Goma User Manual

Release 7.0.0

P. Randall Schunk Rekha R. Rao Ken S. Chen
Duane A. Labreche Amy C. Sun
Matthew M. Hopkins Harry K. Moffat
R. Allen Roach Polly L. Hopkins Patrick K. Notz
S. A. Roberts Kristianto Tjiptowidojo
Andrew Cochrane Weston Ortiz

Mar 02, 2022

CONTENTS

CHAPTER

ONE

GOMA USER MANUAL

1.1 Introduction

"*Goma*," which means rubber, gum, or elastic in Spanish, is a two- or three-dimensional finite element program currently being advanced and specialized for the analysis of manufacturing flows and related processes that involve one or more transport fields, i.e., any combination of heat, mass, momentum (solid and fluid) and species transport fields. Specifically, the processes for which *Goma* is suited are those which contain free or moving boundaries between dissimilar materials or phases. Whether determining the position of an interface whose motion is governed by the underlying physics of the problem, or prescribing the dynamics of a boundary according to user specified kinematics or geometry, the multiphysics approach on which *Goma* is based allows for rapid convergence to the solution. Unique features which make this possible include: (1) a Lagrangian-Eulerian solid mechanics module for mesh motion, (2) energy and chemical species transport modules incorporating convection, diffusion and reaction, (3) fluid momentum transport modules that are fully and mutually coupled, particularly with the mesh motion module through an analytical Jacobian matrix, (3) a Newton-based solution algorithm (full and modified) which exploits that Jacobian matrix, and (4) a structure which allows for different physical descriptions of different materials in the same problem, i.e., conjugate problems. The scope of potentially accessible problems defined by the interaction and close coupling of the individual field equation sets is partially shown in Figure 1 (note that missing from this figure are the fully coupled, partially saturated porous deformable media module and overall variable density mass balance modules). The analytical Jacobian matrix which provides the coupling facilitates a range of computer-aided nonlinear analyses such as parametric sensitivity (stability), design, and optimization as it provides the building blocks (through chain-rule differentiation) for evaluating sensitivities of process variables to processing conditions.

Goma originated in 1994 from an early version of *MP_SALSA* (Shadid, et. al., 1995), a finite element program designed to simulate chemically reacting flows in massively-parallel computing environments. As a point-of-departure, *Goma* was originally extended and adapted to free and moving boundary problems in fluid mechanics, heat transfer, and mass transfer. By virtue of a novel mesh motion algorithm based on Lagrangian solid elasticity, many multiphysics problems involving nonlinear elasticity and viscoplasticity in combination with other transport phenomena are now accessible. The detailed algorithm and underlying physical principles of the moving mesh scheme together with several advanced examples from capillary hydrodynamics, melting and solidification, and polymer processing may be found elsewhere (Sackinger, et. al., 1995; Cairncross, et al., 1995; Chen, et. al., 1995; Cairncross, et. al., 2000; Baer, et. al., 2000; Schunk and Rao, 1994; Bertram, et. al., 1998; Schunk, et. al., 2002).

Since the original publication of the GOMA 2.0 manual (see Schunk, et. al., 1998) work has further focused on concentrated chemical species transport (neutral and charged species) and Eulerian front tracking schemes for large material deformation problems. As in all other developments, these capabilities are being implemented in a fully-coupled way using Newton's method. A concerted effort to bring these capabilities to bear on real-life problems has led to the addition of many esoteric features that address capillary wetting, phase change, charge neutrality, multicomponent species transport, and a host of other physical features. The best way to survey the available features is to consult the large library of reports, technical memoranda, tutorials, and other advanced feature manuals (e.g. Gates, et. al., 2000; Schunk, et. al., 1998; Rao, et. al., 2001; see *Goma* Documentation List in the Appendix), most of which are linked together with this manual in the CD version of the *Goma* Document System currently under development.

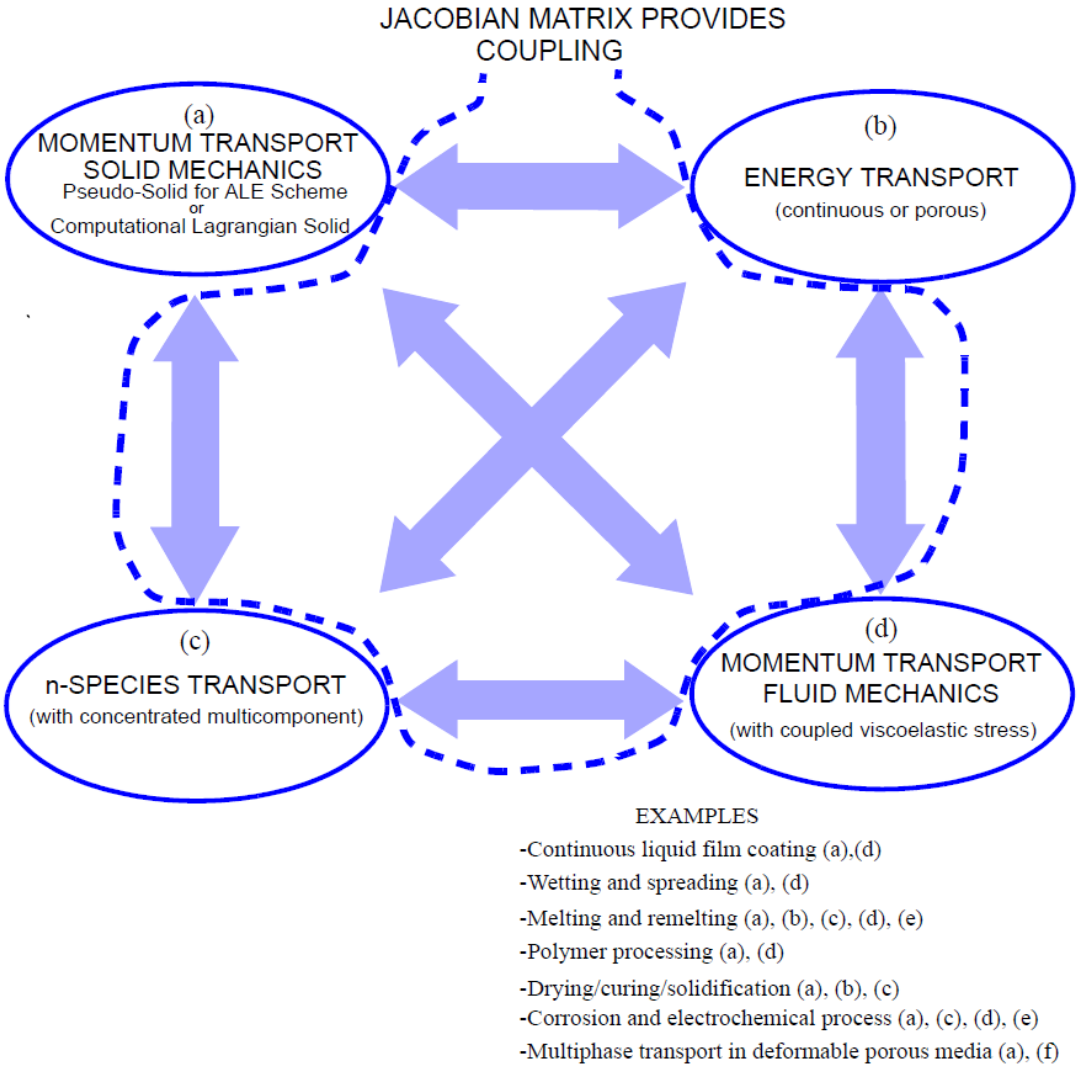


Fig. 1: Main physics modules of Goma, their coupling and examples of potential applications.

Most recent developments, from 2006 through 2012, that are noteworthy are an extensive library of thin-shell physics/equations and accompanying boundary conditions, triangle and tetrahedral elements, phase-field modeling, parallel processing improvements and more. On the thin shell equations, the capability is fully coupled with continuum element equations. We have implemented theory and equations for Reynold's lubrication (laminar or turbulent), thin-shell energy, thin-porous media, and surface rheology.

The purpose of this report is to provide a practical introduction and reference to *Goma*; to introduce the user to the range of options available in *Goma*; to show how easily the code may be adapted to investigate novel situations; and to provide a link to several simple illustrative examples as a tutorial and as a demonstration of the overall utility of the program. By design this is a reference manual which is best navigated together with a tutorial on the class of problems being addressed. It is recommended that perusal be undertaken section by section, consulting the individual input records as needed for a given problem.

1.2 Background Information

1.2.1 Program Features

Free and Moving Boundary Capabilites

Goma is a general purpose program designed for the solution of both steady and transient, two and three-dimensional problems involving heat, mass, and momentum (solid and fluid) transport. A unique feature is the treatment of all boundaries and interfaces as *free* (position unknown) or *moving* (position unknown or prescribed, but variable). If the material domain of interest is a solid, a Lagrangian formulation (i.e., the computational mesh follows the motion of material) of the momentum equations naturally leads to mass conservation and a natural parameterization of the boundaries and interfaces as material surfaces. If the material domain of interest is a fluid, then an Arbitrary-Lagrangian-Eulerian (ALE) formulation allows the boundaries to respond to constraint equations, hereafter referred to as *distinguishing conditions*. These conditions are responsible for determining the location of all boundaries and interfaces, providing the necessary mathematical closure of the system of equations governing the free boundary problem. Distinguishing conditions available to the user fall into several classes, as described below.

Since publication of the *Goma* 2.0 manual in 1998 (and more recently the *Goma* 4.0 manual in 2002), the ALE formulation has been extended to solid-material regions (viz. the TALE algorithm, Schunk, 2000) and purely Eulerian front tracking schemes based on the method of level-sets have been incorporated for free surfaces with large deformations; moreover, both algorithms have been implemented in a completely-coupled way. Of course Eulerian schemes are inherently transient and less accurate in capturing interfacial physics, even though they are more robust and even optimal for a certain class of problems. It is fair to say that of all the available mechanics codes, *Goma* provides the greatest breadth of free and moving boundary tracking formulations and options.

With regard to the ALE algorithms, the fully-implicit, pseudo-solid, unstructured mesh deformation algorithm sets *Goma* apart from other finite element programs. All surfaces, internal and external, together with other geometric features such as corners and junction points, are permitted to move as part of the algorithm. The movement of boundaries, interfaces, and geometric features is dictated by a weighted residual statement of the distinguishing conditions, whether based on specific physical constraints or arbitrary conditions described by the analyst. The internal mesh deforms as if it were embedded in a deforming elastic solid continuum; with the mechanics of the solid governed by either infinitesimal (linear) or finite (nonlinear) deformation theory. Through Newton's method, the deformation of the mesh is determined simultaneously with all of the other physics of the problem.

The key connection between the mesh deformation and the physics of interest is accomplishedthrough a library of distinguishing conditions. Currently, those conditions include (a) kinematic (material surface of a fluid), (b) isotherm (phase transition temperature, such as melting), (c) isoconcentration and (d) geometric (either smooth plane curves or fixed point specifications). As part of the required input for *Goma*, the analyst specifies the associations between the particular distinguishing conditions and corresponding sets of material points of the initial pseudo-solid used to embody the mesh. Chapter 4 describes this process in more detail. Essentially, the algorithm causes smooth boundaries of the pseudo-solid

to slide tangentially in a "frictionless" fashion. Further details of this algorithm and the corresponding equations can be found in several references (e.g., Sackinger, Schunk, and Rao, 1995).

Coordinate Systems and Frames of Reference

Coordinate systems accessible through this version of *Goma* include two-dimensional and threedimensional Cartesian coordinates, cylindrical coordinates for axisymmetric problems, spherical coordinates, and a swirling option for two-dimensional axisymmetric problems with a (swirling) velocity component in the third dimension. A limited framework has been built within *Goma* to use arbitrary orthogonal curvilinear coordinate systems, but this has not yet been extensively tested. As for frame of reference, all conservation equations are cast in an inertial frame (viz. nonaccelerating) but with extensions to allow for arbitrary frame velocities that may or may not be related to the material motion. Hereafter, when we refer to the frame/mesh motion type to be of the *Eulerian* variety, we mean the mesh is fixed with respect to all material motion, which basically means it is fixed in the laboratory frame. For now, we allow this frame of reference for fluid systems and are researching ways to allow this frame for solid systems. The ALE frame of reference, as mentioned above, allows for independent mesh motion in the interior of the domain, but seeks to maintain a material frame of reference on the boundary. This means that the mesh will move to accommodate material boundary motion. Currently, the ALE frame is allowed for all classes of materials (cf. Schunk, 2000). Finally, a pure Lagrangian frame of reference implies that our mesh moves with the material. This formulation is quite common in solid mechanics and is one advocated here for truly solid regions.

Problem Physics and Thermophysical Properties

This brief section summarizes the physics capabilities in **Goma** and the thermophysical properties and constitutive equations available to the user. The rest of the manual is designed to greatly expand on all material parameter options, boundary condition options, and equation options; perusing Chapter 4 and Chapter 5 is recommended to extract more detail.

The class of problems treated by **Goma** are those described by any one or a combination of the incompressible form of the momentum conservation equation for generalized Newtonian fluids, the momentum conservation and differential stress constitutive equations for viscoelastic fluids, saturated and unsaturated flow equations cast for rigid or deformable porous media, the energy conservation equation, the equations of quasi-static equilibrium of an elastic solid, and any number of additional or auxiliary species convection-diffusion-reaction equations. **Goma** has been tested with the following types of fluid mechanics, solid mechanics, and heat transfer problems: (a) mixed convection with mesh parameterization of an isotherm, (b) melting, with a parameterization of the liquidus and solidus isotherms, (c) coating and related flows (slide coating, curtain coating, etc.), (d) polymer processing (viscoelastic) flows (e.g. fountain flow, planar and axisymmetric extrusion, simple mold filling, contraction flow), (e) neutral or charged species transport in multicomponent concentrated systems, (f) partially saturated flow in poroelastic systems, (g) suspension flows, (h) drying and shrinking of gelled polymer films (with creep and elastic recovery), and (i) microfluidic systems with fluid-structure interaction (e.g. MEMS device performance).

Thermophysical properties in the bulk for all equations may be taken as constant or variable, with dependencies on any of the dependent and independent variables of the problem. General property variation models of this sort can be implemented with a user-defined subroutine capability. Moreover, a growing number of often-used standard models are supported within the core routines. These include a Carreau-Yasuda model for the generalized Newtonian viscosity and a Boussinesq source term for the fluid momentum equation that provides a means for simulating flows with thermal and solutal buoyancy. A plethora of other constitutive models and properties are available, including viscoelasticity, elasto-viscoplasticity, nonFickian diffusivity, etc.

To enhance the capability for modeling problems in capillary hydrodynamics, e.g., coating flows, a boundary condition expressing the normal stress balance for two-dimensional Cartesian and axisymmetric problems has been implemented and tested. When capillary forces are activated, a pressure jump term (proportional to the mean curvature) is added to the normal component of the momentum flux balance at specified fluid material interfaces in a natural fashion. At three-phase boundaries (points in two dimensions) a contact angle condition and a surface tangent force condition may be applied. The former is used in place of a specified position on the mesh motion equations and is best used to set static and dynamic contact angles, and the latter is an additional endpoint force which is added to the momentum balance,

necessitated because the curvature term is integrated by parts. The current version of *Goma* also includes the ability to model tangential shear forces along capillary surfaces, i.e., those originating from surface tension gradients caused, for example, by variations in temperature or species concentration. To access this capability requires a constitutive equation for the surface tension. A powerful low-level capability has been implemented which allows the user to select which degree of freedom, or variable, is associated with a particular boundary condition. Such a capability is useful at dynamic contact lines, where it is often desirable to replace the liquid-phase momentum equations with auxiliary constraint conditions.

Generalized interphase boundary conditions that allow for discontinuous field variables are supported through a multiple degree-of-freedom capability. The prime targets for this capability include flowing vapor-liquid equilibrium problems for which there are concentration and velocity jumps between phases due to change in density and solute partitioning through the phase diagram and multiphase/multicomponent corrosion problems. A series of boundary conditions which allow for the application of ideal and non-ideal vapor/liquid equilibrium (e.g. Raoult's law and Flory-Huggins theory), latent heat release/adsorption, and discontinuous velocity components due to evaporation/condensation have been implemented. In the future this capability can be extended to thermal contact resistance, which often involves a temperature jump at an interface.

Recently the solid mechanics module of *Goma*, which was originally installed as a part of the pseudo-solid ALE mesh motion algorithm, has been exploited to solve problems in transport in deformable porous media and other outstanding problems of elastohydrodynamics. For modeling flow in non-deformable porous media, the Brinkman terms in the fluid momentum equations (cf. Gartling, et. al., 1996) may be activated. Since *Goma* 2.0, generalized Darcy transport equations for multiphase components (solid, liquid, gas) have been added and can be used for simulations of deformable poroelastic media. For incompressible but deformable solids, a pressure term was added to the solid momentum balance (e.g. rubber). In continuous shrinking or swelling solids, the dilation is proportional to changes in solvent concentration. In deformable porous media, the solid deformation is coupled to the pressure in the fluid-filled interstices of the porous matrix. Several boundary conditions exist to apply normal tractions (i.e. compressive, tensile, or shear boundary forces) to solid surfaces. To effectively simulate coupled fluid/solid interaction problems, boundary conditions which balance the surface tractions exerted by the liquid and solid phases at the common interface have been incorporated as have been the appropriate interface impregnation/expulsion conditions at boundaries between porous and continuous media.

A complete rewrite of the species transport equations has been undertaken since the release of *Goma* 2.0 that allows for generalized phase/species formulations on multimaterial problems. Accommodating an arbitrary number of species, each of which can exist in an arbitrary number of phases, was the goal of this development in order to model corrosion and charged species transport.

Of course there are many more material property models and constitutive equations, specialized boundary conditions, and more esoteric differential equations that can be solved for just about any mechanics problem. Many of these capabilities are not cited in this manual because they were under development at the time of publication. Interested readers should inquire about the status of the following capabilities: generalized solid-model geometry features, wetting and spreading models for Eulerian front tracking schemes, Eulerian/Eulerian fluid-structural interaction capability, multiphase porous energy equation, Generalized surface and volume user-defined Lagrange multiplier constraints, and much more.

Advanced Capabilities

Several developments in *Goma* that enable advanced engineering analysis of complex systems have been completed since the last major release. These developments include a complete, generalized capability of automated parameter continuation (zeroth-order, first-order, arclength, multiparameter, user-defined parameter continuation, etc.) using the LOCA library (Salinger, et. al., 2002), linear stability analysis of any dynamic system using normal modes, and augmenting condition capability. It is recommended that the user consult a separate manual (Gates et. al., 2000; contact authors for a more recent version) for a complete user description of these features. The input record sections required to activate these features are not covered in this document.

1.2.2 Numerical Methods

With over 150 different boundary conditions for 70 plus differential equation types, *Goma's* algorithms are very extensive for any brief discussion. In this section we simply point out the foundation algorithms. A developer's manual, advanced capabilities manual, and tutorial memos can be consulted for more details (see *Goma* Document List in the Appendix for the citations.).

Goma is based primarily on the Galerkin/finite element method. The element library currently includes (in two dimensions) 4- and 9-node isoparametric quadrilaterals (i.e., Q1 and Q2 interpolations) with available interpolations for linear discontinuous (P1) or piecewise constant (P0) variables, and (in three dimensions) 8-node isoparametric hexahedral elements and 27-node bricks, also available with piecewise constant interpolations. The overall solution algorithm centers around a fully-coupled Newton-Raphson iterative scheme for solving the nonlinear algebraic equations which results from the finite element discretization. That is, all active equations and boundary conditions are solved simultaneously in a single matrix system at the same time plane and during the same Newton iteration. The sparse matrix system is stored in a central element-level matrix data structure that is injected into one of three sparse matrix formats as dictated by the matrix solver chosen. The three formats are modified sparse row, MSR or compressed row format (Hutchinson, et. al., 1995, Schunk and Shadid, 1992), the variable block row, or VBR, format (see Heroux, 1992), or the frontal-solver element-level format (cf. Hood, 1976). If the matrix system is not too poorly conditioned, then iterative solvers of the generalized preconditioned conjugate gradient-type can be used to solve the system (see Tuminaro, et. al., 1999, Schunk and Shadid, 1992). A new matrix-services/solver-services library known as TRILINOS (http://www.cs.sandia.gov/Trilinos), has been installed to handle all iterative solver and preconditioner options. This package has greatly extended the robustness of iterative solvers to the class of problems that *Goma* solves. Virtually all problems and all finite element formulations are now solvable with these iterative schemes (see Schunk, et al., 2002). If all else fails, *Goma* deploys a suite of direct solvers that, even though not always efficient for large threedimensional problems, will always get a solution at the current Newton iteration. These solvers are known as Sparse 1.3 (lu), a classical LU decomposition (Gaussian elimination) method, and two frontal solvers, Umfpack (umf) and front; these are discussed in the next section.

The Galerkin least squares (GLS) method for pressure stabilization of Hughes and Franca (1987) has also been added to *Goma*. The GLS method adds the momentum residual, weighted by the gradient of the Galerkin weight function, to the standard Galerkin continuity equation, thus providing a diagonal term for the pressure. This is a first-order convergent and consistent method that enables the use of iterative solvers for incompressible equations over the entire range of Reynold's numbers.

The overall differential-algebraic system of equations may be advanced in time with implicit time-integration techniques (simple backward Euler and Adams-Bashforth predictor, trapezoidal corrector algorithms for fluid systems, species transport and energy transport; and Newmark-Beta algorithms for solid dynamics). Time marching offers an alternative, albeit indirect, route to attaining solutions to steady equations, as well as providing the capability of simulating process transients directly. Automatic time step control based on current truncation error is also available.

Perhaps the most complicated part of the algorithm is the construction of the Jacobian sensitivity matrix. Because the mesh point positions are actually unknowns in a free or moving boundary problem, that matrix must include sensitivities of each weighted residual equation with respect to each of the mesh variable unknowns that can affect the value of the residual. Unfortunately, almost every term of the bulk equations and many boundary conditions contribute to this sensitivity. This occurs mainly through gradient operators and surface normal and tangent vectors (see Kistler and Scriven, 1983) and through dependencies on mesh position of the determinant of the elemental Jacobian transformation matrix that maps between a fixed unit element and any element in the computational domain. Great care has been taken to include analytical expressions for all of these mesh sensitivities. However, some of this task inevitably falls to the user when implementing user-defined boundary conditions, material property models, and constitutive equations, particularly when any of these quantities depends directly on spatial position or spatial gradients of other variables. In order to maintain the strong convergence properties of Newton's method, these sensitivities must be specified in those user-defined routines. To aid in this task, a debugging option is available which computes a numerical finite-difference approximation of the global Jacobian matrix and compares it with its analytical counterpart. This tool enables users and developers to check the consistency of newly-created equations (whether bulk or boundary constraints) with their corresponding analytic Jacobian contributions.

1.2.3 Portability, Software Library Infrastructure, and Code Accessibility

Goma is written in the C programming language (specifically Kernighan and Ritchie, 1988, C with some ANSI extensions). It has been ported to a number of UNIX platforms including Solaris and Linux, with the Linux Enterprise-4 version being the most actively maintained. Most recent versions are aimed at Red-Hat RHEL5 and RHEL6 levels, almost exclusively. Many of the machine dependencies in the program have been isolated using C preprocessor directives. Some of the machine dependencies that occur in the I/O routines are insulated from the user by software libraries. Building *Goma* requires EXODUS II v2.02 (Schoof and Yarberry, 1994), SPARSE 1.3 (cf. Kundert and Sangiovanni-Vincentelli, 1988), NetCDF v2.3.2 (Rew, et. al., 1993) libraries, Umfpack direct solver libraries (Davis and Duff, 1997), and the TRILINOS 10.0 library (Tuminaro, et. al., 1999; **http://software.sandia.gov/trilinos**). The first of these is part of the SEACAS system at Sandia National Laboratories (Sjaardema, 1993); the latter two libraries are available publicly. Parallel processing is enabled by OPEN-MPI. The user should consult the build instructions for the most recent library revisitions. The most updated library needs are also made clear in the *Goma* makefile: Makefile. There are special versions of this makefile for building for the test suite (Makefile_guts) and debug mode (Makefile_debug). These are the most general makefiles that are deployed. Generally, pre- and post-processing is performed outside of *Goma*, although some post-processing of results is available within the program. This separation of the functionality permits the use of alternative solid-modeling and mesh-generation software and visualization packages of choice, insofar as they may be interfaced with the EXODUS II finite element data model.

Pre-processing options include mesh generation via CUBIT (**http://cubit.sandia.gov**), PATRAN (PDA, 1990), and SolidWorks (**www.solidworks.com**). The latter two require special plug-ins. These mesh generators currently support and will output a finite element database in the EXODUS II format.

Post-processing options include BLOT (see the SEACAS distribution, Gilkey and Glick, 1989), Paraview (**www.paraview.org**), and Ensight (**www.mscsoftware.com.au/products/software/cei/ ensight**).

Since *Goma* is built around the EXODUS II finite element data model, there are numerous options available for communication with other analysis codes that also exchange data via the same EXODUS II data model. Recent modifications to *Goma* permit not only the initialization of unknown values from an EXODUS II file, but also the ability to incorporate field variables into the analysis that are not unknowns. For example, the quasi-static and dynamic electromagnetic fields from codes such as ALEGRA can be used to compute electric fields and current fluxes on a specified finite element mesh that are input to *Goma* through the EXTERNAL FIELD data card.

1.3 Code Structure and I/O

1.3.1 Files for Data Input

The *Goma* file I/O structure is diagrammed in Figure 2. Input to the program is divided into six categories: (1) command-line options, (2) problem description file, (3) material files, (4) ASCII continuation/restart file, (5) EXODUS II database file, and (6) sundry material property or boundary condition table lookup files. *Goma* is basically set up to run in batch mode, i.e., no input is required on the command line or after the run command is issued. There are, however, several command-line switches which can be used to redirect I/O, control the level of I/O, and activate debugging options.

The *problem-description* file is by default called “input” but can be renamed with the -i switch on the command line. A version of this file is also output as an “echo” file, viz. a prefix “echo” prepended to the input file name. The echo file is used to verify input into goma, as it clearly states all default settings for the input file and material files. . The input file itself contains the general description of the problem and directions to *Goma* on how to solve it (see Chapter 4). The file is split into thirteen sections: (1) File Specifications (Section 4.1) which directs I/O, (2) General Specifications (Section 4.2), (3) Time Integration Specifications (Section 4.3), (4) Continuation Specifications (Section 4.4), (5) Hunting Specifications (Section 4.5), (6) Augmenting Condition Specification (Section 4.6), (7) Solver Specifications (Section 4.7), (8) Eigensolver Specifications (Section 4.8), (9) Geometry Specification (Section 4.9), (10) Boundary Condition Specifications (Section 4.10), (11) Rotation Specifications (Section 4.11), (12) Problem Description (Section 4.12), and (13) Post Processing Specifications (Section 4.13); this latter section includes breakouts for fluxes and data (Section

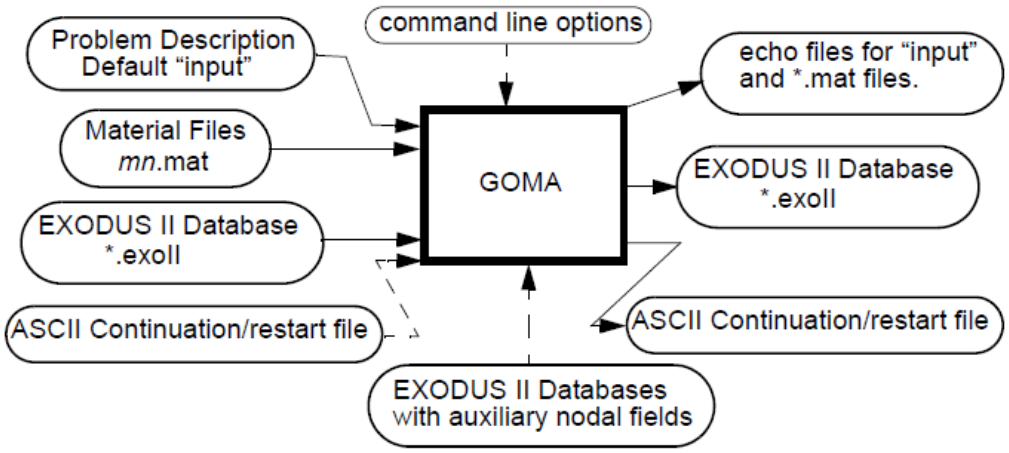


Fig. 2: I/O structure for *Goma*. Dashed lines indicate that the files or commands are not required.

4.14), particle traces (Section 4.15) and for volume-based integrals. The file format is described in detail in Chapter 4. Incidentally, the structure of the data input routines is divided roughly along the same lines as the input data file itself.

The *material description* files (using the nomenclature "[material name].mat") contain all material property data and material property model and constitutive model specifications. The names of these files are specified in the problem description file. The format of these files and the available options are described in Chapter 5. Note that these files are also reproduced as output as "echo" files, with all default settings specified.

The *ASCII continuation/restart files* (may have any name) contain an ASCII list of the solution vector (values of field variables at nodes), which can be used as an initial guess for successive runs of *Goma*. The names of these files are specified in the problem description file, but may be changed with the -c (for input) or -s (for output) command-line options. These restart files are "recyclable", in the sense that output from one *Goma* simulation may be used as input to another *Goma* simulation under certain restrictions.

The *EXODUS II database files* (may have any name but generally end in ".exoII") contain a description of the finite-element structure for the current problem. All EXODUS II files contain a definition of the mesh, material blocks, and boundary sets. In the case of input EXODUS II files created from mesh generator output, this is the sole content of the file. Output EXODUS II database files contain a clone of the input EXODUS II mesh information and also contains the nodal values of all field variables in the solution. The names of these files are specified in the problem description file, but may be changed with the -ix (for input) or -ox (for output) command-line options. The only EXODUS II file required when running *Goma* is the one containing the current problem mesh. All others are either output for postprocessing or used to supply auxiliary external fields (e.g. magnetic fields).

1.3.2 Command-Line Arguments

Goma can be run using only the input files (all four listed above) to describe the problem and to direct the input and output; in this case *Goma* is run using the command "goma" without any arguments. However, command-line arguments offer additional flexibility for redirecting input or output and for adjusting common run-time parameters. The general command line for running *Goma* is:

```
$ goma [-nd] [-se fn] [-so fn] [-i fn] [-c fn] [-s fn] [-ix fn] [-ox fn] [-d int]
       [-n int] [-r dbl] [-a args] [-restart fn] [-h] [-ts dbl] [-te dbl] [-cb dbl]
       [-ce dbl] [-cd dbl] [-cn int] [-cmin dbl] [-cmax dbl] [-cm int] [-ct int] [-c_
↪bc int]
```

(continues on next page)

(continued from previous page)

```
       [-c_df int] [-c_mn int] [-c_mp int] [-bc_list] [-v]
```

Here *fn* denotes "file name", *int* denotes "integer", *dbl* denotes "float or double" and *args* denotes multiple sub-options or file names. The input line is parsed into options, which are preceded by a single hyphen (-) and arguments, which normally are *fn*, *int*, or *dbl* not preceded by a hyphen. The default, if no options are specified, is the input option (e.g. "goma input.alt" is the same as "goma -i input.alt"). The following is a list of the command-line options and their descriptions (two ways are shown to specify each option, an abbreviated and a verbose form).

-a args, -aprepro args Preprocess input files through the APREPRO preprocessor [with args as arguments to APREPRO] before reading into *Goma*. With this option, *Goma* performs a UNIX system() call to run APREPRO which will preprocess the input file and the material data files. The APREPRO input file is preprocessed from "input" or the filenamespecified by the -input option and written to "tmp.input". Likewise, the material data files are preprocessed from "[material name].mat" to "tmp.[material name].mat". After the "-a" on the command line, options for APREPRO are preceded by two hyphens (–). For example, the command line "goma -i *input.pre* -a CONSTANT1=0.2 –vd" will preprocess "input.pre" and the material data files specified in *input.pre* using APREPRO, and will pass the argument *-vd* (which prints version number and values of all variables to the screen) and CONSTANT1=0.2 (which sets the variable CONSTANT1 equal to 0.2 for preprocessing) to APREPRO; the preprocessed files will be "*tmp.input*" and "*tmp*.[material name].*mat*".)

-c fn, -contin fn Change the name of the ASCII continuation/restart input file (specified in Problem-Description File) to *fn*, (e.g. "goma -c *old.soln.dat*" uses the file "*old.soln.dat*" as the ASCII input file). Note that this option has no effect if the initial guess is not read from the ASCII file, i.e. unless "*Initial Guess = read*" is specified in the input file.

-d int, -debug int Change the debug flag to *int*. This option is convenient when debugging and the user wants to see more output from *Goma*. (e.g. "goma - d -2" will run *Goma* with the Debug_Flag set to -2). Higher values generally produce more output.

-h, -help Prints a helpful message with brief descriptions of these command line options.

-i fn, -input fn Redirect *Goma* to read the problem description file from *fn*. The normal default option is to read from a file named "*input*".

-ix fn, -inexoII fn Redirect *Goma* to read the input EXODUS II database file (often called "*in.exoII*") from *fn*.

-kway Use builtin METIS KWAY decomposition

-rcb Use builtin METIS recurisve bisection decomposition

-e, -external_decomp Use external decomposition (such as through SEACAS decomp, or already decomposed files)

-n int Change the maximum number of Newton iterations to *int*. This is especially convenient for setting the number of iterations to zero so that *Goma* just runs the postprocessor on the set of input data.

-nd, -nodisplay Do not display the run-time information on the screen. With this option, *Goma* sends the stdout and stderr output to temporary files that are removed at the end of the run. This command takes no arguments.

-ox fn, -outexoII fn Redirect *Goma* to write the output EXODUS II file (often called "*out.exoII*") to *fn*.

-r dbl, relax dbl Change the value of the Newton relaxation parameter to *dbl*. This is convenient if a few Newton steps with relaxation are desired before using full Newton. (e.g. "goma -r 0.1" will use Newton's method with updates one-tenth of the normal value.

-s fn, -soln fn Redirect Goma to write the output ASCII file (normally called "*soln.dat*") to *fn*.

-se fn, -stderr fn Redirect the standard error from *Goma* to *fn*. This output is comprised of more urgent diagnostic error and timing messages.

-so fn, -stdout fn Redirect the standard output from *Goma* to *fn*. This output is comprised of less urgent informational messages.

-ts dbl Start time of simulation.

-te dbl End time of simulation

-cb dbl Continuation: Start value (see Gates et al., SAND2000-2465)

-ce dbl Continuation: Final value (see Gates et al., SAND2000-2465)

-cd dbl Continuation: Path step, ds (see Gates et al., SAND2000-2465)

-cn dbl Continuation: Max number of path steps (see Gates et al., 2000)

-cm int Continuation: Method (see Gates et al., 2000)

-ct int Continuation: Type (see Gates et al., 2000)

-c_bc int Continuation: Boundary condition ID (see Gates et al., 2000)

-c_df int Continuation: BC Data Float ID (see Gates et al., 2000)

-c_mn int Continuation: Material ID (see Gates et al., 2000)

-c_mp int Continuation: Method property ID (see Gates et al, 2000)

-bc_list Continuation: Method property ID (see Gates et al, 2000)

-v -version Output goma version

-petsc string Use petsc options in quoted string see Solution Algorithm card for more information

---

Note: To get the most up-to-date list, simple issue the* "goma -h" *command at the command line. Also note that the continuation input parameters are explained in the Advanced Capabilities Manual (Gates et al. 2000 or newer version).*

The primary purpose of the command-line options is to allow the user an easy way to redirect the input and output of Goma or to quickly change problem specifications. Most of the options are overrides of information in the problem description file, so in some cases it may be easier to edit the problem description file than to use command-line arguments.

1.4 Problem Description (Input File)

The **input file** for *Goma* contains the overall description of the problem to be solved together with instructions on solution strategy. The file (cf. sample in Figure 3) is split into sixteen sections: (1) File Specifications (Section 4.1) which directs I/O, (2) General Specifications (Section 4.2), (3) Time Integration Specifications (Section 4.3), (4) Continuation Specifications (Section 4.4), (5) Hunting Specifications (Section 4.5), (6) Augmenting conditions (Section 4.6), (7) Solver Specifications (Section 4.7), (8) Eigensolver Specifications (Section 4.8), (9) Geometry Specifications, (Section 4.9) (10) Boundary Condition Specifications (Section 4.10), (11) Rotation Specifications (Section 4.11), (12) Problem Description (Section 4.12), (13) Post Processing Specifications (Section 4.13), (14) Post Processing Fluxes and Data (Section 4.14), (15) Post Processing Particle Traces (Section 4.15) and (16) Volumetric Integration (Section 4.16). Sections (1), (2), (3), (7), (10), (12), and (13) are required. The rest are optional, depending on the problem type being solved.

Each section in this chapter discusses a separate part of the input file specification and it indicates the data cards or input records that may be used, followed by the options available for each individual record (or line in the file) and the necessary input data/parameters. All input data are specified in a free field format with successive data items separated by blanks or tabs. In this version of the user's manual, a new format has been instituted in which each record is presented in a template structure. This template has eight parts: 1) a title, which is also the card name, 2) a syntax, which is enclosed in a framed box and shows the proper contents of the card, 3) a Description/Usage section, which presents the user options and descriptions of proper input records, 4) an Example, 5) a Technical Discussion to provide relevant information to help

the user understand how to select from among various options or how to properly determine the desired parameters, 6) a Theory to provide an understanding of the physics and mechanics that have been implemented or are being exercised, 7) a FAQs section to present important user experience, and 8) a Reference section to identify citations and/or provide background information to the user. This is a more lengthy but a more complete form for documenting and instructing users of *Goma*.

The syntax entry denotes a unique string for each input record which *Goma* parses in the input file. All words in these unique strings are separated by a single white space and because the code parses for these exact strings, the parser becomes case sensitive. The identifying string for a particular specification is followed by an '=' character. Following this character will be all additional data for that record, if any. In the syntax box, this additional data is symbolically represented by one or more variables with some appropriate delimiters. Typically, the user will find a variable called *model_name* enclosed in **curly braces** '{}'; this would then be followed by a description of specific options for *model_name* in the Description/Usage section. The curly braces indicate a required input and that the user must select one of the offered options for *model_name*. Required parameters, if any, for the model option are enclosed in **angle brackets** '< >', while optional parameters for *model_name* are enclosed in **square brackets** '[ ]'. Following the '=' character, the user may use white space freely between and among the remaining parameters on the command line.

The nature of the input parser allows the user to freely comment the input deck in any way, so long as the character strings in the comments do not contain the exact strings described in this section of the manual at the beginning of the comment line. Simply for the sake of uniformity, it is recommended that a comment card convention be adopted, i.e., placing some delimiting symbol (e.g., $, #, , etc.) at the *beginning* of each comment line. Moreover, employing some of the basic text processing capabilities provided in the ACCESS system (Sjaardema, 1993) makes it possible to connect both the model generation input file, e.g., FASTQ or CUBIT, with the input deck for *Goma*. For example, the *"include"* statement in APREPRO (Sjaardema, 1992) makes it convenient to include geometrical information contained in a FASTQ input file into the *Goma* input file for use with commands like *PLANE* and *SPLINE* that make use of global problem geometry. APREPRO also enables a user to generate customized model parameterizations with the algebraic preprocessing capability (demonstrated in *Goma* Tutorials). Finally, employing a text preprocessor like APREPRO enables the analyst to attach more meaningful labels to entities such as side sets, node sets and element blocks than the internal names (which are simple integer identifiers).

The order of the input cards is significant; omitting a required card will often result in an error message from *Goma*. To avoid such errors, a good strategy is to copy a current version of a working input file and then make changes to it. However, as noted below, some cards are optional. Some file sections, such as boundary condition specification section and equation specification section, are not order dependent, but number dependent, as only the number of boundary conditions or equations which are specified by the "Number of BC" and "Number of EQ" cards will be read (regardless of the number of cards in the file). That is, after the specified number of individual equation or boundary condition cards is read, any remaining cards are ignored. Figure 3 shows a sample problem description input deck, indicating some optional and required cards (lines). All possible cards and card sections are not identified in this figure as they are too numerous. The remainder of this chapter describes each card in detail.

A final note to the user (and reader) of this manual pertains to backward compatibility and obsolescence. There are several input records that have been superseded or have simply been dropped from usage as the software has evolved. Rather than eliminate all of these inputs cards immediately and cause some head-scratching about input cards which exist in users old input decks, the decision was made to leave these cards in the current manual and simply document the fact that they are no longer used (and in some cases why this is so). In the CD version of the *Goma* Document System, these cards will be removed at a future date and no reference made to them again.

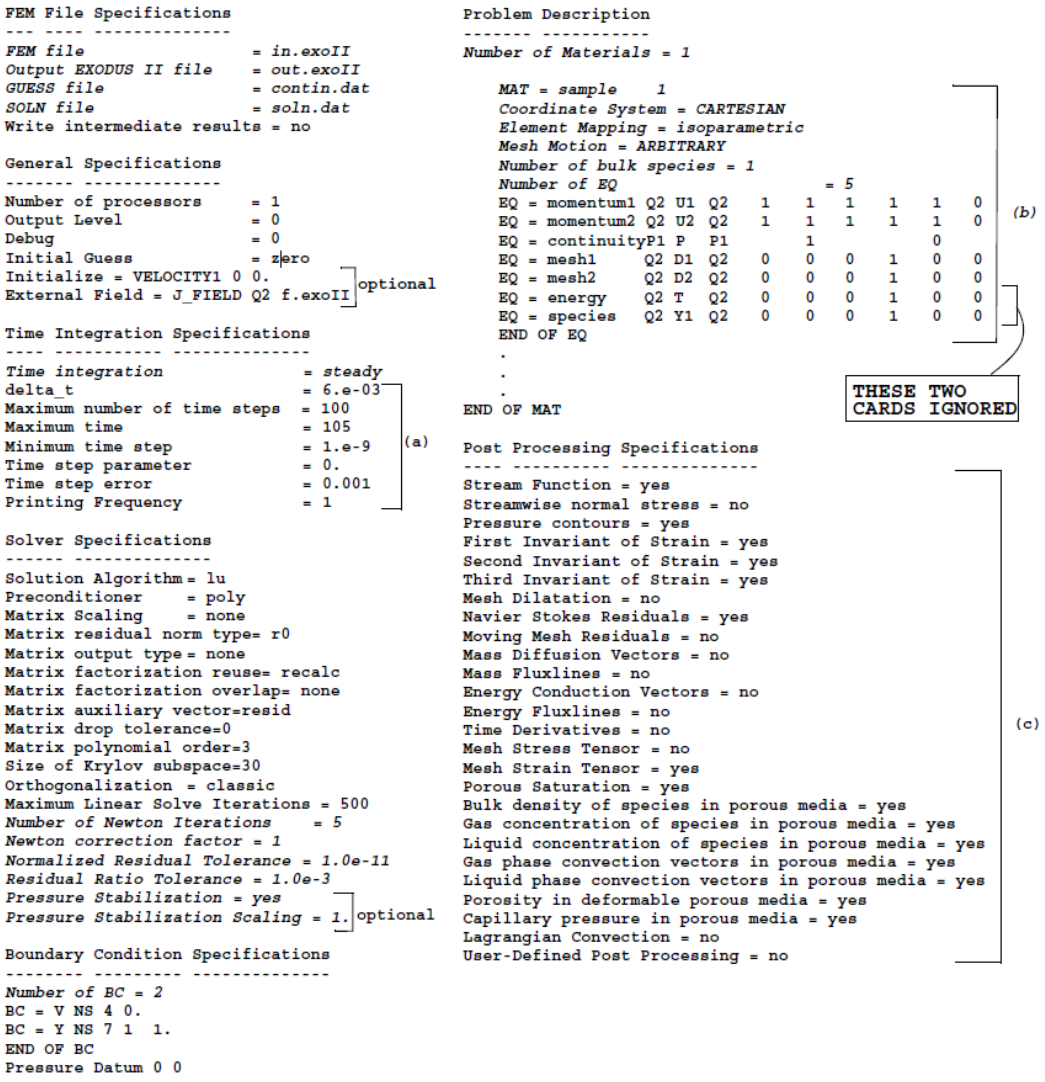


Fig. 3: Sample problem description input deck. Italic type denotes required data cards (lines) and plain type denotes optional cards or cards that in number correspond to the designation above them, e.g., “Number of BC” or “Number of EQ”. (a) These cards are optional if the “steady” option is chosen on the Time Integration card. (b) This group of cards is repeated for each different material block in the EXODUS II database file. (c) These cards are all optional and can appear in any order. Please check this manual for numerous new post processing options.

1.4.1 File Specification

In general, this first section of the main input file is used to direct *Goma* I/O through a series of named external files that contain information about the finite element mesh, the initial guess of a solution vector, and output options for saving solutions for continuation, remesh, etc. The required and optional input records are as follows:

FEM File

```
FEM file = <file_name>
```

Description / Usage

This required card specifies the name of the EXODUS II finite element mesh file. Any EXODUS II file name is permissible, as specified below.

<file_name> A file name of the form *prefix*.exoII. The *prefix* portion is any user-specified alpha-numeric string, which can be used as a problem-type descriptor. Preprocessors and postprocessors (like AVS) might require the ".exoII" suffix so it is a required part of the file designation. The maximum length of the file name is 85 characters.

Examples

Following is a sample card:

```
FEM file = in.exoII
```

Technical Discussion

This file contains the finite element discretization of the problem domain. Finite element mesh files from other preprocessors may be used with *Goma* as long as a translator from the preprocessor's output format to the EXODUS II format is available to the analyst.

References

The EXODUS II format is documented in:

- EXODUS II: A Finite Element Data Model, Schoof, L. A. and V. R. Yarberry, SAND92-2137, Sandia National Laboratories, Albuquerque, NM.

Output EXODUS II File

```
Output EXODUS II file = <file_name>
```

Description / Usage

This required card specifies the name of the output EXODUS II file. Any EXODUS II file name is permissible, as specified below.

<file_name> A file name of the form `*prefix*.exoII`. The *prefix* portion is any user-specified alpha-numeric string, which can be used as an output file descriptor.

This EXODUS II file contains a replica of the input mesh and boundary condition information exactly as it was provided in the *FEM file*, but has appended to it the solution field information appropriate to the problem type. If the name of this output EXODUS II file <file_name> is identical to the name of the input EXODUS II file (as specified in the *FEM file* card), then no replication of the input mesh data is performed and any results are simply appended to it.

Examples

Following is a sample card:

```
Output EXODUS II file = out.exoII
```

Technical Discussion

Although allowed, it is not advisable to make this file name the same as the file name input on the *FEM file* card.

References

The EXODUS II format is documented in:

- EXODUS II: A Finite Element Data Model, Schoof, L. A. and V. R. Yarberry, SAND92-2137, Sandia National Laboratories, Albuquerque, NM.

Guess File

```
GUESS file = <file_name>
```

Description / Usage

This required card identifies the input file that provides the initial guess for the solution vector for continuation or time integration, where

<file_name> Specifies the exact name of the file and can be any file name.

The file <file_name> is read by *Goma* only if the value of the *Initial Guess* (next section on *General Specifications*) card is set to read. The current format of this ASCII file is a list of unformatted floating point numbers (the solution variable followed by the residual value for that degree of freedom) in the order of the unknown map; this is the same format as the file described in the *SOLN* file card. A solution file from a previous simulation may be used.

Examples

Following is a sample card:

```
GUESS file = contin.dat
```

Technical Discussion

This file is typically a copy of the *SOLN file* thus being an exact replica of it. It represents the only way to continue a previous solution from an ASCII file. Typically a continuation proceeds from a converged solution but the result from an intermediate solution could also be used; the user is cautioned about the potential difficulties of restarting from non-converged solution. (See *Initial Guess* card about (re-)starting from a binary file.)

SOLN File

```
SOLN file = <file_name>
```

Description / Usage

This required card identifies the ASCII output file that will provide the initial guess for continuation or time integration, where

<file_name> Specifies the name of the output file, or if no file is desired, a value of **no** or **none** should be entered.

The current format of this ASCII file is a list of unformatted floating point numbers that includes every degree of freedom in the problem in the order specified in the unknown map. Other information (residual for that degree of freedom) may appear beyond the first column of numbers in this file that is sometimes useful in determining the name and location of the corresponding degree-of-freedom. If **no** or **none** is used in place of the file name, no ASCII information is written.

Examples

Following is a sample card:

```
SOLN file = soln.dat
```

Technical Discussion

This file represents the primary ASCII output of the *Goma* solution vector and the primary way to continue or restart a solution from an ASCII file. (See *Write Intermediate Solutions* for related information.) When a continuation run is performed, this file is copied into the file specified in the *GUESS file* input card.

Write Intermediate Results

```
Write Intermediate Results = {yes | no}
```

Description / Usage

This optional card controls the output of intermediate results. The permissible values for this card are

yes The code will output the latest Newton iteration to a file named 'tmp.i.d', where i is the Newton iteration number. The format of tmp.i.d will be similar to the ASCII results data described for the *GUESS file* and *SOLN file* cards. Also, the output EXODUS II database (see the *Output EXODUS II file* card) will accumulate the intermediate iterations as time planes of the solution.

no No intermediate results are written; only the last Newton iteration is written to the file named in the *SOLN file* card, and only the final converged iteration is output to the EXODUS II file.

Examples

Following is a sample card:

```
Write Intermediate Results = no
```

Technical Discussion

This file is useful to guard against machine crashes or accidental job kills, particularly for very large problems, as it can be used to restart a simulation (by using this file as the *Guess file*). The intermediate results in the output EXODUS II database can be a useful debugging tool, giving the analyst the ability to use highly relaxed Newton iterations to see how a free boundary problem diverges.

Write Initial Solution

```
Write initial solution = {yes | no}
```

Description / Usage

This optional card controls the output of an initial solution prior to the start of a time dependent simulation. The permissible values for this card are:

yes This value sets the flag WRITE_INITIAL_SOLUTION variable to "TRUE". The initial solution vector will be written to an EXODUS II file and to an ASCII file (if the number of processors is not greater than DP_PROC_PRINT_LIMIT, currently set to 4 in *rf_io.h*).

no No initial solution is written.

Examples

Following is a sample card:

```
Write Initial Solution = yes
```

Technical Discussion

This option is useful to activate when help is desired in debugging the startup portion of a transient simulation.

External Decomposition

```
External Decomposition = {yes | no}
```

Description / Usage

This optional card controls whether to use the builtin METIS based decomposition in parallel or to use externally decomposed exodus files, such as through SEACAS decomp tool

yes Exodus files are already decomposed and will not automatically be decomposed

no This is the default when built with METIS, internal METIS decomposition will be used. Defaults to RCB when less than 8 processors and KWAY when 8 or more. See Decomposition Type to set preferred internal decomposition.

Examples

Following is a sample card:

```
External Decomposition = yes
```

Technical Discussion

Also available from the command line with `-e, -external_decomp`.

Decomposition Type

```
Decomposition Type = {rcb | kway}
```

Description / Usage

This optional card controls which builtin METIS method for decomposition in parallel

rcb Recurisve Bisection, default when less than 8 processors and this card is not specified

kway KWAY default when 8 or more processors and this card is not specified

Examples

Following is a sample card:

```
Decomposition Type = kway
```

Technical Discussion

Also available from the command line with `-rcb, -kway`.

1.4.2 General Specifications

This section of input records covers additional I/O requests and specifications, including parallel file I/O information, initial-guess directives (viz., whether a restart comes from a neutral file or another exoII file), individual field variable initialization, debugging options, developer diagnostic options, etc. This section and several of its input records are required, as indicated below.

Output Level

```
Output Level = <integer>
```

Description / Usage

This optional card specifies the level of diagnostic information output to the file stderr. The permissible values for <integer> are **0** through **4**, depending on the level of informational (debugging) output desired; higher values of the output level will produce more diagnostic information on the stdout and stderr output channels. The default output level is **0**. Specific output is summarized below.

Level	Results Output
0	No diagnostic output (default).
1	Identifies the degree of freedom, the solution variable, and node at which the maximum value of norm is present.
2,3,4	Currently unused; available for developer output specification.

Examples

Following is a sample card:

```
Output Level = 1
```

Technical Discussion

This specification allows the developer a means to output specific information that would be helpful in diagnosing problems in the software. Currently, the output options are limited.

Debug

```
Debug = <integer>
```

Description / Usage

This optional card specifies the level of information output to files stdout and stderr. The permissible values for <integer> are **-3** through **4**, depending on the level of informational (debugging) output desired; higher values of the output level will produce more diagnostic information output on the stdout and stderr output channels. The default level is 0. Specific results produced for each level are summarized below. The user should exercise caution in using values other than the default for problems with large numbers of unknowns as the volume of information increases very quickly.

Level	Results Output
0	No output (default).
1	Logs activity as the code does problem setup, including setting addresses and sizes, boundary-condition (BC) conflictresolution information, and identification of the rotation conditions at every node with a boundary flag. Prints out surface boundary integral setup information. Lists matrix and solver information for each solution step.
2	Prints same information as level 1, plus provides a summary of BC type information for each BC and logs the beginning and end of matrix fill operations.
3	Prints same information as level 2, but also prints a list of variables/unknowns at each node.
4	Prints same information as for level 3.
-1	Logs activity as the code does problem setup and prints out surface boundary integral setup information as is done for mode 1. Triggers a comparison of the analytical Jacobian and the numerical Jacobian in un-scaled form, which can be used to check the compatibility of the analytical residual equations and Jacobian. Prints results only if the analytical and numerical Jacobian are different. Does not solve any equations; terminates after Jacobian print out.
-2	Same initial information as for level -1. Triggers a comparison of the analytical Jacobian and the numerical Jacobian scaled by the sum of each row of the analytical Jacobian (this helps suppress small errors in large Jacobian entries). Prints results only if the analytical and numerical Jacobian are different.
-3	Similar to level -2 except each row is scaled by the diagonal value which is usually the largest. Prints results only if the analytical and numerical Jacobian are different.

Examples

Following is a sample card:

```
Debug = -2
```

Technical Discussion

For options -1, -2, -3, viz. numerical Jacobian checking, the user must take care when interpreting the cited differences in the numerical and analytical Jacobian. The comparison is made by perturbing each variable and comparing the numerical Jacobian computed between the perturbed and unperturbed states to the analytical Jacobians at the two states. A difference is deemed significant if the numerical Jacobian falls outside the band between the two analytical values with an additional allowance for roundoff error. It is the roundoff error in the residual that is the most difficult for the Jacobian checker to estimate. This is particularly true for problems with zero initial conditions since it is impossible to determine the scale of a velocity, for example, if all the values of velocity are zero. For this reason, it is often better to use a nonzero initial condition or a scaled problem (with values order unity) when using the Jacobian checker.

Currently, there are two parameters output by the Jacobian checker that can help the user decide on the significance of the entry. The first is the relative change in the analytical residual. This quantity, labeled d_{aj}, is the percentage of the acceptance band that comes from changes in the analytical Jacobian from the unperturbed to perturbed states. For a non-linear dependency, the difference between the analytical Jacobians will be significant and it is reasonable to expect that the numerical Jacobian should fall within the band. If the analytical Jacobian is nearly constant over the perturbation, the accuracy of the check becomes increasingly dependent on knowing the roundoff error in the residual. So, as d_{aj} gets closer to unity, the user can have more confidence that the entry is significant.

The second parameter is a confidence measure that is the deviation between the numerical jacobian and analytical values divided by the expected value of the deviation based on roundoff error. Since the roundoff error is only known approximately, this value, called *conf*, is only a qualitative measure of the confidence. A *conf* value of 100 means that the deviation between the numerical jacobian and the analytical values is 100 times larger than the expected deviation based on roundoff error.

Here is a sample of output from a convective heat transfer problem, using the -2 option

```
Eqdof=92 T_0 n=31 Vardof=95 T_0 n=32 x=0
dx=0.0001 aj=-0.008188 nj=-0.008126 aj_1=-0.008188 d_aj=0
conf=1.889e+06

  >>> QCONV on SSID=1
```

This entry can be read as follows: The sensitivity of global equation number 92, which happens to be the T_0 energy equation at node 31, with respect to the temperature variable at node 32 (variable global degree of freedom number 95) has an analytical Jacobian of -0.008188 at the unperturbed state and a computed numerical Jacobian of 0.008126. The analytical jacobian at the perturbed state is -0.008188. For this problem the change in the analytical Jacobian is zero between the unperturbed and perturbed states, so d_{aj} is zero. But even though the difference is small between the analytical and numerical values, it is huge relative to the expected roundoff error, with the deviation being 1.889e+6 times the deviation attributable to roundoff error.

For each node where a deviation is found, the side boundary conditions applied at the node are printed, as shown above. If one of these boundary conditions are applied to the equation that shows an error and have the same dependency that is showing the error, this boundary condition is flagged as shown for the QCONV boundary condition above.

Before the user/developer concludes that there is a discrepancy in the analytical Jacobian, a few things should be tried:

- Giving the problem a nonzero initial guess, either by reading in a STEADY state solution, if one exists, or on transient problems using the “one” option on the Initial Guess card. Sometimes this will make many differences disappear.

- Checking whether the nodes cited in the difference outputs are boundary nodes. Specifically, if they are boundary nodes on which Dirichlet boundary conditions are specified, artificial errors can occur.
- Also, if you are in doubt that there are not reported errors, put one in by a 10 percent perturbation to the residual. The Jacobian checker should hit on those errors and report them to you.
- Check the settings in *mm_numjac.h*.

Number of Jacobian File Dumps

```
Number of Jacobian File Dumps = <integer>
```

Description / Usage

This routine will dump a serial machine independent binary file out to disk containing the Jacobian. The file is meant to be used by the auxiliary program, **checkGomaJac**, to compare two versions of the Jacobian. Ancillary data meant to enhance the printouts in **checkGomaJac** are also output to the file. The card takes one mandatory integer variable.

<integer> If the integer is a positive number, n, then *Goma* will dump the first n Jacobians created (for any reason) to the current directory. If the integer is a negative value, -n, then *Goma* will dump a single Jacobian, the n'th Jacobian created, to the current directory.

The dumped files are named matrix.000, matrix.001, etc. Overwrites of files are allowed to occur. The files themselves are written out using the XDR protocol layer (easy, quick, and machine portable). The VBR format is used to write files out, even if the internal format used by *Goma* is MSR. Thus, VBR and MSR formatted Jacobians may be compared. Frontal Solver Jacobians are not compatible. The algorithm used is also compatible with parallel jobs using *Goma*. In other words, the Jacobian file dumped out for an 8 processor *Goma* run should be identical to the file dumped out by a single processor run.

In order to use this feature, it is necessary to compile *Goma* with the MATRIX_DUMP flag defined.

To compare two Jacobian files previously dumped out for compatibility, run **checkGomaJac** offline:

```
checkGomaJac   matrix1   matrix2
```

checkGomaJac will compare each entry in the row and column scaled matrices and print out in an annotated format the entries containing the largest differences.

Examples

```
Number of Jacobian File Dumps = 2
```

Technical Discussion

This capability has proven itself to be very useful in tracking changes to the Jacobian due to differences in the machine architecture, number of processes, and due to changes in the source code over time. The comparison is done using the standard RTOL, ATOL logic found in ODE solvers. In other words, a weighting vector of the form,

EQUATION

is created for each Jacobian entry, J_i. Then, a determination of the difference between $J_{i`and J:sub:`i}$ by the following formula:

EQUATION

w_i is also used in the Jacobian column scalings, before the standard row sum scaling is applied.

Internal Sandia users can find the auxiliary program, **checkGomaJac**, in the directory **/ home/goma/arch/linux/bin** on the Linux compute server, and in other 'arch' subdirectories for other platforms. External users should contact Goma support staff to obtain the tool.

Initial Guess

```
Initial Guess = {char_string} [filename]
```

Description / Usage

This optional card directs the initialization of the entire unknown vector. Three options are provided to set the entire solution field to numerical values determined by {char_string}. Three additional options are available for reading initial values of the solution vector from data files. The permissible values of {char_string} are:

zero For an initial guess of zero (0.) for each degree of freedom in the unknown vector.

one For an initial guess of one (1.) for each degree of freedom in the unknown vector.

random For a random initial guess (between 0. and 1.) for each degree of freedom in the unknown vector.

read To obtain the initial guess by reading the ASCII data file identified as the *GUESS file*, which must have initially been a *SOLN* file or a tmp_i.d (*Write Intermediate Results*) file.

read_exoII To obtain the initial guess from the EXODUS II file specified by the *FEM file* card that is also used to supply mesh data. Any extraneous variables in the EXODUS II file that are not in the list of active variables for the current problem description are simply ignored.

read_exoII_file *file name* To read the initial guess for the field variables from an EXODUS II database file different from the initial mesh database file. The *file_name* is specified as a single string following the read_exoII_file keyword. As with the read_exoII option, any extraneous variables not specified as active variables for the simulation will be simply ignored.

If this card is omitted, then the default behavior is to assume that a value of zero has been specified for {char_string}.

Examples

Following is a sample card:

```
Initial Guess = zero
```

```
Initial Guess = read_exoII_file      First_Iteration.exoII
```

Technical Discussion

This card provides the specification of the initial vector of unknowns in a problem. In most cases this vector is specified to be identically zero, though in some cases a nonzero vector may be of value (see *Technical Discussion* section of *Debug* card). The first three options (**zero, one, random**) employ an internally-generated vector of initial values, while the **read** option utilizes the values read from an ASCII solution file (see *SOLN* input card) previously calculated by *Goma*, and the **read_exoII** options employ solutions read from binary (exoII) files, not necessarily always generated by *Goma*.

Initialize

```
Initialize = {char_string} <integer> <float>       [units vary]
```

Description / Usage

This optional card provides a mechanism to set one of the field variables to a constant value across the *whole* domain. Definitions of the input parameters are as follows:

{char_string} Permissible values for this input string are any variable names identified in source file rf_fem_const.h beginning at the section labeled Variable Names of unknowns. Examples include, but are not limited to, the following (note the shorthand notation for components):

VELOCITY1, VELOCITY2, VELOCITY3 (V123), MESH_DISPLACEMENT (MD123), SOLID_DISPLACEMENT (SD123), MASS_FRACTION, TEMPERATURE, PRESSURE, VOLTAGE, FILL, LS, POLYMER_STRESS (6 components, 8 modes), VELOCITY_GRADIENT (9 components), SHEAR_RATE, VOLF_PHASE (6 phases), POR_LIQ_PRES, POR_GAS_PRES, POR_POROSITY, POR_SATURATION, POR_LAST, LAGR_MULT (LM123), SURF_CHARGE, EXT_VELOCITY, EFIELD(123), SHELL (4 variables), SPECIES (7 variables).

For a more comprehensive list, see Technical discussion below.

<integer> Species number to be initialized if the value of {char_string} is one of the SPECIES variables (see Technical Discussion); otherwise, set <integer> to zero.

<float> Value to which the variable should be initialized.

Multiple applications of this card are valid; *Goma* automatically counts the number of *Initialize* cards.

Examples

Following is a sample card:

```
Initialize = VELOCITY1 0 0.
```

Technical Discussion

This card provides the means to globally set (i.e., the entire problem domain) initial values for any of the field variables. Since the setting of variables initialized on this card takes place after reading the initial guess (see function *init_vec* in file *rf_util.c*), it can be used to override the value in the *Initial Guess* file.

In order to set a field to a specific value in a particular material only, a similar *Initialize* capability is provided within each material block. Please check in the Material Files section of this manual.

Note, the SPECIES_UNK variables are **NOT** used to initialize any of the species variables. Rather, the special definition called **MASS_FRACTION** representing the various Species Types, is the variables used in Goma input or mat files for this input record. Multiple species are initialized by combining one of these variable types with the second parameter (<integer>) on this card.

The comprehensive list of keyword variable names can be found in *mm_input_util.c*, if you have access to GOMA source code. Search for the function *variable_string_to_int*. A snapshot of the initialize-able variables in that routine is shown here:

```
var = VELOCITY1;
var = VELOCITY2;
var = VELOCITY3;
var = TEMPERATURE;
var = MASS_FRACTION;
var = MESH_DISPLACEMENT1;
var = MESH_DISPLACEMENT2;
var = MESH_DISPLACEMENT3;
var = PRESSURE;
var = POLYMER_STRESS11;
var = POLYMER_STRESS12;
var = POLYMER_STRESS13;
var = POLYMER_STRESS22;
var = POLYMER_STRESS23;
var = POLYMER_STRESS33;
var = SOLID_DISPLACEMENT1;
var = SOLID_DISPLACEMENT2;
var = SOLID_DISPLACEMENT3;
var = VELOCITY_GRADIENT11;
var = VELOCITY_GRADIENT12;
var = VELOCITY_GRADIENT13;
var = VELOCITY_GRADIENT21;
var = VELOCITY_GRADIENT22;
var = VELOCITY_GRADIENT23;
var = VELOCITY_GRADIENT31;
var = VELOCITY_GRADIENT32;
var = VELOCITY_GRADIENT33;
var = VOLTAGE;
var = FILL;
var = SHEAR_RATE;
var = PVELOCITY1;
var = PVELOCITY2;
var = PVELOCITY3;
var = POLYMER_STRESS11_1;
var = POLYMER_STRESS12_1;
var = POLYMER_STRESS22_1;
var = POLYMER_STRESS13_1;
var = POLYMER_STRESS23_1;
var = POLYMER_STRESS33_1;
var = POLYMER_STRESS11_2;
```

(continues on next page)

(continued from previous page)

```
var = POLYMER_STRESS12_2;
var = POLYMER_STRESS22_2;
var = POLYMER_STRESS13_2;
var = POLYMER_STRESS23_2;
var = POLYMER_STRESS33_2;
var = POLYMER_STRESS11_3;
var = POLYMER_STRESS12_3;
var = POLYMER_STRESS22_3;
var = POLYMER_STRESS13_3;
var = POLYMER_STRESS23_3;
var = POLYMER_STRESS33_3;
var = POLYMER_STRESS11_4;
var = POLYMER_STRESS12_4;
var = POLYMER_STRESS22_4;
var = POLYMER_STRESS13_4;
var = POLYMER_STRESS23_4;
var = POLYMER_STRESS33_4;
var = POLYMER_STRESS11_5;
var = POLYMER_STRESS12_5;
var = POLYMER_STRESS22_5;
var = POLYMER_STRESS13_5;
var = POLYMER_STRESS23_5;
var = POLYMER_STRESS33_5;
var = POLYMER_STRESS11_6;
var = POLYMER_STRESS12_6;
var = POLYMER_STRESS22_6;
var = POLYMER_STRESS13_6;
var = POLYMER_STRESS23_6;
var = POLYMER_STRESS33_6;
var = POLYMER_STRESS11_7;
var = POLYMER_STRESS12_7;
var = POLYMER_STRESS22_7;
var = POLYMER_STRESS13_7;
var = POLYMER_STRESS23_7;
var = POLYMER_STRESS33_7;
var = SPECIES_MASS_FRACTION;
var = SPECIES_MOLE_FRACTION;
var = SPECIES_VOL_FRACTION;
var = SPECIES_DENSITY;
var = SPECIES_CONCENTRATION;
var = SPECIES_CAP_PRESSURE;
var = SPECIES_UNDEFINED_FORM;
var = POR_LIQ_PRES;
var = POR_GAS_PRES;
var = POR_POROSITY;
var = POR_TEMP;
var = POR_SATURATION;
var = VORT_DIR1;
var = VORT_DIR2;
var = VORT_DIR3;
var = CURVATURE;
var = BOND_EVOLUTION;
var = SURF_CHARGE;
var = EXT_VELOCITY;
var = EFIELD1;
var = EFIELD2;
var = EFIELD3;
```

(continues on next page)

(continued from previous page)

```
var = ENORM;
var = NORMAL1;
var = NORMAL2;
var = NORMAL3;
var = SHELL_CURVATURE;
var = SHELL_TENSION;
var = SHELL_X;
var = SHELL_Y;
var = SHELL_USER;
var = PHASE1;
var = PHASE2;
var = PHASE3;
var = PHASE4;
var = PHASE5;
var = SHELL_ANGLE1;
var = SHELL_ANGLE2;
var = SHELL_SURF_DIV_V;
var = SHELL_SURF_CURV;
var = N_DOT_CURL_V;
var = GRAD_V_DOT_N1;
var = GRAD_V_DOT_N2;
var = GRAD_V_DOT_N3;
var = ACOUS_PREAL;
var = ACOUS_PIMAG;
var = ACOUS_ENERGY;
var = POR_SINK_MASS;
var = VORT_DIR1
var = VORT_DIR2
var = VORT_DIR3
var = VORT_LAMBDA
var = CURVATURE
var = LAGR_MULT1
var = LAGR_MULT2
var = LAGR_MULT3
var = BOND_EVOLUTION
var = SURF_CHARGE
var = EXT_VELOCITY
var = EFIELD1
var = EFIELD2
var = EFIELD3
var = ENORM
var = NORMAL1
var = NORMAL2
var = NORMAL3
var = SHELL_CURVATURE
var = SHELL_TENSION
var = SHELL_X
var = SHELL_Y
var = SHELL_USER
var = PHASE1
var = PHASE2
var = PHASE3
var = PHASE4
var = PHASE5
var = SHELL_ANGLE1
var = SHELL_ANGLE2
var = SHELL_SURF_DIV_V
```

(continues on next page)

(continued from previous page)

```
var = SHELL_SURF_CURV
var = N_DOT_CURL_V
var = GRAD_S_V_DOT_N1
var = GRAD_S_V_DOT_N2
var = GRAD_S_V_DOT_N3
var = ACOUS_PREAL
var = ACOUS_PIMAG
var = SHELL_DIFF_FLUX
var = SHELL_DIFF_CURVATURE
var = SHELL_NORMAL1
var = SHELL_NORMAL2
var = ACOUS_REYN_STRESS
var = SHELL_BDYVELO
var = SHELL_LUBP
var = LUBP
var = SHELL_FILMP
var = SHELL_FILMH
var = SHELL_PARTC
var = SHELL_SAT_CLOSED
var = SHELL_PRESS_OPEN
var = SHELL_TEMPERATURE
var = SHELL_DELTAH
var = SHELL_LUB_CURV
var = SHELL_SAT_GASN
var = SHELL_SHEAR_TOP
var = SHELL_SHEAR_BOT
var = SHELL_CROSS_SHEAR
var = MAX_STRAIN
var = CUR_STRAIN
var = LUBP_2
var = SHELL_PRESS_OPEN_2
var = SHELL_LUB_CURV_2
```

External Field

```
External Field = <char_string1> {char_string2} <file_name> [char_string_3]
```

Description / Usage

This optional card format provides a mechanism for reading-in nodal field variables stored in an EXODUS II file. Each field variable is specified on a separate input card, with the following input parameters:

<char_string1> Name of the nodal field to be read; it should correspond to a nodal variable name in the EXODUS II file.

{char_string2} Two- to eight-character value that identifies the type of interpolation to be applied to the external variable field. Possible values are as follows:

- **Q1** - Linear
- **Q2** - Quadratic
- **Q2_LSA** - Special quadratic for 3D analysis of 2D LSA
- **Q1_D** - Linear with special surface dofs

- **Q2_D** - Quadratic with special surface dofs
- **Q2_D_LSA - Special quadratic discontinuous for 3D** analysis of 2D LSA
- **PQ1** - Bilinear discontinuous
- **PQ2** - Biquadratic discontinuous
- **P0** - Piecewise constant
- **Pl** - Piecewise linear
- **SP - Subparametric; linear on interior,** quadratic on surface

<file_name> Name of the EXODUS II file from which the nodal field is to be read. When Goma is compiled with LIBRARY_MODE defined (see Appendix 2) and the external field will be passed into Goma from a driver code, this entry will be either IMPORT (for nodal variables) or IMPORT_EV (for element variables), instead of a file name.

[char_string3] Optional character string. Only optional available is **"timedependent"** which enables nodal variables to be interpolated to the current time step. This option is useful for transient coupling, viz. a case in which a transient field variable is used to drive a time-dependent simulation. A good example of this is a transient current density field from a electromagnetics calculation being used to drive a transient thermal calculation with Joule heating.

Examples

The first example:

```
External Field = VX Q2 velocity.exoII
```

```
External Field = VY Q2 velocity.exoII
```

The second example:

```
External Field = JX_FIELD Q2 fields.exoII
```

```
External Field = JY_FIELD Q2 fields.exoII
```

```
External Field = BTHETA_FIELD Q2 fields.exoII
```

The third example:

```
External Field = DMX Q1 IMPORT
```

```
External Field = DMY Q1 IMPORT
```

```
External Field = P_POR Q1 IMPORT_EV
```

The fourth example:

```
External Field = JE_N_1 Q1 emfields.exoII time_dependent
```

Technical Discussion

The field variables read into *Goma* from the Example cards can be accessed in any user-defined subroutine.

In the case of variables named **VX, VY, or VZ**, these fields are automatically loaded to the appropriate velocity component so they can be used in an advection-diffusion analysis, i.e., **VX, VY, VZ** are reserved names for <char_string1> and a user-defined routine is not required. Thus the variables for the two fields, "VX" and "VY", read from the file named "velocity.exoII" in the first example above, would be automatically accessed when the advection term is left on in the *energy* or *species_bulk* equation cards. In other words, without solving the momentum equations, one can access an external velocity field for advection-diffusion problems. These variables would have quadratic interpolation (Q2) applied to the velocity values read.

The three cards in the second example can be used to read two components of a current density field (JX_FIELD, JY_FIELD), and the azimuthal component of a magnetic field (BTHETA_FIELD) from the file "fields.exoII" (generated by some other analysis code). These fields are then accessed in the user-defined subroutines as

fv->external_field[0], fv->external_field[1], and fv->external_field[2],

respectively, as an interpolated value at an integration point. NOTE that these fields are brought in as a part of the BOUSS_JXB_FORCE on the Navier Stokes source card. These are to be distinguished from the electromagnetic fields in the fourth example which correspond to Solid Momentum Source models.

Note that the number of field variables read from the EXODUS II file must not exceed the value MAX_EXTERNAL_FIELD set in the include file rf_fem_const.h. Should that occur, a new version of *Goma* must be compiled with an increased value of MAX_EXTERNAL_FIELD. The user should consult notes on building *Goma* if (s)he has questions regarding how to do this.

The third example assumes that *Goma* has been compiled with LIBRARY_MODE and is linked in to an external driver code along with another program which will compute some variables and pass their values into *Goma*; here the imported fields are the X and Y components of mesh displacement (nodal variables) and porosity (an element variable). There is a naive first order interpolation function in *Goma* to obtain nodal values of fields which are imported as element variables. Although *Goma* does not solve for these variables, their values are included in the output Exodus file.

In the fourth example a field JE_N_1, the x-directed current density field, timedependent, is brought in from em-fields.exoII. Typically, depending on the dimension of the problem, additional fields JE_N_2, JE_N_3 are also brought in as current density is a e. These fields are part of the JXB Solid Momentum Source model, together with the magnetic nodal field quantities, BE_N_1, BE_N_2, and BE_N_3.

Several other standard external fields variables are supported in GOMA. Namely:

FVP11, FVP22, etc. These fields are useful for the elastoviscoplasticity model. Please consult GOMA tutorial GT-019.2 for more details.

SAT, HEIGHT, PERM, CROSS_PERM, SH_SAT_CL_POROSITY, etc. These are specially designated external fields which are mapped to variations in these properties corresponding to thin porous media. Please see GT-038.

References

GT-019.2. Elastoviscoplastic (EVP) Consitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam, and K. S. Chen. Memo to Distribution. March 13, 2003.

GT-038.0: Pixel-to-Mesh Tool Tutorial for GOMA. P R. Schunk, Memo to distribution, 10 November 2009.

Export Field

```
Export Field = <integer1>
```

Description / Usage

Special capability for use in library mode, a mode in which GOMA is called as a library from a driver program. This card is used to indicate which fields will be exported for use in other codes.

<integer1> Name of the nodal field to be read; it should correspond to a nodal variable name in the EXODUS II file.

Technical Discussion

See Appendix 2.

External Pixel Field

```
External Pixel Field = <char_string1> {Q1|Q2} <file_name> <integer1>
```

Description / Usage

This optional card format provides a mechanism for reading-in pixel fields which are converted (mapped, with a least squares algorithm) to finite element fields with the chosen interpolation. After GOMA execution these fields are output in exodusII format in a file map.exoII. Please see discussion below and Tutorial GT-038 for more details and important tips.

<char_string1> Name of the nodal field to be read; it should correspond to a nodal variable name you wish to have in the output EXODUS II file. If you subsequently wish to read the field in again and use as an EXTERNAL_FIELD model on other material property card, the chosen name matters.

{Q1 | Q2} The type of interpolation to be applied to the external pixel field variable field. Possible values are as follows:

- Q1 - Linear
- Q2 - Quadratic

<file_name> Name of the text file name with the pixel points. The pixel field format in this file should be as follows:

- # pixel points
- x_1, y_1, z_1 value
- x_2, y_2, z_2 value
- …
- x_N, y_n, z_n value

<integer1> Material block ID to which the pixel field is mapped.

Examples

```
External Pixel Field = HEIGHT Q1 tread.txt 1
```

Technical Discussion

Please consult the tutorial GT-038 before using this capability. Many user tips are given together with a more thorough explanation on the proper use This capability is extremely memory intensive, and excessive grid sizes and pixel densities can blow out the memory on your machine. As of 12/22/2012 (the end of the Mayan calendar) these fields are used typically to bring in pattern maps for scaling porous media and lubrication height properties.

SAT, HEIGHT, PERM, CROSS_PERM, SH_SAT_CL_POROSITY, etc. These are specially designated external fields which are mapped to variations in these properties corresponding to thin porous media. Please see GT-038.

References

GT-038.0: Pixel-to-Mesh Tool Tutorial for GOMA. P R. Schunk, Memo to distribution, 10 November 2009.

Pressure Datum

```
Pressure Datum = <float> { atm | torr | cgs }
```

Description / Usage

This card is used to set a thermodynamic pressure datum on fluid or solid mechanics problems that calculate equations of state requiring a true value for the total pressure. The total pressure is then defined as the sum of a constant base thermodynamic pressure, specified by this card, and a variable hydrodynamic pressure calculated via the pressure unknown. Definitions of the input parameters are as follows:

<float> Value of the thermodynamic pressure datum.

{ atm | torr | cgs } Units of the float specified above.

Examples

Following is a sample card:

```
Pressure Datum = 1.0 atm
```

Technical Discussion

The value of this variable is stored in the unified problem description structure in cgs units. It is then used in consistency checks and as input into some equation of state routines, such as the ideal gas equation of state routine.

Anneal Mesh on Output

```
Anneal Mesh on Output = {yes | no}
```

Description / Usage

This optional card enables the user to specify that the mesh displacements should be set to zero for the next continuation step. Valid options for this input card are

yes Set the mesh displacements to zero for the next continuation step.

no Do not set the mesh displacements to zero for the next continuation step. This is the default.

There are two important restrictions: 1) annealing the mesh will not work for the TOTAL_ALE mesh motion types, and 2) only the last time step will be annealed for transient problems.

Examples

Following is a sample card:

```
Anneal Mesh on Output = yes
```

Technical Discussion

Annealing a mesh is accomplished by adding the displacements (from the solution) to the base positions (from the FEM file) and writing the resulting nodal positions to a new EXODUS II file, currently anneal.exoII. During the annealing process, the displacement field is also set to zero. This file would be used to restart a subsequent analysis where the anneal.exoII is copied to, or becomes, the file used in a **read_exoII** option for an *Initial Guess*.

1.4.3 Time Integration Specifications

The first card in this section dictates whether the problem is a steady state or transient simulation. This card is required. If the *steady state* option is chosen, then the remaining input records are not required, as the rest of the records are used to set parameters for transient simulations, e.g., time step size, time step error control, etc. Some records are optional even for a transient simulation, as indicated below. It should be noted that the mass-matrix term multiplier in the *Problem Description section* (see, for example, the *EQ*= cards), must be set to one (1) for the transient run to evolve the fields in time. The only equations that are taken as purely quasi static are the *EQ*=*mesh* equations for the situation in which the *Mesh Motion* type is *Arbitrary*.

In addition to the transient parameter information, some Level-Set function information is also supplied to *Goma* in this section. The method of Level-Sets is used to track fluid-fluid or fluidsolid interfaces in an Eulerian fashion, making the problem inherently transient.

Time Integration

```
Time integration = {steady | transient}
```

Description / Usage

This required card is used to specify transient or steady-state calculation. Valid options are:

steady For a solution to the steady (time-derivative free) equations.

transient For transient simulations.

If option **steady** is chosen, then none of the other Time Integration Specification cards in this section are needed.

Examples

This is a sample card for a steady state simulation:

```
Time integration = steady
```

This is a sample card for a transient simulation:

```
Time integration = transient
```

delta_t

```
delta_t = <float>
```

Description / Usage

This card is required for **transient** simulations to set the value of the initial time step. The input parameter is defined as:

<float> Any floating point number that indicates the time step in the appropriate units for your problem.

To specify a fixed time step size for an analysis, set <float> to be a negative number, e.g. -1.0e-6; the code will use a constant (positive) time step. Should convergence problems occur when a fixed step size is specified, the size of the time increment entered for the *delta_t* card will be reduced by half until convergence is achieved. Once a constant time step is reduced, it will not be increased.

Examples

Following is a sample card for an initial time step:

```
delta_t = 6.e-03
```

If a constant time step is desired, use a negative value:

```
delta_t = -6.e-03
```

Maximum Number of Time Steps

```
Maximum number of time steps = <integer>
```

Description / Usage

This card sets the maximum number of time steps that may be performed for a **transient** simulation. *Goma* will stop if this limit is reached. The input parameter is defined as

<integer> Any integer greater than zero, which will limit the number of time steps taken in a simulation.

Examples

The following sample card sets the maximum number of time steps to 100:

```
Maximum number of time steps = 100
```

Maximum Time

```
Maximum time = <float>
```

Description / Usage

This card sets the maximum value of time that may be achieved in a **transient** simulation. *Goma* will stop if this limit is reached. The input parameter is defined as:

<float> Any floating point number in the same units as specified in the *delta_t* card.

The last result written to the EXODUS II and soln.dat file in a successfully completed simulation will always be at the maximum time. This provides a cutoff time beyond which the simulation will terminate.

Examples

The following sample card sets the maximum time to 105 (in units consistent with your simulation):

```
Maximum time = 105.
```

Minimum Time Step

```
Minimum time step = <float>
```

Description / Usage

This card sets the value of the minimum allowable time step size in a **transient** analysis, a useful control if the time step is being decreased due to poor convergence of the transient or iterative algorithm. The input parameter is defined as

<float> Any floating point number in the same units as specified in the *delta_t* card.

Examples

A sample card that sets the minimum time step to 1.e-9 follows:

```
Minimum time step = 1.e-9
```

Technical Discussion

This specification provides a graceful way for the program to terminate based on the computed time step dropping below the minimum value rather than terminating by a segmentation fault or a divide-by-zero error that could result if the time step becomes too small without the benefit of this control.

Maximum Time Step

```
Maximum time step = <float>
```

Description / Usage

This card sets the value of the maximum allowable time step size in a **transient** analysis, where the input parameter is defined as

<float> Any floating point number in the same units as specified in the *delta_t* card.

Examples

A sample card that sets the maximum time step to 10.0 follows:

```
Maximum time step = 10.0
```

Technical Discussion

This setting is useful for advection dominated simulations, such as FILL, where a Courant-like limit must be set on the value of the time step for optimal performance.

Minimum Resolved Time Step

```
Minimum Resolved Time Step = <float>
```

Description / Usage

Its role is to set a lower bound for the time step with respect to the **Time step error** tolerance. When a converged time step is obtained by GOMA, the difference between the predicted solution and final solution for that time step is compared to the **Time step error** tolerance. If the difference exceeds this tolerance the step fails and the time step is cut (usually by a factor of 2), UNLESS the time step falls below the **Minimum Resolved Time Step** size. In this case the step is accepted, even if this error tolerance is not achieved. This provides a mechanism for the modeler to control what phenomena is resolved and what phenomena is ignored.

<float> Any floating point number in the same units as specified in the *delta_t* card.

Examples

A sample card that sets the maximum time step to 10.0 follows:

```
Maximum Resolved Time Step = 10.0
```

Technical Discussion

See GT-034 for a thorough discussion.

References

GT-034: Tutorial on time step parameter selection for level-set problems in GOMA. April 1, 2006. D. R. Noble

Courant Number Limit

```
Courant Number Limit = <float>
```

Description / Usage

This parameter's roll is to control time step growth based on the well-known Courant number criterion. This card applies only to level-set problems. This card imposes an upper limit on the time step size, irrespective of the variable time integrator already in place.

<float> Any floating point number to indicate the Courant number limit.

Examples

A sample card that sets the Courant number to 0.2 is:

```
Courant Number Limit = 0.2
```

Technical Discussion

See GT-034 for a thorough discussion.

Theory

The time step limit imposed by this limit is computed as

$$\mathrm{d}t_{\text{limit}} = C \min_{e} \left| \frac{h_e}{\|\hat{U}\|_e} \right|$$

Here e is the element, h_e is the average size of the element, C is the specified Courant number, and

$$\|\hat{U}\|_e = \frac{\int_e \delta_\alpha(\phi)\, \underline{v} \cdot \underline{n}\, d\Omega}{\int_e \delta_\alpha(\phi)\, d\Omega}$$

References

GT-034: Tutorial on time step parameter selection for level-set problems in GOMA. April 1, 2006. D. R. Noble

Time Step Parameter

```
Time step parameter = <float>
```

Description / Usage

This card allows the user to vary the time integration scheme. The usual settings are:

0.0	Backward Euler method (1st order in time)
0.5	Trapezoid rule (2nd order in time)

Examples

This is a sample card that sets the time integration scheme to Trapezoidal rule:

```
Time step parameter = 0.5
```

Technical Discussion

One should usually use the Trapezoid rule. When a large time step Δt is used the Trapezoid rule can exhibit oscillations. If such a large Δt is required then the Backward Euler method can be used (it will damp oscillations), albeit at a cost of accuracy.

If we designate the time step parameter as θ, the solution at time step n as y^n, and the PDE to be solved as

$$\frac{\partial y}{\partial t} = g\left(y\right)$$

then the time integration method takes the form

$$\frac{y^{n+1} - y^n}{\Delta t} = \frac{2\theta}{1+2\theta}\dot{y}^n + \frac{1}{1+2\theta}g\left(y^{n+1}\right)$$

where

$$\dot{y}^{n+1} = \frac{1+2\theta}{\Delta t}\left(y^{n+1} - y^n\right) - 2\theta\dot{y}^n = g\left(y^{n+1}\right).$$

Note that there is no choice of finite θ that will yield a Forward Euler method. See Gartling (1987) for more information.

FAQs

For porous flow problems with mass lumping, you should always choose backward Euler method.

References

SAND86-1816: NACHOS 2: A Finite Element Computer Program for Incompressible Flow Problems - Part 2 - User's Manual, Gartling, David K. (September, 1987).

Time Step Error

```
Time step error = <float> <integer_list>
```

Description / Usage

The time step error controls the adjustable time step size based on the difference between the solution and the predicted solution (L_2 norm). The first of the eight arguments is a floating point number that indicates the error in the time step selection.

<float> the error value, any floating point number.

The smaller this number is, the smaller the time step will tend to be in the automatic time step control. The original implementation of this capability in *Goma* did not use a normalized value for the norm; to enable this most useful feature, use a negative value of the time step error and a positive, normalized norm will be computed. This way a percentage value of the solution error will be set.

<integer_list> seven integers, with a value either zero (0) or one (1).

A further degree of control is offered by the seven integers (i_1 through i_7) that identify which solution variables will contribute to the error norm calculations. Permissible values for each of these seven integers are 0 and 1. The correspondence between the integers and variables is as follows:

i_1	(pseudo) solid displacement
i_2	fluid velocity
i_3	temperature
i_4	concentration, porous liquid pressure, gas pressure, porosity, saturation
i_5	pressure
i_6	fluid (polymer) extra stress
i_7	voltage

A value of 0 for an integer directs *Goma* to exclude contributions from that variable in the error norm calculation; correspondingly, a value of **1** means that variable should be included.

Examples

A sample time step error card follows:

```
Time step error = 0.01 0 1 1 1 0 0 0
```

In this example, the L_2 norms for the fluid velocity, temperature, and concentration are summed (and scaled) prior to comparison with the target error value of 0.01. If the norms of the velocity, temperature, and concentration variables is greater than 0.01, the time step is halved and the step repeated. Otherwise, the current step size is compared to other step criteria before continuing to the next step.

If the integer values are omitted, the scaled error norm becomes infinite and the analysis will terminate in the error norm calculation with an arithmetic overflow.

Examples

To use the normalized value of the norm, the following would be specified:

```
Time step error = -0.01 0 1 1 1 0 0 0
```

This would set the maximum time step error to be 1%.

Technical Discussion

Note that on porous flow problems the error in step-size is computed as a composite measure of all porous-flow variables, viz. these cannot currently be controlled separately.

Printing Frequency

```
Printing Frequency = <integer> [float]
```

Description / Usage

This card sets the printing frequency, the step or time interval, at which *Goma* will print the solution variables to the *Output EXODUS II file* and the *SOLN file*. Definitions of the <integer> options, and the dependent [float] option when <integer> is set to 0, are:

<integer> Specifies how often the solution will be printed.

> 0	Interval in time steps between successive printings of the solution, any positive integer value.
0	Controls printing of the solution at regularly spaced (uniform) intervals of time (every [float]), regardless of the number of time steps over that time interval

[float] Elapsed time (in the same units as specified in the *delta_t* card) between successive printings of the solution (any positive number).

Examples

Goma will print the solution every five time steps given the following sample card:

```
Printing Frequency = 5
```

Goma will print the solution every ten time units given the following sample card:

```
Printing Frequency = 0 10.
```

Fix Frequency

```
Fix Frequency = <integer>
```

Description / Usage

<integer> fix frequency relative to number of prints (0 is disabled)

Examples

Following are sample cards:

```
# Fix every print
Fix Frequency = 1
```

```
# Fix every 5 prints
Fix Frequency = 5
```

Second Frequency Time

```
Second frequency time = <float1> <float2>
```

Description / Usage

This card allows the time between successive writings of the solution to change after a specified time and is only used if the <integer> in the *Printing Frequency* card is set to 0. Definitions of input parameters are as follows:

<float1> Any number indicating the time at which the printing frequency should shift from that specified in the *Printing Frequency* card to <float2>.

<float2> Printing frequency in time units (same units as specified in the *delta_t* card) for printing the solution at times greater than <float1>.

Examples

The following is a sample card that will change the printing frequency to print every 3 time units after 15 time units:

```
Second frequency time = 15. 3.
```

Initial Time

```
Initial Time = <float>
```

Description / Usage

This card sets the time at which the calculation starts. The input parameter is defined as

<float> Any number indicating the initial solution time (in the same units as specified in the *delta_t* card). An additional feature can be triggered if this float is specified to be negative, which triggers GOMA to look for the nearest restart time in the restart ExodusII database to use as the start time. Note that this option can only be used with Initial Guess options of read_exoII_file or read_exoII.

Normally, the value of <float> will be set to zero unless the problem is a continuation of a previous transient problem.

Examples

The following is a sample card that shows a restart at 45 time units:

```
Initial Time = 45.0
```

The following is a sample card that triggers Goma to look for a restart time of 10 time units, or the closest time value to 10 time units, to start from:

```
Initial Time = -10.0
```

1.4.4 Level Set Specifications

Fill Subcycle

```
Fill Subcycle = <integer>
```

Description / Usage

This is an optional card that sets the number of subcycle-fill time steps between fluidflow time steps in uncoupled level set calculations. The default is 10 subcycle time steps for every flow time step. The input parameter is defined as

<integer> Any nonzero number indicating the subcycling frequency of the fill equation versus the flow equations.

For example, if the value of <integer> is 1, the flow and fill equations are solved every time step. If it is 10, between every transient step in the flow calculation, the fill (advection) equation is solved 10 times with one-tenth of the time step.

Examples

The following is a sample card that sets the fill subcycling rate to 4:

```
Fill Subcycle = 4
```

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, Februray 27, 2001, T.A. Baer

Fill Weight Function

```
Fill Weight Function = {Galerkin | Taylor-Galerkin | SUPG}
```

Description / Usage

Sets the weight function used for the *FILL* equation for either the VOF or Level Set methods. The options for this card are as follows:

Galerkin Name of the weight function formulation. This option requests a standard Galerkin finite element weighted residual treatment. A floating point parameter is not used for this option.

Taylor-Galerkin Name of the weight function formulation

SUPG Name of the weight function formulation. This option requests a Streamwise Upwinding Petrov Galerkin formulation. No floating point parameter is required.

The default value for the *Fill Weight Function* is Taylor-Galerkin.

Examples

This is a sample card:

```
Fill Weight Function = Galerkin
```

Technical Discussion

This card selects the integration/weight function used in solving for the VOF color function or the level set distance function (i.e., the *FILL* unknown). The user should refer to the tutorial on Level Set Computations for a detailed description of level set interface tracking. (See References.)

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

A. N. Brooks and T. J. R. Hughes, "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations," Comp. Math. In Appl. Mechanics and Eng., 32, 199 - 259 (1992).

A. J. A. Unger, P. A. Forsyth and E. A. Sudicky, "Variable spatial and temporal weighting schemes for use in multi-phase compositional problems," Advances in Water Resources, 19, 1 - 27 (1996).

R. Helmig and R. Huber, "Comparison of Galerkin-type discretization techniques for two-phase flow in heterogeneous porous media," Advances in Water Resources, 21, 697-711 (1998).

E. Gundersen and H. P. Langtangen, "Finite Element Methods for Two-Phase Flow in Heterogeneous Porous Media," in Numerical Methods and Software Tools in Industrial Mathematics, Morten Daehlen, Aslak Tveito, Eds., Birkhauser, Boston, 1997.

S. F. Bradford and N. D. Katopodes, "The anti-dissipative, non-monotone behavior of Petrov-Galerkin Upwinding," International J. for Numerical Methods in Fluids, v. 33, 583-608 (2000).

Level Set Interface Tracking

```
Level Set Interface Tracking = {yes | no}
```

Description / Usage

Activates (or deactivates) embedded interface tracking by the level set method. When activated, the set of cards specifying level set run parameters are read; these should appear in the input deck following this card. Also when activated a "level_set" equation type should be included in the list of equations identified in the equations section.

Examples

A sample input card is:

```
Level Set Interface Tracking = yes
```

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Semi_Lagrange

```
Level Set Semi_Lagrange = <char1>
```

Description / Usage

This card is currently inactive because it was developed for decoupled LS fill problems.

<char1> YES | ON | TRUE

Examples

```
Level Set Semi_Lagrange = yes
```

Level Set Subgrid Integration Depth

```
Level Set Subgrid Integration Depth = <integer1>
```

Description / Usage

Subgrid integration is used to improve integration accuracy for all functions which invoke a diffuse level-set interface representation of properties and surfaces. With integration depths greater than zero the elements through which the zero level set crosses are subdivided in a geometric way to achieve more accurate integration. Level- 1 depths implies the smallest grid size is 1/4 of the original, and a level-2 is 1/8th, and so on. Please see usage nodes below.

<integer1> Level of integration depth. Default is zero. See usage notes.

Examples

This example sets the subgrid integration depth to two:

```
Level Set Subgrid Integration Depth = 2
```

Technical Discussion

Each level of subgrid integration leads to precipitous growth in computational load, especially in 3D. Level-2 seems to optimize accuracy and efficiency. Levels higher than 2 is not recommended.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Subelement Integration

```
Level Set Subelement Integration = {ON | YES | OFF | NO}
```

Description / Usage

Subelement integration is used to improve integration accuracy for all functions which invoke a sharp level-set interface. Note here that the Level Set Length Scale option must be zero. This is possible because the subelement integration scheme actually produces a geometric representation of the zero level set surface on which exact line integrals of the surface tension source term term can be peformed. Please see usage nodes below.

{ON | YES} Use subelement integration on surface level set capillary term.

{OFF | NO} Don't use subelement integration.

Examples

This example invokes the subelement integration

```
Level Set Subelement Integration = ON
```

Technical Discussion

- **NOTE**: Level Set Length Scale must be set to zero.
- Because of the construction of an in-element interface meshing to find this representation, subelement integration cannot be used currently for three dimensional problems. Subgrid integration can be, however, but it is inefficient.
- Best to use this integration approach with the property specification method of "Second Level-Set "property_name", e.g. Second Level Set Density, etc.
- Typically this capability greatly improves mass conservation and avoids parasitics for surface tension dominated problems.
- **NOTE** that the Level Set Renormalization method must be set to Huygens.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Adaptive Integration

```
Level Set Adaptive Integration = {ON | YES | OFF | NO}
```

Description / Usage

To be used with Subelement integration to improve integration accuracy. Does not work with subgrid integration or basic level-set. Requires a sharp interface, viz. levelset length scale of zero. Please see usage nodes below.

{ON | YES} Use adaptive integration on surface level set capillary term.

{OFF | NO} Don't use adaptive integration.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Adaptive Order

```
Level Set Adaptive Integration = <integer1>
```

Description / Usage

To be used with Subelement adaptive integration to improve integration accuracy. Does not work with subgrid integration or basic level-set. Requires a sharp interface, viz. level-set length scale of zero. Please see usage nodes below.

<integer1> Adaptive integration order. Single positive integer greater than zero. Default value is 3.

Examples

This example invokes the subelement adaptive integration order:

```
Level Set Adaptive Integration = YES
```

```
Level Set Adaptive Order = 2
```

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Overlap Quadrature Points

```
Overlap Quadrature Points = <integer1>
```

Description / Usage

To be used with the overset grid capability. This function sets the number of overlap quadrature points with this capability. See GT-026 for more details.

<integer1> Overlap quadrature points. Single positive integer greater than zero. Default value is 3.

Examples

This example invokes the number of overlapping quadrature points:

```
Overlap Quadrature Points = 2
```

Technical Discussion

Please consult the overset grid capability tutorial for futher discussion. (Ref. below). This is to be use with AC_OVERLAP, or the augmenting condition of type AC = OV.

References

GT-026.4: “GOMA’s Overset Mesh Method”, P R. Schunk and E. D. Wilkes, 11 Jan. 2006

Level Set PSPP Filtering

```
Level Set PSPP filtering = <YES | NO>
```

Description / Usage

On this card, the user specifies a single char_string.

<YES | ON> This string turns on level set PSPP filtering if it is “yes” or “on”.

Examples

A typical PSPP filtering input card looks like:

```
Level Set PSPP filtering = yes
```

Technical Discussion

Not entirely clear what this card does, but in the vicinity of the level-set interface, the Bochev PSPP stabilization scheme is altered. This is recommended when this pressure stabilization scheme is deployed. See the Pressure Stabilization card.

Level Set Length Scale

```
Level Set Length Scale = <float>
```

Description / Usage

On this card, the user specifies a single char_string.

<float> This value represents the size of the region around the zero level set function contour in which interfacial physical quantities, for example, surface tension, will be present.

Stability and conservation of phase volume are dependent upon this value to a significant degree. Experimentation has revealed that this float value should be between two and three times the average linear dimension of the elements in the mesh.

Examples

A typical length scale input card looks like:

```
Level Set Length Scale = 0.3
```

Technical Discussion

The level set method is an *embedded* interface method. That is, the location of the interface is not known explicitly as a geometric parameter of the problem, but rather it is abstracted as a level contour of a higher dimensional function. This is convenient in many ways, but it does mean that phenomena associated with the interface, for example, surface tension, must enter the problem spread over a region near the zero level set contour. The *Level Set Length Scale* sets the size of this region.

A good example of the application of the *Level Set Length Scale* parameter is in how surface tension is included in problems using level set interface tracking. The following tensor is added to the fluid momentum equation:

$$\underline{\underline{T}} = \sigma \delta_\alpha \left(F \right) \left(\underline{\underline{I}} - \underline{n}\,\underline{n} \right)$$

where F is the level set function itself, $\underline{n}$ is the unit normal to the level set contour, $\underline{\underline{I}}$ is the unit tensor, σ the surface tension, and $\delta_\alpha \left(F \right)$ is a "smooth" Dirac function given by:

$$\delta_\alpha \left(F \right) = |\nabla F| \frac{1 + \cos \left(\pi F / \alpha \right)}{2\alpha}, \quad |F| \leq \alpha$$

In this example, the parameter α would be equal to one-half the *Level Set Length Scale* value specified on this card.

FAQs

How should the Length Scale value be chosen? Trial and error is often the best method to determine an appropriate value for this parameter. However, experience has shown that values for Level Set Length Scale that are between two and three times the average element linear dimension seem to work best.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Initialize

```
Level Set Initialize = <char_string> <float1> <float2>
```

Description / Usage

This card is used to initialize fields around the zero level set.

<Char_string> A character string which identifies dependent variable to be initialized. It is taken from the list of names on the Initialize card.

<float1> Value of the variable on the negative side of the zero level set.

<float2> Value of the field on the positive side of the zero level set.

Examples

Two examples of initialization methods are provide below:

```
Level Set Initialize = TEMPERATURE 0. 100.
```

Technical Discussion

Not clear whether this capability has been used and tested much. (12/3/2012)

Level Set Adaptive Mesh

```
Level Set Adaptive Mesh = {yes|no}
```

Description / Usage

This card specifies whether to enable level set adaptive mesh through `Omega_h` library

yes|on Adaptivity is enabled

no|off Default, no adaptive mesh

Examples

This is a sample card:

```
Level Set Adaptive Mesh = on
```

Technical Discussion

This requires first order trianglular or tetrahedral meshes

References

Level Set Adapt Width

```
Level Set Adapt Width = <float>
```

Description / Usage

This card specifies the width to apply an "Inner" adaptive size using `Omega_h` library

<float> Width around the level set interface to apply the inner size

Examples

This is a sample card:

```
Level Set Adapt Width = 0.5
Level Set Adapt Inner Size = 0.1
Level Set Adapt Outer Size = 0.8
```

Technical Discussion

This requires first order trianglular or tetrahedral meshes

References

Level Set Adapt Inner Size

```
Level Set Adapt Inner Size = <float>
```

Description / Usage

This card specifies the size within "Adapt Width" near the interface, using `Omega_h` library

<float> ISO element size inside of the Adapt Width region

Examples

This is a sample card:

```
Level Set Adapt Width = 0.5
Level Set Adapt Inner Size = 0.1
Level Set Adapt Outer Size = 0.8
```

Technical Discussion

This requires first order trianglular or tetrahedral meshes

References

Level Set Adapt Outer Size

```
Level Set Adapt Outer Size = <float>
```

Description / Usage

This card specifies the size outside of the "Adapt Width" range near the interface, using `Omega_h` library

<float> ISO element size outside of the Adapt Width region

Examples

This is a sample card:

```
Level Set Adapt Width = 0.5
Level Set Adapt Inner Size = 0.1
Level Set Adapt Outer Size = 0.8
```

Technical Discussion

This requires first order trianglular or tetrahedral meshes

References

Level Set Adapt Frequency

```
Level Set Adapt Frequency = <integer>
```

Description / Usage

This card specifies how often to adapt the mesh

<integer> Will adapt the mesh after <integer> time steps

Examples

This is a sample card:

```
Level Set Adapt Width = 0.5
Level Set Adapt Inner Size = 0.1
Level Set Adapt Outer Size = 0.8
Level Set Adapt Frequency = 10
```

Technical Discussion

This requires first order trianglular or tetrahedral meshes

References

Level Set Initialization Method

```
Level Set Initialization Method = {method_name} {parameter list}
```

Description / Usage

This card specifies the means by which the level set function is initialized. That is, it constructs from a representation of the starting interface shape, a value for the distance function at every node in the mesh. The syntax of the card is as follows:

{method_name} A character string which identifies the initialization option desired. Choices for this string are: **Projection, Exodus, Nodeset, Surfaces, SM_object.**

{parameter list} This is a variable parameter list specific to each option. The nature of it for each method is detailed in the syntax descriptions below.

Below are the exact syntax used for each initialization method, a brief description of the method and a specification of any additional required parameters.

Projection	This method computes the initial level set field by calling a user-specified routine which returns the signed distance function for a given point. It has no parameter list after its name.
Exodus	Using this card indicates that the initial level set field is to be read from the exodus file specified earlier (see *FEM file* and *Initial Guess* cards for **read_exoII** option). This card has no parameter list after its name.
Nodeset <integer1> **EB** <integer2>	This method establishes the initial location of the interface as the boundary between two element blocks. The value <integer1> is the nodeset identification number for an internal nodeset defined to exist at the interface between the two element blocks. The character string **EB** is required. The integer <integer2> is the element block id number to which positive values of level set function is going to be assigned.
Surfaces <integer>	This card establishes the initial level set function by referring to a set of primitive geometric objects. It is the easiest to use and the most general. The integer value <integer> is the number of surface objects that are used to construct the initial interface. This number of **SURF** object cards must follow this card. This is the syntax of the **SURF** object card: SURF = {object_name} {float list} {object_name}: a character string identifying the type of geometric object. Options are: **PLANE**, **CIRCLE, SPHERE,** SS, **USER.** {float list}: geometric parameters associated with each object as float values

The following is the syntax and description for each geometric object option, i.e., the "{object_name} {float list}" part of **SURF**

PLANE <nx. <ny> <nz> <d>	This card constructs a planar interface surface. The float values <nx>, <ny>, <nz> define a vector normal to this plane with the restriction that the sign of the vector must be such that it points from the negative side of the interface to the positive side of the interface. The float value <d> effectively represents the distance of the plane from the origin. Its value must be set, however, so that the dot product of any position vector to a point on the desired plane and the vector (nx,ny,nz) must be equal to <d> (it is a property of planes that this number is independent of the point on the plane that is chosen).
CIRCLE <cx> <cy> <radius>	This card constructs a circular interface surface in a two-dimensional domain. The float values <cx> <cy> identify the coordinates of the center of the circle. The float value <radius> establishes the radius of the curve. By definition, points interior to the circle are assigned negative level set function values.
SPHERE <cx> <cy> <cz> <radius>	This card constructs a spherical interface surface in a three-dimensional domain. The float values <cx> <cy> <*cz*> identify the coordinates of the center of the circle. The float value <radius> establishes the radius of the sphere. By definition, points interior to the sphere are assigned negative level set function values.
SS {ss_id}	This card uses an existing sideset in the problem as a defined geometric object for construction of an interface. The parameter <ss_id> identifies this sideset.
USER {user-defined float list}	This card indicates the user has defined an object function using the supplied parameter float list that returns a signed distance value when supplied with the coordinates of a point in space. This object function should appear in the function call *user_init_object* in the file **user_pre.c.**
SM_object {object_type} {object_name}	This card allows the user to initialize the level set location by using a piece of solid model geometry. The solid model object_type can be either **FACE** or **BODY.** A 2D initialization uses the boundary of the specified FACE (or surface) as the 0 level set. A 3D initialization uses the boundary of the specified BODY (or volume) as the 0 level set.

Examples

Two examples of initialization methods are provide below:

```
Level Set Initialization Method = Nodeset 20 EB 1
```

```
Level Set Initialization Method = Surfaces 3
    SURF = PLANE -1. 0. 0. -3.
        SURF = CIRCLE -2 0 1
        SURF = CIRCLE -3 0 0.5
```

```
Level Set Initialization Method = SM_object BODY my_blob
```

Technical Discussion

The **Projection** initialization method was developed early in the level set development process. It has since been superseded by other more easily used methods. It is still supported primarily for the use of developers. Users wanting a complicated interface shape for which they can supply an appropriate distance function should user the USER surface object option under the Surfaces initialization method.

The **Exodus** method deserves little comment. It should be used when restarting level set computations from a preexisting solution.

The **Nodeset** method allows the user to make use of the sophisticated solid body manipulation software in meshing packages like CUBIT. The procedure for using this method is to create a domain which contains two element blocks. The desired starting point for the interface should lie on the curve or surface which these two blocks have in common. A single nodeset should be defined over this entire curve or surface. The nodeset identification number should be the first integer parameter specified on the card. Also note that one of the blocks must be designated as the "positive" block. This means then when initialized the values of the level set function in this block will be positive. The values in the other block will be negative. Note that this initialization method can only by used for problems that have exactly two blocks, no more.

The **Surfaces** initialization method is the most useful method for initialization. It draws from the fact that it is relatively easy to determine the distance to simple geometric objects (planes, circles, spheres, etc.). Further, it permits initialization using more than one of these objects so that relatively complicated initial interface locations can be constructed. However, the user should recognize that this method is still somewhat unsophisticated in its approach so there are some caveats associated with its use. The primary point is that surface objects should never intersect anywhere within the domain of interest, otherwise it is more than likely that the starting interface shape will not be what the user expects.

The **SM_object** initialization method allows the user to use solid model geometry to initialize 2D and 3D level sets. Certain 2D geometries can be created using only Goma input commands (see *FACE*). Other 2D geometries, and all 3D geometries, can be accessed via an ACIS .sat file. The usual way to do this is for the user to create their desired geometry within Cubit (or, import solid model geometry from elsewhere into Cubit). Faces (or surfaces) should be created for 2D initialization, and bodies (or volumes) should be created for 3D initialization. The *boundary* of the object is used to initialize the level set. The geometry should be named within Cubit and exported to an ACIS .sat file via Cubit's export acis "filename" ascii command. This same file should be read in via the *ACIS file* command in the Geometry Specifications section. The solid model geometry is then available for the *Level Set Initialization Method* command. (Note that the Geometry Specifications section usually comes after the *Level Set Initialization Method* command; this is OK).

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Periodic Planes

```
Level Set Periodic Planes = <float1> <float2> <float3> <float4> <float5> <float6>
```

Description / Usage

This card directs the level-set renormalization to accommodate peroidic boundary conditions (see Advanced Capabilities Manual AC_Periodic capability). The periodic boundary conditions on the level set field are not compatible with renormalization unless this capability is specified.

<float1> x-coordinate value of first periodic boundary.

<float2> x-coordinate value of second periodic boundary. If equivalent to float1 than this direction is not periodic.

<float3> y-coordinate value of first periodic boundary.

<float4> y-coordinate value of second periodic boundary. If equivalent to float3 than this direction is not periodic.

<float5> y-coordinate value of first periodic boundary.

<float6> y-coordinate value of second periodic boundary. If equivalent to float5 than this direction is not periodic.

Examples

Two examples of initialization methods are provide below:

```
Level Set Periodic Boundary = -0.5 0.5 0 0 0 0
```

This card instructs renormalization to accommodate the x-directed-boundaries to be considered as periodic relative to the level-set field.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Level Set Control Width

```
Level Set Control Width = <float>
```

Description / Usage

This card is a multiplier on the *Level Set Length Scale* to determine the size of the region around the zero level set contour over which the level set gradient is averaged. The value of this parameter defaults to 1.0 if this card is not included.

Examples

This sample card sets the control width to be equivalent to the length scale:

```
Level Set Control Width = 0.5
```

Technical Discussion

As noted in the description of the *Level Set Renormalization Tolerance* card, renormalization is triggered when the average of the level set gradient magnitude has departed sufficiently from unity. The region over which this average is obtained is approximately a narrow fixed-width strip on either side of the zero level set contour. The width of this strip is twice the *Level Set Length Scale* multiplied by the float value supplied on this card.

FAQs

Usually it is best practice to leave this parameter at its default setting and control the frequency of renormalization with the renormalization tolerance.

Level Set Timestep Control

```
Level Set Timestep Control = <YES | NO>
```

Description / Usage

On this card, the user specifies a single char_string.

<YES | ON> This string turns on level set timestep control if it is “yes” or “on”.

Examples

A typical length scale input card looks like:

```
Level Set Timestep Control = yes
```

Technical Discussion

In normal operations, the error norm of the level set function is not included in controlling the size of the time step decided upon by the variable timestep size integrator. Inclusion of this card will add the level set unknown to the list of update error norms used to decide the time step size. In other words, use this card when you want the changes of the level set function to affect the timestep size. If this card is not used, the default behavior is to ignore the level set degrees of freedom in controlling the timestep size.

Level Set Renormalization Tolerance

```
Level Set Renormalization Tolerance = <float>
```

Description / Usage

This parameter provides a means for controlling how often renormalization (redistancing) operations are performed on the level set function as it evolves by fixing the size of the deviation allowed between the average absolute magnitude of the level set function gradient near the level set interface and unity, the theoretical value observed for a pure distance function.

<float> Value of the tolerance, the allowable deviation.

The range of this parameter is any positive real number, however, it is rare to use values smaller than 0.1 or larger than 5.0. The value of the tolerance defaults to 0.5 if this card is not specified.

Examples

This is a sample renormalization card:

```
Level Set Renormalization Tolerance = 0.05
```

Technical Discussion

One of the key properties of the level set function is that it is a smooth function near to the interface. In particular, if the level set function is a distance function then the magnitude of its gradient on the zero level contour should always be unity. This fact is used to provide a criterion for invoking a renormalization procedure. The gradient of the level set function is found within a fixed region around the zero level set contour (see *Level Set Control Width*). The integrated average of the magnitude of this vector is determined and compare to unity. Should this difference differ by greater than the value for Renormalization Tolerance identified on this card, a renormalization procedure will presently be initiated.

FAQs

What is a proper value for this parameter? Values on the order of unity should work well. Renormalization based on gradient can be disabled completely by choosing a very large value for this parameter. Conversely, a very small value will always result in a renormalization step.

Is it possible to renormalize too often? Yes. Renormalization is an extraphysical procedure designed solely to improve the numerical performance of the interface tracker. As such, it can add or subtract volume to or from the phases represented by the interface contour. Renormalizing too often, therefore, can result in errors being introduced. The renormalization procedure, Huygens_Constrained, attempts to mitigate this effect.

Level Set Renormalization Method

```
Level Set Renormalization Method = {char_string}
```

Description / Usage

This card indicates the method to be used to renormalize the level set function during the course of the computation. The syntax of this card is as follows:

{char_string} A character string which specifies the type of method for renormalization. Choices for this string are: **Huygens, Huygens_Constrained, Correction.**

Each method is described below; see also the Technical Discussion.

Huygens In this method a set of m points P is constructed:

$$\mathbf{P} = \{(x_i, y_i, z_i)\,,\, i = 1, 2, \ldots m| \quad \phi_j\,(x_i, y_i, z_i) = 0\}$$

which in a sense represent a discretization of the interface location. The finite element interpolation functions are used to find exact locations for these points. For each mesh node j, a minimum distance D_j, can be found to this set of points. Renormalization is accomplished by replacing the level set value at this node ϕ_j with D_j multiplied by the sign of the previous value for the level set function. This method is fast and robust and reasonably accurate given sufficiently refined meshes using high order level set interpolation. However, this method is prone to losing material if low order level set interpolation is employed.

Huygens_Constrained This method renormalizes the function in much the same way as the **Huygens** method, except it employs a Lagrange multiplier to enforce a global integrated constraint that requires the volume occupied by the "negative" phase to remain unchanged before and after renormalization. This requirement makes this method better at conserving mass. However, since it enforces a global constraint, it is possible that material might be moved nonphysically around the computational domain.

Examples

This is a sample renormalization method input card:

```
Level Set Renormalization Method = Huygens_Constrained
```

Technical Discussion

Renormalization is an operation particular to level set embedded interface tracking. The level set function ϕ is usually specified in terms of a signed distance to the interface. This type of function has very nice properties in terms of smoothness and a unitary gradient magnitude in the vicinity of the interface. All of which are beneficial in accurately integrating the function and applying interfacial physics such as surface tension. The difficulty appears because of the velocity field $\underline{u}$ used to evolve the level set function via the relation:

$$\frac{\partial \phi}{\partial t} + \underline{u} \cdot \nabla \phi = 0$$

There is nothing that requires that this velocity preserve the level set function as a distance function during its evolution. As a result, large gradients in the level set function might appear that would degrade the accuracy of both its time evolution and the accuracy of the interfacial terms related to the level set function. To remedy this problem, periodically the level set function must be reconstructed as a distance function; this process is referred to as renormalization. The criteria for determining when renormalization should occur is discussed under *Level Set Renormalization Tolerance.*

Level Set Renormalization Frequency

```
Level Set Renormalization Frequency = <integer>
```

Description / Usage

This card sets an upper limit to the number of time steps which are allowed to pass between renormalization procedures. Possible values for <integer> are listed below:

<integer>

-1	never renormalize (default)
0	renormalize every step
n	a positive integer >1, renormalize every n^{th} time step

Examples

This is a sample input:

```
Level Set Renormalization Frequency = 50
```

Technical Discussion

Renormalization procedures are normally triggered by the average gradient exceeding one by a specified amount (*see Level Set Renormalization Tolerance*). However, at times it might be advantageous to trigger a renormalization independent of the size of the average level set gradient. For example, it might occur that in a very small region near the interface, the level set gradient is becoming large but elsewhere the gradient is still relatively small. Since the average gradient is used, this condition might not trigger renormalization. By setting an upper limit for the number of time steps that can pass before renormalization, situations such as this can be remedied.

Restart Time Integration After Renormalization

```
Restart Time Integration After Renormalization = {yes | no}
```

Description / Usage

This card is used to specify whether or not to restart time integration each time Goma renormalizes the level set function during the course of the computation. When time integration is restarted, the time step is reset to its initial size and held at this step size for the following 3 time steps. If this card is not present, the default is yes (time integration will be restarted after each renormalization). The syntax of this card is as follows:

{yes | no} Indicates the specified choice. {yes | on | true}can all be used to specify restarting of time integration. {no | off | false} can all be used to specify no restart.

Examples

This is a sample renormalization method input card:

```
Restart Time Integration After Renormalization = no
```

Level Set Reconstruction Method

```
Level Set Reconstruction Method = {char_string}
```

Description / Usage

This card indicates the method used to perform the Huygens renormalization of the level set function. This card applies only if *Level Set Renormalization Method* is set to **Huygens** or **Huygens_Constrained.** Permissible values of {char_string} are:

POINTS A list of points on the interface is formed and the renormalized distance is computed as the distance to the nearest point in this list; this is the default method.

FACETS A list of connected facets on the interface is formed and the renormalized distance is computed as the distance to the nearest point on the nearest facet in this list. Currently this option is not supported for 3-dimensional calculations.

Examples

This is a sample input card:

```
Level Set Reconstruction Method = FACETS
```

Technical Discussion

As described for the *Level Set Renormalization Method* card, Huygens based renormalization is performed by reconstructing the level set surface and computing the distance to the nearest point on this surface. Here, the method of reconstructing the level set surface is addressed. Either a set of points on the interface is formed or a connected set of facets is formed. The advantage to using connected facets is that the interface is better described between the points on the interface. However, the calculation of the faceted geometry is slightly more expensive computationally. Also, the current implementation is limited to 2-dimensional simulations.

Level Set Contact Extension

```
Level Set Contact Extension = {yes|no}
```

Description / Usage

This card specifies whether the level set surface is considered to extend into boundaries when performing renormalization of the level set distance function. This card applies only if *Level Set Renormalization Method* = **Huygens_Constrained.** Permissible values for this option are:

yes|on The level set interface is considered to extend smoothly into the boundaries.

no|off The level set interface ends at the boundaries; this is the default.

Examples

This is a sample input card:

```
Level Set Contact Extension = no
```

Technical Discussion

When renormalizing the level set distance function, the behavior of the interface near boundaries is important. When the interface is considered to end at the boundary, a large number of grid points may be closest to this boundary point. This appears as a cusp in the interface and can make it difficult to achieve sharp contact angles because of the very large capillary force that results. One method to alleviate this is to extend the interface smoothly into the boundaries to eliminate the cusp in the interface. The current algorithm, however, can cause errors when employed near corners of the domain. Until this is resolved, this option can only be recommended for domains without interior corners.

Level Set Slave Surface

```
Level Set Slave Surface = {yes|no}
```

Description / Usage

This card specifies whether the level set distance function is constrained during the calculation or evolves with the typical advection equation. Permissible values for this option are:

yes|on The surface is constrained to remain on the initial surfaces throughout the calculation (moving with these surfaces if they are moving).

no|off The surface evolves normally according to the local velocity field; this is the default.

Examples

This is a sample card:

```
Level Set Slave Surface = on
```

Technical Discussion

In a typical level set simulation, the surface is first initialized with the *Level Set Initialization Method* card, and then the surface evolves in time according to the local velocity field. Using this card, however, the surface is constrained to remain on the initial surfaces. If the initial surfaces are static, then the level set surface remains stationary. For moving interfaces such as those defined by an isosurface or a side set, the level set function is reinitialized at each Newton iteration to match the moving surface.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Ignore Level Set Dependencies

```
Ignore Level Set Dependencies = {yes | no}
```

Description / Usage

Including this card in your input deck with the string parameter set to "yes" instructs *Goma* to discard the sensitivities of all equations to the level set variable when constructing the Jacobian matrix. This may have benefits when it comes to stability and convergence; although, the effectiveness of this card is very much case by case. Note also that use of this card is consistent only with **Fill Weight Function = Explicit.** Any other choice will result in an error.

Examples

A sample input card is:

```
Ignore Level Set Dependencies = yes
```

Force Initial Level Set Renormalization

```
Force Initial Level Set Renormalization = <char_string>
```

Description / Usage

This card is used to invoke a renormalization step prior to the first time step of any transient computation.

<char_string> `YES|ON` (not case sensitive) will cause the renormalization procedure to occur on the first step. If this card is not included or some other string is used here a renormalization will automatically occur on the first time step.

Examples

A typical length scale input card looks like:

```
Force Initial Level Set Renormalization = yes
```

Technical Discussion

Restarts occur fairly frequently during level set computations. It has been discovered that the robustness of the subsequent computation can be improved by quite a bit if the level set field is renormalized at the start of the restart, regardless of the current average gradient norm error. This card is employed to invoke a renormalization at the start of any computation, that is, a renormalization procedure is conducted prior to the initial time step if this card is present in the input deck. It has become standard operating procedure that when a level set computation runs into computational difficulty the first step in recovery should be to restart with a forced initial renormalization using this card.

1.4.5 Phase Field Specifications

Number of Phase Functions

```
Number of phase functions = {integer}
```

Description / Usage

Activates generalized phase function capability. Currently, the number of phase functions cannot exceed five. Phase function fields are essentially identical to level set fields, but more than one can be activated for various purposes. Please see technical discussion below.

Examples

A sample input card is:

```
Number of phase functions = 1
```

Technical Discussion

Various uses of the phase function approach have been explored. To track multiple interface types from multiple fluids requires more than one level-set field. This capability can also be deployed for tracking imprinted solid surfaces (moving) together with capillary free surfaces. Consult the tutorials.

References

GT-026.3 GOMA's Overset Mesh Method: User Tutorial, November 19 2003. P. R. Schunk and E. D. Wilkes

Phase Function Slave Surface

```
Phase Function Slave Surface = <char_string>
```

Description / Usage

This card is used to designate that the phase function degree of freedom is being slaved to a boundary. This card is used primarily in the overset grid algorithm in which a phase function field is slaved to the surface of the embedded body.

<char_string> YES|ON (not case sensitive) will allow the phase function field to be slaved to a surface. Currently, no support is given to more than one slaved function fields or to problems in which there are slaved and unslaved (free?) phase function fields.

Examples

A typical length scale input card looks like:

```
Phase Function Slave Surface = yes
```

Technical Discussion

One of the nice properties of level set/phase function fields is that they can be used to find distances from surfaces. This function can be used quite apart from their abilities to track interfaces. Including this card informs Goma that the phase function 1 field is going to be used in this capacity and that no PDE is going to be solved to evolve it. Instead, the values of this field will be "slaved" to a specific surface in the problem and their values will be determined in reference to this surface in a process very reminicent of renormalization.

The overset grid method makes use of a slaved phase function field. In that case, the phase function field is slaved to the surface of the embedded object. As the embedded object moves through the flow field, the slaved phase function values will be updated by determining the distance of a given node to the object's surface. This slaved phase function field is then used in a variety of ways to compute the influence of the embedded object on the flow and stresses of the surrounding fluid.

Phase Function Initialization Method

```
Phase Function Initialization Method = {method_name} {parameter list}
```

Description / Usage

This card specifies the means by which the phase functions are initialized. After the initial instance, subsequent instances of {model_name} {parameter_list} are used to describe initializations of phase fields 2 through 5. This card constructs from a representation of the starting interface shape, a value for the distance function at every node in the mesh. The syntax of the card is as follows:

{method_name} A character string which identifies the initialization option desired. Choices for this string are: **Projection, Exodus, Nodeset, Surfaces, SM_object.**

{parameter list} This is a variable parameter list specific to each option. The nature of it for each method is detailed in the syntax descriptions below.

Below are the exact syntax used for each initialization method, a brief description of the method and a specification of any additional required parameters.

Projection	This method computes the initial phase function field by calling a user-specified routine which returns the signed distance function for a given point. It has no parameter list after its name.
Exodus	Using this card indicates that the initial phase function field is to be read from the exodus file specified earlier (see *FEM file* and *Initial Guess* cards for ** read_exoII** option). This card has no parameter list after its name.
Nodeset <integer1> **EB** <integer2>	This method establishes the initial location of the interface as the boundary between two element blocks. The value <integer1> is the nodeset identification number for an internal nodeset defined to exist at the interface between the two element blocks. The character string **EB** is required. The integer <integer2> is the element block id number to which positive values of phase function function is going to be assigned.
Surfaces <integer>	This card establishes the initial phase function function by referring to a set of primitive geometric objects. It is the easiest to use and the most general. The integer value <integer> is the number of surface objects that are used to construct the initial interface. This number of **SURF** object cards must follow this card. This is the syntax of the **SURF** object card: **SURF** = {object_name} {float list} {object_name}: a character string identifying the type of geometric object. Options are: **PLANE, CIRCLE, SPHERE,** SS, **USER.** {float list}: geometric parameters associated with each object as float values.

The following is the syntax and description for each geometric object option, i.e., the "{object_name} {float list}" part of **SURF**

PLANE <nx. <ny> <nz> <d>	This card constructs a planar interface surface.The float values <nx>, <ny>, <nz> define a vector normal to this plane with the restriction that the sign of the vector must be such that it points from the negative side of the interface to the positive side of the interface. The float value <d> effectively represents the distance of the plane from the origin. Its value must be set, however, so that the dot product of any position vector to a point on the desired plane and the vector (nx,ny,nz) must be equal to <d> (it is a property of planes that this number is independent of the point on the plane that is chosen).
CIRCLE <cx> <cy> <radius>	This card constructs a circular interface surface in a two-dimensional domain. The float values <cx> <cy> identify the coordinates of the center of the circle. The float value <radius> establishes the radius of the curve. By definition, points interior to the circle are assigned negative phase function function values.
SPHERE <cx> <cy> <cz> <radius>	This card constructs a spherical interface surface in a three-dimensional domain. The float values <cx> <cy> <cz> identify the coordinates of the center of the circle. The float value <radius> establishes the radius of the sphere. By definition, points interior to the sphere are assigned negative phase function function values.
SS {ss_id}	This card uses an existing sideset in the problem as a defined geometric object for construction of an interface. The parameter <ss_id> identifies this sideset.
USER {user-defined float list}	This card indicates the user has defined an object function using the supplied parameter float list that returns a signed distance value when supplied with the coordinates of a point in space. This object function should appear in the function call *user_init_object * in the file **user_pre.c.**
SM_object {object_type} {object_name}	This card allows the user to initialize the phase function location by using a piece of solid model geometry. The solid model object_type can be either **FACE** or ** BODY.** A 2D initialization uses the boundary of the specified FACE (or surface) as the 0 phase function. A 3D initialization uses the boundary of the specified BODY (or volume) as the 0 phase function.

Examples

Three examples of initialization methods for a single phase function are provide below:

```
Phase Function Initialization Method = Nodeset 20 EB 1
```

```
Phase Function Initialization Method = Surfaces 3
        SURF = PLANE -1. 0. 0. -3.
        SURF = CIRCLE -2 0 1
        SURF = CIRCLE -3 0 0.5
```

```
Phase Function Initialization Method = SM_object BODY my_blob
```

Technical Discussion

Please consult Level Set Initialization Method card for discussion.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Phase Function Renormalization Tolerance

```
Phase Funtion Renormalization Tolerance = <float>
```

Description / Usage

This parameter provides a means for controlling how often renormalization (redistancing) operations are performed on the phase function fields as they evolve by fixing the size of the deviation allowed between the average absolute magnitude of the phase function gradient near each respecitve interface and unity, the theoretical value observed for a pure distance function.

<float> Value of the tolerance, the allowable deviation.

The range of this parameter is any positive real number, however, it is rare to use values smaller than 0.1 or larger than 5.0. The value of the tolerance defaults to 0.5 if this card is not specified. Note that a global parameter value is applied to all phase function fields in the problem. Currently, there is no provision for each phase function field to have a unique value for this parameter.

This parameter is exactly analogous to the similarly named parameter used in standard level set interface tracking.

Examples

This is a sample renormalization card:

```
Phase Function Renormalization Tolerance = 0.25
```

Technical Discussion

The reader is referred to the Technical Discussion associated with Level Set Renormalization Tolerance card as it is virtually identical to the operation of it in the current context. The only thing to note is that each phase function is evaluted separately against this tolerance and each function is renormalized independently if the tolerance is exceeded. That is, exceeding the tolerance by one phase function field only triggers renormalization for that field. The other phase function fields are left unaltered.

FAQs

What is a proper value for this parameter? Values on the order of unity should work well. Renormalization based on gradient can be disabled completely by choosing a very large value for this parameter. Conversely, a very small value will always result in a renormalization step.

Is it possible to renormalize too often? Yes. Renormalization is an extraphysical procedure designed solely to improve the numerical performance of the interface tracker. As such, it can add or subtract volume to or from the phases represented by the interface contour. Renormalizing too often, therefore, can result in errors being introduced. The renormalization procedure, Huygens_Constrained, attempts to mitigate this effect.

Phase Function Renormalization Method

```
Phase Function Renormalization Method = <char_string>
```

Description / Usage

This card indicates the method to be used to renormalize the phase function fields during the course of the computation.

{char_string} A character string which specifies the type of method for renormalization. Choices for this string are: **Huygens, Huygens_Constrained, Correction.**

Each method is described below; see also the Technical Discussion.

Huygens In this method a set of m points P is constructed:

$$\mathbf{P} = \{(x_i, y_i, z_i)\,,\, i = 1, 2, \ldots m| \quad \phi_j\,(x_i, y_i, z_i) = 0\}$$

which in a sense represent a discretization of the interface location. The finite element interpolation functions are used to find exact locations for these points. For each mesh node j, a minimum distance D_j, can be found to this set of points. Renormalization is accomplished by replacing the phase field value at this node ϕ_j with D_j multiplied by the sign of the previous value for the phase field function. This method is fast and robust and reasonably accurate given sufficiently refined meshes using high order phase field interpolation. However, this method is prone to losing material if low order phase field interpolation is employed.

Huygens_Constrained This method renormalizes the function in much the same way as the **Huygens** method, except it employs a Lagrange multiplier to enforce a global integrated constraint that requires the volume occupied by the "negative" phase to remain unchanged before and after renormalization. This requirement makes this method better at conserving mass. However, since it enforces a global constraint, it is possible that material might be moved nonphysically around the computational domain.

Examples

This is a sample card:

```
Phase Function Renormalization Method = Huygens_Constrained
```

Technical Discussion

Renormalization is an operation particular to phase function (and level set) embedded interface tracking. The phase function fields are defined originally as distanes from a known curve or surface. This type of function offers benefits in terms of smoothness of representation and the easy with which interfacial physics can be included. However, typically we are evolving these functions using the commonplace advection operator:

$$\frac{D\phi_j}{Dt} = 0$$

which does not necessarily perpetuate the phase field as a distance function. Sharp gradients or flat regions in the function may therefore appear near the interface which have various detrimental effects on the accuracy of the solution. The solution that is most often used is to periodically construct the interfaces from the phase function field and renormalize the phase function fields, i.e. reevaluated them so that they return to being distance functions from the interface. In general, this is a satisfactory solution if the frequency of renormalization is not too great. To set the criteria for determining when to renormalize the phase functions see the *Phase Function Renormalization Tolerance* card.

1.4.6 Continuation Specifications

This section of input records is used to direct all automatic continuation procedures. The entire section is completely optional. Basically, automatic continuation can be accomplished in steady state simulations (see *Time Integration* card) through any one or combination of parameters. These parameters can be any one or combination of the input floats required on the boundary condition cards (see Section 4.10) or material property cards (see Chapter 5). The cards in this section are used to specify the parameters that will be marched automatically, the method of marching (e.g. zero-order, first-order, multiparameter first-order, etc.), the limits of parameter values, and other sundry options. Much of this capability can now be managed from the LOCA library package (Library of Continuation Algorithms - Salinger et al. 2002).

Input specifications for this section of input records is discussed in a separate, comprehensive manual (Gates, et. al., 2000); an update to this manual has been completed during the summer of 2006 (Labreche, et. al., 2006).

1.4.7 Hunting Specifications

The cards in this section are used to direct multiparameter continuation with the hunting technique, which is a linear, multiparameter capability. The user is referred to discussions for the *Continuation Specifications* for the important details of Continuation. As is true for the *Continuation Specifications*, this entire section is completely optional. *Hunting Specification* cards are used in conjunction with *Continuation Specifications.*

Input specifications for this section of input records is discussed in a separate, comprehensive manual (Gates, et. al., 2000); an update to this manual has been completed during the summer of 2006 (Labreche, et. al., 2006).

1.4.8 Augmenting Conditions Specifications

Input records in this section are used to direct the solution of augmenting constraints on the base system of differential equations. Addition of these conditions may require some programming in the file user_ac.c. This entire section of the input deck is optional.

Input specifications for this section of input records is discussed in a separate, comprehensive manual (Gates, et. al., 2000); an update to this manual has been completed during the summer of 2006 (Labreche, et. al., 2006).

1.4.9 Solver Specifications

This required section directs the nonlinear iteration strategy with associated parameters (e.g., Newton's method options), matrix solution strategy and parameters, and other sundry options and toggles for the pressure stabilization approach and linear stability analysis capability. With regard to the parameters associated with matrix solution methods, it is important to understand that there are two major classes of solvers - direct and iterative solvers. Direct solvers are the most robust, but can be computationally impractical for some larger systems. Iterative solvers and associated preconditioners are the only practical options for large-scale problems (viz., very large twodimensional problems and virtually all three-dimensional problems). Choosing the solver settings for good convergence of iterative matrix solvers can be an artful task for Navier-Stokes problems and other poorly conditioned systems. It is recommended that the user consult the comprehensive report by Schunk, et al. (2002) for an overview and further usage tips.

Total Number of Matrices

```
Total Number of Matrices = <integer> (1 default)
```

Description / Usage

<integer> Number of matrices to be used, should be 1 for fully coupled or > 1 for segregated solves

Examples

Following is a sample card for 2 matrices:

```
Total Number of Matrices = 2
```

Technical Discussion

References

Solution Algorithm

```
Solution Algorithm = {char_string}
```

Description / Usage

This required card selects an algorithm for the solution of the linear matrix system that arises at each Newton iteration (either for a steady-state solution or for the solution at each discrete time). Please note that at the time of this writing, new solver capabilities were being generated; although the following information was complete and accurate, it will likely be out of date by the time of publishing. Users should consult the CD version of this document in the Goma Documentation System for up to date options.

There are three major matrix solver packages accessible in *Goma*, two direct factorization collections and an iterative solver package. The first collection of direct factorization methods in *Goma* include the Sparse1.3 package (Kundert and Sangiovanni-Vincentelli, 1988) and Y12M direct factorization technique (Zlatev, Wasniewski and Schaumburg, 1981) accessible via the Aztec linear solver package. The second collection of direct factorization methods include two frontal

solvers, SNL_MPFRONT, an adaptation of R. Benner's implementation of Hood's (1976) frontal method, and UMFPACK (Davis and Duff, 1997). SNL_MPFRONT is a traditional frontal method while UMFPACK is a multi-frontal solver.

The Aztec 2.x linear solver package (Tuminaro, et. al., 1999) is the iterative solver component of *Goma*. A successor to the krysolve 1.0 package (Schunk and Shadid, 1992) and the Aztec 1.0 package (Hutchinson, Shadid and Tuminaro, 1995), Aztec 2.x includes support for distributed memory architectures and for matrices in either a modified sparse row (MSR) format or a variable block row (VBR) format, as well as their distributed memory extensions. Generally, convergence of these iterative methods can be accelerated by judicious use of a preconditioner (which many of the other *Solver Specifications* cards address).

The options for this input card are listed below, but additional usage comments are included as part of the Technical Discussion section of this card. These comments provide assistance in choosing the *Solution Algorithm* for your problem.

Valid options for {char_string} are as follows:

lu Direct factorization via Gaussian elimination using Sparse 1.3. This solver is robust even for poorly conditioned matrix systems. It is unavailable when running *Goma* on multiple processors.

front Direct factorization based on Benner's SNL_MPFRONT that eliminates equations and variables as the fully assembled rows of the matrix are acquired. This is the latest solver installed within *Goma* and users are encouraged to report their successes and failures with this option as part of testing. It is unavailable when running *Goma* on multiple processors.

umf/umff Direct factorization using UMFPACK. This multi-frontal solver has been hardwired to perform elimination only upon complete assembly. The **umff** option forces a full factorization every time, whereas **umf** does not. It is unavailable when running *Goma* on multiple processors.

y12m Direct factorization using the Y12M package. This package is accessible through the Aztec matrix solver interface and cannot be used for multiple processor computations. Other direct solvers are recommended against this one.

gmres Iterative solver from the Aztec package using the restarted generalized minimum residual method. Iterative solver options are important to convergence of this method, e.g. *Preconditioner, Size of Krylov subspace, Matrix,* etc.

cg Iterative solver from the Aztec package using the conjugate gradient method. Like other iterative solvers, the successful convergence of the conjugate gradient method for a linear system depends on preconditioners and other cards in the *Solver Specifications* section.

cgs Iterative solver from the Aztec package using the conjugate gradient squared method. Convergence of this method is frequently contingent on the linear system and on the choice of other cards in the *Solver Specifications* section.

tfqmr Iterative solver from the Aztec package using the transposefree quasi-minimum residual method. Convergence of this method is frequently contingent on the linear system and on the choice of other cards in the *Solver Specifications* section.

bicgstab Iterative solver from the Aztec package using the biconjugate gradient with stabilization. Convergence of this method is frequently contingent on the linear system and on the choice of other cards in the Solver Specifications section.

amesos Allows access to direct solver options implemented in parallel. Please see the user-notes below for Goma build options that must be exercised. This package is part of the Trilinos 6.0 framework. With this option, you must add an additional input card to specify the parallel direct solvers:

```
Amesos Solver Package = {superlu | mumps | klu | umfpack}
```

Of these four options, we currently recommend **mumps**. All options can be run in parallel.

stratimikos Interface to Trilino's Stratimikos package requires:

```
Matrix storage format = epetra
```

Allows block solvers, see also ref:*Stratimikos File*

petsc PETSc solver and preconditioner, will use *petscrc* file or *-petsc* command line, see Technical Discussion for more information

Examples

Following is a sample card:

```
Solution Algorithm = lu
```

Another example (two cards) shows how to invoke a parallel direct solver:

```
Solution Algorithm = amesos
```

```
Amesos Solver Package = superlu
```

Technical Discussion

The direct factorization options are the most robust but consume the most computational resources (CPU time and memory, particularly for large and 3D problems). The iterative methods consume less resources but may take some experimentation to obtain convergence to the solution of the linear system. For example, a poorly conditioned linear system may require a lot of preconditioning. The conjugate gradient method may not be very useful on linear systems that are not symmetric positive definite. Although the following guidelines are useful, selection of the "right" linear solver requires experience, understanding and sometimes, luck.

- **lu** - The Sparse1.3 direct solver, is the most robust solver in *Goma* in terms of obtaining successful convergence for even poorly conditioned matrix systems. A significant disadvantage, however, is that it can be computationally expensive for large problems. Not only do the memory and CPU requirements grow with problem size, but the initial symbolic factorization that seeks optimal reordering also consumes greater CPU resources with larger problem sizes. For example, a problem with 70,000 degrees of freedom that required 22 hours of CPU for the initial factorization required only 1/2 hour for subsequent factorizations. Furthermore, this solver is unavailable when *Goma* is run on multiple processors. Its robustness makes it an excellent choice for small- and medium-sized problems.
- **front** - This solver is an adaptation for *Goma* of R. Benner's frontal solver, which itself includes considerable improvements compared to the pioneering frontal solvers (Irons, 1970; Hood, 1976). The SNL_MPFRONT library is compiled and linked into *Goma* only by choice. Direct factorization is done as the fully assembled rows of the matrix are acquired. The frontal solver consumes CPU time roughly comparable to Sparse 1.3, with the noted advantage of eliminating intraelement fully summed equations as they are encountered and only keeping the active working matrix in-core, thereby reducing memory requirements and possible storage of matrix components to disk.
- **umf/umff** - UMFPACK 2.0d is a powerful direct solver that is generally faster than Sparse 1.3a, though it might lack the robustness of the latter on infrequent occasions. The implementation of UMFPACK within *Goma* is only barebones, i.e. the multi-frontal solver has been hardwired to perform elimination only upon complete assembly. Finally, usage of UMFPACK is governed by a license that limits usage to educational, research and benchmarking purposes by nonprofit organizations and the U.S. government. Please refer to the license statement contained in the UMFPACK distribution for exact details. This solver was implemented prior to **front** so it was the only direct solver alternative to lu for a period of time. User's should now evaluate performance of this solver against **front** on a case by case basis.
- **gmres, cg, cgs, tfqmr, bicgstab** - The convergence of each of these iterative solvers is highly influenced by the kind of preconditioning selected. Often, the method(s) will not converge at all without an appropriate level of preconditioning. GMRES is considered one of the best iterative methods available, although there are instances where each of the others is superior. It is a Krylov-based method and has an additional input card, *Size of Krylov subspace*.

As mentioned earlier, CG should only be used on systems that are symmetric positive definite. See the *Matrix subdomain* solver card, and other *Solver Specifications* cards for guidance on appropriate use of preconditioners; also consult Schunk, et. al. (2002).

- **amesos**: superlu, klu, umfpack - These solvers are all direct (not iterative, but based on Gaussian elimination) and can be run in parallel with mpi. We recommend these solvers when robustness is required over iterative solvers and when the matrix assembly time is excessive, which is often the case when overloaded equations like species diffusion, porous media equations, etc. are used. This option also performs well for three-dimensional problems of small to moderate size.
- **stratimikos**: mostly used for interfacing with Trilino's *Teko* but can also call full solver suite that is supported in Trilinos through xml files
- **petsc**: There are quite a lot of linear solvers and preconditioners available through PETSc and most are configured through either command line arguments using *-petsc* or using a *petscrc* file in your goma problem directory specifying petsc options

  Options are specified using the usual *ksp_type* and *pc_type* etc

```
-ksp_type gmres
-pc_type asm
... etc
```

  When in a segregated solve *ksp* and *pc* options should be prefixed with a 0-indexed *-sys#* corresponding to each matrix

```
-sys0_ksp_type gmres
-sys0_pc_type asm
-sys1_ksp_type gmres
-sys1_pc_type hypre
... etc
```

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

G. H. Golub and C. F. V. Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD 3rd ed. (1996)

For all other references, please see *References* at the end of this manual.

Matrix Storage Format

```
Matrix storage format = {msr | vbr}
```

Description / Usage

This optional card can be used to choose between two formats accepted by the Aztec 2.1 solver package. Valid options are:

msr modified sparse row format (see Schunk and Shadid, 1992). This option is the default option and is automatically used for all direct solver options.

vbr variable block row format (see Heroux, 1992). This option should only be selected when an Aztec iterative solver is chosen.

epetra Compressed Sparse Row format using the Epetra library from Trilinos

Examples

Following is a sample card:

```
Matrix storage format = msr
```

Technical Discussion

Goma supports two global matrix formats for its linear solvers. The advantage of choosing **vbr** over the default **msr** format is a matter of which preconditioner option is selected. (See Schunk, et al., 2002 on iterative methods.) When using the front solver package, another format known as **estifm** is employed internally but not specified by this card, which is not used in this case.

References

SAND92-1158: Iterative Solvers in Implicit Finite Element Codes, Sandia Technical Report, Schunk, P. R. and Shadid, J. N. (1992)

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

TR/PA/92/90: M. A. Heroux, A proposal for a sparse BLAS toolkit, Technical Report, CERFACS, December 1992.

Stratimikos File

```
Stratimikos File = <filename> (stratimikos.xml default)
```

Description / Usage

<filename> XML file where Stratimikos options are stored

Examples

Following is a sample card:

```
Stratimikos File = teko_ml_simplec.xml
```

Technical Discussion

References

Preconditioner

```
Preconditioner = {char_string}
```

Description / Usage

Iterative techniques for solving a linear matrix system (see above) often benefit from preconditioning to aid convergence. This optional card provides for the selection of a preconditioner from those available through Aztec. For direct factorization *Solution Algorithm* specifications, the *Preconditioner* specification is immaterial since none is performed; in such cases, this card should be omitted.

Valid options for {char_string} are listed below.

none No preconditioning is performed. This is the default specification if no preconditioner has been specified.

Jacobi A k-step Jacobi preconditioner is used (block Jacobi for VBR matrices). The number of Jacobi steps, k, is set using the *Matrix polynomial order* card.

Neumann A Neumann series polynomial preconditioner is used, where the order of the polynomial, k, is set using the *Matrix polynomial order* card.

ls A least-squares polynomial preconditioner is used, where the order of the polynomial, k, is set using the *Matrix polynomial order* card.

sym_GS A k-step symmetric Gauss-Seidel preconditioner is used for non-overlapping domain decomposition (additive Schwarz). In parallel, each processor performs one step of symmetric Gauss-Seidel on its local matrix, followed by communication to update boundary values from adjacent processors before performing the next local symmetric Gauss-Seidel step. The number of steps, k, is set using the *Matrix polynomial order* card.

lu Approximately solve the processor's local matrix via direct factorization using Sparse 1.3 in conjunction with a user-specified *Matrix drop tolerance.*

dom_decomp A domain-decomposition-based preconditioner (additive Schwarz). Each processor augments its local matrix according to the *Matrix factorization overlap* card and then approximately solves the resulting linear system using the solver specified by the *Matrix subdomain solver* card. This is the most often used *Preconditioner* card.

Examples

Following is a sample card:

```
Preconditioner = dom_decomp
```

Technical Discussion

Note that prior to Aztec 2.x, certain subdomain solvers were specified simply as arguments to the *Preconditioner* card. While this historical usage is permitted via limited backward compatibility in order to ease the transition from Aztec 1 usage, the preferred usage is to specify ILU (and similar) preconditioners as a subdomain solver using the more powerful and flexible options that are available using Aztec 2.x together with this option for the preconditioner. Since subdomain solvers such as ILU and ILUT are powerful and frequently used, this preconditioner option will predominate when iterative solvers are being used, even in serial execution.

The most popular setting is **dom_decomp**, with a subdomain solver specified in the *Matrix Subdomain Solver* card. For further details, consult Mike Heroux's recipe for applying preconditioners and what to dial the knobs to (in Schunk, et. al., 2002).

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

Matrix Subdomain Solver

```
Matrix subdomain solver = {char_string}
```

Description / Usage

This optional card selects a solver to use in constructing a preconditioner. It is used in conjunction with a *Preconditioner* setting.

```
Preconditioner = dom_decomp
```

All of these preconditioners are available through the Aztec library. Valid options for {char_string} are listed below.

lu Approximately solve the processor's local matrix via direct factorization using Sparse 1.3 in conjunction with a user-specified *Matrix drop tolerance.*

ilut Approximately solve the processor's local matrix via ILUT (Saad, 1994.) The factorization is affected by user-specified options for *Matrix drop tolerance* as well as *Matrix ILUT fill factor.*

This subdomain solver is among the more robust to recommend as a first attempt; thus it has been chosen as the default if no subdomain solver is specified.

ilu Approximately factor the processor's local matrix using ILU(k), where k is specified by the user in the argument to *Matrix graph fillin.*

rilu Approximately factor the processor's local matrix using RILU(k,ω), where k is specified by the user in the argument to *Matrix graph fillin* and ω is specified by the user in the argument to *Matrix RILU relax factor.* (This option applies only to Trilinos.)

bilu Approximately factor the processor's local matrix using block ILU(k) for a VBR format matrix, where k is specified by the user in the argument to *Matrix graph fillin.* While not the most efficient preconditioner, **bilu** is very robust. (This option applies only to Trilinos.)

icc Incomplete Cholesky factorization. See the Aztec manual for a reference.

If this *Matrix subdomain solver* card is omitted, then the default selection is **ilut.**

Examples

Following is a sample card:

```
Matrix subdomain solver = ilut
```

Technical Discussion

There is no real recipe to follow when choosing a preconditioner. In general, the cheapest preconditioner that works should be used. If ILUT(1) does the job, great. Sometimes the only preconditioner(s) that will work are very expensive. When the preconditioner seems to take too much time, remember that you may not be choosing the "wrong" preconditioner; the problem may just be that difficult.

Although Aztec 2.1 is being maintained and supported as a solver package for Goma, the interative solvers and preconditioners are now primarily accessed through the Trilinos library (as AztecOO), which is actively being developed and maintained at Sandia National Laboratories. Note that some features can only be accessed through Trilinos, as indicated above.

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

Saad, Y., 1994. "ILUT: a dual threshold incomplete ILU factorization", Numerical Linear Algebra with Applications, 1:387-402.

Matrix Scaling

```
Matrix scaling = {char_string}
```

Description / Usage

This optional card selects a scaling for the linear matrix system solution step.Valid options for {char_string} are listed below.

none No scaling is performed. This is the default if no *Matrix Scaling* card is present.

Jacobi Point Jacobi scaling is performed.

BJacobi Block Jacobi scaling is performed if the underlying matrix format is VBR. If the MSR matrix format is used, the scaling reverts to point Jacobi.

row_sum Scale each row so the sum of the magnitudes of the nonzero elements is 1.

sym_diag Symmetric scaling so that diagonal elements are 1.

sym_row_sum Symmetric scaling using the matrix row sums.

If the *Matrix Scaling* card is omitted, the default selection is **none.**

Examples

Following is a sample card:

```
Matrix scaling = sym_diag
```

Technical Discussion

All of these scalings are supplied via the Aztec library and thus will not affect the linear systems that are solved by other means (using **front**, for example). In an odd twist of fate, the linear system always undergoes a row sum scaling (equivalent to the **row_sum** option) before these other scalings are applied. Note that when a nontrivial scaling is selected, the matrix is overwritten with a rescaled system.

Matrix Residual Norm Type

```
Matrix residual norm type = {char_string}
```

Description / Usage

This optional card selects the type of norm that is used to measure the size of the residuals occurring during the solution of the linear matrix system $r\left(z\right) = b - Az$, where z is an approximation to the solution x of the linear matrix problem $Ax = b$. The types of norms used by the linear solver are controlled by values of {char_string}:

{char_string}	Norm type
r0	$\frac{\|r\|_2}{\|r^0\|_2}$
rhs	$\frac{\|r\|_2}{\|b\|_2}$
Anorm	$\frac{\|r\|_2}{\|A\|_\infty}$
sol	$\frac{\|r\|_\infty}{\|A\|_\infty \|x\|_1 + \|b\|_\infty}$
noscaled	$\|r\|_2$

The (0) superscript for the **r0** specification indicates the initial value of the residual.

If the *Matrix residual norm type* card is omitted, the default is **r0.**

Examples

Following is a sample card:

```
Matrix residual norm type = r0
```

Technical Discussion

For direct factorization linear solution algorithms, the norm should become very small in the single iteration that is performed. This card is more pertinent when an iterative solution algorithm has been specified.

Note the distinction between the residual for the overall global Newton iteration and use of the term residual to describe an aspect of the linear solver iteration. For the linear matrix systems, a residual r may be computed for any guess of the solution to $Ax = b$ as $r(z) = b - Az$. If $z = x$, the actual solution, then the residual is zero; otherwise, it is some vector with a nonzero norm.

Matrix Output Type

```
Matrix output type = {char_string}
```

Description / Usage

This optional card indicates a level of diagnostic output for Aztec. The valid input parameters for {char_string} are either a string or a positive integer:

all Print matrix and indexing vectors for each processor and all intermediate residual expressions.

none No intermediate results are printed. This is the default.

warnings Only Aztec warnings are printed.

last Only the final residual expression is printed.

k Residual expressions are printed every *k* iterations, k > 0.

If the *Matrix output type* card is omitted, the default is **none.**

Examples

Following is a sample card:

```
Matrix output type = 10
```

Matrix Factorization Reuse

```
Matrix factorization reuse = {char_string}
```

Description / Usage

This optional card directs the approximate factorization solvers used in preconditioner construction to reuse matrix information that may have been obtained during previous linear solution stages. This card only has an effect when using an Aztec solver. Valid options for {char_string} are:

calc Use no information from previous linear solutions.

recalc Use information from previous linear solutions but recalculate the preconditioning factors, with the implication that the symbolic factorization will be similar.

reuse Use information from previous linear solution; do not recalculate preconditioner factorizations. However, use scaling factors from previous linear solutions to scale righthand sides, initial guesses, and final solutions.

If the *Matrix factorization* reuse card is omitted, the default is **recalc.**

Examples

Following is a sample card:

```
Matrix factorization reuse = recalc
```

Technical Discussion

See related discussions for *Matrix factorization save*.

Matrix Graph Fillin

```
Matrix graph fillin = <integer>4.7.10_matrix_factorization_overlap.txt
```

Description / Usage

This optional card sets the graph level of fill-in for approximate factorizations used in preconditioner construction for ILU(k), ICC(k) and BILU(k). The input parameter is defined as

<integer> k, specifies the graph level of fill-in, k > 0.

If the *Matrix graph fillin* card is omitted, the default value of **k** is **0.**

Examples

Following is a sample card:

```
Matrix graph fillin = 2
```

Technical Discussion

As the level of graph fill-in increases, the accuracy (usefulness) of the preconditioner increases; however, so does memory usage as well as the time required to compute the preconditioner.

Matrix Factorization Overlap

```
Matrix factorization overlap = {char_string}
```

Description / Usage

This optional card determines how much matrix factorization overlap occurs with other processors. This specification is only relevant for parallel computations. The valid options for {char_string} are:

none No augmentation is performed, equivalent to a setting of k=0. This is the default.

diag Augment the processor's local matrix to include the diagonal (MSR) or diagonal blocks (VBR) for external rows.

k Augment the processor's local matrix to include external rows. The rows are selected by examining non-zero columns from the current local system that refer to offprocessor unknowns, and including the rows associated with those off-processor unknowns. This process is repeated *k* times, where k > 0. When complete, all non-zero columns whose associated rows have not been included are discarded. A value of 0 is equivalent to a setting of none.

If the *Matrix factorization overlap* card is omitted, the default is **none.**

Examples

Following is a sample card:

```
Matrix factorization overlap = 1
```

Technical Discussion

This optional card determines how much a processor's local matrix is to be augmented with information from adjacent processors during the approximate factorizations used to build preconditioners. This card should be omitted or given a value of **none** for serial executions.

Matrix Overlap Type

```
Matrix overlap type = {standard | symmetric}
```

Description / Usage

This card selects the kind of matrix overlap that occurs (for parallel computations). Valid options are:

standard The local processor considers only its own estimate for any unknown; results from adjacent processors are ignored. This is the default.

symmetric The local processor adds its own estimate together with estimates from adjacent processors, retaining symmetry of preconditioners if a symmetric technique is being employed.

If the *Matrix ovelap type* card is omitted, the default is **standard.**

Examples

Following is a sample card:

```
Matrix overlap type= symmetric
```

Technical Discussion

This optional card determines how overlapping subdomain solver results are combined when different processors derive different estimates for the same solution unknown.

This overlap option is moot for serial problems whose data decomposition is trivial.

Matrix Auxiliary Vector

```
Matrix auxiliary vector = {resid | rand}
```

Description / Usage

This optional card indicates to Aztec how the auxiliary vector r is determined. Permissible options are:

resid The auxiliary vector is set to the initial residual vector, viz. $r = r(0)$.

rand The auxiliary vector is filled with random numbers, each in the range $[-1, 1]$.

If the *Matrix auxiliary vector* card is omitted, the default is **resid.**

Examples

Following is a sample card:

```
Matrix auxiliary vector = rand
```

Technical Discussion

The auxiliary vector is only used for certain iterative linear matrix solution algorithms.

The **rand** option may cause difficulties with initial iterative solver steps because different processors may have different initial unknown values at shared unknowns.

Matrix Drop Tolerance

```
Matrix drop tolerance = <float>
```

Description / Usage

This optional card indicates to Aztec a drop tolerance to be used in conjunction with preconditioners based on LU or on ILUT. The <float> input parameter is:

<float> tol, a floating point number ($tol \geq 0$) that specifies the drop tolerance.

If the *Matrix drop tolerance* card is omitted, the default is **0.0.**

Examples

Following is a sample card:

```
Matrix drop tolerance = 0.01
```

Technical Discussion

When constructing the partial factorization(s), any value less than tol is dropped. If set to 0.0, then other parameters will govern preconditioner size and components (e.g., *Matrix ILUT fill factor* for the ILUT preconditioner).

The two main parameters when using the ILUT preconditioner are this card and the *Matrix ILUT fill factor* card. The restrictions in *Matrix ILUT fill factor* take precedence over the dropped entries caused by this card.

Matrix Polynomial Order

```
Matrix polynomial order = <integer>
```

Description / Usage

This optional card allows selection of polynomial order when a polynomial preconditioning option is selected (see the *Preconditioner* card). The input parameter is defined as:

<integer> Number of steps, **k** (≥ 0), to take when using matrix polynomial based preconditioners (Jacobi and symmetric Gauss-Seidel, for example).

If the Matrix polynomial order card is omitted, then the default selection is **k=3.**

Examples

Following is a sample card:

```
Matrix polynomial order = 4
```

Technical Discussion

When used, the value of this parameter should be greater than 0, and probably no more than 10. In some, if not all, cases, a value of 0 is meaningless.

This card is not used if the preconditioner does not use matrix polynomials.

Matrix Reorder

```
Matrix reorder = {none | rcm}
```

Description / Usage

This optional card determines whether RCM (Reverse Cuthill-McKee) reordering of the linear system is to be performed. Valid options are:

none the equations are not reordered.

rcm the equations are reordered using an RCM scheme.

If the *Matrix reorder* card is omitted, then the default selection is **none.**

Examples

Following is a sample card:

```
Matrix reorder = rcm
```

Technical Discussion

Note that reordering frequently is helpful in achieving convergence for iterative solution of linear systems. In a few instances, however, *Goma* users have noted that RCM reordering hinders convergence for selected problems. The default for *Goma* is to not use the RCM reordering so that quantitatively comparable results are obtained using either Aztec 1 (which did not have RCM reordering as an option) or Aztec 2.x. In summary, users are encouraged to try RCM reordering when using iterative solvers, foregoing the option only as a further resort in the face of repeated convergence failures.

Matrix Factorization Save

```
Matrix factorization save = {0 | 1}
```

Description / Usage

This optional card is a boolean specification that determines whether the preconditioner factorization information should be kept after a solve. Valid options are

0 Factorization information is discarded.

1 Factorization information is kept for that step.

If the *Matrix factorization save* card is omitted, then the default selection is **0.**

Examples

Following is a sample card:

```
Matrix factorization save = 1
```

Technical Discussion

This option is most useful for iterative solution techniques where the computed preconditioning matrix found from an incomplete factorization requires significant computational resources. Such a preconditioner may be useful in later matrix solves and obviate the need to compute another expensive preconditioner at the later stage. Although a lot of time may be saved by re-using a previous factorization, the loss in accuracy may cause convergence problems.

Matrix ILUT Fill Factor

```
Matrix ILUT fill factor = <float>
```

Description / Usage

This optional card provides a second criterion to Aztec to be used in conjunction with preconditioners based on ILUT approximate factorization, where

<float> fac, a floating point value (fac ≥ 0) that specifies, very crudely, how many nonzero entries the approximate factorization will contain relative to the number of nonzero entries in the original matrix.

If the *Matrix ILUT fill factor* card is omitted, the default is **1.**

Examples

Following is a sample card:

```
Matrix ILUT fill factor = 2.0
```

Technical Discussion

By increasing this factor, the preconditioner becomes more accurate because more terms in the preconditioner (pseudo-inverse) are retained. A value of 1.0 indicates that the preconditioner would contain approximately the same number of nonzero entries as the original matrix.

The two main parameters when using the ILUT preconditioner are this card and the *Matrix drop tolerance* card. If the *Matrix drop tolerance* is 0.0, then this card determines the size of the preconditioner. If *Matrix drop tolerance* is greater than 0.0, then the approximate factorization is first created subject to this card's restriction, and then the drop tolerance is applied. This can result in a preconditioner with significantly fewer nonzero entries.

Matrix RILU Relax Factor

```
Matrix RILU relax factor = <float>
```

Description / Usage

This optional card provides a relaxation factor to Aztec to be used in conjunction with preconditioners based on RILU(k,ω) approximate factorization. The input parameter <float> is defined as

<float> fac, a floating point number (fac ≥ 0) that specifies a relaxation factor.

If the *Matrix RILU relax factor* card is omitted, the default is **1.**

Examples

Following is a sample card:

```
Matrix RILU relax factor = 0.5
```

Technical Discussion

Some limiting values for fac provide specific behavior:

- for a value of zero, the ILU(k) is obtained
- for a value of one, the MILU(k) is obtained.

The value of *k* is set by the *Matrix graph fillin* card.

Matrix BILU Threshold

```
Matrix BILU threshold = <float>
```

Description / Usage

This capability is only present within the Trilinos library. This optional card provides a means to modify the way the block ILU preconditioner (*Matrix subdomain solver* = **bilu**) is constructed. The input parameter is defined as:

<float> t, a floating point number (t ≥ 0.0) that sets the *Matrix Relative Threshold* and *Matrix Absolute Threshold* thresholds.

When the *Matrix BILU threshold* card is omitted, the default value is 0.0.

Examples

Following is a sample card:

```
Matrix BILU Threshold = 1.0e-14
```

Technical Discussion

Using this card is equivalent to supplying both the *Matrix Relative Threshold* and *Matrix Absolute Threshold* with the value specified with this card.

The value of **t** defaults to zero, and if given a small value, say 1.0e-14, the condition number of the preconditioner, as reported when using the **bilu** option, should decrease. Try increasing up to around 1.0e-3 to get added benefit. The **bilu** preconditioner is not actually the cheapest or most efficient preconditioner, but it is very robust.

Matrix Relative Threshold

```
Matrix Relative Threshold = <float>
```

Description / Usage

This card is only available with the Trilinos library. The effect of this card is to impose a relative lower bound to either a diagonal value or a singular value. The legal values for <float> are:

<float> **r**, a floating point number (r ≥ 0.0) that specifies a relative threshold.

If this card is omitted, the default is 0.0.

Examples

A sample input card follows:

```
Matrix Relative Threshold = 1.e-4
```

Technical Discussion

This card, along with the *Matrix Absolute Threshold* card, allow the user to modify the linear system prior to calculation of the preconditioner. Note that the modification is only to change the "initial condition" of the preconditioner–it does not actually change the linear system.

Let **t** be the value specified with the Matrix Absolute Threshold card. For a scalar-based preconditioner (**ilut, ilu, rilu, icc**), each value on the diagonal undergoes the following substitution:

$$d_{\text{new}} = r * d_{\text{old}} + \text{sgn}\left(d_{\text{old}}\right) * t$$

For the **bilu** preconditioner, each singular value of the diagonal block preconditioner is compared to:

$$\sigma_{\text{min}} = r * \sigma_1 + t$$

where σ1 is the largest singular value of the diagonal block under consideration. All σ_k are modified (if necessary) to be at least as large as σ_{min}.

The appropriate values for the threshold can vary over many orders of magnitude depending on the situation. Refer to Schunk, et. al., 2002 for information and for further guidance.

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

Matrix Absolute Threshold

```
Matrix Absolute Threshold = <float>
```

Description / Usage

This card is only available with the Trilinos library. It allows the user to specify a lower bound for either a diagonal entry or a singular value. The exact meaning depends on the kind of preconditioner used (scalar-based or block-based). The legal values are:

<float> t, a floating point number ($t \geq 0.0$) that specifies a minimum threshold value for diagonal or singular value.

Along with the *Matrix Relative Threshold* card, this card gives the user the ability to modify what matrix the preconditioner operates on. See the *Matrix Relative Threshold* card for a full description.

If this card is omitted, the default is 0.0.

Examples

A sample input card follows:

```
Matrix Absolute Threshold = 1.e-4
```

Technical Discussion

Refer to the discussion for card *Matrix Relative Threshold*. The appropriate values for the threshold can vary over many orders of magnitude depending on the situation. Refer to Schunk, et. al., 2002 for information and for further guidance.

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun, March 2002.

Size of Krylov Subspace

```
Size of Krylov subspace = <integer>
```

Description / Usage

This optional card allows the user to specify the dimension (size) of the Krylov subspace for the **gmres** option of the *Solution Algorithm* card, where

<integer> m, specifies the number of orthogonalization directions and can be any positive integer less than or equal to the order of the matrix.

If the *Size of Krylov subspace* card is omitted, then the default dimension is **m** = 30.

Examples

The following is a sample input card:

```
Size of Krylov subspace = 128
```

Technical Discussion

If the size of the subspace is at least as large as the maximum number of iterations permitted by the solver then the **gmres** iteration will not include any restarts. Depending on the problem, restarts may be beneficial, and then again they may not. Particularly poorly conditioned linear systems may never converge below a certain tolerance if **gmres** is allowed to restart (i.e. they "level off"). However, some linear systems will admit a converged solution more rapidly with restarts than without. Consequently, the user may wish to experiment with different values of this parameter. See the Orthogonalization card for related information.

gmres' internal iterations create a Krylov subspace up to dimension m (less in some circumstances, such as convergence). The time and space required by the internal iterations increases nonlinearly with **m** (but see the *Orthogonalization* card) - a doubling of **m** will result in more than a doubling of space and time requirements. So simply choosing a very large dimension is generally not recommended.

Orthogonalization

```
Orthogonalization = {classic | modified}
```

Description / Usage

This optional card selects the orthogonalization scheme used internally for the gmres solution algorithm (see the *Solution Algorithm* card). Valid options are

classic | classical Two steps of classical Gram-Schmidt orthogonalization.

modified A modified Gram-Schmidt orthogonalization.

If the *Orthogonalization* card is omitted, then the default selection is **classic.** Goma's parser will accept **classical** as equivalent to **classic.**

Examples

Following is a sample card:

```
Orthogonalization = modified
```

Technical Discussion

By specifying **modified,** the user is greatly speeding up the **gmres** algorithm at the expense of possibly losing convergence. A good indication that you should not have used the **modified** setting is a premature "leveling off" of the sequence of residuals produced internally within **gmres.**

Maximum Linear Solve Iterations

```
Maximum Linear Solve Iterations = <integer>
```

Description / Usage

This optional card limits the maximum number of iterations used by iterative linear solver algorithms. The input parameter is defined as

<integer> <integer> **n,** any positive integer (n > 0) that specifies the maximum number of iterations.

If the *Maximum Linear Solve Iterations* card is omitted, the default selection is 500.

Examples

Following is a sample card:

```
Maximum Linear Solve Iterations = 5
```

Technical Discussion

If the linear system can be solved within a specified tolerance (see the *Residual Ratio Tolerance* card) in less than **n** iterations, then a normal return from Aztec occurs and the actual number of iterations required to obtain convergence will be printed on the status line. If the specified convergence tolerance is not met within **n** iterations, then an abnormal return status occurs and, in place of the number of iterations, the string "max" will be printed on the status line under the LIS (linear iteration status) heading. Other abnormal returns from Aztec are possible and are indicated on the LIS status line; see the Aztec User's Guide (Hutchinson, Shadid and Tuminaro, 1995) for further interpretation of different abnormal return status indicators.

References

SAND95-1559: Aztec User's Guide Version 1.0, Sandia Internal Report, Hutchinson, S. A., Shadid, J. N. and Tuminaro, R. S., 1995.

Number of Newton Iterations

```
Number of Newton Iterations = <integer1> [integer2]
```

Description / Usage

This required card sets the maximum number of iterations allowed for convergence of the Newton nonlinear iteration loop. It also provides an optional parameter for setting the reformation stride for the Jacobian matrix. Definitions of the input parameters are as follows:

<integer1> n_1, any integer indicating the maximum number of iterations allowed for convergence of the Newton nonlinear iteration loop.

[integer2] n_2, an optional parameter indicating the reformation stride for the Jacobian matrix.

The *Number of Newton Iterations* card is required, there is no default.

See the *Jacobian Reform Time Stride* card for some detailed examples of the interaction amongst various input parameters that influence when a Jacobian reformation occurs.

Examples

Following is a sample card:

```
Number of Newton Iterations = 5
```

Technical Discussion

For an unrelaxed Newton iteration with a good initial guess, five or six iterations (for n_1) should be sufficient to achieve convergence for most problems. One iteration will suffice for problems that are linear; two can be specified, with the second iteration verifying that the residual norms are small. More iterations may be required for relaxed Newton iteration schemes using the correction factor described in the *Newton correction factor* card. This parameter can also be controlled from the command line (see the **-n** option in the section on Command-line Arguments, Chapter 3).

The optional second parameter can be used to invoke a modified Newton iteration. If this value is missing, the stride is set to unity. This capability enables the user to save on assembly time when near a solution, particularly when doing transient simulations.

Modified Newton Tolerance

```
Modified Newton Tolerance = <float1> <float2>
```

Description / Usage

This optional card allows the user to exert finer control over Jacobian formation than a stride specification (as with the *Number of Newton Iterations* card's second parameter or the *Jacobian Reform Time Stride* card). Input parameters are defined as:

<float1> r, if the convergence rate is below this level (r > 0.0), a Jacobian reformation will be forced.

<float2> t, if the residual norm is above this level (t ≥ 0.0), a Jacobian reformation will be forced.

If the *Modified Newton Tolerance* card is omitted, then reformations are always computed, subject to the *Number of Newton Iterations'* second parameter and the *Jacobian Reform Time Stride* value.

See the *Jacobian Reform Time Stride* card for some detailed examples of the interaction amongst various cards that influence when a Jacobian reformation occurs.

Examples

Following is a sample card:

```
Modified Newton Tolerance = 1.5 1.0e-8
```

Technical Discussion

The convergence rate is defined as:

$$\text{convergence rate} = \frac{\log\left(\text{current} L_1 \text{norm}\right)}{\log\left(\text{previous} L_1 \text{norm}\right)}$$

This rate should be equal to 2 when Newton's method is in its region of convergence (this is what it means to converge quadratically). A secant method would have a convergence rate of $1 + \sqrt{5}/2$ (the golden ratio!), approximately 1.6.

The residual norm is simply the L_1 norm of the residual after a Newton iteration.

The method used to determine if a Jacobian reformation should take place is conservative. If either test condition for reformation is satisfied, a reformation occurs. Often, this card will allow you to speed up your runs by foregoing a fresh Jacobian reformation, but still maintain strong convergence. Moreover, without a Jacobian reformation, the **lu** solver (see the *Solution Algorithm* card) can use a previously factored matrix and simply do a resolve.

Jacobian Reform Time Stride

```
Jacobian Reform Time Stride = <integer>
```

Description / Usage

This optional card has a single input parameter:

<integer> k, the stride length for Jacobian reformations ($k \geq 1$).

The *Jacobian Reform Time Stride* card is optional; there is no default.

Examples

Three examples are provided to illustrate how to use this card.

Example #1:

```
Number of Newton Iterations = 12 1

Modified Newton Tolerance = 1.9 0.1

Jacobian Reform Time Stride = 2

Newton correction factor = 1
```

This will reform the Jacobian every 2 steps. Furthermore, if the convergence rate falls below 1.9 or the L_1 residual is greater than 0.1 on an off-stride step a Jacobian reformation will occur. Specifically, the*Modified Newton Tolerance* takes precedence over a reformation stride setting (from either *Number of Newton Iterations or Jacobian Reform Time Stride*).

Example #2:

```
Number of Newton Iterations = 12 1

# Modified Newton Tolerance = 1.9 0.1

Jacobian Reform Time Stride = 2

Newton correction factor = 1
```

Note this differs from the previous example only by omitting the *Modified Newton Tolerance* card. This causes the Jacobian to be reformed every other time step.

Example #3:

```
Number of Newton Iterations = 12 2

# Modified Newton Tolerance = 1.9 0.1

Jacobian Reform Time Stride = 1

Newton correction factor = 1
```

We've changed the *Jacobian Reform Time Stride* from 2 to 1 and changed the second parameter of the *Number of Newton Iterations* card from 1 to 2. This will cause the Jacobian to be reformed every other step.

Technical Discussion

If the second parameter on the *Number of Newton Iterations* card is present and greater than 1, this *Jacobian Reform Time Stride* card is ignored. Otherwise, this card simply forces the Jacobian to be rebuilt every **k** Newton steps. Often, this card will allow you to speed up your runs by foregoing a fresh Jacobian formation, but still maintain strong convergence. Moreover, without a Jacobian formation, the **lu** solver (see the *Solution Algorithm* card) can use a previously factored matrix and simply do a resolve.

Newton Correction Factor

```
Newton correction factor = <float_list>
```

Description / Usage

This required card indicates the damping (or relaxation) factor for the Newton updates and offers customization of the relaxation choice based on the size of the nonlinear residual from the Newton iteration. Definitions of the <float_list> input parameters, from one (f_1) to six (f_2, … f_6) floating point numbers (one required and five optional), are as follows:

<float1> f_1, damping factor for the Newton updates, where ($0.0 < f_1 \leq 1.0$). A value of 1.0 gives the usual Newton's method, otherwise, only a portion of the Newton update is applied to the solution. Values near 0 (e.g., 0.1) may be used effectively to aid convergence for sensitive problems where the initial guess is not very close to the final solution for the first several Newton iterations. This parameter can also be controlled from the command line (see **-r** option, Command-line Arguments, Chapter 3).

[floatn] These five floats [f_2, … f_6] are optional but give a way to more finely control the amount of relaxation applied to Newton updates. See the description below and the example for an explanation.

Examples

A simple example is the following:

```
Newton correction factor = 0.1
```

This tells *Goma* to take the specified number of Newton iterations (from the *Number of Newton Iterations* card) at a fixed relaxation parameter of 0.1. This is a moderately large amount of relaxation, but of course "moderately large" is always problem dependent.

A more interesting example:

```
Newton correction factor = 0.8 1.0e-6 0.4 1.0e-4 0.1 1.0e-3
```

causes the following relaxation scheme to be used according to the L_∞ norm of the nonlinear residual:

- If $L_\infty > 1.0e-3$, the relaxation factor is taken as 0.1.
- If $1.0e-4 < L_\infty \leq 1.0e-3$, the relaxation factor is taken as 0.4.
- If $1.0e-6 < L_\infty \leq 1.0e-4$, the relaxation factor is taken as 0.8.
- If $L_\infty \leq 1.0e-6$, the relaxation factor is taken as the usual Newton's method relaxation of 1.0.

The default relaxation level for small residuals is 1.0.

Technical Discussion

The relaxation factor is used to intentionally shorten the solution update vector computed by the Newton iteration. There are many factors that can cause the effective radius of convergence of Newton's method to be quite small or malformed:

- the underlying nonlinear problem is stiff,
- the initial solution is poor,
- non-analytic constitutive models or boundary conditions,
- poor linear solver performance, etc.

Under these kinds of circumstances, the update computed by Newton's method may be too large and end up not improving the overall solution. In such cases it is recommended that one uses some relaxation (e.g., 0.9), and possibly a lot (e.g., 0.05).

What one really wishes to do is to use shortened updates when far from convergence, and full updates as the solution converges. This is the capability that the optional five parameters makes available. While they don't directly measure how far the solution is from convergence, it does use the residual as an indicator. The full set of six parameters allows the user to specify four different residual intervals with four different relaxation factors. The f_1, f_3 and f_5 values are relaxation factors and must lie in $0.0 < f_i \leq 1.0$, while the f_2, f_4, and f_6 values are interval endpoints. The supplied interval endpoints must be in ascending order, $0 < f_2 < f_4 < f_6$. Although no such restriction is put on the relaxation factors, they should generally satisfy $0 < f_5 \leq f_3 \leq f_1 \leq 1.0$.

Normalized Residual Tolerance

```
Normalized Residual Tolerance = <float>
```

Description / Usage

This required card indicates the value of the L_2 norm of the global nonlinear residual vector that indicates termination of Newton's method (i.e., convergence). The input parameter is defined as

<float> tol, a non-negative floating point number (tol $\geq$ 0.0) specifying the L_2 convergence tolerance for the global nonlinear residual vector.

The *Normalized Residual Tolerance* card is required; there is no default.

Examples

Following is a sample card:

```
Normalized Residual Tolerance = 1.0e-11
```

Technical Discussion

Newton's method is terminated when the global nonlinear residual falls below tol, or the maximum number of iterations specified in the *Number of Newton Iterations* is reached.

Normalized Correction Tolerance

```
Normalized Correction Tolerance = <float>
```

Description / Usage

This optional card sets the tolerance for a mixed measure of the size of the update vector which must be satisfied for the solution to be considered converged. The input parameter is defined as

<float> rel, a floating point value (rel ≥ 0.0) used as the convergence tolerance for the mixed measure of the update vector (defined in the Technical Discussion).

When the *Normalized Correction Tolerance* card is omitted, the default value of **rel** is 1.0e+10.

Examples

Following is a sample card:

```
Normalized Correction Tolerance = 1.0e-4
```

Technical Discussion

The mixed measure used here is:

$$\sqrt{\sum \frac{{\Delta x_i}^2}{1 + {x_i}^2}}$$

This measures the relative size of the update vector when the solution vector is large (i.e., size of unknowns is greater than 1), and measures the absolute size of the update vector when the solution vector is small (i.e., size of unknowns is much less than 1).

This mixed measure must be less than **rel,** in addition to the nonlinear residual satisfying the absolute residual tolerance specified in the *Normalized Residual Tolerance* card for a solution to be considered converged.

If rel < 1.0 (larger values are not really imposing any restrictions), mixed measure values are output instead of the update vector norms.

Residual Ratio Tolerance

```
Residual Ratio Tolerance = <float>
```

Description / Usage

This optional card sets the convergence criterion for the iterative solution of the linear matrix system solved at each Newton iteration. The input parameter is defined as

<float> tol, a non-negative real number (tol ≥ 0.0) specifying the value of the convergence criterion.

The default value of **tol** is 1.0e-6.

Examples

Following is a sample card:

```
Residual Ratio Tolerance = 1.0e-3
```

Technical Discussion

The value of **tol** is ignored when a direct factorization algorithm (such as **lu**) for the linear solve is specified in the *Solution Algorithm* card. When an iterative matrix solution technique is specified (such as **gmres**), **tol** acts as the inner iteration termination relative tolerance. Letting r_0 represent the initial residual norm, when the n_{th} iteration's linear residual norm r_n satisfies r_n / r:sub :*0* ≤ tol, the iterative solution is deemed acceptable and the inner iterations terminate. The number of iterations required is reported under the LIS column of the Newton iteration output. If the maximum number of iterations (specified in the *Maximum Linear Solve Iterations* card) is reached, then **max** appears instead of a number. Although the standard residual is usually used as the residual norm, the type of matrix residual norm used can be changed through the *Matrix residual norm type* card.

Pressure Stabilization

```
Pressure Stabilization = {yes | no | local | pspp | pspp_e}
```

Description / Usage

This optional card indicates whether or not pressure stabilization should be used. Valid options are

yes Use the Galerkin Least square pressure stabilization method developed by Hughes, et. al. (1986).

local Use the Galerkin Least square pressure stabilization method with local scaling.

pspp Use polynomial stabilized pressure projection stabilization method developed by Dohrmann and Bochev (2004). Please see Level Set PSPP filtering card if using with the level-set front tracking technique.

pspp_e Use polynomial stabilized pressure projection method with upgrade for nonuniform/graded meshes (recommended)

no Do not use any pressure stabilization.

The amount of pressure stabilization to use is specified with the *Pressure Stabilization Scaling* card.

The default is **no**, to not use pressure stabilization.

Examples

Following is a sample card:

```
Pressure Stabilization = yes
```

Technical Discussion

If input for this card is **yes**, the Hughes, et al. (1986) method adds the residual of the momentum equation weighted by the gradient of the Galerkin weight function to the Galerkin continuity equation. The result is that the continuity equation now has a diagonal term to stabilize it and improve the condition of the matrix, allowing for the use of iterative solvers. When pressure stabilization is used, equal-order interpolation can (and should) be used for velocity and pressure, e.g., velocity and pressure both Q2 or both Q1. If input for this card is **no**, then the standard Galerkin finite-element weight functions are used and velocity and pressure interpolations should be chosen to satisfy the Babuska-Brezzi condition, e.g., velocity Q2 and pressure Q1 or P1, or velocity Q1 and pressure P0.

An improvement on the Hughes approach was developed by Bochev and Dohrmann (2004) called the polynomial stabilized pressure projection. In its fundamental form, it is like PSPG just an additional term on the continuity equation residual that helps stabilize the pressure, and it is predicated on the fact that the pressure field is governed by an elliptical equation known as the pressure Poisson equation. Please consult this paper for details. An additional improvement to that technique was developed internally to Sandia which better accommodates graded meshes. This technique is invoked with the pspp_e option, which we recommend.

References

Hughes, T. J. R., L. P. Franca and M. Balestra, "A New Finite Element Formulation for Computational Fluid Dynamics: V. Circumventing the Babuska-Brezzi Condition: A Stable Petrov-Galerkin Formulation of the Stokes Problem Accommodating Equal- Order Interpolations," *Comput. Methods Appl. Mech. Engrg.*, 59 (1986) 85-99.

Pressure Stabilization Scaling

```
Pressure Stabilization Scaling = <float>
```

Description / Usage

This optional card is only used if the Pressure Stabilization card is set to **yes,** where

<float> tau, a positive real value (tau > 0.0) that scales the momentum residual being added to the continuity equation for pressure stabilization.

The default value of **tau** is 0.1. If the *Pressure Stabilization* card is omitted, or set to **no,** then **tau** is ignored.

Examples

Following is a sample card:

```
Pressure Stabilization Scaling = 0.01
```

Technical Discussion

Generally, if **tau** is small, then more accurate solutions may be obtained at the cost of a more ill-conditioned matrix system that may not be easily amenable to iterative solvers (but stay tuned!). Conversely, larger values of this parameter result in equation systems that are easier to solve using the available iterative matrix solvers, but the solution thus obtained may be less accurate. A good choice for **tau** is 0.1.

The scaling value, **tau,** is further scaled inside of *Goma*. Knowledge of this scaling is sometimes useful. First, an average Reynolds number (*Re*) is computed according to:

$$\mathrm{Re} = \frac{\rho \|U\| \langle h \rangle}{2\mu}$$

where ρ and μ are local values for density and viscosity, $\|U\|$ is a norm of the velocity field, and $\langle h \rangle$ is a global average value for element size. If Re < 3.0, the pressure stabilization scaling is given by this expression:

$$\frac{\mathrm{tau} \langle h \rangle^2}{12\mu}$$

On the other hand, if Re > 3.0, the following scales the pressure stabilization terms in the continuity equation:

$$\frac{\mathrm{tau} \langle h \rangle}{2\rho \|U\|}$$

Linear Stability

```
Linear Stability = {char_list}
```

Description / Usage

This optional card indicates whether or not linear stability analysis should be performed, as well as what kind.

The valid options for {char_list} are:

no Do not perform any kind of linear stability analysis.

yes Perform regular linear stability analysis. If your problem was 2D, then 2D analysis is performed. If your problem was 3D, then 3D analysis is performed.

inline Same as yes, perform regular linear stability analysis.

3D Subject the 2D flow to 3D linear stability analysis by normal mode expansion for the modes specified with the *Eigen Wave Numbers* card.

file Set up the problem as in **yes** or **inline,** but output the matrices involved instead of determining stability.

3Dfile Set up the problem as in **3D,** but output the matrices involved instead of determining stability.

The default value is **no.**

Examples

Here is a sample card:

```
Linear Stability = yes
```

Technical Discussion

When linear stability analysis is performed, a steady-state solution is first acquired, and then the eigenvalue/eigenvector spectrum is computed subject to the choices made in the *Eigensolver Specifications* section. In the case of **file** or **3Dfile,** the steady-state solution is acquired and then the matrices that would have been used to compute the spectrum are exported to file and no spectrum is actually computed. Refer to the Advanced Capabilities (Gates, et. al., 2001) document for a more thorough description.

The name of the output files when **file** is specified are:

- LSA_mass_coo.out for the mass matrix, B or M,
- LSA_jac_coo.out for the jacobian matrix, J,
- LSA_vars.out for variable names associated with unknowns.

When **3Dfile** is specified, the names are:

- LSA_mass_coo-<f>.out, for the mass matrix, B or M,
- LSA_jac_coo-<f>.out, for the jacobian matrix, J,
- LSA_vars.out, for variable names associated with unknowns.

where <f> is the value of the requested normal mode (see the *Eigen Wave Numbers* card). The *Eigen Matrix Output* card must be set to **yes** in order to create and write these files.

When computing the 3D stability of a base 2D flow, other modifications need to be made (see the 3D stability of 2D flow memo).

See the Advanced Capabilities document (Gates, et. al., 2001), or it's replacement (Labreche, et. al., 2002).

References

SAND2000-2465: Advanced Capabilities in Goma 3.0 - Augmenting Conditions, Automatic Continuation, and Linear Stability Analysis, I. D. Gates, D. A. Labreche and M. M. Hopkins (January 2001)

SAND2002-xxxx: Advanced Capabilities in Goma 4.0 - Augmenting Conditions, Automatic Continuation, and Linear Stability Analysis, Labreche, D. A., Wilkes, E. D., Hopkins, M. M. and Sun, A. C., (in preparation)

Filter Concentration

```
Filter Concentration = <integer> <float1> <float2>
```

Description / Usage

This optional card allows the user to enforce strict bounds on the concentration of a specific species. The input parameters are defined as:

<integer> i, this integer indicates which species (i ≥ 0) receives this special restriction.

<float1> min, a real number indicating the minimum concentration.

<float2> max, a real number indicating the maximum concentration.

There are no default values; concentrations take on whatever values are naturally dictated by the Newton iterations.

Examples

The following is a sample card:

```
Filter Concentration = 0 0.0 1.0
```

Technical Discussion

Although a correct solution should not have concentrations less than 0 or greater than 1.0, such values may arise in the solution vector due to various sources. Intermediate solutions during the Newton iteration may cause non-physical values to arise. Numerical error due to inexact linear solves, rounding, etc., may cause the values to be inexact. This card allows the user to force the concentration of species **i** to be corrected to fall within a strict concentration range **[min,max]** after the Newton iterations have terminated.

Disable Viscosity Sensitivities

```
Disable Viscosity Sensitivities = {yes | no}
```

Description / Usage

This optional card permits the analyst to omit the sensitivities of a shear-thinning viscosity model with respect to shear rate from the Jacobian.Valid options for this card are

yes Omit the sensitivities of a shear-thinning viscosity model with respect to shear rate from the Jacobian

no Form the complete Jacobian.

Currently, this card will have an effect only when using the following viscosity models: **POWER_LAW, CARREAU, BINGHAM** (see the *Liquid Constitutive Equation* card).

The default value is **no**.

Examples

The following is a sample card:

```
Disable Viscosity Sensitivities = yes
```

Technical Discussion

It has been observed that when these terms are included for very highly shear-thinning models the result can be non-convergence. In such situations, disabling these terms can often result in a convergent answer but at a convergence rate far less than the usual quadratic.

1.4.10 Eigensolver Specifications

The ability to solve for the stability of a base flow is a very powerful tool. Often, the important characteristics of a flow can be summarized in the answer to the question "is the flow stable?". Although the following cards are in active use at the time of this writing, sweeping changes are coming to the eigensolver sections of *Goma*. In particular, the old code (called "eggroll") is being replaced with newer methods (in the ARPACK library), as well as being coupled to the continuation and tracking algorithms (in the LOCA library).

Input specifications for this section of input records is discussed in a separate, comprehensive manual (Gates, et. al., 2000); an update to this manual will be completed during the summer of 2006 (Labreche, et. al., 2006). Either of these manuals contains a thorough discussion of how to successfully compute the stability and interesting modes of an underlying base flow.

1.4.11 Boundary Condition Specifications

The broad range of mechanics capabilities that has been built into *Goma* necessitates an equally broad range of boundary conditions (BCs) to provide all boundary condition information that the differential equations specified in the *Problem Description* section will require for a well-posed system. The BCs for *Goma* have been categorized according to the differential equation set to which they apply. First are listed those boundary conditions which can be applied to any equation followed by BCs for mesh, real solid, fluid momentum, energy, mass, continuity, porous, stress, gradient, shear rate, fill and potential equations. Each boundary condition (BC) card follows a general syntax as follows:

```
BC = <bc_name> <bc_type> <bc_id> {integer_list}/{float_list}
```

The <bc_name> identifies the desired control of the physics/mechanics at the boundary as identified by the <bc_type> and its associated <bc_id>. The <bc_type> is either nodeset, NS (NODEBC or POINBC in EXODUS II) or sideset, SS (ELEMBC in EXODUS II) depending on the <bc_name> and can be located in the problem domain by means of its flag or <bc_id> number (set in EXODUS II). The {integer_list} and/or {float_list} specify parameters of the boundary condition. Within each equation category are Dirichlet nodeset boundary conditions (i.e. T, U, V, W, DX, DY, DZ, Y, S11, S12, S13, S22, S23, S33, G11, G12, G13, G21, G22, G23, G31, G32, G33) that can be handled (i.e., processed) in two ways in *Goma*. The first way is application of the BC as a "hard-set" on the primitive variable, and the second as a residual equation; differences in these methods are discussed below. The cards belonging to this category have the following general syntax:

```
BC = <bc_name> <bc_type> <bc_id> <float1> <float2>
```

where <float2> flags whether a hard-set or residual equation is to be used.

Prior to introducing individual boundary conditions and their parameters, some general comments regarding the first category of BCs, boundary condition types and the resolution of boundary condition conflicts will be made.

Any Equation Boundary Conditions - There are several boundary condition types that are not necessarily best binned with a specific equation type. The FIX, GD_* and TABLE boundary condition types are general and can be applied to any equation type. A general description of these types (called Category 1) is given below.

Boundary Condition Types - Beyond the generalized boundary conditions types and the Dirichlet types, *Goma* has strong-collocated, weak form, and several others that are intrinsic to the Galerkin finite element method; these are applied in a variety of ways. Because of this, boundary conditions at a given node might interact in ways that produce unexpected results. For this reason, it is important to understand the differing methods of application that occur in *Goma* and how each affects the other. In addition, by cleverly mixing boundary conditions, the analyst is often able to achieve a desired result, but only if the nature of each boundary condition is understood. Toward this end, the user will find a *special label* assigned to each boundary condition, which, with the ensuing explanation below, will provide each user with an understanding of how that BC is applied within *Goma*.

On each boundary condition card, the boundary condition type appears in the **Description/Usage** section. These are the following boundary condition types that will be found here:

- **DIRICHLET (DC)**
- **STRONGLY INTEGRATED (SIC)**
- **STRONGLY INTEGRATED EDGE (SIC_EDGE)**
- **COLLOCATED (PCC)**
- **COLLOCATED EDGE (PCC_EDGE)**
- **WEAKLY INTEGRATED (WIC)**

The following sections discuss the method of application of each boundary condition type along with the implications of using each.

DIRICHLET (DC):

In the hierarchy of boundary conditions, Dirichlet conditions are at the top. Nothing trumps a Dirichlet conditions. A Dirichlet condition is applied by discarding all mechanics information related to a particular field variable that has been accumulated at a given node and replacing it with a direct assignment of the nodal unknown of that field with a fixed *a priori* value. Algorithmically, applying a Dirichlet condition on a degree of freedom at a node involves zeroing the entire equation row, inserting a unity value on the diagonal element of the Jacobian matrix, inserting a zero value at the appropriate place in the residual vector, and inserting the known boundary condition value at the appropriate place in the solution vector. This is referred to in many places as the "*hard set*" method. An alternate formulation imposes the boundary condition by replacing the mechanics equation at a node with the simple *residual equation*, EQUATION , where φ and $\varphi 0$ are the nodal unknown field and its assigned value, respectively.The sensitivities of this residual equation are entered into the Jacobian appropriately and solution takes place normally.

Dirichlet conditions are strictly node-based. Neighbor nodes and shared elements have no influence on them. For this reason, all Dirichlet conditions are applied to nodesets. Furthermore, Dirichlet conditions are assigned the highest precedence in terms of boundary conditions. If a Dirichlet condition appears at a node, it will be applied. Any other boundary condition that could be applied will be discarded (at that node).

Dirichlet conditions are limited, however in that they can only affect the nodal value of a degree of freedom. Derived quantities cannot be set with a Dirichlet condition. You will never see a Dirichlet condition being applied to a heat flux for example.

STRONGLY INTEGRATED (SIC):

The next class of boundary condition is referred to within *Goma* as the strongly integrated boundary conditions. These boundary conditions replace the mechanics equation at the i^{th} node with a surface integral of some derived quantity. The general form of these conditions is:

$$\int_S \phi_i \, g(\mathbf{x}) \, \mathrm{d}S = 0$$

where in this case $g(\mathbf{x})$ is not a residual equation but some derived quantity. Unlike strong constraints, this term is not multiplied by a penalizing factor before it is added to the accumulated mechanics equation at node *i*. Consequently, it represents boundary contributions to the mechanics at that node. Note also that since these conditions only make additions to the boundary mechanics, if a strongly enforced condition (SIC or PCC) is also present at the node, the weakly integrated constraint will be clobbered along with the rest of the mechanics. As an example, a CAPILLARY boundary condition that is applied to the same sideset as a VELO_NORMAL condition will have no effect in the final answer.

Weakly integrated boundary conditions are also very much a consequence of the "natural" boundary conditions that emerge from the finite element formulation. As anyone familiar with the finite element method knows, these are the ghostly boundary terms that enforce zero boundary fluxes or forces as a convenient default. The weakly integrated boundary condition step into the space afforded by the natural boundary conditions and allow the user to specify values for these boundary fluxes or forces as functions of conditions on those boundaries.

In addition, to the various classes of boundary conditions detailed above, there are special cases that arise when applying boundary conditions to the "vector" degrees of freedom. Currently, the only "vector" degrees of freedom are the mesh displacement and fluid velocity unknowns. When a boundary condition is applied to these degrees of freedom, it may be ROTATED, VECTOR or SCALAR. These labels appear in the boundary condition documentation along with the class of the condition.

ROTATED:

When a boundary condition is designated as "ROTATED," the vector components of the appropriate equations for the surface nodes are projected into a new coordinate system that is locally based on the surface normal vector and tangent vectors. It is the presence of the "ROTATED" boundary condition that prompts this process. Usually, only one of these rotated components is then affected by the boundary condition constraint and in this sense ROTATED conditions are SCALAR conditions (see below). Also generally speaking, ROTATED boundary conditions are strongly enforced as described above.

VECTOR:

When a boundary condition is designated as a "VECTOR" condition, the implication is that a vector quantity will be added to the vector components of the original mechanics equations. "VECTOR" boundary conditions are generally always applied weakly.

SCALAR:

When a boundary condition is designated a "SCALAR" condition, only a single mechanics equation is going to be influenced by the boundary condition. In the case of the vector degrees of freedom, only a single component would be affected by the boundary condition. Boundary conditions that apply to degrees of freedom that are naturally scalars, for instance temperature and species, are by default SCALAR conditions.

An example of these special labels for the *VELO_NORMAL_EDGE* condition (found on the line with the **Description/Usage** section header) is **PCC-EDGE/ROTATED MOMENTUM** indicating a rotated collocated edge condition applied to the fluid momentum equation. Given this labeling convention, boundary conditions which are not specified to be rotated or vector conditions can be presumed to be unrotated scalar conditions. Boundary conditions that may be applied to any equation are labeled "varied."

The user will not find "periodic boundary conditions" discussed in this manual. Those interested in such conditions should consult the Advanced Capabilities Manual (SAND2006-7304).

Resolving Conflicts between Boundary Conditions - In *Goma*, the bulk equations and boundary conditions are evaluated on an element-by-element basis. After the residual and Jacobian entries for the bulk equations have been calculated, the boundary conditions are used to modify or replace the bulk entries where necessary. Often the selection of boundary conditions from the input deck may cause two boundary conditions to be applied to the same equation (equation associated with a nodal point); this is especially true at junction points. Frequently the multiple boundary conditions perform the same function (i.e. duplicates) but in some important instances they are different (i.e. conflicts). In *Goma*, a decision making process was developed for determining which boundary conditions have priority. The flow chart for this decision-making is shown in Figure 4. While this process resolves boundary-condition conflicts, it **does not** eliminate the possibility of setting boundary conditions that are incompatible and lead to errors in solving the problem. However,

this method should clarify how BC's are chosen from the input deck and should enable the user to determine why a given combination of boundary conditions does not work.

The flow chart in Figure 3 shows the procedure for resolving what boundary conditions get applied to a given equation at a given node. The starting point assumes that a list of all the potential boundary conditions for the equation are known. Boundary conditions in *Goma* fall into several classes: *Dirichlet, Pointwise Collocation, Strong Integrated, Weak Integrated and Special conditions*, in order of priority. For boundary conditions applied to vector equations (mesh or momentum), a boundary condition can cause the bulk equations to be rotated prior to applying the boundary condition; in conflicts between boundary conditions, conditions which do not rotate the bulk equations (*unrotated conditions*) have priority over conditions which rotate the bulk equations (*rotated conditions*). In certain cases (e.g. two PLANE conditions which intersect at a point), conflicting boundary conditions can be checked to determine if they are duplicates, in which case only the first of the duplicates in the input deck is applied. Most boundary conditions are designed to apply by themselves, but a special class of boundary conditions, the generalized dirichlet (GD) conditions, are designed so that multiple GD conditions can apply along the same boundary and to the same equation.

While running, *Goma* prints the results of conflict resolution for every node at which it found at least two boundary conditions being applied to the same equation. The results indicate the node number, equation type, boundary conditions chosen by *Goma*, and the side-set or node-set numbers to which the boundary conditions apply. Thus to determine what boundary conditions are actually used by *Goma*, carefully check the output from conflict resolution. Setting the *Debug_Flag* = 1 causes *Goma* to print out more information regarding which boundary conditions apply and which do not. Despite the complexity of the logic built into *Goma* to resolve conflicts between boundary conditions, there are several combinations of boundary conditions that do not have a clear resolution. It is up to the user to resolve the final conflicts.

And finally, the first (*Number of BC*) and last (*END OF BC*) boundary condition cards are a pair and stand alone; the remaining cards belong to the categories of conditions discussed above. The ordering of input cards within this collection of BC input records (i.e., section) is sequential and some sections of interspersed comments accompany each boundary condition category.

Number of BC

```
Number of BC = <integer>
```

Description / Usage

This required card indicates how many boundary condition (BC) cards are contained in the *Problem Description File*. The single input parameter is defined as

<integer> The number of BC cards that follow.

If <integer> is set to -1, *Goma* will automatically count the number of BC cards between the *Number of BC* card and the *End of BC* card. This latter usage is generally preferred if a large number of BCs are to be specified.

Examples

Following is a sample card, indicating that there are two BC cards that follow this card.

```
Number of BC = 2
```

Technical Discussion

If there are more BC cards listed in an input deck than specified on this card, *Goma* ignores the extras; in other words, only the first <integer> cards are read by *Goma*. If the number of BCs is fewer than the amount specified by <integer>, *Goma* will stop with an error.

Also note, that if more than one BC on the same variable is specified, only the last one is applied.

Category 1: Any Equation

This category includes a set of cards that are used to provide all boundary condition information for a generalized dirichlet (GD) boundary condition. The condition is applied as a pointwise collocation along a given node set. The general syntax for the GD_cards is as follows:

```
BC = <bc_name> <bc_type> <bc_id> <equation_name> <integer1> <variable_name> <integer2>
↪ {float_list}
```

The current allowable definitions and/or values for < bc_name>, <bc_type>, <bc_id>, <integer1>, <integer2> and {float_list} are provided in the individual cards. As a general note, <integer1> and <integer2> are the species number of the mass transport equation and concentration variable, respectively; they should be zero for other equation and variable types. Currently these conditions assume that the variable is defined at all the nodes at which the equation is defined (no subparametric mapping).

However, the values for <equation_name> and <variable_name>, which apply generally to all cards in this category (except as subsequently noted), are given here:

<equation_name> A character string indicating the equation to which this boundary condition is applied, which can be

- R_MOMENTUM1 R_MOMENTUM2 R_MOMENTUM3
- R_MESH1 R_MESH2 R_MESH3
- R_MASS R_ENERGY R_MASS_SURF
- R_PRESSURE
- R_STRESS11 R_STRESS12 R_STRESS13
- R_STRESS22 R_STRESS23 R_STRESS33
- R_GRADIENT11 R_GRADIENT12
- R_GRADIENT13 R_GRADIENT21
- R_GRADIENT22 R_GRADIENT23
- R_GRADIENT31 R_GRADIENT32
- R_GRADIENT33 R_POTENTIAL R_FILL
- R_SHEAR_RATE R_MESH_NORMAL (rotate mesh equations and apply this condition to normal component)
- R_MESH_TANG1 R_MESH_TANG2
- R_MOM_NORMAL (rotate momentum equations and apply this condition to normal component)
- R_MOM_TANG1 R_MOM_TANG2
- R_POR_LIQ_PRESS R_POR_GAS_PRESS
- R_POR_POROSITY R_POR_SATURATION
- R_POR_ENERGY R_POR_LAST

- R_POR_SINK_MASS R_VORT_DIR1
- R_VORT_DIR2 R_VORT_DIR3 R_VORT_LAMBDA
- R_CURVATURE R_LAGR_MULT1
- R_LAGR_MULT2 R_LAGR_MULT3
- R_BOND_EVOLUTION R_SURF_CHARGE
- R_EXT_VELOCITY R_EFIELD1 R_EFIELD2
- R_EFIELD3 R_ENORM R_NORMAL1
- R_NORMAL2 R_NORMAL3 R_ _CURVATURE
- R_SHELL_TENSION R_SHELL_X R_SHELL_Y
- R_SHELL_USER R_PHASE1 R_PHASE2
- R_PHASE3 R_PHASE4 R_PHASE5
- R_SHELL_ANGLE1 R_SHELL_ANGLE2
- R_SHELL_SURF_DIV_V R_SHELL_SURF_CURV
- R_N_DOT_CURL_V R_GRAD_S_V_DOT_N1
- R_GRAD_S_V_DOT_N2 R_GRAD_S_V_DOT_N3
- R_ACOUS_PREAL R_ACOUS_PIMAG
- R_SHELL_DIFF_FLUX
- R_SHELL_DIFF_CURVATURE
- R_SHELL_NORMAL1 R_SHELL_NORMAL2
- R_ACOUS_REYN_STRESS R_SHELL_BDYVELO
- R_SHELL_LUBP R_LUBP R_SHELL_FILMP
- R_SHELL_FILMH R_SHELL_PARTC
- R_SHELL_SAT_CLOSED R_SHELL_SAT_OPEN
- R_SHELL_ENERGY R_SHELL_DELTAH
- R_SHELL_LUB_CURV R_SHELL_SAT_GASN
- R_SHELL_SHEAR_TOP R_SHELL_SHEAR_BOT
- R_SHELL_CROSS_SHEAR R_MAX_STRAIN
- R_CUR_STRAIN R_LUBP_2
- R_SHELL_SAT_OPEN_2 or
- R_SHELL_LUB_CURV_2

<variable_name> A character string indicating the variable which should be fixed, which can be

- VELOCITY1 VELOCITY2 VELOCITY3
- MESH_DISPLACEMENT1
- MESH_DISPLACEMENT2
- MESH_DISPLACEMENT3 MESH_POSITION1
- MESH_POSITION2 MESH_POSITION3

- MASS_FRACTION SURFACE TEMPERATURE or
- PRESSURE (pressure will have no effect if not using Q1 or Q2 basis functions)
- POLYMER_STRESS11
- POLYMER_STRESS12 POLYMER_STRESS13
- POLYMER_STRESS22 POLYMER_STRESS23
- POLYMER_STRESS33 VOLTAGE FILL
- SHEAR_RATE VEL_NORM D_VEL1_DT
- D_VEL2_DT D_VEL3_DT D_T_DT D_C_DT
- D_X1_DT D_X2_DT D_X3_DT D_S_DT D_P_DT
- VELOCITY_GRADIENT11
- VELOCITY_GRADIENT12
- VELOCITY_GRADIENT13
- VELOCITY_GRADIENT21
- VELOCITY_GRADIENT22
- VELOCITY_GRADIENT23
- VELOCITY_GRADIENT31
- VELOCITY_GRADIENT32
- VELOCITY_GRADIENT33 POR_LIQ_PRESS
- POR_GAS_PRESS POR_POROSITY
- POR_POROSITY POR_TEMP POR_SATURATION
- POR_LAST MAX_POROUS_NUM
- POR_SINK_MASS VORT_DIR1 VORT_DIR2
- VORT_DIR3 VORT_LAMBDA CURVATURE
- LAGR_MULT1 LAGR_MULT2 LAGR_MULT3
- BOND_EVOLUTION SURF_CHARGE
- EXT_VELOCITY EFIELD1 EFIELD2 EFIELD3
- ENORM NORMAL1 NORMAL2 NORMAL3
- SHELL_CURVATURE SHELL_TENSION
- SHELL_X SHELL_Y SHELL_USER PHASE1
- PHASE2 PHASE3 PHASE4 PHASE5
- SHELL_ANGLE1 SHELL_ANGLE2
- SHELL_SURF_DIV_V SHELL_SURF_CURV
- N_DOT_CURL_V GRAD_S_V_DOT_N1
- GRAD_S_V_DOT_N2 GRAD_S_V_DOT_N3
- ACOUS_PREAL ACOUS_PIMAG
- SHELL_DIFF_FLUX SHELL_DIFF_CURVATURE

- SHELL_NORMAL1 SHELL_NORMAL2
- ACOUS_REYN_STRESS SHELL_BDYVELO
- SHELL_LUBP LUBP SHELL_FILMP
- SHELL_FILMH SHELL_PARTC
- SHELL_SAT_CLOSED SHELL_PRESS_OPEN
- SHELL_TEMPERATURE SHELL_DELTAH
- SHELL_LUB_CURV SHELL_SAT_GASN
- SHELL_SHEAR_TOP SHELL_SHEAR_BOT
- SHELL_CROSS_SHEAR MAX_STRAIN
- CUR_STRAIN LUBP_2 SHELL_PRESS_OPEN2
- SHELL_LUB_CURV_2

EXCEPTIONS to the above parameter definitions: For the *GD_TIME* card, the <variable_names> of **LINEAR, EXPONENTIAL**, or **SINUSOIDAL** are acceptable (see examples below). There are also differences in the use of the *GD_TABLE* card, which are explained in the description of that card below.

A GD boundary condition can be applied multiple times to the same side set and equation to build up a general multiparameter condition. When this is done, the function is built by expanding the equations sequentially in the order specified in the BC list.

Descriptions of the GD cards are given next. An insert entitled "Usage Notes on the GD Cards" follows the descriptions, explaining how the cards are used together in various combinations.

FIX

```
BC = FIX NS <bc_id> {char_string} <integer1>
```

Description / Usage

(DC/VARIED)

This boundary condition card is used to fix the value of a nodal variable along a node set to the value it receives from an initial guess file (viz. either from the neutral file specified by the *Initial Guess* card or an input EXODUS II file as also specified by the *read_exoII_file* option on the *Initial Guess* card). The boundary condition is applied as a Dirichlet condition (see technical discussion below).

Definitions of the input parameters are as follows:

FIX Name of the boundary condition (<bc_name>).

NS Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.

{char_string} Variable name that is to be fixed. This parameter can have the following permissible values:

- VELOCITY1
- VELOCITY2 VELOCITY3
- MESH_DISPLACEMENT1

- MESH_DISPLACEMENT2
- MESH_DISPLACEMENT3
- SOLID_DISPLACEMENT1
- SOLID_DISPLACEMENT2
- SOLID_DISPLACEMENT3 MASS_FRACTION
- TEMPERATURE PRESSURE VOLTAGE FILL
- POLYMER_STRESS11 POLYMER_STRESS12
- POLYMER_STRESS13 POLYMER_STRESS22
- POLYMER_STRESS23 POLYMER_STRESS33
- VELOCITY_GRADIENT11
- VELOCITY_GRADIENT12
- VELOCITY_GRADIENT13
- VELOCITY_GRADIENT21
- VELOCITY_GRADIENT22
- VELOCITY_GRADIENT23
- VELOCITY_GRADIENT31
- VELOCITY_GRADIENT32
- VELOCITY_GRADIENT33 POR_LIQ_PRES
- POR_GAS_PRES POR_POROSITY
- POR_POROSITY POR_TEMP POR_SATURATION
- POR_LAST MAX_POROUS_NUM
- POR_SINK_MASS VORT_DIR1 VORT_DIR2
- VORT_DIR3 VORT_LAMBDA CURVATURE
- LAGR_MULT1 LAGR_MULT2 LAGR_MULT3
- BOND_EVOLUTION SURF_CHARGE
- EXT_VELOCITY EFIELD1 EFIELD2 EFIELD3
- ENORM NORMAL1 NORMAL2 NORMAL3
- SHELL_CURVATURE SHELL_TENSION
- SHELL_X SHELL_Y SHELL_USER PHASE1
- PHASE2 PHASE3 PHASE4 PHASE5
- SHELL_ANGLE1 SHELL_ANGLE2
- SHELL_SURF_DIV_V SHELL_SURF_CURV
- N_DOT_CURL_V GRAD_S_V_DOT_N1
- GRAD_S_V_DOT_N2 GRAD_S_V_DOT_N3
- ACOUS_PREAL ACOUS_PIMAG
- SHELL_DIFF_FLUX SHELL_DIFF_CURVATURE

- SHELL_NORMAL1 SHELL_NORMAL2
- ACOUS_REYN_STRESS SHELL_BDYVELO
- SHELL_LUBP LUBP SHELL_FILMP
- SHELL_FILMH SHELL_PARTC
- SHELL_SAT_CLOSED SHELL_PRESS_OPEN
- SHELL_TEMPERATURE SHELL_DELTAH
- SHELL_LUB_CURV SHELL_SAT_GASN
- SHELL_SHEAR_TOP SHELL_SHEAR_BOT
- SHELL_CROSS_SHEAR MAX_STRAIN
- CUR_STRAIN LUBP_2 SHELL_PRESS_OPEN2
- SHELL_LUB_CURV_2

<integer1> Species number of concentration, or zero if variable is not concentration.

Examples

The following is an example of using this card to set the mesh displacement components in a 2-D problem:

```
BC =    FIX    NS    4    MESH_DISPLACEMENT1    0
```

```
BC =    FIX    NS    4    MESH_DISPLACEMENT2    0
```

In this example, several continuation steps were taken to deform part of an elastic block of material. The displacements on boundary node set 4 were then held constant while moving another boundary (because the current displacements were not known, FIX was a convenient tool).

Technical Discussion

This boundary condition capability is indispensable for moving-mesh problems when the dependent variable is the mesh displacement from a stress free state. If one were to try to use the DX/DY/DZ type Dirichlet condition to suddenly freeze a mesh along a node set after a parameter continuation or transient problem restart, then they would be faced with figuring out the displacement of each node and defining individual node sets for each node for boundary condition application. This capability is also beneficial when using previous simulation results to generate boundary conditions for more complex analysis. We have on occasion used this boundary condition for most of the variable types shown.

GD_CONST

```
BC = GD_CONST SS <bc_id> <equation_name> <integer1> <variable_name> <integer2> <float>
```

Description / Usage

(PCC/VARIED)

This boundary condition of type *Category 1* (see discussion) is used to impose a constant value for any nodal variable, using the residual function form

$$x - C_1 = 0$$

C_1 being the constant value (<float>) and x being the <variable_name>. This boundary condition card can be used in combination with any of the other *GD_** conditions as a building block to construct more complicated conditions. Please see the examples on all of these cards for details and instructive uses. Definitions of the input parameters are as follows:

GD_CONST Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied (see the list of permissible values in the discussion above for *Category 1*).

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<variable_name> A character string indicating the variable that should be fixed (see the list of permissible values in the discussion above for *Category 1*).

<integer2> Species number of the concentration variable.The value should be 0 unless the <variable_name> is of type MASS_FRACTION.

<float> Value of variable C_1.

Examples

Following is a sample card:

```
BC = GD_CONST SS 2 R_MESH_NORMAL 0 MASS_FRACTION 0 0.2
```

This boundary condition results in the equation $C_1 - 0.2 = 0$ being applied as a boundary condition to the mesh-motion equation and being rotated into a normal-tangential basis. C_1 is the concentration of the zeroth species. The equation is actually applied as a replacement to the normal component of the mesh motion equation and in this case would cause the mesh surface, defined by side set 2, to move as the isoconcentration surface of $C_1 = 0.2$.

Technical Discussion

Note that this collocated boundary condition may be applied as a rotated, vector or scalar condition depending on the equation to which this condition applies. The example above is a powerful demonstration of this boundary condition as a distinguishing condition. Please consult the example discussions on the other *GD_** options for more detailed examples, as this boundary condition card can be used in an additive way with other *GD_** cards.

GD_LINEAR

```
BC = GD_LINEAR SS <bc_id> <equation_name> <integer1> <variable_name> <integer2>
↪<float1> <float2>
```

Description / Usage

(PCC/VARIED)

This boundary condition of type *Category 1* (see discussion) is used to impose a linear function for any nodal variable, using the residual function form

$$C_1 + C_2 x = 0$$

where C_1 and C_2 being the constant values and x representing any variable (<variable_name>). This boundary condition card can be used in combination with any of the other *GD_** conditions as a building block to construct more complicated conditions. Moreover, the resulting boundary condition can be applied as a strong residual replacement to any differential equation type. Please see the examples on all of these cards for details and instructive uses. Definitions of the input parameters are as follows:

GD_LINEAR Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied (see the list of permissible values in the discussion above for *Category 1*).

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<variable_name> A character string indicating the variable that should be fixed (see the list of permissible values in the discussion above for *Category 1*).

<integer2> Species number of the concentration variable.The value should be 0 unless the <variable_name> is of type MASS_FRACTION.

<float1> Intercept C_1

<float2> Slope C_2

Examples

Following is a sample card:

```
BC = GD_LINEAR SS 1 R_MESH1 0 MESH_POSITION1 0 -1. 2.
```

This boundary condition results in the equation $2.0 * x - 1.0 = 0$ to be applied as a boundary condition to the x-component of the mesh motion equation. x is the xcomponent of the mesh position (N.B. not displacement, as *MESH_POSITION1* would be replaced by *MESH_DISPLACEMENT1* in the above). The equation is actually applied as a replacement to the x-component of the mesh motion equation and in this case would lead to the mesh surface, defined by side set 1, to move or position itself according to this linear relationship.

Technical Discussion

Note that this collocated boundary condition may be applied as a rotated, vector or scalar condition depending on the equation to which this condition applies. Please consult the example discussions on the other *GD_** options for more detailed examples, as this boundary condition card can be used in an additive way with those.

GD_PARAB

```
BC = GD_PARAB SS <bc_id> <equation_name> <integer1> <variable_name> <integer2>
↪<float1> <float2> <float3>
```

Description / Usage

(PCC/VARIED)

This boundary condition of type *Category 1* (see discussion) is used to impose a quadratic function for any nodal variable, using the residual function form

$$C_1 + C_2 x + C_3 x^2 = 0$$

where C_1, C_2, and C_3 are the constant values (<*float*>) and x represents any variable (<variable_name>). This boundary condition card can be used in combination with any of the other *GD_** conditions as a building block to construct more complicated conditions. Moreover, the resulting boundary condition can be applied as a strong residual replacement to any differential equation type. Please see the examples on all of these cards for details and instructive uses. Definitions of the input parameters are as follows:

GD_PARAB Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied (see the list of permissible values in the discussion above for *Category 1*).

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<variable_name> A character string indicating the variable that should be used in the function (see the list of permissible values in the discussion above for *Category 1*).

<integer2> Species number of the concentration variable.The value should be 0 unless the <variable_name> is of type MASS_FRACTION.

<float1> Intercept C_1.

<float2> Slope C_2.

<float3> Acceleration C_3.

Examples

Following is a sample card:

```
BC = GD_PARAB SS 4 R_MESH1 0 MESH_POSITION2 0 1. -2. -3.
```

```
BC = GD_LINEAR SS 4 R_MESH1 0 MESH_DISPLACEMENT1 0   0. -1.
```

This boundary condition results in the equation $-3 * y^2 - 2.0 * y + 1.0 = 0$ to be applied as a boundary condition to the x-component of the mesh motion equation. y is the ycomponent of the mesh position (N.B. not displacement, as *MESH_POSITION2* would be replaced by *MESH_DISPLACEMENT2* in the above). The equation is actually applied as a replacement to the x-component of the mesh motion equation and in this case would lead to the mesh surface, defined by side set 4, to move or position itself according to this quadratic relationship.

Technical Discussion

This condition is convenient for applying Poiseuille velocity profiles, as a circular condition on geometry, together with many other uses.

Note that this collocated boundary condition may be applied as a rotated, vector or scalar condition depending on the equation to which this condition applies. Please consult the example discussions on the other *GD_** options and the examples below for more detailed examples, as this boundary condition card can be used in an additive way with those.

References

Please consult the following reference (on Roll Coating) for examples of roll surface geometry.

GT-003.1: Roll coating templates and tutorial for GOMA and SEAMS, February 29, 2000, P. R. Schunk and Matt Stay

---

GD_POLYN

```
BC = GD_POLYN SS <bc_id> <equation_name> <integer1> <variable_name> <integer2> <float_
↪list>
```

Description / Usage

(PCC/VARIED)

This boundary condition of type *Category 1* (see discussion) is used to impose a polynomial function for any nodal variable, using the residual function form of a 6^{th} order polynomial dependence on a variable

$$C_1 + C_2 x + C_3 x^2 + C_4 x^3 + C_5 x^4 + C_6 x^5 + C_7 x^6 = 0$$

There are three required and four optional parameters in the <float_list>; definitions of the input parameters are as follows:

GD_POLYN Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied (see the list of permissible values in the discussion above for *Category 1*).

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<variable_name> A character string indicating the variable that should be fixed (see the list of permissible values in the discussion above for *Category 1*).

<integer2> Species number of the concentration variable. The value should be 0 unless the <variable_name> is of type MASS_FRACTION.

<float1> Intercept C_1.

<float2> Slope C_2.

<float3> Acceleration C_3.

<float4> Coefficient for 3rd-order term C_4.

<float5> Coefficient for 4th-order term C_5.

<float6> Coefficient for 5th-order term C_6.

<float7> Coefficient for 6th-order term C_7.

Examples

Following is a set of sample cards:

```
BC = GD_POLYN SS 2 R_ENERGY 0 MESH_POSITION1 0 {c1} {c2} {c3} {c4} {c5} {c6} {c7}
```

```
BC = GD_LINEAR SS 2 R_ENERGY 0 TEMPERATURE 0   0. -1.
```

This boundary condition results in the equation

$$C_1 + C_2 x + C_3 x^2 + C_4 x^3 + C_5 x^4 + C_6 x^5 + C_7 x^6 = T$$

to be applied as a boundary condition on the energy equation, i.e., made a boundary condition on temperature with second card, which brings in a dependence on temperature. Here the coefficients are set by APREPRO, x is the x-component of the mesh position (N.B. not displacement, as *MESH_POSITION2* would be replaced by *MESH_DISPLACEMENT2* in the above).

Technical Discussion

This condition is not used as often as *GD_LINEAR* and *GD_PARAB*, and in fact supersedes those conditions. Please consult the example discussions on the other *GD_** options and the example section after *GD_TABLE* for more descriptive examples.

GD_TIME

```
BC = BC = GD_TIME SS <bc_id> <equation_name> <integer1> <time_func_name> <integer2>
↪<float1> <float2> [float3]
```

Description / Usage

(PCC/VARIED)

This boundary condition card is actually a multiplicative building block that can be used to impose a multiplicative time modulation of a specified functional form on any set of *GD_** conditions. *NOTE: unlike the other GD_* cards which are additive, this card is multiplicative*. This condition must be placed after any single or set of *GD_** cards for which the user wishes to modulate (viz. *GD_LINEAR, GD_PARAB*, etc.). The card can be used as many times as needed to construct the desired function. The examples below will clarify its use. Definitions of the input parameters are as follows:

GD_TIME Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied (see the list of permissible values in the discussion above for *Category 1*).

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<time_func_name> Keyword to identify the functional form of the time modulation. Permissible values for this parameter are LINEAR, EXPONENTIAL, and SINUSOIDAL.

<integer2> Set this required but unused parameter to zero.

<float1> C_0 model parameter

<float2> C_1 model parameter

[float3] Optional parameter to add a maximum time to be applied if $t > t_{max}$ then t is set to t_{max}

The functional form of each time-modulation model is as follows:

LINEAR:	$f(t) = C_0 + C_1 t$
EXPONENTIAL:	$f(t) = \exp(C_0 + C_1 t)$
SINUSOIDAL:	$f(t) = \sin(C_0 + C_1 t)$

Examples

Following is a sample card set:

```
BC = GD_LINEAR SS 1 R_MESH_NORMAL 0 MESH_DISPLACEMENT1 0 1. 0.
BC = GD_TIME SS 1 R_MESH_NORMAL 0 SINUSOIDAL 0 10. 2.
BC = GD_LINEAR SS 1 R_MESH_NORMAL 0 MESH_POSITION1 0 0. -1.
```

This set of cards leads to the application of $x = sin(10.0 + 2t)$ to the normal component of the mesh displacement at side set 1. If side set 1 were a surface of constant x (viz. normal in the x-direction) then this condition could be used to impose a piston motion to the surface. Recall that *GD_LINEAR* cards are additive with each other and *GD_TIME* is multiplicative with the previous cards. The first card is used to put a constant of 1.0 in the equation, the second card

(*GD_TIME* card) multiplies that constant with the sinusoidal time function, and the third card is used to put the linear term on mesh position. Note carefully the signs used.

Invoking with a maximum time is done using the optional parameter:

```
BC = GD_LINEAR SS 1 R_MOMENTUM1 0 VELOCITY1 0 {web_speed/time_max} 0.
BC = GD_TIME SS 1 R_MOMENTUM1 0 LINEAR 0 0 1. {time_max}
BC = GD_LINEAR SS 1 R_MOMENTUM1 0 VELOCITY1 0 0. -1.
```

Technical Discussion

This boundary condition building block is very useful for imposing time-dependent boundary conditions with some fairly standard functional forms without the inconvenience of writing a user-defined boundary condition. Boundary conditions for pulsating flow, piston motion, roll-eccentricity effects in coating, time-evolving temperature transients, etc. can all be constructed using this card. The examples at the end of this section on *GD_** options will help the user construct such functions.

GD_CIRC

```
BC = GD_CIRC SS <bc_id> <equation_name> <integer1> <variable_name> <integer2> <float1>
↪ <float2> <float3>
```

Description / Usage

(PCC/VARIED)

This boundary condition of type *Category 1* (see discussion) is used to impose a quadratic function for any nodal variable using the residual function form

$$-{C_1}^2 + C_3 \left(x - C_2\right)^2 = 0.$$

where C_1, C_2, and C_3 are the constant values (<*float*>) and x represents any variable (<variable_name>). This boundary condition card can be used in combination with any of the other *GD_** conditions as a building block to construct more complicated conditions. *GD_CIRC* happens to be a convenient building block for circles or elliptical functions (see examples below). Moreover, the resulting boundary condition can be applied as a strong residual replacement to any differential equation type. Please see the examples on all of these cards for details and instructive uses. Definitions of the input parameters are as follows: (convenient for circles):

GD_CIRC Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied. See the list of permissible values in the introduction to the *Category 1 BCs* following the *Number of BC* card.

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type R_MASS.

<variable_name> A character string indicating the variable that should be fixed. See the list of permissible values in the introduction to the *Category 1 BCs* following the *Number of BC&* card.

<integer2> Species number of the concentration variable. The value should be 0 unless the <variable_name> is of type *MASS_FRACTION*.

<float1> Radius C_1. This *should appear in only one* GD_CIRC *condition on each boundary*.

<float2> Origin C_2.

<float3> Ellipticity C_3.

Examples

Following is a sample set of cards:

```
BC = GD_CIRC SS 1 R_MESH_NORMAL 0 MESH_POSITION1 0 1. 1. 1.
BC = GD_CIRC SS 1 R_MESH_NORMAL 0 MESH_POSITION2 0 0. 1. 1.
```

This set of cards can be used to prescribe a mesh distinguishing condition for a mesh surface with a quadratic dependence on x and y, a circle center at [1., 1.], and a radius of 1.0 (note the radius only appears on one card).

GD_TABLE

```
BC = GD_TABLE SS <bc_id> <equation_name> <integer1> <variable_name> <integer2> <scale>
↪ <interpolation> [FILE = <fname>]
```

Description / Usage

(PCC/VARIED)

This card is used to specify arbitrary, univariate (one abscissa and one ordinate: x1 - x2) data for boundary conditions on two-dimensional boundaries, e.g., the inlet velocity profile of a non-Newtonian fluid in a two-dimensional channel. The *GD_TABLE* specification differs slightly from the other cards in this category: the data are scalable and the data can be read from a file. Like the other *GD_** cards, this card can be used as an additive building block for more complicated conditions. The examples below and at the end of the *GD_** section will provide more detailed guidance.

Definitions of the input parameters are described next. Differences between this card and other *GD_** cards are pointed out.

GD_TIME Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<equation_name> A character string indicating the equation to which this boundary condition is applied. See the list of permissible values in the discussion above for *Category 1*. In contrast to other *GD_** cards, this parameter also serves to identify the equation that is being supplanted.

<integer1> Species number of the mass transport equation. The value should be 0 unless the <equation_name> is of type *R_MASS*.

<variable_name> A character string indicating the variable that should be used in the function. See the list of permissible values in the discussion above for *Category 1*. For this card, in contrast to other *GD_** cards, this parameter also identifies what value is to serve as abscissa when interpolating the table.

<integer2> Species number of the concentration variable.The value should be 0 unless the <variable_name> is of type *MASS_FRACTION*.

<scale> A floating point value by which to multiply the ordinate list after interpolation. It can be used to scale the table values or change their sign, *e.g.* C_0, scale factor in $f(x_1) = C_0 * x_2$

<interpolation> Specifies the method to use in interpolating between supplied data points. Currently the only choice available is *LINEAR*, which invokes a simple linear interpolation method. Alternative methods will/can be added latter as required or requested.

The table data will be read from within the input deck itself (following the GD_TABLE BC card). The end of the table is signaled by the keywords "END TABLE." (See the second example below.) An alternative to this method is to read a file with table data.

[FILE = <fname>] The optional keyword '*FILE =*' indicates that the table data is to be read from a separate file identified by <fname>.

Note that this boundary condition card functions as every other GD condition, be it *LINEAR, QUADRATIC, POLYNOMIAL*, or in this case *TABULAR*. It is used simple as a piece of a residual on the appropriate equation. Hence, it usually requires more than one GD card to completely specify the boundary condition.

Examples

Following is a sample card set in which the table data is to be read from an external file called upstream_land.dat:

```
BC = GD_LINEAR SS 1 R_MESH_NORMAL 0 MESH_POSITION2 0 0. -1.
BC = GD_TABLE SS 1 R_MESH_NORMAL 0 MESH_POSITION1 0 1.0 LINEAR
FILE=upstream_land.dat
```

This card set first creates a linear term in *MESH_POSITION2*, which is the y-coordinate of the mesh points along side set 1. The second, *GD_TABLE* card then creates a table of y-coordinate values based on x-mesh position. This boundary condition describes a land/filet composite geometry with x-y data points.

Following is a sample card, where the table data is to be read directly from the input deck:

```
BC = GD_TABLE SS 1 R_MOMENTUM1 0 MESH_POSITION2 0 1.0 LINEAR

$ r/R          Uz
0.000000       1.666667
0.050000       1.666458
0.100000       1.665000
0.150000       1.661042
0.200000       1.653333
0.250000       1.640625
0.300000       1.621667
.
.
0.900000       0.451667
0.950000       0.237708
1.000000       0.000000
END TABLE
```

This table is used to specify the radial dependence of an axial velocity profile along the specified side set.

Technical Discussion

This capability is widely used for geometry and velocity profile boundary conditions that do not have a convenient closed form. Note that for geometry specifications you cannot specify multi-valued functions, like for a cutback angle.

References

GTM-021.0: Multiparameter continuation and linear stability analysis on highly deformable meshes in Goma, M. M. Hopkins, June 22, 2000

Usage Notes on the GD Cards

Following are several examples of uses of the Generalized Dirichlet conditions:

- For a circular boundary (with radius 1, center at (0,0), $x^2 + y^2 = 1$):

```
BC = GD_PARAB SS   1   R_MESH2 0 MESH_POSITION2     0   -1.   0.   1.
BC = GD_PARAB SS   1   R_MESH2 0 MESH_POSITION1     0   -0.   0.   1.
```

- For a planar boundary ($2x + y = 1$)

```
BC = GD_LINEAR SS   1   R_MESH1 0 MESH_POSITION1     0   -1.   2.
BC = GD_LINEAR SS   1   R_MESH1 0 MESH_POSITION2     0   0.   1.
```

- For a parabolic inflow velocity profile ($u = 1–2y–3y^2$):

```
BC = GD_LINEAR SS   4   R_MOMENTUM1 0 VELOCITY1     0   0.   -1.
BC = GD_PARAB SS   4   R_MOMENTUM1 0 MESH_POSITION2     0   1.   -2.   -3.
```

- For a distinguishing condition where the mesh is an iso-concentration surface ($C = 0.2$ with mesh equations rotated):

```
BC = GD_CONST SS   2   R_MESH_NORMAL 0 MASS_FRACTION     0   0.  2
```

- For a temperature boundary condition with APREPRO constants C_i of the form

$$T = C_1 + C_2x + C_3x^2 + C_4x^3 + C_5x^4 + C_6x^5 + C_7x^6$$

```
BC = GD_LINEAR SS   2   R_ENERGY 0 TEMPERATURE     0   -1
BC = GD_POLYN SS   2   R_ENERGY 0 MESH_POSITION1     0   {c1 c2 c3 c4 c5 c6 c7}
```

Note, in the first three examples, two cards are combined to create a single boundary condition that is a function of two variables. Thus, with a little creativity, the Generalized Dirichlet conditions can replace many of the other boundary condition types.

To help generalize the Dirichlet conditions even more, GD_TIME can be used to modulate any combination of spatial GD conditions (the CONST, LINEAR, PARAB, POLYN, CIRC and TABLE options above) which appears prior to the set. Some examples here are warranted:

- For a parabolic inflow velocity profile which is ramped from zero to a linearly growing multiplier times ($u = 1–2y–3y^2$):

```
BC = GD_PARAB SS   4   R_MOMENTUM1 0 MESH_POSITION2     0   1.   -2.   -3.
BC = GD_TIME SS   4   R_MOMENTUM1   0   LINEAR     0   0.   1.
BC = GD_LINEAR SS   4   R_MOMENTUM1   0   VELOCITY1      0   0.   -1.
```

(This set of 3 conditions actually applies $f(x,y,z,t,u) = 1t\left(1–2y–3y^2\right) –u = 0$ in place of the x-momentum equation.)

- For a sinusoidally time-varying roller surface with equation $(x–x_0)^2 + (y–y_0)^2 = {R_0}^2$ with a frequency of 2. and a phase lag of 10:

```
BC = GD_PARAB SS   1   R_MESH_NORMAL   0   MESH_POSITION2 0 {x0*x0 + y0*y0 - R0*R0} {-
↪2.*y0} 1
BC = GD_PARAB SS   1   R_MESH_NORMAL   0   MESH_POSITION1     0 {0.}   {-2.*x0} 1
BC = GD_TIME SS   1   R_MESH_NORMAL   0   SINUSOIDAL     0   10.   2.
```

This set of cards applies $f(x,y,z,t)(x–x_0)^2 + (y–y_0)^2 - \sin(2t+10){R_0}^2 = 0$ to the normal component of the mesh equations along side set 1.

- For a sinusoidally varying gap on a slot coater, the substrate has been made to oscillate according to $f(x,y,t) = y–3\sin(t/4+5) = 0$:

```
BC = GD_LINEAR SS   9   R_MESH2   0   MESH_POSITION1     0   -3.  0   0.
BC = GD_TIME SS   9   R_MESH2   0   SINUSOIDAL     0   5.   0.25
BC = GD_LINEAR SS   9   R_MESH2   0   MESH_POSITION2     0   0.   1.0
```

- Setting the u-velocity on an inlet boundary for a power law fluid:

```
BC = GD_LINEAR SS   1   R_MOMENTUM1   0   VELOCITY1     0   0.   -1.
BC = GD_TABLE SS   1   R_MOMENTUM1   0   MESH_POSITION2 0 1.0 LINEAR

$ r/R         Uz
0.000000      1.666667
0.050000      1.666458
0.100000      1.665000
0.150000      1.661042
0.200000      1.653333
0.250000      1.640625
0.300000      1.621667
.             .
.             .
.             .
0.900000      0.451667
0.950000      0.237708
1.000000      0.000000
END TABLE
```

- Setting the inlet concentration profile for species 0 from data in y0.table

```
BC = GD_LINEAR SS   1   R_MASS   0   MASS_FRACTION     0   0.0   -1.0
BC = GD_TABLE SS   1   R_MASS   0   MESH_POSITION2 0 1.0 LINEAR FILE = y0.table
```

- Setting the inlet concentration profile for species 0 from an implicit relation.

Occasionally, we have analytic representations that are in the wrong form. For example, in particulate suspension modelling, a relation exists that gives the radial coordinate as a function of the concentration, i.e. $r = F(C)$, where F is a non-linear relation. We would prefer it the other way around. We can use GD_TABLE to solve this dilemma. First, a file is prepared with the two columns, eqn.table for example:

C_0	F(C_0)
C_1	F(C_1)
.	.
.	.
C_N	F(C_N)

This just requires function evaluation. In the input deck, we then use the following cards

```
BC = GD_LINEAR SS   1   R_MASS   0   MESH_POSITION2     0   0.0   -1.0
BC = GD_TABLE SS   1   R_MASS   0   MASS_FRACTION 0 1.0 LINEAR FILE = eqn.table
```

and the right inlet concentration profile results.

TABLE_WICV

```
BC = TABLE_WICV SS <bc_id> {abscissa} {ordinate} {scale} {interpolation} [FILE =
↪<fname>]
```

Description / Usage

(WIC/VECTOR VARIED)

This boundary allows the user to supply boundary data for vector weak integrated boundary conditions. See the *TABLE_WICS* card for scalar weak integrated boundary conditions. A prime example of the use of the *TABLE_WICV* card is application of a force for a solid deformation problem.

Definitions of the input parameters are as follows:

TABLE_WICV Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with *TABLE_WICV* that identifies the boundary location (side set in EXODUS II) in the problem domain.

{abscissa} For one-dimensional tables (i.e. for use in 2D problems), the choices are restricted to one of the three coordinate directions. Use the strings X, Y or Z to identify the direction of choice. For two-dimensional tables (i.e. for use in 3D problems) use XY, XZ, YX, YZ, ZX, or ZY to denote the coordinate of the first and second columns in the table.

{ordinate} This string identifies the equation of the weak integrated boundary term that the boundary data is added to. For example, use of the VELOCITY1 string will cause the table data to be used for all components of the liquid traction in the boundary integral for the liquid momentum equations. See the following table.

String	replaces	Equation
VELOCITY1 or U	liquid tractions	R_MOMENTUM[1-3]
MESH_DISPLACEMENT1 or DX or MESH_POSITION1	mesh tractions	R_MESH[1-3]
SOLID_DISPLACEMENT1	solid tractions	R_SOLID[1-3]

{scale} A floating point scale multiplier which can be used to scale the tabular data. The boundary data used will be the product of {scale} and the tabular data.

{interpolation} This is the method chosen to interpolate between supplied data points.

For one-dimensional tables, the choices are **LINEAR**, which denotes linear interpolation, **QUADRATIC**, which denotes quadratic Lagrangian interpolation and requires an odd number of data points, and **QUAD_GP**, which denotes quadratic interpolation where the data points represent Gauss point values. 3N data points (see Technical Discussion) are required for **QUAD_GP** interpolation.

For two-dimensional tables, **BIQUADRATIC** is currently the only choice. The first two columns of the table should define a rectangular, mapped grid where the second coordinate changes more quickly than the first. More complicated methods could be added latter.

[FILE = <fname>] The keyword "**FILE** =" indicates that the table data be read from a separate file identified by <fname>. This parameter is optional and if it is left out the table data will be read from the input deck itself following the *TABLE_WICV* card. In this latter case, the end of the table is signaled by the keywords "END TABLE". Note that the file specified by FILE = is fully *apreproable*, i.e., it will be preprocessed by APREPRO before reading if APREPRO is enabled.

Examples

Following is a sample card:

```
BC = TABLE_WICV SS 12 ZX MESH_DISPLACEMENT1 BIQUADRATIC FILE = load.table
```

```
load.table:

0   0   0   6   0
0   1   0   4   0
1   0   0   3   0
1   1   0   1   0
```

Technical Discussion

The table data itself appears as columns of numbers. One-dimensional *TABLE_WICV* tables have three columns (column1=abscissa, column2=ordinate component1, column3=ordinate component2), whereas two-dimensional *TABLE_WICV* tables have five columns (column1=abscissa1, column2=abscissa2, column3=ordinate component1, column4=ordinate component2, column5=ordinate component3). *Goma* will try to read float values from any line whose first parameter can be converted to a float.

The QUAD_GP interpolation option is meant for the case when the table data comes from another finite element model or another *Goma* run and the data is most readily available at the integration points of the finite element mesh. Hence, with quadratic Gaussian quadrature, there are three data points per element. N is the number of elements from the model that the data is coming from and therefore 3N data points are the total expected.

The user is also referred to the section on **Boundary Condition Types** at the beginning of the *Boundary Condition Specifications*. In particular, look at the discussion of Weakly Integrated Conditions (WIC).

TABLE_WICS

```
BC = TABLE_WICS SS <bc_id> {abscissa} {ordinate} {scale} {interpolation} [FILE =
↪<fname>]
```

Description / Usage

(WIC/VARIED)

This boundary allows the user to supply boundary data for scalar weak integrated boundary conditions. See the *TABLE_WICV* card for vector weak integrated boundary conditions. A prime example of the use of the *TABLE_WICS* card is application of heat flux for a thermal problem.

Definitions of the input parameters are as follows:

TABLE_WICS Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with *TABLE_WICS* that identifies the boundary location (side set in EXODUS II) in the problem domain.

{abscissa} For one-dimensional tables (i.e. for use in 2D problems), the choices are restricted to one of the three coordinate directions. Use the strings X, Y or Z to identify the direction of choice. For two-dimensional tables (i.e. for use in 3D problems) use XY, XZ, YX, YZ, ZX, or ZY to denote the coordinate of the first and second columns in the table.

{ordinate} This string identifies the equation of the weak integrated boundary term that the boundary data is added to. For example, use of the VELOCITY1 string will cause the table data to be used for the x-component of the liquid traction in the boundary integral for the x-momentum equation. See the following table.

String	replaces	Equation
VELOCITY1 or U	liquid x-traction	R_MOMENTUM1
VELOCITY2 or V	liquid y-traction	R_MOMENTUM2
VELOCITY3 or W	liquid z-traction	R_MOMENTUM3
TEMPERATURE	diffusive energy flux	R_ENERGY
MESH_DISPLACEMENT1 or DX or MESH_POSITION1	mesh x-traction	R_MESH1
MESH_DISPLACEMENT2 or DY or MESH_POSITION2	mesh y-traction	R_MESH2
MESH_DISPLACEMENT3 or DZ or MESH_POSITION3	mesh z-traction	R_MESH3
SOLID_DISPLACEMENT1	solid x-traction	R_SOLID1
SOLID_DISPLACEMENT2	solid y-traction	R_SOLID2
SOLID_DISPLACEMENT3	solid z-traction	R_SOLID3
S[1-3][1-3]_[1-7]	polymer mode traction	R_STRESS[1-3][1-3]_[1-7]

{scale} A floating point scale multiplier which can be used to scale the tabular data. The boundary data used will be the product of {scale} and the tabular data.

{interpolation} This is the method chosen to interpolate between supplied data points.

For one-dimensional tables, the choices are **LINEAR**, which denotes linear interpolation, **QUADRATIC**, which denotes quadratic Lagrangian interpolation and requires an odd number of data points, and **QUAD_GP**, which

denotes quadratic interpolation where the data points represent Gauss point values. 3N data points (see Technical Discussion) are required for **QUAD_GP** interpolation.

For two-dimensional tables, **BIQUADRATIC** is currently the only choice. The first two columns of the table should define a rectangular, mapped grid where the second coordinate changes more quickly than the first. More complicated methods could be added latter.

[FILE = <fname>] The keyword "**FILE =**" indicates that the table data be read from a separate file identified by <fname>. This parameter is optional and if it is left out the table data will be read from the input deck itself following the *TABLE_WICS* card. In this latter case, the end of the table is signaled by the keywords "END TABLE". Note that the file specified by FILE = is fully *apreproable*, i.e., it will be preprocessed by APREPRO before reading if APREPRO is enabled.

Examples

Following is a sample card:

```
BC = TABLE_WICS SS 12 X TEMPERATURE QUADRATIC FILE =heatflux.table
```

```
heatflux.table:

0.0    1.0
0.5    1.5
1.0    1.75
1.5    2.0
2.0    2.0
```

Technical Discussion

The table data itself appears as columns of numbers. One-dimensional *TABLE_WICS* tables have two columns (column1=abscissa, column2=ordinate), whereas twodimensional *TABLE_WICS* tables have three columns (column1=abscissa1, column2=abscissa2, column3=ordinate). *Goma* will try to read float values from any line whose first parameter can be converted to a float.

The QUAD_GP interpolation option is meant for the case when the table data comes from another finite element model or another *Goma* run and the data is most readily available at the integration points of the finite element mesh. Hence, with quadratic Gaussian quadrature, there are three data points per element. N is the number of elements from the model that the data is coming from and therefore 3N data points are the total expected.

The user is also referred to the section on **Boundary Condition Types** at the beginning of the *Boundary Condition Specifications*. In particular, look at the discussion of Weakly Integrated Conditions (WIC).

TABLE

```
BC = TABLE SS <bc_id> {X|Y|Z|TIME} {ordinate} [species] {interpolation} [FILE =
↪<fname>] [NAME = <identifier>]
```

Description / Usage

(PCC/VARIED))

This boundary condition is a stand-alone version of the more complicated *GD_TABLE* card. It allows the user to supply arbitrary univariate (one abscissa and one ordinate) data about the spatial variation of unknowns fields on a boundary. The abscissa will be one of the three spatial coordinates or time and the ordinate is one of a choice of unknown field variables. All *TABLE_BC* conditions must have attached tabular data as a list of paired float values either directly following the card or in a separate file (identified on the card). The list of data pairs is terminated by the string "END TABLE" on its own line.

Definitions of the input parameters are as follows:

TABLE Name of the boundary condition.

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

{X|Y|Z|TIME} A char_string that identifies the independent table variable (abscissa). The strings X,Y, and Z refer of course to the three spatial coordinates. Depending on the choice here, the x, y, or z coordinate value at a given point, respectively, is used to obtain an interpolated ordinate value using the attached table data. If the TIME string appears here, however, the current simulation time is used to interpolate an ordinate value. This single value is applied uniformly to the sideset.

{ordinate} This string associates a variable type with the values of the ordinate in the attached table. It also identifies the equation that is supplanted by the boundary condition on the sideset. The following table lists the available string choices and the corresponding equation component clobbered by the boundary condition.

String	replaces	Equation*
VELOCITY1 or U		R_MOMENTUM1
VELOCITY2 or V		R_MOMENTUM2
VELOCITY3 or W		R_MOMENTUM3
MASS_FRACTION or Y or SPECIES		R_MASS
TEMPERATURE		R_ENERGY
MESH_DISPLACEMENT1 or DX		R_MESH1
MESH_DISPLACEMENT2 or DY		R_MESH2
MESH_DISPLACEMENT3 or DZ		R_MESH3
PRESSURE or P		R_PRESSURE
SOLID_DISPLACEMENT1 or DX_RS		R_SOLID1
SOLID_DISPLACEMENT2 or DY_RS		R_SOLID2
SOLID_DISPLACEMENT3 or DZ_RS		R_SOLID3
SHEAR_RATE or SH		R_SHEAR_RATE

String	replaces	Equation*
S11		R_STRESS11
S12		R_STRESS11
S22		R_STRESS11
S13		R_STRESS11
S23		R_STRESS11
S33		R_STRESS11

String	replaces	Equation*
S11_1		R_STRESS11_1
S12_1		R_STRESS12_1
S22_1		R_STRESS22_1
S13_1		R_STRESS13_1
S23_1		R_STRESS23_1
S33_1		R_STRESS33_1

String	replaces	Equation*
S11_2		R_STRESS11_2
S12_2		R_STRESS12_2
S22_2		R_STRESS22_2
S13_2		R_STRESS13_2
S23_2		R_STRESS23_2
S33_2		R_STRESS33_2

String	replaces	Equation*
S11_3		R_STRESS11_3
S12_3		R_STRESS12_3
S22_3		R_STRESS22_3
S13_3		R_STRESS13_3
S23_3		R_STRESS23_3
S33_3		R_STRESS33_3

String	replaces	Equation*
S11_4		R_STRESS11_4
S12_4		R_STRESS12_4
S22_4		R_STRESS22_4
S13_4		R_STRESS13_4
S23_4		R_STRESS23_4
S33_4		R_STRESS33_4

String	replaces	Equation*
S11_5		R_STRESS11_5
S12_5		R_STRESS12_5
S22_5		R_STRESS22_5
S13_5		R_STRESS13_5
S23_5		R_STRESS23_5
S33_5		R_STRESS33_5

String	replaces	Equation*
S11_6		R_STRESS11_6
S12_6		R_STRESS12_6
S22_6		R_STRESS22_6
S13_6		R_STRESS13_6
S23_6		R_STRESS23_6
S33_6		R_STRESS33_6

String	replaces	Equation*
S11_7		R_STRESS11_7
S12_7		R_STRESS12_7
S22_7		R_STRESS22_7
S13_7		R_STRESS13_7
S23_7		R_STRESS23_7
S33_7		R_STRESS33_7

[species] An optional integer parameter that identifies the index of the appropriate species. Note, it should appear only when the <ordinate> string is *MASS_FRACTION*.

{interpolation} A char_string parameter that identifies the method chosen to interpolate between the attached table data points. For one-dimensional tables, the choices are *LINEAR*, which denotes simple linear interpolation, and *QUADRATIC*, which denotes quadratic Lagrangian interpolation. Note that the latter requires an odd number of data points be supplied in the table.

[FILE = <fname>] The optional char_string keyword "**FILE** =" indicates that the table data be read from a separate file identified by <fname>. This parameter is optional and if it is left out the table data will be read from the input deck itself following the *TABLE BC* card. Note that the file specified by <fname> will be first preprocessed by APREPRO if that option was enabled on the command line. This is a useful feature that allows for a quick way to introduce analytic expressions onto boundaries.

[NAME = <identifier>] The optional char_string keyword *NAME* = allows for a set of table data to be attached to the char_string parameter <identifier>. This option can only be used if the table data is read from a separate file identified by *FILE* = <*fname*>. In this case, the file <fname> is scanned for the char_string "identifier:" (note the colon). Once found the table data is read until encountering *END TABLE*. This option permits multiple sets of data in the same file.

The second half of the *TABLE_BC* implementation is the tabular data itself. In the *TABLE* boundary condition, it consists of a set of paired float values, each pair on its own line. This data should follow directly after the *TABLE* boundary condition card if the *FILE* = option is not used. If a value for <fname> is supplied, the table data should be written in the file so indicated. Note that in most implementations of UNIX, <fname> can include a complete path specification in case the datafile is in a different directory than the run directory. In either case, input deck or separate file, the set of data table pairs should always be terminated by the string *END TABLE* to terminate reading of the data. When reading the table data, *Goma* attempts to read a float value on each line. If it is unsuccessful, e.g., a string might start the line, it will proceed to the next line. If it is successful, it will attempt to read a second float value to complete the data pair. An unsuccessful read here is an error. Once the second value is read, however, the remainder of the line is discarded and the next line is read. This procedure permits inclusion of comments within. See the next section for some examples.

Thus,

```
3. 1.e-4
1. 3. % this is a good example
$ 1. 40.0
$ I have no idea where the following data came from
    3.4   2.1
    1.e-2   6000.0
```

will result in four data points being read, whereas, both of the following

```
6.443   3.43c
5.4099   % 099.0
```

will result in an error.

Examples

The following is an example of a tabular data set that will be read correctly

```
$ This data came from M. Hobbs. God only knows where he got it.
T   k
0.5 1.e-4
1. 15.   % I'm not particularly sure about this one.
3.4   8.1
5.6   23.0
$ 1.0 40.0
```

In this case, four data pairs will be read to form the table.

Example usage of the *TABLE* card follows:

- Setting the u-velocity on an inlet boundary for a power law fluid:

```
BC = TABLE SS 1 Y U LINEAR
$ r/R Ux
0.000000 1.666667
0.050000 1.666458
0.100000 1.665000
0.150000 1.661042
0.200000 1.653333
0.250000 1.640625
0.300000 1.621667
..
..
0.900000 0.451667
0.950000 0.237708
1.000000 0.000000
END TABLE
```

- Setting the inlet concentration profiles for species 0 and species 1 from data in y.table:

```
BC = TABLE SS 1 Y SPECIES 0 QUADRATIC FILE = y.table NAME = y0
BC = TABLE SS 1 Y SPECIES 1 QUADRATIC FILE = y.table NAME = y1
```

- The file y.table contains:

```
y0:
        0.   1.0
        0.25 0.75
        0.5 0.60
        0.75 0.30
        1.0 0.20
END TABLE
y1:
```

(continues on next page)

(continued from previous page)

```
        0. 0.0
        0.25 0.2
        0.5 0.3
        0.75 0.5
        1.0 0.8
END TABLE
```

- Setting a temperature history on a sideset

```
BC = TABLE SS 1 TIME TEMPERATURE LINEAR
0.0   0.0
10.0   373.0
40.0   373.0
50.0   500.0
100.0   500.0
150    0.0
100000.0   0.0
END TABLE
```

Technical Discussion

The *TABLE* boundary condition provides similar functionality to the *GD_TABLE* boundary condition but with a simplified interface the notion behind both cards is that often information on boundaries is known only as a set of data points at specific positions on the boundary. The *TABLE* boundary condition can use that boundary information to provide interpolated values at nodal locations and then impose them as a strong point collocated condition.

Interpolation orders for this method are limited to *LINEAR* and *QUADRATIC* with the latter requiring an odd number of data points be supplied in the table.

Category 2: Mesh Equations

The boundary conditions in this section involve the mesh motion equations in LAGRANGIAN or ARBITRARY form (cf. *Mesh Motion* card). These conditions can be used to pin the mesh, specify its slope at some boundary intersection, apply a traction to a surface, etc. Several more boundary conditions that are applied to the mesh motion equations but include other problem physics are also available.

DISTNG

```
BC = DISTNG SS <bc_id> <float>
```

Description / Usage

(PCC/ROTATED MESH)

This boundary condition card is used to specify a distinguishing condition for mesh motion based on an isotherm, viz. the distinguishing condition forces the mesh boundary to which it is applied to take on a position such that the temperature is constant and at the specified value, all along the boundary. This condition causes the vector mesh motion equations (viz. *mesh1, mesh2, and mesh3* on *EQ* cards) to be rotated into normal-tangential form. In two dimensions, this condition is applied to the normal component automatically; in three dimensions it is suggested to put it on the normal component, as specified by the *ROT* conditions. Definitions of the input parameters are as follows:

DISTNG	Name of the boundary condition (<bcname>).
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	Value of temperature/isotherm. To apply a variable temperature, e.g., as a function of the concentration, it is suggested that the user-defined boundary conditions be used, like *SPLINE* or *GEOM*.

Examples

The following is a sample input card:

```
BC = DISTNG SS 123 273.0
```

This card forces the boundary defined by EXODUS II side set number 123 to conform to the isotherm temperature of 273.0.

Technical Discussion

The mathematical form of this distinguishing condition is as follows:

$$T - T_{\text{mp}} = 0$$

where T_{mp} is the specified temperature parameter. This condition has been used extensively for macroscale and microscale melting problems, whereby one needs to distinguish a molten region from a solidified or mushy region with liquidus and solidus temperatures. In three dimensions, usage needs to be completed with a companion *ROT* input card which directs the equation application of the condition.

FAQs

Continuation Strategies for Free Surface Flows In free surface problems, there exists one or more boundaries or internal surfaces whose position(s) are unknown a *priori*. As such, the geometry of the problem becomes part of the problem and must be determined together with the internal physics. Most problems of this sort cannot be solved with a trivial initial guess to the solution vector, mainly because the conditions which determine the surface position are closely coupled to the active physics in the bulk. Thus, these problems require continuation (zero or higher order) to achieve a converged solution to a desired state. The continuation strategy typically involves turning on and off the conditions which distinguish the position of the free surface(s); one such strategy is described in this FAQ.

Distinguishing conditions in *Goma* serve two purposes: (1) they can be used to locate a surface whose position depends on internal and interfacial transport phenomena, and (2) they can be used to prescribe solid boundary position or motion. The

first type of condition contains field variables needed to locate the interface or free surface position, and hence ties the mesh motion to the problem physics, i.e., mass, momentum, and energy transport phenomena. Currently, the side-set boundary conditions of type *DISTNG, KINEMATIC,* and *KIN_LEAK* fall into this class. The second type of condition requires only geometrical information from the mesh, and, although geometrically couples the mesh motion to the problem physics, it tends not to be so tightly coupled. Currently, boundary conditions *PLANE, PLANEX, PLANEY, PLANEZ, SPLINE, SPLINEX, SPLINEY,* and *SPLINEZ* fall into this class.

In two dimensions, there is no need to use *PLANEX, PLANEY, PLANEZ, SPLINEX, SPLINEY,* and *SPLINEZ*. Because the code automatically rotates the mesh residual equations and the corresponding Jacobian entries into normal-tangential form on the boundary, *SPLINE, PLANE,* and *DISTNG* are the only cards required to specify the position of the boundary. Currently, in three dimensions, the logic for the same rotation concept is not totally functional, and one must use the *PLANEX*, etc. cards to designate which component of the mesh stress residual equation receives the distinguishing conditions.

If cards *DISTNG, KINEMATIC,* and *KIN_LEAK*, i.e., distinguishing conditions of type 1, are absent in any simulation, then any initial guess for the transport field equations, i.e., energy and momentum, has a chance of converging, as long as the initial mesh displacement guess is within the radius of convergence of the mesh equations and associated boundary conditions. For example, if the side sets of the EXODUS II database mesh correspond somewhat closely to what is prescribed with *PLANE* and *SPLINE*-type conditions, then an initial guess of the NULL vector has a good chance of converging, so long as the velocities and temperatures are within "converging distance."

When conditions from the first class are present, i.e., either *DISTNG, KIN_LEAK* or *KINEMATIC*, then the following procedure should be followed:

- Set the keyword for the *Initial Guess* character_string to **zero**, **one**, or **random**.
- Obtain a solution (run *Goma*) with the initial guess for the free surfaces distinguished as *KINEMATIC* (or other) coming from the EXODUS II database, but without the *KINEMATIC* (or other) card(s). That is, "fix" those surfaces with either a *PLANE* or *SPLINE* command, or simply place no distinguishing condition on them (this works only if the grid has not been previously "stressed", i.e., all the displacements are zero). The rest of the "desired" physics should be maintained. If any surface is distinguished as *KINEMATIC*, then it is highly advantageous to place a *VELO_NORMAL* condition on that surface for startup, and set the corresponding floating point datum to zero. This effectively allows the fluid to "slip" along that boundary as if it were a shear free condition.
- Set the keyword in the *Initial Guess* character_string to **read**.
- Copy the file named in *SOLN file* into the file named in *GUESS file*.
- Release the free boundaries by taking off any current distinguishing condition cards and adding the appropriate *KINEMATIC* (or other) card. Adjust all other boundary conditions appropriately.
- Run *Goma*, using a relaxed Newton approach (factor less than unity but greater than zero - e.g., 0.1) for complex flows.

When dealing with material surface boundaries distinguished by the kinematic boundary condition, the nature of that condition requires a non-zero and substantial component of velocity tangent to the surface upon start-up. In this case, it can be advantageous to use the *VELO_TANGENT* card to set the velocity along the free surface to some appropriate value prior to releasing the free surface (in the third step above). Of course this card will be removed in subsequent steps. Also, although not necessary, a smooth, "kinkless", initial guess to the free surface shape is helpful because it reduces the amount of relaxation required on the Newton iteration.

Obtaining start-up solutions of most coating flow configurations is still an art. The best way to start up a coating flow analysis may be to acquire a "template" developed from a previous analysis of some closely related flows.

References

Allen Roach's or Randy's ESR tutorials. Perhaps these need to be put into the repository.

DXDYDZ

```
BC = {DX | DY | DZ} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/MESH)

This boundary condition format is used to set a constant X, Y, or Z displacement. Each such specification is made on a separate input card. These boundary conditions must be applied to node sets. Definitions of the input parameters are as follows:

{DX | DY | DZ}	Two-character boundary condition name (<bc_name>) that defines the displacement, where: • **DX** - X displacement • **DY** - Y displacement • **DZ** - Z displacement
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of the displacement (X, Y, or Z) defined above.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

Following is a sample card which applies an X-displacement boundary condition to the nodes in node set 100, specifically an X-Displacement of 1.0. These displacements are applied immediately to the unknowns and hence result in immediate mesh displacements from the initial state.

```
BC = DX NS 100   1.0
```

This sample card applies the same condition as above, except as a residual equation that is iterated upon with Newton's method.

```
BC = DX NS 100 1.0   1.0
```

The second float 1.0 forces this application. This approach is advisable in most situations, as the nodes are gradually moved as a part of the mesh deformation process; sudden movements, as in the first example, can lead to folds in the mesh.

Technical Discussion

Application of boundary conditions of the Dirichlet type on mesh motion requires different considerations than those on non-mesh degrees of freedom. Sudden displacements at a point, without any motion in the mesh surrounding that point, can lead to poorly shaped elements. It is advisable to apply these sorts of boundary conditions as residual equations, as discussed above. Examples of how these conditions are used to move solid structures relative to a fluid, as in a roll-coating flow, are contained in the references below.

References

GT-003.1: Roll coating templates and tutorial for GOMA and SEAMS, February 29, 2000, P. R. Schunk and M. S. Stay

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk.

DXUSER DYUSER DZUSER

```
BC = {DXUSER | DYUSER | DZUSER} SS <bc_id> <float_list>
```

Description / Usage

(PCC/MESH)

This boundary condition format is used to set a constant X, Y, or Z displacement as a function of any independent variable available in Goma. These boundary conditions require the user to edit the routines *dx_user_surf*, *dy_user_surf*, and/or *dz_user_surf* to add the desired models. These routines are located in the file *user_bc.c*. In the input deck each such specification is made on a separate input card. These boundary conditions must be applied to side sets. Definitions of the input parameters are as follows:

{DX_USER | DY_USER | DZ_USER}	Seven-character boundary condition name (<bc_name>) that defines the displacement, where: • **DX_USER** - X displacement, user-defined • **DY_USER** - Y displacement, user-defined • **DZ_USER** - Z displacement, user-defined
SS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutine so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

Following is a sample card which applies an X-displacement boundary condition to the nodes in node set 100, with a functional form set by the user and parameterized by the single floating point number . These displacements are applied immediately to the unknowns and hence result in immediate mesh displacement from the initial state.

```
BC = DX_USER SS 100 1.0
```

Please consult the user-definition subroutines for examples.

References

No References.

DXYZDISTNG

```
BC = {DXDISTNG | DYDISTNG | DZDISTNG} SS <bc_id> <float>
```

Description / Usage

(PCC/MESH)

This boundary condition card is used to specify a distinguishing condition for mesh motion based on an isotherm, viz. the distinguishing condition forces the mesh boundary to which it is applied to take on a position such that the temperature is constant and at the specified value, all along the boundary. Although of the same mathematical form as the *DISTNG* boundary condition, this condition does not force a boundary rotation of the vector mesh residuals. Instead, it is recommended that the condition be chosen such that the predominant direction of the normal vector is close to one of the three Cartesian coordinates, X, Y, or Z. For example, if the boundary in question is basically oriented so that the normal vector is mostly in the positive or negative Y-direction, then *DYDISTNG* should be chosen. Definitions of the input parameters are as follows:

{DXDISTNG | DYDISTNG | DZDISTNG}	Eight-character boundary condition name (<bc_name>) that defines the distinguishing condition, where: • **DXDISTNG** - X condition • **DYDISTNG** - Y condition • **DZDISTNG** - Z condition
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	Value of temperature isotherm. If one wanted to apply a variable temperature, e.g. as a function of the concentration, it is suggested that the user-defined boundary conditions be used.

Examples

The following is a sample input card:

```
BC = DYDISTNG SS 123 273.0
```

This card forces the boundary defined by EXODUS II side set number 123 to conform to the isotherm temperature of 273.0. Most importantly, the y-component of the mesh equation residuals is replaced by this condition.

Technical Discussion

The mathematical form of this distinguishing condition is as follows:

$$T - T_{\text{mp}} = 0$$

where T_{mp} is the specified temperature parameter. This condition has been used extensively for macroscale and microscale melting problems, whereby one needs to distinguish a molten region from a solidified or mushy region with liquidus and solidus temperatures. In three dimensions usage needs to be completed with a companion *ROT* input card which directs the equation application of the condition, even though rotations are not actually performed.

As a bit of software trivia, this is the first distinguishing condition ever written in *Goma*, and one of the first boundary conditions, period.

SPLINEXYZ/GEOMXYZ

```
BC = {bc_name} SS <bc_id> [floatlist]
```

Description / Usage

(PCC/MESH)

This card is used to specify a general surface (solid) boundary description for ALE (or in special cases LAGRANGIAN) type mesh motion (see *Mesh Motion* card). These boundary conditions are tantamount to *SPLINE* or *GEOM*, except that they do *not* invoke a mesh-equation vector residual rotation into normal-tangential form. Instead, *SPLINEX* or, equivalently, *GEOMX* invokes the geometric boundary condition on the x-component of the mesh equation residual, and so on. The card requires user-defined subroutines. Templates for these routines are currently located in the routine "user_bc.c". Both a function routine, fnc, for function evaluation and corresponding routines dfncd1, dfncd2, and dfncd3 for the derivative of the function with respect to global coordinates are required. *GEOMX* and *SPLINEX* are exactly the same condition. SPLINE* usage is being deprecated. Note that it takes an arbitrary number of floating-point parameters, depending on the user's needs.

Definitions of the input parameters are as follows:

{bc_name}	Boundary condition name that defines the general surface; the options are: • **SPLINEX/GEOMX** - X general surface • **SPLINEY/GEOMY** - Y general surface • **SPLINEZ/GEOMZ** - Z general surface
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
[floatlist]	Constants to parameterize any f(x,y,z) = 0 function input in user-defined routine fnc.

Examples

The following is a sample input card:

```
BC = GEOMZ SS 10 1.0 100. 20.0 1001.0 32.0
```

applies a user-defined distinguishing condition parameterized by the list of floating points to the boundary defined by side set 10. Most importantly, the condition replaces the Z-component of the momentum equation.

Technical Discussion

The mathematical form of this distinguishing condition is arbitrary and is specified by the user in the fnc routine in user_bc.c. Derivatives of the user-specified function must also be provided so as to maintain strong convergence in the Newton iteration process. These functions are located next to fnc and are named dfncd1, dfncd2, and dfncd3.Several examples for simple surfaces exist in the template routine. In three dimensions, usage needs to be completed with a companion *ROT* input card which directs the equation application of the condition, even though rotations are not actually performed.

SPLINE/GEOM

```
BC = {SPLINE|GEOM} SS <bc_id> [floatlist]
```

Description / Usage

(PCC/ROTATED MESH)

This card is used to specify a general surface (solid) boundary description for ALE (or in special cases LAGRANGIAN) type mesh motion (see *Mesh Motion* card). Like most other distinguishing conditions, this condition causes the mesh-motion equations, viz. *mesh1, mesh2,* and *mesh3*, to be rotated into boundary normal-tangential form. The card requires user-defined subroutines. Templates for these routines are currently located in the routine "user_bc.c". Both a function routine, fnc, for function evaluation and corresponding routines dfncd1, dfncd2, and dfncd3 for the derivative of the function with respect to global coordinates are required. The SPLINE condition is exactly the same and uses the same routine as the GEOM card option, and hence as of the time of this writing we are deprecating the use of SPLINE. Note that it takes an arbitrary number of floating-point parameters, depending on the user's needs.

Definitions of the input parameters are as follows:

SPLINE/GEOM	Name of the boundary condition <bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
[floatlist]	Constants to parameterize any f(x,y,z) = 0 function input in user-defined routine fnc.

Examples

The following sample input card:

```
BC = SPLINE SS 10 1.0 100. 20.0 1001.0 32.0
```

applies a user-defined distinguishing condition, parameterized by the list of five floating point values, to the boundary defined by side set 10.

Technical Discussion

This condition, like *DISTNG, PLANE*, and others that can be applied to geometry, is applied to the normal component of the mesh motion equations along a boundary in two dimensions; in three dimensions application needs to be further directed with the *ROT* conditions. Examples of typical distinguishing conditions can be found in user_bc.c in the fnc routine and companion derivative routines.

PLANEXYZ

```
BC = {PLANEX | PLANEY | PLANEZ} SS <bc_id> <floatlist>
```

Description / Usage

(PCC/ MESH)

This boundary condition card is used to specify a planar surface (solid) boundary description as a replacement on the X, Y, or Z-component (*PLANEX, PLANEY, PLANEZ*, respectively) of the mesh equations (see *EQ* cards *mesh1, mesh2*, or *mesh3*). The form of this equation is given by

$$f(x, y, z) = ax + by + cz + d = 0$$

This mathematical form and its usage is exactly like the BC = PLANE boundary condition card (see PLANE for description), but is applied to the mesh motion equations without rotation. Definitions of the input parameters are given below; note that <floatlist> has four parameters corresponding to the four constants in the equation:

{PLANEX | PLANEY | PLANEZ}	Boundary condition name (<bc_name>) where: • **PLANEX** - normal predominantly in X direction • **PLANEY** - normal predominantly in Y direction • **PLANEZ** - normal predominantly in Z direction
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	a in function $f(x, y, z)$
<float2>	b in function $f(x, y, z)$
<float3>	c in function $f(x, y, z)$
<float4>	d in function $f(x, y, z)$

Examples

Following is a sample input card for a predominantly X-directed surface (viz, as planar surface whose normal has a dominant component in the positive or negative X direction):

```
BC = PLANEX SS 101 1.0 1.0 -2.0   100.0
```

This boundary condition leads to the application of the equation $1.0x + 1.0y – 2.0z = –100.0$ to the *mesh1* equation on EXODUS II side set number 101.

Technical Discussion

These conditions are sometimes used instead of the more general *PLANE* boundary condition in situations where *ROTATION* (see *ROT* command section) leads to poor convergence of the matrix solvers or is not desirable for some other reason. In general, the *PLANE* condition should be used instead of these, but in special cases these can be used to force the application of the planar geometry to a specific component of the mesh stress equation residuals. Full understanding of the boundary rotation concept is necessary to understand these reasons (see *Rotation Specifications*).

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GT-007.2: Tutorial on droplet on incline problem, July 30, 1999, T. A. Baer

GT-013.2: Computations for slot coater edge section, October 10, 2002, T.A. Baer

GT-018.1: ROT card tutorial, January 22, 2001, T. A. Baer

PLANE

```
BC = PLANE SS <bc_id> <floatlist>
```

Description / Usage

(PCC/ROTATED MESH)

This card is used to specify a surface (solid) boundary position of a planar surface. It is applied as a rotated condition on the mesh equations (see *EQ* cards *mesh1, mesh2 mesh3*). The form of this equation is given by

$$f(x, y, z) = ax + by + cz + d = 0$$

Definitions of the input parameters are given below; note that <floatlist> has four parameters corresponding to the four constants in the equation:

PLANE	Name of the boundary condition name (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	a in function $f(x, y, z)$
<float2>	b in function $f(x, y, z)$
<float3>	c in function $f(x, y, z)$
<float4>	d in function $f(x, y, z)$

Examples

Following is a sample input card:

```
BC = PLANE SS 3 0.0 1.0 0.0 -0.3
```

results in setting the side set elements along the side set 3 to a plane described by the equation $f(x, y, z, t) = y–0.3 = 0$.

Technical Discussion

This, like most boundary conditions on geometry with arbitrary grid motion, is applied to the weighted residuals of the mesh equation rotated into the normal-tangential basis on the boundary. Specifically, this boundary condition displaces the normal component after rotation of the vector residual equation, leaving the tangential component to satisfy the natural mesh-stress free state. That is to say, this boundary condition allows for mesh to slide freely in the tangential direction of the plane surface.

This boundary condition can be applied regardless of the *Mesh Motion* type, and is convenient to use when one desires to move the plane with time normal to itself.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GT-013.2: Computations for slot coater edge section, October 10, 2002, T.A. Baer

MOVING_PLANE

```
BC = MOVING_PLANE <bc_id> <floatlist>
```

Description / Usage

(PCC/ROTATED MESH)

The *MOVING_PLANE* card is used to specify a surface (solid) boundary position versus time for a planar surface (cf. *PLANE* boundary condition card). It is applied as a rotated condition on the mesh equations (see *EQ* cards *mesh1, mesh2, mesh3*). The form of the equation is given by

$$f(x, y, z, t) = ax + by + cz + d + g(t) = 0$$

and the function $g(t)$ is defined as

$$g(t) = \lambda_1 t + \lambda_2 t^2 + \lambda_3 t^3$$

Definitions of the input parameters are given below; note that <floatlist> has seven parameters corresponding to the seven constants in the above equations:

MOVING_ PLANE	Name of the boundary condition name (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	a in function $f(x, y, z, t)$
<float2>	b in function $f(x, y, z, t)$
<float3>	c in function $f(x, y, z, t)$
<float4>	d in function $f(x, y, z, t)$
<float5>	λ_1 coefficient in $g(t)$
<float6>	λ_2 coefficient in $g(t)$
<float7>	λ_3 coefficient in $g(t)$

Examples

The boundary condition card

```
BC = MOVING_PLANE SS 3 0. 1. 0. -0.3 0.1 0.0 0.0
```

results in a plane originally positioned at y = 0.3 to move at a velocity of *-0.1*, viz. the position of all nodes on the plane will follow:

$$f(x, y, z, t) = y - 0.3 + 0.1t = 0$$

Technical Discussion

This, like most boundary conditions on geometry with arbitrary grid motion, is applied to the weighted residuals of the mesh equation rotated into the normal-tangential basis on the boundary. Specifically, this boundary condition displaces the normal component after rotation of the vector residual equation, leaving the tangential component to satisfy the natural mesh-stress free state. That is to say, this boundary condition allows for mesh to slide freely in the tangential direction of the plane surface.

This boundary condition can be applied regardless of the *Mesh Motion* type, and is convenient to use in place of *PLANE* when one desires to move the plane with time normal to itself.

SLOPEXYZ

```
BC = {SLOPEX | SLOPEY | SLOPEZ} SS <bc_id> <floatlist>
```

Description / Usage

(SIC/MESH)

This boundary condition card applies a slope at the boundary of a *LAGRANGIAN, TALE*, or *ARBITRARY* solid (see *Mesh Motion* card) such that the normal vector to the surface is colinear with the vector specified as input, viz $\underline{n} \cdot \underline{n}_{\text{spec}} = 0$. Here $\underline{n}_{\text{spec}}$ is the vector specified component-wise via the three <floatlist> parameters on the input card. Definitions of the input parameters are as follows:

{SLOPEX | SLOPEY | SLOPEZ}	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	X-component of the slope vector $\underline{n}_{\text{spec}}$
<float2>	Y-component of the slope vector $\underline{n}_{\text{spec}}$
<float3>	Z-component of the slope vector $\underline{n}_{\text{spec}}$

Examples

The following is a sample input card:

```
BC = SLOPEX SS 10 1.0 1.0 0.0
```

This card invokes a boundary condition on the normal component of the mesh residual momentum equations such that the outward facing surface normal vector along side set 10 is colinear with the vector [1.0, 1.0, 0.0]. This condition is applied to the x-component of the mesh residual equations.

Technical Discussion

See discussion for *BC* card *SLOPE*. The only difference in these conditions and the *SLOPE* conditions, is that the latter invokes rotation of the vector mesh residual equations on the boundary.

SLOPE

```
BC = SLOPE SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card applies a slope at the boundary of a *LAGRANGIAN, TALE*, or *ARBITRARY* solid (see *Mesh Motion* card) such that the normal vector to the surface is colinear with the vector specified as input, viz $\underline{n} \cdot \underline{n}_{\text{spec}} = 0$. Here $\underline{n}_{\text{spec}}$ the vector specified component-wise via the three <float> parameters on the input card. Definitions of the input parameters are as follows:

SLOPE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	X-component of the slope vector $\underline{n}_{\text{spec}}$
<float2>	Y-component of the slope vector $\underline{n}_{\text{spec}}$
<float3>	Z-component of the slope vector $\underline{n}_{\text{spec}}$

Examples

The following is a sample input card:

```
BC = SLOPE SS 10 1.0 1.0 0.0
```

This card invokes a boundary condition on the normal component of the mesh residual momentum equations such that the outward facing surface normal vector along side set 10 is colinear with the vector [1.0, 1.0, 0.0].

Technical Discussion

This condition, although not often used, allows for a planar boundary condition (cf. *PLANE, PLANEX*, etc.) to be specified in terms of a slope, rather than a specific equation. Clearly, at some point along the surface (most likely at the ends), the geometry has to be pinned with some other boundary condition (cf. *DX, DY, DZ*) so as to make the equation unique. This condition has the following mathematical form:

$$\underline{n} \cdot \underline{n}_{\text{spec}} = 0$$

and is applied in place of the normal component of the mesh motion equations, i.e., it is a rotated type boundary condition. If used in three dimensions, it will require a rotation description with the *ROT* cards.

KINEMATIC

```
BC = KINEMATIC SS <bc_id> <float1> [integer]
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card). It enforces the boundary of the mesh defined by the side set to conform to a transient or steady material surface, with an optional, pre-specified mass loss/gain rate. In two dimensions, this condition is automatically applied to the normal component of the vector mesh equations, which is rotated into normal-tangential form. In three dimensions, the application of this boundary condition needs to be further directed with the *ROT* cards (see *Rotation Specifications*). The application of this condition should be compared with *KINEMATIC_PETROV* and *KINEMATIC_COLLOC*.

Definitions of the input parameters are as follows:

KINEMATIC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Mass-loss (positive) or mass-gain (negative) velocity at the free boundary.
[integer]	Optional integer value indicating the element block id from which to apply the boundary condition.

Examples

The following sample card

```
BC = KINEMATIC SS 7 0.0
```

leads to the application of the kinematic boundary condition to the boundary-normal component of the mesh-stress equation on the boundary defined by side set 7.

Technical Discussion

The functional form of the kinematic boundary condition is:

$$\underline{n} \cdot (\underline{v} - \underline{v}_s) = \dot{m}$$

Here $\underline{n}$ is the unit normal vector to the free surface, $\underline{v}$ is the velocity of the fluid, $\underline{v}_s$ is the velocity of the surface (or mesh), and $\dot{m}$ is the mass loss/gain rate. In two dimensions this equation is applied to the normal component of the vector mesh position equation, and hence is considered as a distinguishing condition on the location of the mesh relative to the fluid domain.

FAQs

See the FAQ pertaining to "Continuation Strategies for Free Surface Flows" on the *DISTNG* boundary condition card.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

KINEMATIC_PETROV

```
BC = KINEMATIC_PETROV SS <bc_id> <float1> [integer]
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card). It enforces the boundary of the mesh defined by the side set to conform to a transient or steady material surface, with an optional, pre-specified mass loss/gain rate. In two dimensions, this condition is automatically applied to the normal component of the vector mesh equations, which is rotated into normal-tangential form. In three dimensions, the application of this boundary condition needs to be further directed with the *ROT* cards (see *ROTATION Specifications*). Please consult the Technical Discussion for important inofrmation.

Definitions of the input parameters are as follows:

KINEMATIC_PETROV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Mass-loss (positive) or mass-gain (negative) velocity at the free boundary.
[integer]	Optional integer value indicating the element block id from which to apply the boundary condition.

Examples

The following sample card

```
BC = KINEMATIC_PETROV SS 7 0.0
```

leads to the application of the kinematic boundary condition to the boundary-normal component of the mesh-stress equation to the boundary defined by side set 7.

Technical Discussion

Important note: This condition is actually the same as the *KINEMATIC* condition but is applied with different numerics for special cases. Specifically, rather than treated in a Galerkin fashion with a weighting function equal to the interpolation function for velocity, the residual of the equation is formed as weighted by the directional derivative of the basis functions along the free surface. Specifically,

$$\int \{\underline{n} \cdot (\underline{v} - \underline{v}_s) - \dot{m}\} \, \phi^i \, \mathrm{d}A = R^i = 0$$

where the nodal basis function ϕ^i is replaced by $\frac{\partial}{\partial s}\phi^i$ in the residual equation. Compare this to the *KINEMATIC* boundary condition description.

KINEMATIC_COLLOC

```
BC = KINEMATIC_COLLOC SS <bc_id> <float1>
```

Description / Usage

(PCC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card). It enforces the boundary of the mesh defined by the side set to conform to a transient or steady material surface, with an optional, pre-specified mass loss/gain rate. In two dimensions this condition is automatically applied to the normal component of the vector mesh equations, which is rotated into normal-tangential form. In three dimensions the application of this boundary condition needs to be further directed with the *ROT* cards (see *Rotation Specifications*). Definitions of the input parameters are as follows:

KINEMATIC_COLLOC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Mass-loss (positive) or mass-gain (negative) velocity at the free boundary.

Examples

The following sample card

```
BC = KINEMATIC_COLLOC SS 7 0.0
```

leads to the application of the kinematic boundary condition to the boundary-normal component of the mesh-stress equation to the boundary defined by side set 7.

Technical Discussion

Important note: This condition is actually the same as the *KINEMATIC* condition but is applied with different numerics for special cases. Specifically, rather than treated in a Galerkin fashion, with a weighting function equal to the interpolation function for velocity, the residual equation is formed at each node directly, in a collocated fashion, without Galerkin integration. This method is better suited for high-capillary number cases in which Galerkin's method is often not the best approach.

KINEMATIC_DISC

```
BC = KINEMATIC_DISC SS <bc_id> <float1>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card) in the special case of an interface between two fluids of different density (e.g. a gas and a liquid, both meshed up as *Goma* materials) through which a phase transition is occurring and there is a discontinuous velocity (see the mathematical form in the technical discussion below). Like the *KINEMATIC* boundary condition, it is used to distinguish a material surface between two phases exchanging mass. In two dimensions, this condition is automatically applied to the normal component of the vector mesh equations which is rotated into normal-tangential form. In three dimensions, the application of this boundary condition needs to be further directed with the *ROT* cards (see *Rotation Specifications*). The application of this condition should be compared with *KINEMATIC_PETROV* and *KINEMATIC_COLLOC*.

This condition must be applied to problem description regions using the Q1_D or Q2_D interpolation type, indicating a discontinuous variable treatment at the interface (see *EQ* card).

Definitions of the input parameters are as follows:

KINEMATIC_DISC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Set to zero for internal interfaces; otherwise used to specify the mass average velocity across the interface for external boundaries.

Examples

The following sample card

```
BC = KINEMATIC_DISC SS 10 0.0
```

is used at internal side set 10 (note, it is important that this side set include elements from both abutting materials) to enforce the overall conservation of mass exchange.

Technical Discussion

This boundary condition is typically applied to multicomponent two-phase flows that have rapid mass exchange between phases, rapid enough to induce a diffusion velocity at the interface. The best example of this is rapid evaporation of a liquid component into a gas.

This boundary condition card is used for a distinguishing condition and its functional form is:

$$\rho_1 \underline{n} \cdot (\underline{v} - \underline{v}_s)|_1 = \rho_2 \underline{n} \cdot (\underline{v} - \underline{v}_s)|_2$$

where 1 denotes evaluation in phase 1 and 2 denotes evaluation in phase 2.

This condition is applied to the rotated form of the mesh equations. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question (see for example boundary condition *VL_EQUIL_PSEUDORXN*).

References

GTM-015.1: Implementation Plan for Upgrading Boundary Conditions at Discontinuous-Variable Interfaces, January 8, 2001, H. K. Moffat

KINEMATIC_EDGE

```
BC = KINEMATIC_EDGE <bc_id1> <bc_id2> <float1>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card). It enforces the boundary of the mesh defined by the side set to conform to a transient or steady material surface, with an optional, pre-specified mass loss/gain rate. This condition is applied only in three-dimensional problems along contact lines that define the intersection of a freesurface and a geometrical solid, the intersection of which is partially characterized by the binormal tangent as described below.

Definitions of the input parameters are as follows:

KINEMATIC_EDGE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This surface is the "primary solid surface"
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This surface is the "free surface"
<float1>	Mass-loss (positive) or mass-gain (negative) velocity at the free boundary.

Examples

```
BC = KINEMATIC_SPECIES SS 10 2 0.0
```

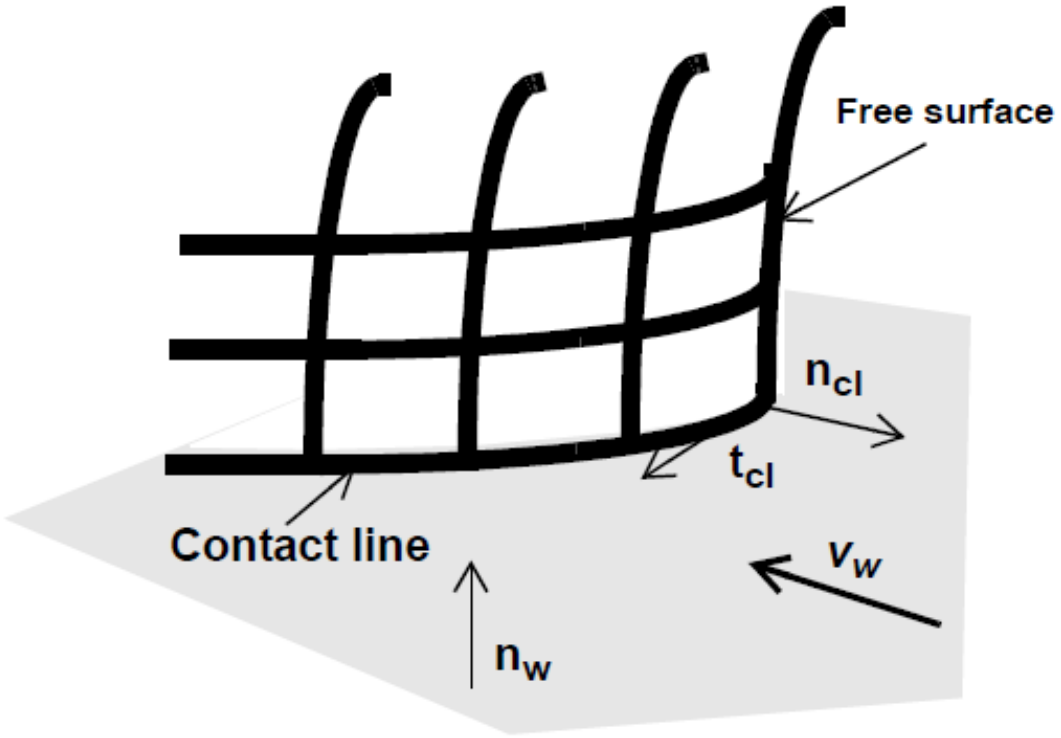
In this example, the *KINEMATIC_EDGE* boundary condition is applied to the line defined by the intersection of side sets 10 and 20. The normal vector used in application of this condition is the one in the plane of side-set 10, viz. it is tangent to the surface delineated by side set 10.

Technical Discussion

The functional form of the kinematic boundary condition is:

$$\underline{n}_{\text{cl}} \cdot (\underline{v} - \underline{v}_s) = 0$$

Here $\underline{n}_{\text{cl}}$ is the unit normal tangent vector to a line in space defined by two surfaces, in the plane of the primary surface, viz. tangent to that surface. $\underline{v}$ is the velocity of the fluid, $\underline{v}_s$ is the velocity of the surface (or mesh). This condition only makes sense in three dimensions, and needs to be directed with *ROT* conditions for proper application.



References

GT-007.2: Tutorial on droplet on incline problem, July 30, 1999, T. A. Baer

KINEMATIC_SPECIES

```
BC = KINEMATIC_SPECIES SS <bc_id> <integer>
```

Description / Usage

(WIC/MASS)

This boundary condition card is used to impose an interphase species flux continuity constraint on species components undergoing phase change between two materials. The species conservation equation (see *EQ* card and *species_bulk*) for a single gas or liquid phase component requires two boundary conditions because of the multivalued, discontinuous concentration at the interface. This condition should be used in conjunction with *VL_EQUIL* tie condition for each species. Definitions of the input parameters are as follows:

KINEMATIC_SPECIES	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number.
<float1>	Unused floating point number.

This boundary condition is typically applied to multicomponent two-phase flows that have rapid mass exchange between phases, rapid enough to induce a diffusion velocity at the interface, and to thermal contact resistance type problems. The best example of this is rapid evaporation of a liquid component into a gas.

Examples

Following is a sample card:

```
BC = KINEMATIC_SPECIES SS 10 2 0.0
```

This card invokes the species flux balance condition on species 2 at shared side set 10 to be applied to the liquid phase convective diffusion equation. It should be used in conjunction with a *VL_EQUIL* type condition on the same species, but from the bounding phase. Note: side set 10 must be a double-sided side set between two materials (i.e., must be attached to both materials), each deploying basis function interpolation of type **Q1_D** or **Q2_D**.

Technical Discussion

The condition only applies to interphase mass transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question. The mathematical form is

$$\underline{n} \cdot \left[\left(\underline{v}_s - \underline{v}^l \right) y_i{}^l \rho^l - \underline{j}_i{}^l \right] = \underline{n} \cdot \left[\left(\underline{v}_s - \underline{v}^g \right) y_i{}^g \rho^g - \underline{j}_i{}^g \right]$$

Here $\underline{v}^l$ and $\underline{v}^g$ are the gas and liquid velocity vectors at the free surface, respectively; $\underline{v}_s$ is the mesh velocity at the same location; ρ^l and ρ^g are the liquid and gas phase densities, respectively; $y_i{}^l$ and $y_i{}^g$ are the liquid and gas phase volume fractions of component i; and $\underline{j_i}^l$ and $\underline{j_i}^g$ the mass fluxes of component i. This condition constrains only one of two phase concentrations at the discontinuous interface. The other needs to come from a Dirichlet boundary condition like (BC =) Y, or an equilibrium boundary condition like *VL_EQUIL*.

References

Schunk, P. R. and Rao, R. R. 1994. "Finite element analysis of multicomponent twophase flows with interphase mass and momentum transport", Int. J. Numer. Meth. Fluids, 18, 821-842.

GTM-007.1: New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

KIN_DISPLACEMENT_PETROV

```
BC = KIN_DISPLACEMENT_PETROV SS <bc_id> <integer>
```

Description / Usage

(SIC/ROTATED MESH)

The *KIN_DISPLACEMENT_PETROV* boundary condition is exactly the same as *KIN_DISPLACEMENT* except in the way in which it is applied numerically to a problem. See *KIN_DISPLACEMENT* for a full discussion.

Definitions of the input parameters are as follows:

KIN_DISPLACEMENT_PETROV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Element block identification number for the region of TALE solid mesh motion.

Sometimes this condition is a better alternative to *KIN_DISPLACEMENT* to stabilize the surface and prevent wiggles. If the user wants to know more regarding numerical issues and implementation, consult the description for the fluid-counterpart *KINEMATIC_PETROV* card.

Examples

The following sample card:

```
BC = KIN_DISPLACEMENT_PETROV SS 7 12
```

leads to the application of the kinematic boundary condition (displacement form, see below) to the boundary-normal component of the mesh-stress equation to the boundary defined by side set 7. The element block ID number which shares this boundary with a neighboring *TALE* or fluid *ARBITRARY* region is 12.

Technical Discussion

See discussions on the *KINEMATIC_PETROV* and *KIN_DISPLACEMENT* cards.

References

No References.

KIN_DISPLACEMENT_COLLOC

```
BC = KIN_DISPLACEMENT_COLLOC SS <bc_id> <integer>
```

Description / Usage

(SIC/ROTATED MESH)

The *KIN_DISPLACEMENT_COLLOC* boundary condition is exactly the same as *KIN_DISPLACEMENT* except in the way in which it is applied numerically to a problem. See *KIN_DISPLACEMENT* for a full discussion.

Definitions of the input parameters are as follows:

KIN_DISPLACEMENT_COLLOC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Element block identification number for the region of TALE solid mesh motion.

Sometimes this condition is a better alternative to *KIN_DISPLACEMENT* to stabilize the surface and prevent wiggles. If the user wants to know more regarding numerical issues and implementation, consult the description for the fluid-counterpart *KINEMATIC_COLLOC* card.

Examples

The following sample card:

```
BC = KIN_DISPLACEMENT_COLLOC SS 7 12
```

leads to the application of the kinematic boundary condition (displacement form, see below) to the boundary-normal component of the mesh-stress equation to the boundary defined by side set 7. The element block ID number which shares this boundary with a neighboring *TALE* or fluid *ARBITRARY* region is 12.

Technical Discussion

See discussions on the *KINEMATIC_COLLOC* and *KIN_DISPLACEMENT* cards.

References

No References.

KIN_DISPLACEMENT

```
BC = KIN_DISPLACEMENT SS <bc_id> <integer>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition on the mesh motion equations (viz. *mesh1, mesh2*, and *mesh3* under the *EQ* card). It forces the boundary of the mesh defined by the side set to conform to a transient or steady material surface. Unlike the *KINEMATIC* condition, which is designed for material surfaces between two fluids, or the external material boundary of a fluid, this condition is applied to solid materials to which the *TOTAL_ALE* mesh motion scheme is applied (see technical discussion below and the *Mesh Motion* card). In two dimensions, this condition is automatically applied to the normal component of the vector mesh equations, which is rotated into normal-tangential form. In three dimensions, the application of this boundary condition needs to be further directed with the *ROT* cards (see *ROTATION specifications). The application of this condition should be compared with KIN_DISPLACEMENT_PETROV* and *KIN_DISPLACEMENT_COLLOC*.

Definitions of the input parameters are as follows:

KIN_DISPLACEMENT	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Element block identification number for the region of TALE solid mesh motion.

Examples

The following sample card:

```
BC = KIN_DISPLACEMENT SS 7 12
```

leads to the application of the kinematic boundary condition (displacement form, see below) to the boundary-normal component of the mesh-stress equation to the boundary defined by side set 7. The element block ID number which shares this boundary with a neighboring *TALE* or fluid *ARBITRARY* region is 12.

Technical Discussion

The functional form of the kinematic boundary condition is:

$$\underset{\sim}{n} \bullet (\underset{\sim}{d}_m - \underset{\sim}{d}^0_m) - \underset{\sim}{n} \bullet (\underset{\sim}{d} - \underset{\sim}{d}^0) = 0$$

Here **EQUATION** is the unit normal vector to the solid-fluid free surface, **EQUATION** is the mesh displacement at the boundary **EQUATION**, is the mesh displacement from the base reference state (which is automatically updated from the stress-free state coordinates and for remeshes, etc. in *Goma* and need not be specified), **EQUATION** is the real solid displacement, and **EQUATION** is the real solid displacement from the base reference state (or mesh). In stark contrast with the *KINEMATIC* condition, which too is used to distinguish a material fluid surface) this condition is written in Lagrangian displacement variables for *TALE* mesh motion and is applied as a distinguishing condition on the mesh between a fluid and *TALE* solid region. In essence, it maintains a real solid displacement field such that no real-solid mass penetrates the boundary described by this condition.

References

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk, May 2000

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

KIN_LEAK

```
BC = KIN_LEAK SS <bc_id> <float1> <float2>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used as a distinguishing condition - kinematic with mass transfer on mesh equations. The flux quantity is specified on a per mass basis so heat and mass transfer coefficients are in units of L/t.

Definitions of the input parameters are as follows:

KIN_LEAK	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Mass transfer coefficient for bulk fluid (species n +1).
<float2>	Driving force concentration in external phase.

Please see Technical Discussion regarding the appropriate units for the mass transfer coefficient and concentration in the external phase. For a pure liquid case, these inputs are read directly from this card, while for a multi-component case these values are read from *YFLUX* boundary conditions corresponding to each species that is needed. See following examples.

Examples

Following are two sample input cards:

Pure Liquid Case

```
BC = KIN_LEAK SS 3   0.1   0.
```

Two Component Case

```
BC = KIN_LEAK SS 3   0.   0.
```

```
BC = YFLUX SS 3   0   0.12   0.
```

Note, in the two component case, when *Goma* finds the *KIN_LEAK* card, it scans the input deck to locate the applicable *YFLUX* conditions associated with side set 3 and creates a linked list which is used by the applying function (*kin_bc_leak*). The existence of this list is denoted in *Goma* by the addition of an integer into an unused field of the BC structure for side set 3. The bulk fluid constitutes the second component and is non-volatile so it requires no *YFLUX* card; a second volatile species would require a second *YFLUX* input card.

Technical Discussion

Functionally, the *KIN_LEAK* boundary condition can be represented as the following:

where **EQUATION** is the vector velocity; **EQUATION** is the velocity of the boundary itself (not independent from the mesh velocity); **EQUATION** is the normal vector to the surface; **EQUATION** is the concentration of species *i*; **EQUATION** is the ambient concentration of species *i* at a distance from the surface of interest and **EQUATION** is the mass transfer coefficient for species *i*. This function returns a volume flux term to the equation assembly function.

KIN_LEAK is implemented through function *kin_bc_leak*; it sums the fluxes for all species plus the bulk phase evaporation. These fluxes are computed via several other function calls depending on the particular flux condition imposed on the boundary. (See various *YFLUX* * cards for Mass Equations.) However, at the end of the *kin_bc_leak* function, the accumulated flux value is assigned to variable *vnormal*, i.e., the velocity of fluid relative to the mesh. The apparent

$$\underset{\sim}{n} \bullet (\underset{\sim}{v} - \underset{\sim}{v}_s) = \sum_i h_i (y_i - y^0_i)$$

absence of a density factor here to convert a volume flux to a mass flux is the crucial element in the proper usage of the flux boundary conditions. The explanation is rooted in the formulation of the convective-diffusion equation.

The convective-diffusion equation in *Goma* is given as

$$\frac{dy_i}{dt} = -(\boldsymbol{v} - \boldsymbol{v}_m) \bullet \nabla y_i - \nabla \bullet \boldsymbol{J}_i + \boldsymbol{R}_i$$

with mass being entirely left out of the expression. *J* is divided by density before adding into the balance equation; this presumes that volume fraction and mass fraction are equivalent. The users must be aware of this. This formulation is certainly inconvenient for problems where volume fraction and mass fraction are not equal and multicomponent molar fluxes are active elements of an analysis. However, *kin_bc_leak* is entirely consistent with the convective-diffusion equation as a velocity is a volume flux, and multiplied by a density gives a proper mass flux. If y_i is a mass concentration, and h_i were in its typical velocity units, the result is a mass flux; if y_i is a volume fraction, then we have a volume flux. So *kin_bc_leak* is consistent.

The burden here lies with the user to be consistent with a chosen set of units. A common approach is to build density into the mass transfer coefficient h_i .

FAQs

1. See the FAQ pertaining to "Continuation Strategies for Free Surface Flows" on the *DISTNG* boundary condition card.
2. A question was raised regarding the use of volume flux in *Goma*; the following portion of the question and response elucidate this topic and the subject of units. Being from several emails exchanged during January 1998, the deficiencies or lack of clarity have since been remedied prior to *Goma* 4.0, but the discussions are relevant for each user of the code.

Question: … I know what you are calling volume flux is mass flux divided by density. The point I am trying to make is that the conservation equations in the books I am familiar with talk about mass, energy, momentum, and heat fluxes. Why

do you not write your conservation equations in their naturally occurring form? If density just so happens to be common in all of the terms, then it will be obvious to the user that the problem does not depend on density. You get the same answer no matter whether you input rho=1.0 or rho=6.9834, provided of course this does not impact iterative convergence. This way, you write fluxes in terms of gradients with the transport properties (viscosity, thermal conductivity, diffusion coefficient, etc.) being in familiar units.

Answer: … First let me state the only error in the manual that exists with regard to the convection-diffusion equation (CDE) is the following:

> J_i in the nomenclature table should be described as a volume flux with units of L/t, i.e., $D \cdot \nabla y_i$, where D is in L^2/t units.

Now, this is actually stated correctly elsewhere, as it states the J_i is a diffusion flux (without being specific); to be more specific here, we should say it is a "volume flux of species i." So, in this case D is in $L \cdot L/t$ units y_i, is dimensionless and it is immaterial that the CDE is multiplied by density or not, *as long as density is constant*.

Now, in *Goma* we actually code it with no densities anywhere for the FICKIAN diffusion model. For the HYDRO diffusion model, we actually compute a J_i / ρ in the code, and handle variable density changes through that ρ. In that case J_i as computed in *Goma* is a mass flux vector, not a volume flux vector, but by dividing it by ρ and sending it back up to the CDE it changes back into a volume flux. i. e., everything is the same.

Concerning the units of the mass transfer coefficient on the YFLUX boundary condition, the above discussion now sets those. *Goma* clearly needs the flux in the following form:

$$\underset{\sim}{n} \cdot D \nabla Y = K \cdot (y_i - y_i^{\infty})$$

and dimensionally for the left hand side

$$(L^2/t) \cdot (1/L) = L/t$$

where D is in units L^2/t, the gradient operator has units of 1/L so K *has* to be in units of L/t (period!) because y_i is a fraction.

So, if you want a formulation as follows:

$$\underset{\sim}{n} \cdot D\nabla Y = \hat{K}(p_i - p_i^{\infty})$$

then K's units will have to accommodate for the relationship between p_i and y_i in the liquid, hopefully a linear one as in Raoult's law, i.e. if $p_i = PvV_i$ where Pv is the vapor pressure, then

$$\underset{\sim}{n} \cdot D\nabla Y = KP_V(y_i - y_i^{\infty})$$

and so K on the YFLUX command has to be KPv….and so on.

Finally, you will note, since we do not multiply through by density, you will have to take care of that, i. e., in the Price paper he gives K in units of t/L. So, that must be converted as follows:

$$K_{price}(P_V/\rho) = K_{goma} \quad : \quad (t/L)(M/Lt^2)(L^3/M) = L/t$$

This checks out!

References

Price, P. E., Jr., S. Wang, I. H. Romdhane, "Extracting Effective Diffusion Parameters from Drying Experiments," AIChE Journal, 43, 8, 1925-1934 (1997)

KIN_CHEM

```
BC = KIN_CHEM SS <bc_id> <float1> ... <floatn>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition card is used to establish the sign of flux contributions to the overall mass balance on boundaries so that movements are appropriately advancing or receding depending on whether a species is a reactant or product in a surface reaction.

Definitions of the input parameters are as follows:

KIN_CHEM	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (set in EXODUS II) in the problem domain.
<float1>	Stoichiometric coefficient for species 0.
<floatn>	Stoichiometric coefficient for species n +1.

The input function will read as many stoichiometric coefficients as specified by the user for this card; the number of coefficients read is counted and saved. The stoichiometric coefficient is +1 for products or -1 for reactants. When a species is a product, the surface will advance corresponding to production/creation of mass of that species, versus recession of that interface when a reaction leads to consumption of that species.

Examples

Following is a sample card for two reactant and one product species:

```
BC = KIN_CHEM SS 25   -1.0   -1.0   1.0
```

Technical Discussion

This function is built from the same function as boundary condition *KIN_LEAK*, i.e., *kin_bc_leak*, so the user is referred to discussions for this boundary condition for appropriate details. The stoichiometric coefficients are read from the *KIN_CHEM* card or set equal to 1.0 in the absence of *KIN_CHEM*.

References

No References.

FORCE

```
BC = FORCE SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(WIC/VECTOR MESH)

This boundary condition card applies a force per unit area (traction) on a Lagrangian mesh region. The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied force is of course a vector. Definitions of the input parameters are as follows:

FORCE	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	X-component of traction in units of force/area.
<float2>	Y-component of traction in units of force/area.
<float3>	Z-component of traction in units of force/area.

Examples

Following is a sample card:

```
BC = FORCE SS 10 0. 1.0 1.0
```

This card results in a vector traction defined by **EQUATION** being applied to the side set boundary delineated by the number 10.

Technical Discussion

Important note: this boundary condition can only be applied to *LAGRANGIAN, DYNAMIC_LAGRANGIAN* or *ARBITRARY* mesh motion types (cf. *Mesh Motion* card). For real-solid mesh motion types, refer to *FORCE_RS*. Furthermore, it is rare and unlikely that this boundary condition be applied to *ARBITRARY* mesh motion regions. An example application of this boundary condition card is to address the need to apply some load pressure to a solid Lagrangian region, like a rubber roller, so as to squeeze and drive flow in a liquid region.

FAQs

On internal two-sided side sets, this boundary condition results in double the force in the same direction.

References

A MEMS Ejector for Printing Applications, A. Gooray, G. Roller, P. Galambos, K. Zavadil, R. Givler, F. Peter and J. Crowley, Proceedings of the Society of Imaging Science & Technology, Ft. Lauderdale FL, September 2001.

NORM_FORCE

```
BC = NORM_FORCE SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(WIC/VECTOR MESH)

This boundary condition card applies a force per unit area (traction) on a Lagrangian mesh region. The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied traction is of course a vector. Unlike the *FORCE* boundary condition card, the vector traction here is defined in normal-tangent vector basis. Definitions of the input parameters are as follows:

NORM_FORCE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set.
<bc_id>	The boundary flag identifier, or a side set number which is an integer that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Normal component of traction in units of force/area.
<float2>	Tangential component of traction in units of force/area.
<float3>	Second tangential component of traction in units of force/area (in 3-D).

This card actually applies a traction that is then naturally integrated over the entire side set of elements. Hence, the units on the floating point input must be force/area.

Examples

Following is a sample card:

```
BC = NORM_FORCE SS 10   0.   1.0   1.0
```

This card results in a vector traction defined by **EQUATION** being applied to the side set boundary delineated by the number 10. The normal vector is defined as the outward pointing normal to the surface. For internal surfaces defined by side sets which include both sides of the interface, this condition will result in exactly a zero traction, i.e., internal surface side sets must be attached to one element block only to get a net effect.

Technical Discussion

Important note: this boundary condition can only be applied to *LAGRANGIAN, DYNAMIC_LAGRANGIAN* or *ARBITRARY* mesh motion types (cf. *Mesh Motion* card). For real-solid mesh motion types, refer to *NORM_FORCE_RS*. Furthermore, it is rare and unlikely that this boundary condition be applied to *ARBITRARY* mesh motion regions. An example application of this boundary condition card is to apply some load pressure uniformly on the inside of a solid-membrane (like a pressurized balloon). In more advanced usage, one could tie this force to an augmenting condition on the pressure, as dictated by the ideal gas law.

This boundary condition is not used as often as the *FORCE* or *FORCE_USER* counterparts.

REP_FORCE

```
BC = REP_FORCE SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MESH)

This boundary condition card applies a force per unit area (traction) that varies as the inverse of the fourth power of the distance from a planar surface to a Lagrangian or dynamic Lagrangian mesh region. This boundary condition can be used to impose a normal contact condition (repulsion) or attraction condition (negative force) between a planar surface and the surface of a Lagrangian region. The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied force is a vector in the normal direction to the Lagrangian interface.

Definitions of the input parameters are as follows, with <float_list> having five parameters:

REP_FORCE	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Coefficient of repulsion, λ.
<float2>	Coefficient *a* of plane equation.
<float3>	Coefficient *b* of plane equation.
<float4>	Coefficient *c* of plane equation.
<float5>	Coefficient *d* of plane equation.

Refer to the Technical Discussion for an explanation of the various coefficients.

Examples

The following sample card:

```
BC = FORCE_REP SS   10  1.e+03.   1.0   0.0 0.0   -3.0
```

results in a vector traction of magnitude $-1.0e3 / h^4$ in the normal direction to surface side set 10 and the distance h is measured from side set 10 to the plane defined by $1.0x - 3. = 0$.

Technical Discussion

The *REP_FORCE* boundary condition produces a vector traction in the normal direction to a surface side set, defined by:

$$\vec{F} = F(\vec{n}) = -\frac{\lambda}{h^4}$$

where F is a force per unit area that varies with the distance h from a plane defined by

$$ax + by + cz + d = 0$$

The normal vector is defined as the outward pointing normal to the surface. For internal surfaces defined by side sets which include both sides of the interface, this condition will result in exactly a zero traction, i.e., internal surface side sets must be attached to one element block only to get a net effect.

Important note: this boundary condition can only be applied to *LAGRANGIAN, DYNAMIC_LAGRANGIAN* or *ARBITRARY* mesh motion types (cf. *Mesh Motion* card). For real-solid mesh motion types, refer to *REP_FORCE_RS*. Furthermore, it is rare and unlikely that this boundary condition be applied to *ARBITRARY* mesh motion regions. An example application of this boundary condition card is to apply some load pressure uniformly on a surface that is large enough such that this surface never penetrates a predefined planar boundary. Hence, this condition can be use to impose an impenetrable contact condition.

FAQs

On internal two-sided side sets, this boundary condition results in double the force in the same direction.

FORCE_USER

```
BC = FORCE_USER SS <bc_id> <float1> ...<floatn>
```

Description / Usage

(WIC/VECTOR MESH)

This boundary condition card applies a user-defined force per unit area (traction) on a Lagrangian or dynamic Lagrangian (see *Mesh Motion* card) mesh region. The functional form of the force is programmed in the function force_user_surf in bc_user.c, and can be made a function of any of the independent or dependent variables of the problem, including position (see example below). The force per unit area is applied to boundary delineated by the side set ID. Definitions of the input parameters are as follows:

FORCE_USR	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>…<floatn>	Parameters list (length arbitrary) for parameterizing the user defined force. These parameters are accessed through the p[]array in force_user_surf.

Examples

The input card

```
BC = FORCE_USER SS 3   {delta_t} 0.   1000.0 0.
```

used in conjuction with the following snippet of code in force_user_surf:

```
/* Comment this out FIRST!!!!! */
   /* EH(-1,"No FORCE_USER model implemented"); */
   /***************************** EXECUTION BEGINS
   *******************************/
   if (time <= p[0])
   {
           func[0] = p[1]*time/p[0];
           func[1] = p[2]*time/p[0];
           func[2] = p[3]*time/p[0];
   }
   else
   {
           func[0] = p[1];
           func[1] = p[2];
           func[2] = p[3];
   }
```

applies a time-dependent force ramped from zero to 1000.0 in the +y direction over the time period {*delta_t*}.

Technical Discussion

Used commonly to apply a force per unit area to an external surface of a solid region (*LAGRANGIAN* type, cf. *FORCE_USER_RS*), that is nonconstant, viz. time varying or spatially varying. The *FORCE* and *NORM_FORCE* boundary conditions can be used for constant forces. This condition is applied as a weak integrated condition in *Goma*, and hence will be additive with others of its kind.

FAQs

On internal two-sided side sets, this boundary condition results in double the force in the same direction.

References

No References.

CA

```
BC = CA NS <bc_id> <float_list>
```

Description / Usage

(PCC/ROTATED MESH)

This boundary condition card applies a specified contact-angle on the mesh at a single node nodeset. It is used exclusively in two dimensional computations. Its primary application is imposing contact angles at static or dynamic contact lines. Consequently, the nodeset is usually found where a free-surface boundary intersects a fixed, "geometry" boundary.

The <float_list> for this boundary condition has four values; definitions of the input parameters are as follows:

CA	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	θ, angle subtended by wall normal and free surface normal, in units of radians.
<float2>	n_x, x-component of normal vector to the geometry boundary (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float3>	n_y, y-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float4>	n_z, z-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).

Examples

Following is a sample card:

```
BC = CA NS 100 1.4 0. 1. 0.
```

This condition applies a contact angle of 1.4 radians between the free surface normal at the 100 nodeset and the vector (0,1,0). Normally, this latter vector is the normal to the solid surface in contact with the free surface at this point.

Technical Discussion

- The constraint that is imposed at the nodeset node is:

$$\underset{\sim}{n} \cdot \underset{\sim}{n}_{fs} = \cos\theta$$

  where *n* is the normal to the geometry specified on the card itself, and n_{fs} is the normal to the outward free surface computed internally by *Goma*. Also see the *CA_OR_FIX* card for an extension to this condition and *CA_EDGE* for its extension to three dimensions.

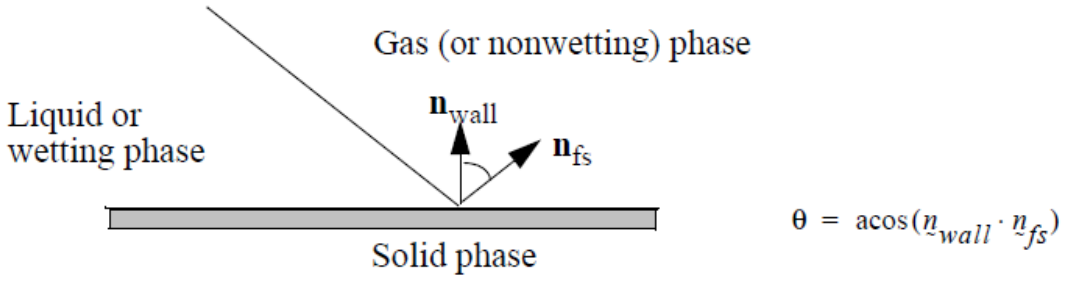
- In addition for the case in which the geometry normal components are set to zero, the wall normal is allowed to vary with a geometrical boundary condition, i.e., *GD_TABLE, SPLINE, PLANE*, etc. The geometry normal is found on the same or on a neighboring element that contains the dynamic contact angle in question. If a *GD*_ type boundary condition is used to describe the wall (i.e., *GD_TABLE*), one must specify the *R_MESH_NORMAL* equation type on that equation for the variable wall normal to take effect.

- **Important: Variable Wall Normals**. Situations for which the wall shape is nonplanar, meaning that the normal vector is not invariant as the contact line moves, there is an option to leave all of the normal-vector components zero. In this case *Goma* then seeks to determine the local wall normal vector from the geometry it is currently on, using the element facets. It is recommended that this option not be used unless the geometry is truly nonplanar, as the logic is complex and not 100% reliable. An example of such a case is as follows:

```
BC = CA NS 100 1.4 0. 0. 0.
```

  Notice how all three components of the normal vector are set to zero.

- **Important: Wall Normal convention**. The wall normal vector on an external solid boundary is defined in goma as the inward facing normal to the mesh, and the free surface normal to the liquid (or wetting phase for two-liquid systems) is defined as the outward facing normal to the free surface. Put another way and referring to the picture below, the wall normal is directed from the "solid phase" to the "liquid phase", and the free surface normal is directed from the "liquid phase" or "wetting phase" to the "vapor phase" or "Non-wetting phase". Note that for zero contact angle the liquid is "perfectly wetting". The air-entrainment limit (viz. the hydrodynamic theory interpretation) would occure at a 180 degree contact angle. Recall that the angle is specified in radians on this card.



References

No References.

CA_OR_FIX

```
BC = CA_OR_FIX NS <bc_id> <float_list>
```

Description / Usage

(PCC/ROTATED MESH)

This boundary condition card allows the application of Gibb's inequality condition in conjuction with a contact angle. This allows for a point to be specified at which a contact line will attach itself and no longer move. Up to that point, the contact line will advance or recede with a specified fixed contact angle. When the contact line attaches, its contact angle is allowed to vary permitting the user to include discontinuities in surface slope as features of the problem. The Gibb's condition also permits the contact line to detach from its fixed point if the contact angle enters a certain range after attaching. This boundary condition is applicable only to two-dimensional problems; see *CA_EDGE_OR_FIX* for details on three dimensional implementations.

The <float_list> has seven values, with definition of the input parameters as follows:

CA_OR_FIX	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	θ_{dcl}, dynamic contact angle, in radians.
<float2>	n_x, x-component of outward-pointing wall surface normal.
<float3>	n_y, y-component of outward-pointing wall surface normal.
<float4>	n_z, z-component of outward-pointing wall surface normal.
<float5>	**x_0, x-coordinate of the point or feature at which the** meniscus will pin.
<float6>	**y_0, y-coordinate of the point or feature at which the** meniscus will pin.
<float7>	**z_0, z-coordinate of the point or feature at which the** meniscus will pin.

Examples

Following is a sample card:

```
BC = CA_OR_FIX NS 100 1.3 0. 1. 0. -0.5 1. 0.
```

Technical Discussion

The Gibb's inequality condition is illustrated in the accompanying figure. The fixed point is indicated by the plane, x = x_0. Initially, the contact line is far from this point as the condition at the contact line fixes the contact angle to the value θ_{dcl}. However, when the contact line approaches to within ε (1.e-6) of the fixed point, it attaches there and stops moving. The contact angle condition is no longer enforced and the angle of the free surface with respect to the solid normal vector is allowed to vary freely. The other part of the Gibb's inequality is illustrated (above) by the last sketch. Here, by virtue of the overall fluid mechanics, the contact angle withdraws until it is larger than θ_{dcl}. When this happens the contact line is no longer affixed at x = x_0 and is allowed to move freely. Once again the contact angle condition is enforced.

Also, please see the important note under the BC = CA card regarding the convention used for specifying wall and free surface normal vectors.

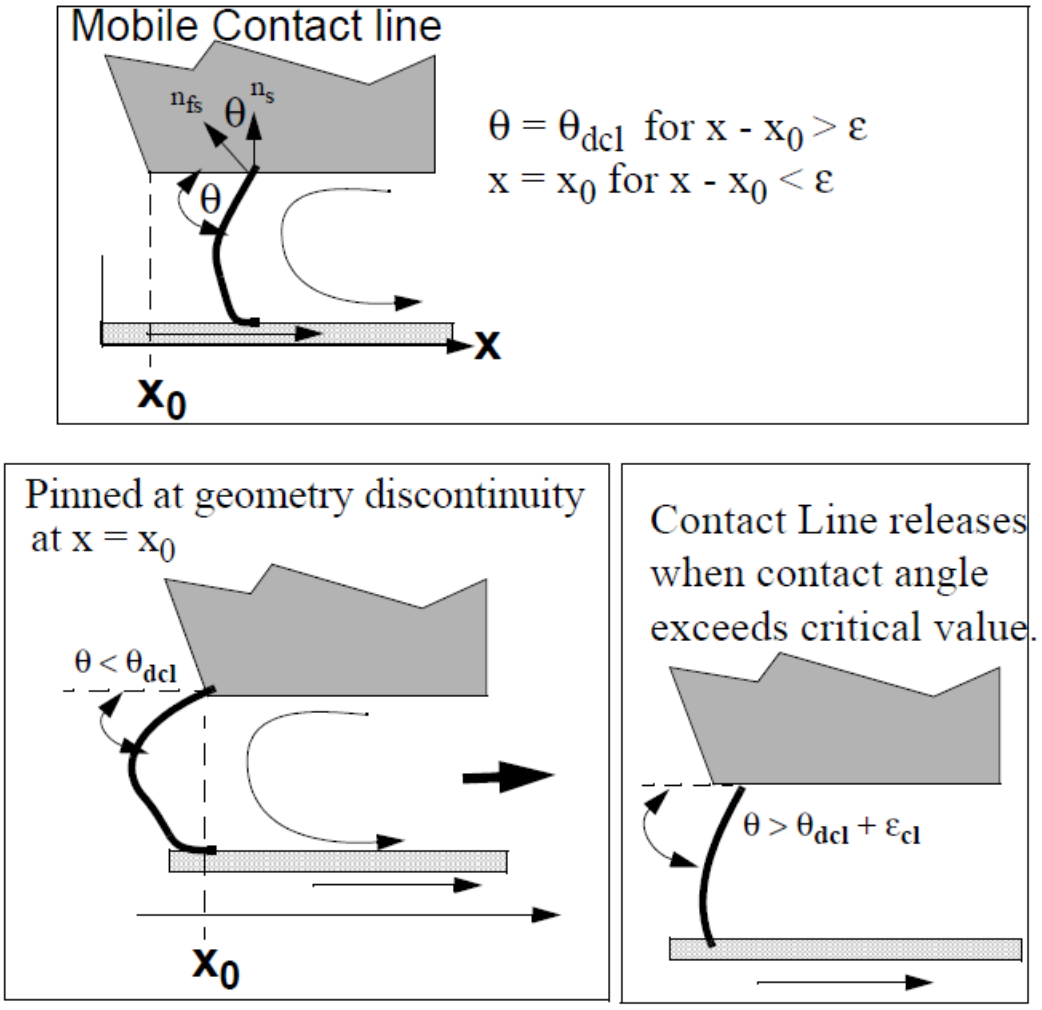


Fig. 4: Contact angles and Gibb's inequality condition in Goma, for the special case when the meniscus is moving along a surface of constant x.

Theory

The principle behind this condition applies when a contact line encounters a sharp feature on a surface. The feature from a distance might appear as a sharp corner at which the meniscus/contact line prefers to locate rather than undergo continued migration. Actually on a smaller scale, the corner feature is not infinitely small, and the contact line undergoes no perceptable movement on the macroscale in order to satisfy a true contact angle. Rather than resolving this feature with a fine mesh, it is an expedient to pin the contact line there and allow it to take on any macroscale contact angle within a certain range. The line can release again if the meniscus pulls the contact line sufficiently to overcome specified bounds.

References

No References.

CA_EDGE

```
BC = CA_EDGE SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(PCC-EDGE/ROTATED MESH)

This boundary condition card specifies a constant contact angle on the edge defined by the intersection of the primary and secondary side sets. This card is used most often to enforce contact angle conditions on three-dimensional static contact lines. *It should not be used in two-dimensional problems*, where the *CA* boundary condition is the appropriate choice.

The contact angle supplied on the card will be enforced so that it is the angle between the outward-pointing normal of the primary side set and the unit vector supplied on the card. It is important to note that this outward-pointing normal should be variable, that is to say, the primary side set is most likely a free-surface.

Definitions of the input parameters are as follows:

CA_EDGE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; in almost all cases it should also be a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*, which plays no other role in this boundary condition than to provide a means of defining the appropriate edge geometry in conjunction with the primary side set. Thus, the secondary side set will often represent a solid boundary.
<float1>	angle, value specifying the enforced angle, in degrees; it should lie in the range $0 \leq$ angle ≤ 180.
<float2>	n_x, the x-component of the fixed unit vector.
<float3>	n_y, the y-component of the fixed unit vector.
<float4>	n_z, the z-component of the fixed unit vector.

This boundary condition is a point collocated condition so it will be enforced exactly at every node that lies on the edge (subject to overriding *ROT* cards or Dirichlet conditions).

Examples

The following is a sample input card:

```
BC = CA_EDGE SS 40 50 33.0 0. 1. 0.
```

This card will result in an angle of 33 degrees between the outward-pointing normal to side set 40 and the vector (0,1,0) at all points on the edge defined by the intersection of side set 40 and side set 50.

Technical Discussion

- Although this constraint deals with vector quantities, it is a scalar constraint. The actual requirement that is imposed is:

$$n_f \cdot n = \cos(\theta)$$

  where n_f is the outward-pointing normal to the primary side set, *n* is the vector supplied on the card, and θ is the angle supplied on the card. It should be recognized that there are usually two orientations for n_f which would satisfy this constraint. Most often the surrounding physics will choose the correct one, but there is nothing to guarantee this in special situations, for example, values for θ near zero or near 180.

- This boundary condition is a point collocated condition so the preceding constraint, will be enforce exactly and strongly for each node on the edge. The actual free surface normal is an average of vectors supplied by adjacent elements sharing a given node.

- As noted above, this boundary condition is most often used in three-dimensional free surface problems to enforce static contact angle conditions at the junction of a free, capillary surface and a solid boundary. The normal vector supplied on the card would be the normal to this solid boundary. Since this vector is a constant, there is the restriction that in this application this boundary condition can only be used to specify a contact angle with respect to a *planar* solid boundary. A different boundary condition, *CA_EDGE_CURVE*, should be used if the solid boundary is not planar.

- Related boundary conditions: *CA_EDGE_INT, CA_EDGE_CURVE, CA_EDGE_CURVE_INT, VAR_CA_EDGE, VAR_CA_USER*.

References

No References.

CA_EDGE_INT

```
BC = CA_EDGE_INT SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(SIC-EDGE/ROTATED MESH)

This boundary condition card specifies a constant contact angle on the edge defined by the intersection of the primary and secondary side sets. It is identical in format and function as the *CA_EDGE* boundary condition. The only difference is that this boundary condition is a strong integrated constraint.

Definitions of the input parameters are as follows:

CA_EDGE_INT	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; in almost all cases it should also be a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*, which plays no other role in this boundary condition than to provide a means of defining the appropriate edge geometry in conjunction with the primary side set. Thus, the secondary side set will often represent a solid boundary.
<float1>	angle, value specifying the enforced angle, in degrees; it should lie in the range $0 \leq$ angle ≤ 180.
<float2>	n_x, the x-component of the fixed unit vector.
<float3>	n_y, the y-component of the fixed unit vector.
<float4>	n_z, the z-component of the fixed unit vector.

Examples

The following is a sample input card:

```
BC = CA_EDGE_INT SS 40 50 33.0 0. 1. 0.
```

This card will result in an angle of 33 degrees between the outward-pointing normal to side set 40 and the vector (0,1,0) at all points on the edge defined by the intersection of side set 40 and side set 50.

Technical Discussion

- As noted above, this boundary condition is identical in function to the *CA_EDGE* condition. It differs only in the manner of its application. Whereas, the former was a point collocated constraint, this boundary condition strongly enforces the following *integrated* constraint at a node i**:

$$\int_{\Gamma} \phi_i (n_f \cdot n - \cos(\theta)) d\Gamma = 0$$

  where φ_i is the finite element trial function for node i, Γ is the edge space curve, n:sub: f is the outward-pointing normal to the primary sideset, n is the vector supplied on the card, and θ is the angle supplied on the card. Because it is an integrated constraint, evaluation of the free-surface normal vector is done at integration points between nodes on the edge. Therefore, there is no averaging of normal vectors. This is sometimes advantageous when there are discontinuities in the slope of the edge curve.

- Related boundary conditions: *CA_EDGE, CA_EDGE_CURVE, CA_EDGE_CURVE_INT, VAR_CA_EDGE, VAR_CA_USER*.

References

No References.

CA_EDGE_OR_FIX

```
BC = CA_EDGE_OR_FIX SS <bc_id1> <bc_id2> <type_string> {float_list}
```

Description / Usage

(PCC/ROTATED MESH)

In analogy to the two-dimensional condition, *CA_OR_FIX*, boundary condition, this boundary condition imposes a contact angle on an edge feature in a three-dimensional mesh. However, this condition also permits the user to specify a closed curve on the substrate plane on which the contact line will attach and not move past. This permits modeling of geometric features in which the substrate slope is discontinuous. When contact lines encounter such sharp features, usually they arrest. The boundary condition also permits the contact line to release from the curve if the overall fluid mechanics would promote a recession of the contact line.

Description of the card parameters is as follows:

CA_EDGE_OR_FIX	Name of boundary condition.	
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.	
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set* defining the edge curve on which this condition applies.	
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set* defining the edge curve on which this condition applies. Taken together, the edge curve is the intersection of the primary and secondary sidesets.	
<type_string>	A string identifying the type of *feature curve* being defined; currently, there are only two choices: **CIRCLE** and **USER**. The **CIRCLE** options indicates that the surface feature on which a Gibb's criterion is applied is a circle in the substrate plane. The **USER** option indicates that the user will have to provide a geometric definition in the user subroutine user_gibbs_criterion in the file user_bc.c.	
{float_list}	A list of float parameters to be used in defining the contact angle, the normal to the substrate, and other geometric parameters used to define the feature curve. For each <type_string> choice there is a different set of float parameters:	
	CIRCLE <float_list>	
	<float1>	θ_{dcl}, contact angle at dynamic contact line, in radians
	<float2>	n_x, x-component of outward substrate normal
	<float3>	n_y, y-component of outward substrate normal
	<float4>	n_z, z-component of outward substrate normal
	<float5>	c_x, x coordinate of circle center
	<float6>	c_y, y-coordinate of circle center
	<float7>	c_z, z-coordinate of circle center
	<float8>	r, radius of circle
	The sign of this last parameter is important. If negative, the implication is that the starting location of the contact line is outside of the circle. If positive, the original location is assumed to be completely inside the circle.	
	USER <float_list>	
	<floati>	a list of float values that are passed to the function user_gibbs_criterion in the one-dimensional array *p* in the order in which they appear on the card from left to right. The user must be certain that the parameters appearing here are sufficient for applying the Gibbs criterion as well as imposing the appropriate contact angle.

Examples

An example making use of the **CIRCLE** feature curve option is as follows:

```
BC = CA_EDGE_OR_FIX SS 10 20 CIRCLE 1.3   0. -1. 0.   0. 0. 0.   1.0
```

This card applies to the intersection between side sets 10 and 20. The constant contact angle applied is 1.3 radians. The substrate outward normal is (0, -1, 0). The feature is a circle of radius 1.0 centered at (0.0, 0.0, 0.0). The original location for the contact line must be completely inside of the feature circle. Note also that the circle center should lie in the substrate plane.

Technical Discussion

- See the Technical Discussion under the boundary condition *CA_OR_FIX* for a detailed discussion of the nature of the Gibb's criterion as it applies to contact lines. In a nutshell, however, the basic notion is that the contact line is free to advance over the substrate with an imposed contact angle, constant or dependent on the local conditions. When the contact angle encounters the geometric feature defined in the function user_gibbs_criterion, it is captured at that point and no longer advances. The contact angle is allowed to vary as long as it is held at the feature. The boundary condition also permits the contact line to release from the feature curve and recede the way it came if the contact angle ever becomes larger than its mobile value.
- So the phenomena that can be modeled with this boundary condition are those in which a contact line moves to, for example, the edge of cylinder. At the edge, the very small curvature of this feature effectively presents a barrier to further advance of the contact line provided the deformation of the free surface beyond the vertical boundaries of the cylinder is not too large. In the fullness of time, it might also be the case that the free surface is drawn backwards in the direction of the cylinder axis. The contact line should also recede and this boundary condition permits this once the contact angle it makes with the cylinder top exceeds the mobile contact angle by a small amount.

References

No References.

CA_EDGE_CURVE

```
BC = CA_EDGE_CURVE SS <bc_id1> <bc_id2> <float1>
```

Description / Usage

(PCC-EDGE/ROTATED MESH)

This boundary condition allows the user to specify a constant contact angle along an edge in three-dimensions. It is similar in function to the *CA_EDGE* boundary condition in which the contact angle is enforced with respect to a fixed vector. However, for this boundary condition, the contact angle is enforced with respect to the normal of the *secondary* side set thereby permitting a contact angle constraint to be applied on a curving surface. The boundary condition is applied to the edge curve defined by the intersection of the primary and secondary side sets.

Definitions of the input parameters are as follows:

CA_EDGE_CURVE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; in almost all cases it should also be a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*. The outwards-pointing normal vector to this side set is used as the *substrate* vector when enforcing the contact angle constraint.
<float1>	the enforced contact angle, in degrees. Its value should lie in the range $0 \leq$ angle ≤ 180.

Examples

The following is a sample input card:

```
BC = CA_EDGE_CURVE SS 40 50 135.0
```

This boundary condition will enforce a 135 degree angle between the normal to the free surface on side set 40 and the outward-pointing normal to side set 50 at all points along the edge defined by side set 40 and 50. There is no restriction on whether side set 50's normal vectors must be constant.

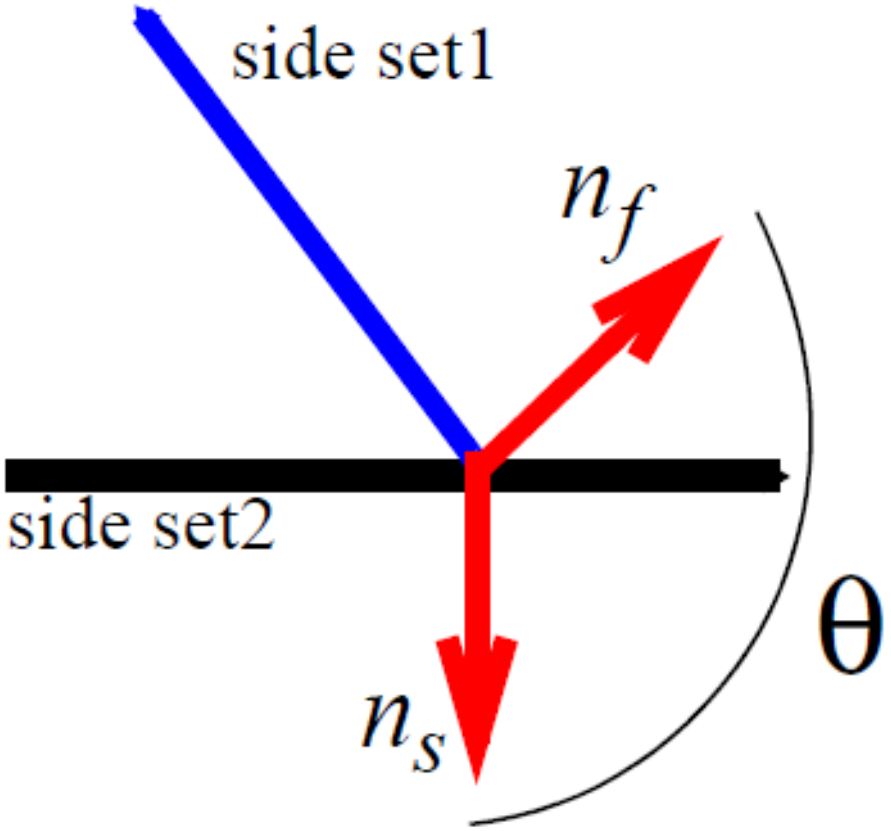
Technical Discussion

- Although this boundary condition deals with vector quantities it is a scalar constraint. The actual requirement that is imposed is:

$$n_f \cdot n_s = \cos(\theta)$$

  where n_f is the outward-pointing normal to the primary side set, n_s is the outward-pointing normal to the secondary side set, and θ is the angle supplied on the card. There is always some confusion regarding the sense of the angle; use the figure to the right for guidance. Note that the sense depicted here is at odds with the usual contact angle convention. Keep this in mind when using this card.

- As in the case of the *CA_EDGE* condition, this condition is also a strongly enforced point collocated condition.
- Related boundary conditions: *CA_EDGE, CA_EDGE_INT, CA_EDGE_CURVE_INT, VAR_CA_EDGE, VAR_CA_USER*.


side set1
n_f
side set2
θ
n_s

References

No References.

CA_EDGE_CURVE_INT

```
BC = CA_EDGE_CURVE_INT SS <bc_id1> <bc_id2> <float1>
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition allows the user to specify a constant contact angle along an edge in three-dimensions. It is identical in function to *CA_EDGE_CURVE* boundary condition, but applies as a strongly integrated constraint. The boundary condition is applied to the edge curve defined by the intersection of the primary and secondary side sets.

Definitions of the input parameters are as follows:

CA_EDGE_CURVE_INT	Card with the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; in almost all cases it should also be a free-surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*. The outwards-pointing normal vector to this side set is used as the *substrate* vector when enforcing the contact angle constraint.
<float1>	the enforced contact angle, in degrees. Its value should lie in the range $0 \leq$ angle ≤ 180.

Examples

The following is a sample input card:

```
BC = CA_EDGE_CURVE_INT SS 40 50 135.0
```

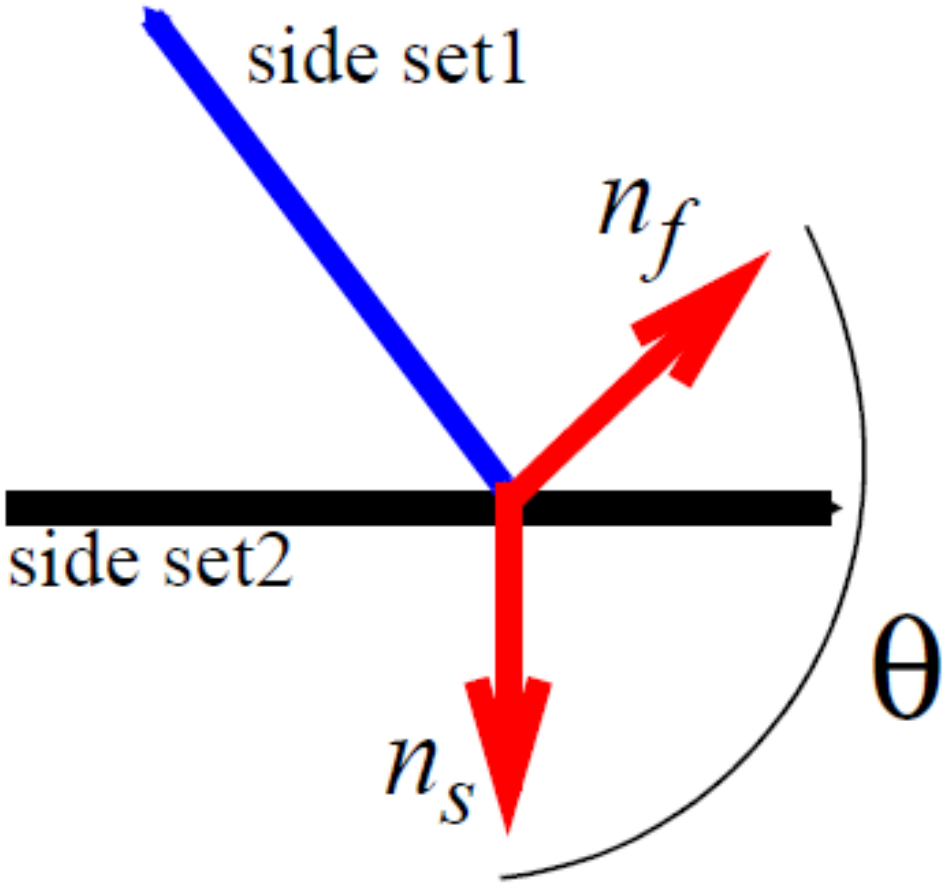
This boundary condition will enforce a 135 degree angle between the normal to the free surface on side set 40 and the outward-pointing normal to side set 50 at all points along the edge defined by side set 40 and 50. The is no restriction on whether side set 50's normal vectors must be constant.

Technical Discussion

- Although this boundary condition deals with vector quantities it is a scalar constraint. As noted above the form of the constraint is identical to that in the *CA_EDGE_CURVE* boundary. In this case, it is applied as a strong integrated constraint:

  where φ_i is the finite element trial function for node *i*, Γ is the edge space curve, n_f is the outwardpointing normal to the primary sideset, n_s is the outward-pointing normal to the secondary sideset, and θ is the angle supplied on the card. There is always some confusion regarding the sense of the angle. Use the figure to the right for guidance. Note that the sense depicted here is at odds with the usual contact angle convention. Keep this in mind when using this card.

$$\int_{\Gamma} \phi_i (n_f \cdot n_s - \cos(\theta)) d\Gamma = 0$$



- As in the case of the *CA_EDGE_INT* condition, this condition is also a strongly integrated constraint.
- Related boundary conditions: *CA_EDGE, CA_EDGE_INT, CA_EDGE_CURVE, VAR_CA_EDGE, VAR_CA_USER*.

References

No References.

VAR_CA_EDGE

```
BC = VAR_ CA_EDGE SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(SIC-EDGE/ROTATED MESH)

This card is used to set a variable contact angle on a dynamic three-dimensional contact line. A local contact angle is determined based upon the local rate of advance/recession of the contact line with respect to a web, and is always associated with the secondary sideset. This card specifies the static contact angle, θ_s, and a linear proportionality constant c_T between the local advance/recession rate and the cosine of the contact angle. The speed of the moving web is specified by components of the web velocity. The contact angle is imposed between the outward-pointing normal of the primary sideset and the outward-pointing normal of the secondary sideset.

Definitions of the input parameters are as follows:

VAR_CA_EDGE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; it should be a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*, which should be a "fixed" geometric entity, e.g. PLANE or SPLINE. Taken together, the primary and secondary sidesets define an edge over which this boundary is applicable.
<float1>	θ_s, parameter that is the static contact angle, in degrees. This is the contact angle that the fluid approaches when the relative motion of the contact line and substrate is zero.
<float2>	c_T, parameter that is the linear proportionality constant between the local advance/recession rate and the cosine of the contact angle; see details below in the Technical Discussion.
<float3>	W_x, x-component of the substrate velocity.
<float4>	W_y, y-component of the substrate velocity.
<float5>	W_z, z-component of the substrate velocity.

Examples

The following is a sample input card:

```
BC = VAR_CA_EDGE SS 60 20   135. 0.02   0. -1. 0.
```

This card sets a variable contact angle condition on the edge between side sets 60 and 20. The static contact angle is 135 degrees and the slope parameter is 0.02. The solid substrate is moving at the fixed velocity (0, -1., 0.).

Technical Discussion

- A contact line that moves relative to its underlying solid substrate is referred to as a dynamic contact line. For a dynamic contact line associated with threedimensional flows, it is recognized that the dynamic contact angle must change from point to point along the curve because the local advance/recession rate of the contact line *with respect to the substrate changes*. Taking this variability into account is the function of this card.

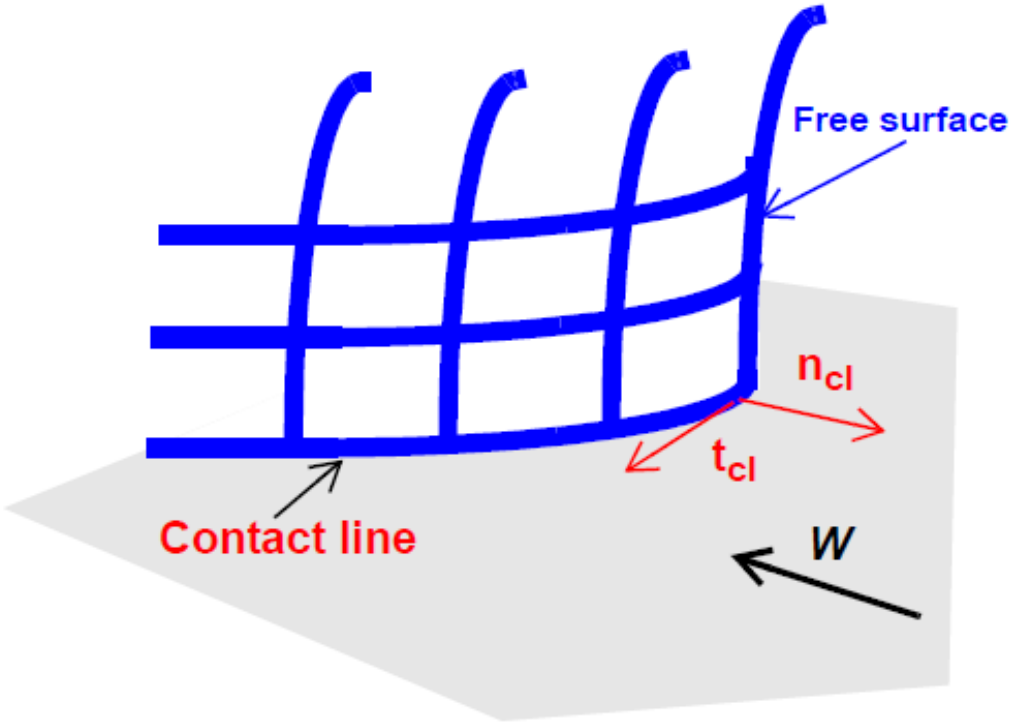
  To understand the function of this card, we first define that the advance (or recession) rate of the contact line, u_{wet}, as the normal component of the contact line velocity, x_{cl} , relative to the substrate velocity, W:

$$u_{wet} = n_{cl} \cdot (W - \dot{x}_{cl})$$

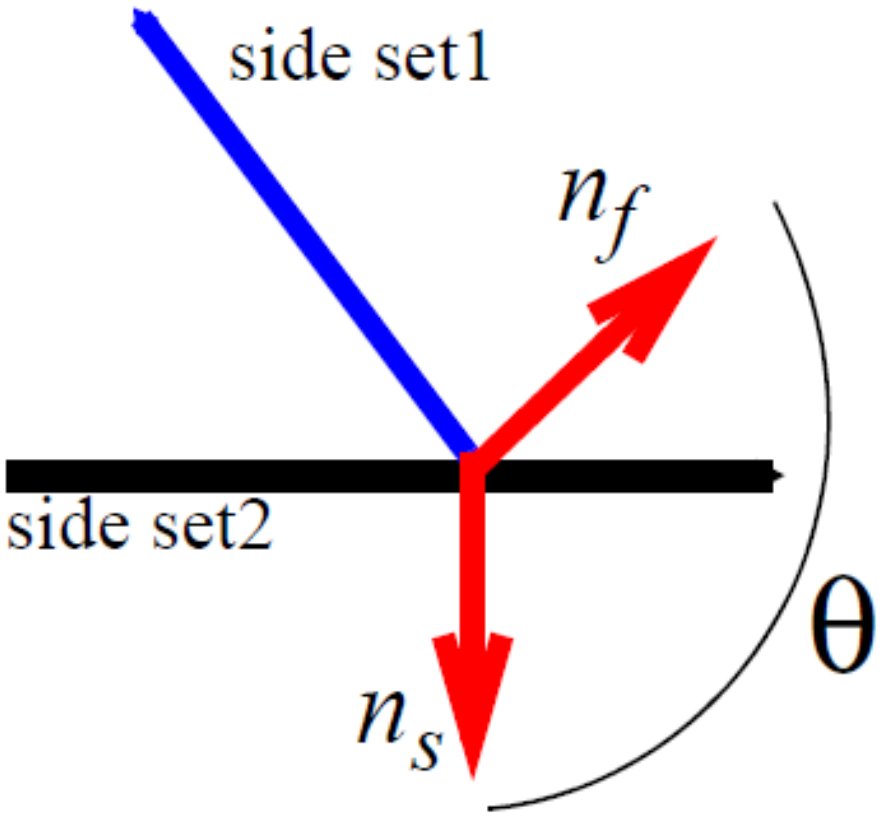
  where n_{cl} is a unit vector normal to the contact line in the plane of the substrate as illustrated in the sketch at right. For an advancing contact line u_{wet} is negative and the converse. We can also define a local capillary number by nondimensionalizing the advance rate as follows,

  where μ is the viscosity and σ the surface tension.

  We choose to define the contact angle as the angle between the outward normal to the free-surface and the substrate normal pointing away from the fluid phase as illustrate here. From direct observation of contact lines, we know that increasing the advance rate will decrease the contact angle towards zero. Conversely, a decrease in the advance rate or increase of recession rate will increase the contact angle towards 180. We capture the essence of this behavior via a simple linear relationship between the local capillary number and the cosine of the contact angle:



$$Ca_L = \mu u_{wet} / \sigma$$



$$\cos\theta = \cos\theta_s - c_T Ca_L$$

where θ_s and c_T are two input parameters. The function of this card is to apply this model for contact angle on the contact line curve.

- This model has many restrictions. It is really only valid for very very small | Ca_L | and also does not predict that the contact angle asymptotically approaches 0 or 180 for | Ca_L | very large. Instead, it is algorithmically restricted to returning 0 or 180 if the above linear relation would predict an angle outside of these bounds.
- Unlike the *CA_EDGE* boundary condition, the *VAR_CA_EDGE* condition is applied as a strong integrated constraint. The equation associated with each node on the edge is:

$$\int_{\Gamma} \phi_i (n_f \cdot n_s - (\cos\theta_s - c_T Ca_L)) d\Gamma = 0$$

where φ_i is the shape function associated with node *i*.

References

No References.

VAR_CA_USER

```
BC = VAR_CA_USER SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(SIC-EDGE/ROTATED MESH)

This card is used to set a variable contact angle on a dynamic three-dimensional contact line. It is identical in function to the *VAR_CA_USER* except that it allows the user to provide a contact angle model to relate local contact angle to local capillary number.

Definitions of the input parameters are as follows:

VAR_CA_USER	Name for the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *primary side set*; it should be a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This identifies the *secondary side set*, which should be a "fixed" geometric entity, e.g. PLANE or SPLINE. Taken together, the primary and secondary sidesets define an edge over which this boundary is applicable.
<float1>	W_x, x-component of the substrate velocity.
<float2>	W_y, y-component of the substrate velocity.
<float3>	W_z, z-component of the substrate velocity.
[float4-floatn]	An optional list of floats which will be passed to the user-supplied function for use with the user model.

Examples

The following is a sample input card:

```
BC = VAR_CA_USER SS 60 20   -1. 0. 0. 1.e-3 135.0
```

This card sets a variable contact angle condition on the edge between side sets 60 and 20. The solid substrate is moving at the fixed velocity (-1., 0., 0.). The var_CA_user function is passed the constants 1.e-3 and 135.0 in variable locations p[0] and p[1], respectively.

Technical Discussion

- *VAR_CA_USER* function is identical to *VAR_CA_EDGE*. It is applied to threedimensional dynamic contact lines in order to set a variable contact angle. The user must supply internal coding for the function var_CA_user in the file *user*_bc.c. This function receives as parameters the local capillary number as described under *VAR_CA_EDGE* and a double array containing the optional list of float parameters. It should return the cosine of the desired contact angle.
- What follows is an example that implements the linear contact angle model described in *VAR_CA_EDGE*.

```
 double
 var_CA_user(double Ca_local,
             int num,
             const double *a,
             double *d_cos_CA_Ca_local)
{
 double cos_CA;
 double static_CA;
 double cT;
 static_CA = a[0]*M_PIE/180.0;
 cT = a[1];
 cos_CA = cos(static_CA) - cT * Ca_local;
 *d_cos_CA_Ca_local = cT;
 return ( cos_CA );
}
```

References

No References.

FRICTION

```
BC = FRICTION SS <bc_id> <float1> [integer1]>
```

Description / Usage

(WIC/VECTOR MESH)

This boundary condition card applies a force per unit area (traction) on a Lagrangian mesh region. The force per unit area is applied according to Coulomb's friction law over the boundary delineated by the side set ID. The applied traction is of course a vector. The vector traction is defined in normal-tangent vector basis. Definitions of the input parameters are as follows:

FRICTION	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set.
<bc_id>	The boundary flag identifier, or a side set number which is an integer that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	μ, Coulombic coefficient of friction.
[integer1]	optional specification of the element block id to which this condition will be applied.

This card actually applies a traction that is then naturally integrated over the entire side set of elements.

Examples

Following is a sample card:

```
BC = FRICTION SS 10 0.1 2
```

Technical Discussion

Important note: this boundary condition can only be applied to *LAGRANGIAN, DYNAMIC_LAGRANGIAN* or *ARBITRARY* mesh motion types (cf. *Mesh Motion* card). For real-solid mesh motion types, refer to *FRICTION_RS*.

This condition should be utilized in conjunction with a rotated condition such as SPLINE in order to apply a tangential force which is proportional to the normal force;

where μ is the coefficient of friction and $\underline{v}$ is the velocity of the convected solid. Note that the direction of the frictional force is determined by the velocity direction.

$$\underline{F} = \mu F_n \frac{\underline{v}}{|\underline{v}|}$$

SOLID_FLUID

```
BC = SOLID_FLUID SS <bc_id> <integer1> <integer2> [float]
```

Description / Usage

(PCC/VECTOR REALSOLID)

The *SOLID_FLUID* condition performs the exact same task as the *FLUID_SOLID* condition. The usage and example are also the same, so consult the discussion on that card for further information.

At one time this condition applied the stress balance between solid and fluid phases in a different fashion that proved not to be useful. To preserve backward compatibility, we have kept this boundary condition around even though it invokes the exact same function that the *FLUID_SOLID* boundary condition does.

Definitions of the input parameters are as follows:

SOLID_FLUID	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid phase from the EXODUS II database.
<integer2>	Element block ID of liquid phase from the EXODUS II database.
[float]	Scale factor for stress balance for non-dimensionalization. This parameter, which multiplies the liquid phase contribution of stress, is optional. The default is 1.0.

Examples

See *FLUID_SOLID* description.

Technical Discussion

See *FLUID_SOLID* description.

References

No References.

PENETRATION

```
BC = PENETRATION
```

Description / Usage

()
No longer supported/used in *Goma*. See *DARCY_CONTINUOUS* boundary condition card.

Examples

No Example.

Technical Discussion

No Discussion.

References

No References.

POROUS_KIN

```
BC = POROUS_KIN
```

Description / Usage

()

This boundary condition card was used as a distinguishing condition for the Darcy- Flow in porous medium, in an arbitrary frame of reference.

This boundary condition was disabled in November of 2001 due to the new formulation in *Goma* for poroelasticity; this boundary condition was poorly formulated.

Examples

No Example.

Technical Discussion

No Discussion.

References

No References.

SDC_KIN_SF

```
BC = SDC_KIN_SF SS <bc_id> <integer> {char_string}
```

Description / Usage

(SIC/ROTATED MESH)

This boundary condition represents the specification of the normal component of the mesh velocity. This is a DVI_MULTI_PHASE_SINGLE boundary condition that has an additional property. The first time encountered in the formation of the residual, the results of a subcalculation are stored either at the node structure level or at the surface gauss point level. The surface reaction and surface species are specified as part of a surface domain within Chemkin.

The SURFDOMAINCHEMKIN_KIN_STEFAN_FLOW boundary condition (shortened to SDC_KIN_SF in the *name2* member of the *BC_descriptions* struct in mm_names.h) solves the following equation representing Stefan flow at a boundary.

where n_1 is the outward facing normal to the liquid material, p^1 is the liquid density, u^1 is the (mass average) velocity at the current surface quadrature point, and u_s the velocity of the mesh (i.e., the interface if the mesh is fixed at the interface). The summation over *N* species is for the product of molecular weight (W_k) and the source term for creation of species k in the liquid (S_k^1). SDC_KIN_SF is linked to the SDC_SPECIES_RXN boundary conditions just as the KINEMATIC_CHEM boundary conditions are by the expression for the interface reaction. The sum is over all of the interfacial source terms for species in the phase.

Definitions of the input parameters are as follows:

$$n_l \bullet [\rho^l (\boldsymbol{u}^l - \boldsymbol{u}_s)] = \sum_{k=1}^{N} -W_k S_k^l$$

SDC_KIN_SF	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	lement Block ID of the phase on whose side of the interface this boundary condition will be applied.
char_string	S_k^1 string indicating where the surface source term information for this boundary condition will be obtained. Three options exist: • **IS_EQUIL_PSEUDORXN** • **VL_EQUIL_PSEUDORXN** • **SDC_SURFRXN** These are boundary conditions that apply to the *Species Equations*. The last boundary condition is not yet implemented, so **SDC_SURFRXN** currently does nothing.

Examples

Following is a sample card:

```
BC = SDC_KIN_SF SS 1    0 VL_EQUIL_PSEUDORXN
```

The above card will create a strongly integrated boundary condition specifying the normal component of the velocity on side set 1 on the element block 0 side of the interface. The source term to be used in the above equation will be taken from multiple previously specified multiple VL_EQUIL_PSEUDORXN cards.

Technical Discussion

- This boundary condition is exactly the same as SDC_STEFANFLOW, except for the fact that it is applied on the normal component of the mesh velocity instead of the normal component of the mass averaged velocity. It is similar to a single phase boundary condition, because all of its input comes from one side of the interface. Thus, it can equally be applied to external surfaces as well as internal ones with some development work.
- Currently, it has only been tested out on internal boundaries using the IS_EQUIL_PSEUDORXN source term.
- The DVI_MULTI_PHASE_SINGLE variable is a nomenclature adopted by Moffat (2001) in his development of a revised discontinuous variable implementation for *Goma*. It pertains to Discontinuous Variable Interfaces (**DVI**)

and boundary conditions that involve the addition of a surface integral to each side of an internal boundary for a variable that is continuous across the interface. The user is referred to Moffat (2001) for detailed presentation on discontinuous variables.

References

GTM-015.1: Implementation Plan for Upgrading Boundary Conditions at Discontinuous-Variable Interfaces, January 8, 2001, H. K. Moffat

---

Category 3: Real Solid Equations

The reader is referred to a report by Schunk (2000) for a complete description of this equation type. Briefly, these boundary conditions pertain to the *TOTAL_ALE* mesh motion type (see *Mesh Motion* card), and are applied to the real solid only, viz. the boundary conditions applied to the companion mesh motion equations are still needed to control the mesh, independent of the realsolid material.

DXDYDZ_RS

```
BC = {DX_RS | DY_RS | DZ_RS} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/REALSOLID)

This boundary condition format is used to set a constant X, Y, or Z real-solid displacement on the real-solid mesh motion equations (see *TOTAL_ALE* option on the *Mesh Motion* card). Each such specification is made on a separate input card. These boundary conditions are of the Dirichlet type and must be applied on EXODUS II node sets. Definitions of the input parameters are as follows:

{**DX_RS** | **DY_RS** | **DZ_RS**}	Boundary condition name (<bc_name>) that defines the displacement, where: • **DX_RS** - real solid X displacement • **DY_RS** - real solid Y displacement • **DZ_RS** - real solid Z displacement
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of the real_solid displacement (X, Y, or Z) defined above.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

Following is a sample card which applies in an X-displacement boundary condition to the real-solid to the nodes in node set 100, specifically an X- real-solid Displacement of 0.1. These displacements are applied immediately to the unknowns, and hence result in immediate mesh displacement from the initial state.

```
BC = DX_RS NS 100 1.0
```

This sample card applies the same condition above, but as a residual equation that is iterated upon with Newton's method.

```
BC = DX_RS NS 100 1.0   1.0
```

The second float 1.0 forces this application. This approach is advisable in most situations, as the nodes are gradually moved as a part of the mesh deformation process. Sudden movements, as in the first example, can lead to folds in the mesh.

Technical Discussion

This condition performs the same function as *DX|DY|DZ* boundary conditions, except that it is applied to the real-solid of a *TOTAL_ALE* solid mesh motion model (see *Mesh Motion* card). More than likely, these conditions are applied together with geometry conditions on the mesh equations, e.g. *PLANE, DX, DY, GEOM*, etc., on the same boundary. *TOTAL_ALE* mesh motion involves two sets of elasticity equations: mesh motion equations (*mesh1* and *mesh2*), and real-solid elasticity equations (*mom_solid1* and *mom_solid2*).

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

FORCE_RS

```
BC = FORCE_RS SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(WIC/VECTOR REALSOLID)

This boundary condition card applies a force per unit area (traction) on a real-solid material region (as opposed to a Lagrangian solid region), as is the case with *TOTAL_ALE* mesh motion type (see *Mesh Motion* card). The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied force is of course a vector. Definitions of the input parameters are as follows:

FORCE_RS	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	X-component of traction in units of force/area.
<float2>	Y-component of traction in units of force/area.
<float3>	Z-component of traction in units of force/area.

Examples

Following is a sample card:

```
BC = FORCE_RS SS 10 0. 1.0 1.0
```

This card results in a vector traction defined by $\vec{F} = 0.0(\vec{e}x) + 1.0(\vec{e}y) + 1.0(\vec{e}z)$ applied to the side set boundary delineated by flag 10, where the element block bounded by this boundary is of a *TOTAL_ALE* mesh motion type.

Technical Discussion

It is important to note that this boundary condition can only be applied to *TOTAL_ALE* mesh motion types (cf. *Mesh Motion* card). (see *FORCE* for all other mesh motion types). Furthermore, it is rare and unlikely that this boundary condition be applied to *ARBITRARY* mesh motion regions. As an example of how this boundary condition card is used, consider the need to apply some load pressure to a real solid of a *TOTAL_ALE* region, like a rubber roller, so as to squeeze and drive flow in a liquid region. Some of the usage tutorials cited below will direct you to some specifics.

FAQs

On internal two-sided side sets, this boundary condition results in double the force in the same direction.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

NORM_FORCE_RS

```
BC = NORM_FORCE_RS SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(WIC/VECTOR REALSOLID)

This boundary condition card applies a force per unit area (traction) on a real-solid in a *TOTAL_ALE* mesh region (see *Mesh Motion* card). The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied traction is of course a vector. Unlike the *FORCE_RS* boundary condition card, the vector traction here is defined in normal-tangent vector basis. Definitions of the input parameters are as follows:

NORM_FORCE_RS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Normal component of traction in units of force/area.
<float2>	Tangential component of traction in units of force/area.
<float3>	Second tangential component of traction in units of force/area (in 3-D).

This card actually applies a traction that is then naturally integrated over the entire side set of elements. Hence, the units on the floating point input must be force/area.

Examples

The following is a sample input card:

```
BC = NORM_FORCE_RS SS 10 0. 1.0 1.0
```

This card results in a vector traction to the real-solid in a *TOTAL_ALE* mesh motion type (not the mesh) defined by $\vec{F}$ = 0.0($\vec{n}$) + 1.0($\vec{t}_1$) + 1.0($\vec{t}_2$) applied to the side set boundary delineated by flag 10. The normal vector is defined as the outward pointing normal to the surface. For internal surfaces defined by side sets which include both sides of the interface, this condition will result in exactly a zero traction, i.e., internal surface side sets must be attached to one element block only to get a net effect.

Technical Discussion

It is important to note that this boundary condition can only be applied to *TOTAL_ALE* mesh motion types (cf. *Mesh Motion* card). As an example of how this boundary condition card is used, consider the need to apply some load pressure uniformly on the inside of a solid-membrane (like a pressurized balloon). In more advanced usage, one could tie this force to an augmenting condition on the pressure, as dictated by the ideal gas law.

This boundary condition is not used as often as the *FORCE_RS* or *FORCE_USER_RS* counterparts.

References

No References.

REP_FORCE_RS

```
BC = REP_FORCE_RS SS <bc_id> <floatlist>
```

Description / Usage

(WIC/VECTOR REALSOLID)

This boundary condition card applies a force per unit area (traction) that varies as the inverse of the fourth power of the distance from a planar surface (see Technical Discussion below) on a *TALE* or Dynamic Lagrangian mesh region. This boundary condition can be used to impose a normal contact condition (repulsion) or attraction condition (negative force) between a planar surface and the surface of a *TALE* region. It differs from REP_FORCE card only in the mesh-motion type to which it applies. The force per unit area is applied uniformly over the boundary delineated by the side set ID. The applied force is a vector in the normal direction to the Lagrangian interface.

Definitions of the input parameters are as follows, where <floatlist> has five parameters:

REP_FORCE_RS	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Coefficient of repulsion, λ.
<float2>	Coefficient a of plane equation.
<float3>	Coefficient b of plane equation.
<float4>	Coefficient c of plane equation.
<float5>	Coefficient d of plane equation.

Examples

The following is a sample input card:

```
BC = REP_FORCE_RS SS 10 1.e+03. 1.0 0.0 0.0 -3.0
```

This card results in a vector traction in the normal direction on surface side set 10 defined by $\vec{F}$ = –1.0e3$/h^4$ where F is a force per unit area that varies with the distance h from the plane specified by 1.0x – 3.0 = 0.0 .

Technical Discussion

The repulsive force is defined by $\vec{F}$ = F($\vec{n}$) where F is a force per unit area that varies with the distance h from a plane defined by the equation ax + by + cz + d = 0. The magnitude of the function $\vec{F}$ is defined as:

$$F = -\frac{\lambda}{h^4}$$

The normal vector is defined as the outward pointing normal to the surface. For internal surfaces defined by side sets

which include both sides of the interface, this condition will result in exactly a zero traction, i.e., internal surface side sets must be attached to one material only to get a net effect.

It is important to note that this boundary condition can only be applied to *TALE* mesh motion types (cf. *Mesh Motion* card). As an example of how this boundary condition card is used, consider the need to apply some load pressure uniformly on a surface that is large enough such that this surface never penetrates a predefined planar boundary. This condition hence can be use to impose an impenetrable contact condition.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

FORCE_USER_RS

```
BC = FORCE_USER_RS SS <bc_id> <float1>...<floatn>
```

Description / Usage

(WIC/VECTOR REALSOLID)

This boundary condition card applies a user-defined force per unit area (traction) on a *TOTAL_ALE* real solid region (see *Mesh Motion* card). It differs from its counterpart *FORCE_USER* only in the type of material to which the force is applied, as described on the *Mesh Motion* card. The functional form of the force is programmed in the function force_user_surf in bc_user.c, and can be made a function of any of the independent or dependent variables of the problem, including position (see example below). The force per unit area is applied to boundary delineated by the side set ID. Definitions of the input parameters are as follows:

FORCE_USER_RS	Name of the boundary condition (<bc_name>)
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>…<floatn>	Parameters list (length arbitrary) for parameterizing the user defined force. These parameters are accessed through the p[]array in force_user_surf.

Examples

The input card

```
BC = FORCE_USER_RS SS 3   {delta_t} 0.   1000.0 0.
```

used in conjuction with the following snippet of code in force_user_surf:

```
/* Comment this out FIRST!!!!! */
/* EH(-1,"No FORCE_USER model implemented"); */
/***************************** EXECUTION BEGINS
*******************************/
   if (time <= p[0])
      {
         func[0] = p[1]*time/p[0];
```

(continues on next page)

(continued from previous page)

```
            func[1] = p[2]*time/p[0];
            func[2] = p[3]*time/p[0];
      }
  else
      {
            func[0] = p[1];
            func[1] = p[2];
            func[2] = p[3];
      }
```

applies a time-dependent force ramped from zero to 1000.0 in the +y direction over the time period {*delta_t*}. Note how p[0] is the time period, viz. {*delta_t*}, over which the force is ramped up.

Technical Discussion

Used commonly to apply a force per unit area to an external surface of a solid region (*TOTAL_ALE* type, cf. *FORCE_USER*), that is nonconstant, viz. time varying or spatially varying. The *FORCE_RS* and *NORM_FORCE_RS* boundary conditions can be used for constant forces. This condition is applied as a weak integrated condition in Goma, and hence will be additive with others of its kind.

References

No References.

SOLID_FLUID_RS

```
BC = SOLID_FLUID_RS SS <bc_id> <integer1> <integer2> [float]
```

Description / Usage

(PCC/VECTOR REALSOLID)

Used for fluid-structure interaction problems, the *SOLID_FLUID_RS* condition equates the normal traction between adjacent fluid and solid materials. (By "normal traction" we mean the tangential and normal force components, per unit area.) This condition is only to be used on boundaries between regions of *ARBITRARY* mesh motion with fluid-momentum equations and of *TOTAL_ALE* mesh motion (cf. *SOLID_FLUID* boundary condition card for *LAGRANGIAN* mesh motion regions), with solid momentum equations (or mesh equations) - see *Mesh Motion* and *EQ* cards. All elements on both sides of the interface must have the same element type (the same order of interpolation and basis functions) e.g., Q1 or Q2. Also, such interfaces must include element sides from both sides of the interface in the defining side set.

Definitions of the input parameters are as follows:

SOLID_FLUID_RS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of the solid phase (of TOTAL_ALE motion type) from the EXODUS II database.
<integer2>	Element block ID of the liquid phase from the EXODUS II database.
[float]	Scale factor for stress balance for nondimensionalization. This parameter, a multiplier on the liquid phase contribution, is optional; the default is 1.0.

Examples

The following set of input cards is a sample specification for a fluid-structure interaction problem:

```
BC = SOLID_FLUID_RS SS 5   2   1
```

```
BC = NO_SLIP_RS SS 5   2   1
```

```
BC = KIN_DISPLACEMENT SS   5   2
```

In this example, side set 5 is a boundary between a solid rubber blade and a liquid; the material in element block 2 is the blade, and the material in element block 1 is the fluid. Along the blade, a companion boundary condition is applied to ensure no slip on the same side set. Also, because this condition involves a *TOTAL_ALE* mesh region, a *KIN_DISPLACEMENT* boundary condition is needed on the same side set to force the solid boundary to follow the side set.

Technical Discussion

The functional form of the *SOLID_FLUID_RS* boundary condition is:

$$\lambda(\underset{\sim}{n} \cdot \underset{\sim}{T}) = \underset{\sim}{n} \cdot \underset{\sim}{\sigma}$$

where $\underline{T}$ is the fluid phase stress tensor given by any one of the specified fluid-phase constitutive equations, and $\underline{\sigma}$ is the real-solid solid phase stress tensor, also given by any one of the solid-phase constitutive equation (see *Mat* file specifications). λ is a scaling factor that defaults to unity (and is usually best taken as such unless some scaling is invoked). With this boundary condition, the local residual and Jacobian contributions from the fluid mechanics momentum equations (on the *ARBITRARY* side of the boundary) are added into the weak form of the residual and Jacobian entries for the real-solid solid mechanics equations (viz. the *EQ = mom_solid* options on the real-solid *TOTAL_ALE* side of the boundary).

TOTAL_ALE mesh motion regions cannot be porous and deformable (as of 11/19/ 2001).

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

SPLINEXYZ_RS

Description / Usage

(PCC/MESH)

This card is used to specify a general surface (solid) boundary description for TOTAL_ALE real solid equations (see *Mesh Motion* card). These boundary conditions are tantamount to *SPLINE_RS* , except that they do *not* invoke a vector residual rotation into normal-tangential form. Instead, *SPLINEX_RS* invokes the geometric boundary condition on the x-component of the real solid equation residual, and so on. The card requires user-defined subroutines. Templates for these routines are currently located in the routine "user_bc.c". Both a function routine, fnc, for function evaluation and corresponding routines dfncd1, dfncd2, and dfncd3 for the derivative of the function with respect to global coordinates are required. Note that it takes an arbitrary number of floating-point parameters, depending on the user's needs.

Definitions of the input parameters are as follows:

{bc_name}	Boundary condition name that defines the general surface; the options are: • **SPLINEX_RS** - X general surface • **SPLINEY_RS** - Y general surface • **SPLINEZ_RS** - Z general surface
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
[floatlist]	Constants to parameterize any f(x,y,z) = 0 function input in user-defined routine fnc.

Examples

The following is a sample input card:

```
BC = SPLINEZ_RS SS 10 1.0 100. 20.0 1001.0 32.0
```

applies a user-defined distinguishing condition parameterized by the list of floating points to the boundary defined by side set 10. Most importantly, the condition replaces the Z-component of the real solid equation.

Technical Discussion

The mathematical form of this distinguishing condition is arbitrary and is specified by the user in the fnc routine in user_bc.c. Derivatives of the user-specified function must also be provided so as to maintain strong convergence in the Newton iteration process. These functions are located next to fnc and are named dfncd1, dfncd2, and dfncd3.Several examples for simple surfaces exist in the template routine. In three dimensions, usage needs to be completed with a companion *ROT* input card which directs the equation application of the condition, even though rotations are not actually performed.

References

No References.

SPLINE_RS

```
BC = SPLINE_RS SS <bc_id> [floatlist]
```

Description / Usage

(PCC/VECTOR REALSOLID)

This card is used to specify a general surface (solid) boundary description for TOTAL_ALE type mesh motion (see *Mesh Motion* card). Like most other distinguishing conditions, this condition causes the real-solid equations, viz. *solid1, solid2*, and *solid3*, to be rotated into boundary normal-tangential form. The card requires user-defined subroutines. Templates for these routines are currently located in the routine "user_bc.c". Both a function routine, fnc, for function evaluation and corresponding routines dfncd1, dfncd2, and dfncd3 for the derivative of the function with respect to global coordinates are required. . Note that it takes an arbitrary number of floating-point parameters, depending on the user's needs.

Definitions of the input parameters are as follows:

SPLINE_RS	Name of the boundary condition <bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
[floatlist]	Constants to parameterize any f(x,y,z) = 0 function input in user-defined routine fnc.

Examples

The following is a sample input card:

```
BC = SPLINE_RS SS 10 1.0 100. 20.0 1001.0 32.0
```

applies a user-defined distinguishing condition, parameterized by the list of five floating point values, to the boundary defined by side set 10.

Technical Discussion

This condition is applied to the normal component of the real solid equations along a boundary in two dimensions; in three dimensions application needs to be further directed with the *ROT* conditions. Examples of typical distinguishing conditions can be found in user_bc.c in the fnc routine and companion derivative routines.

References

No References.

FRICTION_RS

```
BC = FRICTION_RS SS <bc_id> <float1> [integer1]>
```

Description / Usage

(WIC/VECTOR REAL SOLID)

This boundary condition card applies a force per unit area (traction) on the TOTAL_ALE solid mechanics equations. The force per unit area is applied according to Coulomb's friction law over the boundary delineated by the side set ID. The applied traction is of course a vector. The vector traction is defined in normal-tangent vector basis. Definitions of the input parameters are as follows:

FRIC-TION_RS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set.
<bc_id>	The boundary flag identifier, or a side set number which is an integer that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	μ, Coulombic coefficient of friction.
[integer1]	optional specification of the element block id to which this condition will be applied.

This card actually applies a traction that is then naturally integrated over the entire side set of elements.

Examples

Following is a sample card:

```
BC = FRICTION_RS SS 10 0.1 2
```

Technical Discussion

Important note: this boundary condition can only be applied to *TOTAL_ALE* mesh motion types (cf. *Mesh Motion* card). For other mesh motion types, refer to *FRICTION*.

This condition should be utilized in conjunction with a rotated condition such as SPLINE_RS in order to apply a tangential force which is proportional to the normal force;

where μ is the coefficient of friction and $\underline{v}$ is the velocity of the convected solid. Note that the direction of the frictional force is determined by the velocity direction.

$$\underline{F} = \mu F_n \frac{\underline{v}}{|\underline{v}|}$$

Category 4: Fluid Momentum Equations

The fluid-momentum equations, e.g., the momentum equations in the Navier-Stokes system for incompressible flows, require many boundary conditions mainly because they are formulated in an arbitrary frame of reference. The plethora of boundary conditions here contain Dirichlet, finiteelement weak form, finite-element strong form, and many other boundary condition types.

UVW

```
BC = {U | V | W} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/MOMENTUM)

This Dirichlet boundary condition specification is used to set a constant velocity in the X-, Y-, or Z-direction. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{U | V | W}	One-character boundary condition name (<bc_name>) that defines the velocity direction, where: • **U** - Indicates X velocity component • **V** - Indicates Y velocity component • **W** - Indicates Z velocity component
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of velocity component.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following are sample input cards for the X velocity component Dirichlet card:

```
BC = U NS 7   1.50
```

```
BC = U NS 7   1.50   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

This class of card is used to set Dirichlet conditions on the velocity components. When the second optional float parameter is not present, the matrix rows corresponding to the appropriate velocity component for nodes on this node set are filled with zeros, the diagonal element is set to one, the corresponding residual entry is also set to zero, and in the solution vector the appropriate degree of freedom is set to the value specified by <float1>. This is the so-called "hard set" method for specifying Dirichlet conditions.

An alternate method for specifying Dirichlet conditions is applied when the second float parameter is present (the actual value is not important except that it be different from -1.0). In this case, the Dirichlet constraint is applied as a residual equation. That is, the momentum equation for the appropriate component at each node in the nodeset is replaced by the residual equation,

This residual equation is included in the Newton's method iteration scheme like any other residual equation. Note that in this case, nothing is set in the solution vector since that will occur automatically as part of the iteration method.

$$R = v - \langle float1 \rangle$$

References

No References.

PUVW

```
BC = {PU | PV | PW}
```

Description / Usage

(DC/PMOMENTUM)

This card is currently not implemented.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

UVWVARY

```
BC = {UVARY | VVARY | WVARY} SS <bc_id> [float_list]
```

Description / Usage

(PCC/MOMENTUM)

The *UVARY, VVARY* and *WVARY* boundary condition format is used to set variation in X, Y, or Z velocity component, respectively, with respect to coordinates and time on a specified sideset. Each such specification is made on a separate input card.

The *UVARY, VVARY*, and *WVARY* cards each require user-defined functions be supplied in the file user_bc.c. Four separate C functions must be defined for a boundary condition: velo_vary_fnc, dvelo_vary_fnc_d1, dvelo_vary_fnc_d2, and dvelo_vary_fnc_d3. The first function returns the velocity component at a specified coordinate and time value, the second, third, and fourth functions return the derivative of the velocity component with x, y and z respectively.

A description of the syntax of this card follows:

{UVARY | VVARY | WVARY}	Five-character boundary condition name (<bc_name>) identifies the velocity component: • **UVARY** -X velocity component • **VVARY** -Y velocity component • **WVARY** -Z velocity component
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
[float_list]	An optional list of float values separated by spaces which will be passed to the user-defined subroutines to allow the user to vary the parameters of the boundary condition. This list of float values is passed as a onedimensional double array designated p in the parameter list of all four C functions.

Examples

Following is a sample card for an X component

```
BC = UVARY SS 10 2.0 4.0
```

Following are the C functions that would have to be implemented in "user_bc.c" to apply the preceding boundary condition card to set a parabolic velocity profile along a sideset.

```
double velo_vary_fnc( const int velo_condition, const double x,
  const double y, const double z, const double p[], const double
  time )
{
double f = 0;
double height = p[0];
```

(continues on next page)

(continued from previous page)

```
double max_speed = p[1];
  if ( velo_condition == UVARY ) {
  f = max_speed*( 1.0 - pow(y/height, 2 ) );
  }
return(f);
}
/* */
double dvelo_vary_fnc_d1( const int velo_condition, const double
  x, const double y, const double z, const double p[], const
  double time )
{
double f = 0;
return(f);
}
/* */
double dvelo_vary_fnc_d2( const int velo_condition, const double
  x, const double y, const double z, const double p[], const
  double time )
{
double f = 0;
double height = p[0];
double max_speed = p[1];
  if ( velo_condition == UVARY ) {
  f = -2.0*max_speed*(y/height)/height;
}
return(f);
}
/* */
double dvelo_vary_fnc_d3( const int velo_condition, const double
  x, const double y, const double z, const double p[], const
  double time )
{
double f = 0;
return(f);
}
/* */
```

Technical Discussion

- Including the sensitivities is a pain, but required since *Goma* has no provision for computing Jacobian entries numerically.
- Note that the type of boundary condition (*UVARY, VVARY*, or *WVARY*) is sent to each function in the velo_condition parameter. Since there can be only one set of definition functions in user_bc.c, this allows the user to overload these functions to allow for more than one component defined in this manner. It would also be possible to use these functions to make multiple definitions of the same velocity component on different sidesets. However, this would have to be done by sending an identifier through the p array.
- This is a collocated-type boundary condition. It is applied exactly at nodal locations but has lower precedence of application than direct Dirichlet conditions.

References

No References.

UVWUSER

```
BC = {UUSER | VUSER | WUSER} SS <bc_id> <float_list>
```

Description / Usage

(SIC/MOMENTUM)

This card permits the user to specify an arbitrary integrated condition to replace a component of the fluid momentum equations on a bounding surface. Specification of the integrand is done via the functions uuser_surf, vuser_surf and wuser_surf in file "user_bc.c.", respectively.

A description of the syntax of this card follows:

{UUSER | VUSER | WUSER}	Five-character boundary condition name (<bc_name>) identifies the momentum equation component: • **UUSER** - X momentum component • **VUSER** - Y momentum component • **WUSER** - Z momentum component
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following is an example of card syntax:

```
BC = VUSER SS 10 1.0
```

Implementing the user-defined functions requires knowledge of basic data structures in *Goma* and their appropriate use. The uninitiated will not be able to do this without guidance.

References

No References.

NO_SLIP/NO_SLIP_RS

```
BC = {NO_SLIP | NO_SLIP_RS} SS <bc_id> <integer1> <integer2>
```

Description / Usage

(SIC/ VECTOR MOMENTUM)

This card invokes a special boundary condition that applies a no-slip condition to the fluid velocity at an interface between a liquid phase and a solid phase so that the fluid velocity and solid velocity will be in concert. The solid phase must be treated as a Lagrangian solid and may be in a convected frame of reference. The fluid velocity is equal to the velocity of the stress-free state mapped into the deformed state (for steadystate problems).

In general, a *SOLID_FLUID* boundary condition must also be applied to the same boundary so that the force balance between liquid and solid is enforced. Note that a *FLUID_SOLID* boundary condition will have no effect since the strongly enforced *NO_SLIP/NO_SLIP_RS* on the fluid momentum equation will clobber it.

All elements on both sides of the interface must have the same element type, i.e., the same order of interpolation and basis functions, e.g., Q1 or Q2.

Definitions of the input parameters are as follows:

{NO_SLIP | NO_SLIP_RS}	Boundary condition name applied in the following formulations: • **NO_SLIP** - this condition applies when the solid phase is a purely *LAGRANGIAN* solid. • **NO_SLIP_RS** - this condition should be used instead when the displacements in the solid phase are determined via a *TALE* formulation.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This side set should be the intersection of liquid and solid element blocks and be defined so that it is present in both element blocks.
<integer1>	the element block ID number of the solid phase material.
<integer2>	the element block ID number of the liquid phase material.

Examples

The following is a sample input card:

```
BC= NO_SLIP SS 10   2 1
```

This card will enforce continuity of velocity between the solid phase in element block 2 with the fluid phase in element block 1. Side set 10 should be in common with both element blocks.

Technical Discussion

- This boundary condition is a vector condition meaning that all three components of the fluid momentum equation are affected by use of a single boundary condition. The actual constraints that are imposed at node *j* are:

$$\int \phi_j (v_f - v_s) \cdot \delta_x d\Gamma = 0 \quad \int \phi_j (v_f - v_s) \cdot \delta_y d\Gamma = 0 \quad \int \phi_j (v_f - v_s) \cdot \delta_z d\Gamma = 0$$

where ϕ_j is the finite element trial function, v_f is the fluid velocity, and v_s is the solid phase velocity. These three constraints are strongly enforced so they replace completely the x, y, and z fluid momentum components. The boundary condition is not rotated since all three components of the momentum equation are supplanted.

- As mentioned above this boundary condition is used most often in conjunction with the *SOLID_FLUID* boundary condition which equates stresses across fluid/ solid interfaces. As described in the section discussing this card, this latter card imposes these forces by using the residuals of the fluid momentum equation as surrogates for the fluid phase forces. These forces however are imposed on the solid equations prior to imposition of the *NO_SLIP* boundary condition.
- As noted above, for this boundary condition to function properly it is necessary that the side set between the fluid and solid element block be present in both element blocks. To explain this it is necessary to recognize that side sets are defined as a set of faces attached to specific elements. This is in contrast to node sets which are simply a list of node numbers. Therefore, in the case of a side set that lies at the interface of two element blocks, it is possible for a given face in that side set to appear twice, once attached to the element in the first element block and a second time attached to the adjoining element in the second element block. This is the condition that is required for the proper execution of this boundary condition. Fortunately, this is the default of most meshing tools that interface with *Goma*.
- It is also important to reiterate that another necessary condition for the proper function of this boundary condition is that the interpolation order of the pseudosolid mesh unknowns *and the fluid velocity unknowns* in the *ALE* fluid phase block be identical to the interpolation order of the solid displacement unknowns in the *LAGRANGIAN* or *TALE* adjoining solid phase block. This usually means that the element type must be the same in both phases. In two-dimensions this generally is not a problem, but in three dimensions it can impose a considerable hardship on the analyst.

References

No References.

VELO_NORMAL

```
BC = VELO_NORMAL SS <bc_id> <float> [integer]
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition allows the user to set the outward velocity component normal to a surface.

Definitions of the input parameters are as follows:

VELO_NORMAL	Boundary condition designation.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	V_n, value of the normal velocity component. Note that this velocity component is relative to the motion of the underlying mesh.
[integer]	*blk_id*, an optional parameter that is the element block number in conjugate problems that identifies the material region where the *VELO_NORMAL* condition will be applied (usually the liquid element block in solid/liquid conjugate problems). For external boundaries, this optional parameter can be set to unity to force the condition to be kept at a corner between two side sets (2D only). This is handy for corner conditions. Please see GTM-004.0 for details.

Examples

The following is a sample input card:

```
BC = VELO_NORMAL SS 10   0.0
```

This boundary condition will enforce an impenetrability constraint over side set 10 as it excludes normal velocity of the fluid relative to the mesh. This is by far the most common context for this boundary condition.

Technical Discussion

- The actual weighted residual equation that is applied to a node, *j*, on the surface in question is as follows:

where ϕ_j is the finite element trial function, n the outward-pointing normal to the surface, v the fluid velocity, v_s the velocity of the underlying mesh, and v_n is the normal velocity set by V_n (the input value).

- This constraint is a rotated strongly integrated equation so that it will replace one of the rotated components of the fluid momentum equation. This component should generally always be the normal rotated component. In two dimensions, this replacement is automatic. In three dimensions, this replacement must be specified by a *ROT* condition.

$$\int \phi_j n \cdot (v - v_s) d\Gamma = \int \phi_j v_n d\Gamma$$

- This card applies the identical constraint that is applied by the *KINEMATIC* boundary condition. The only difference is that this card replaces the normal component of the rotated *fluid momentum* equation, while the latter card replaces the normal component of the rotated (*pseudo-solid*) *mesh momentum* equation.
- In conjugate liquid/solid problems, the *VELO_NORMAL* condition is often used to enforce the impenetrability condition of the liquid/solid interface. The optional *blk_id* parameter can be used to insure that the *VELO_NORMAL* condition is correctly applied to the liquid side of the interface. *blk_id* should be set equal to the element block ID of the liquid in this case. This also applies to the *KINEMATIC* and *KINEMATIC_PETROV* boundary conditions.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GTM-004.1: Corners and Outflow Boundary Conditions in Goma, April 24, 2001, P. R. Schunk

VELO_NORMAL_LS

```
BC = VELO_NORMAL_LS SS <bc_id> 0.0 <blk_id> <float1> <float2>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition relaxes the VELO_NORMAL condition in the light phase of a level-set simulation, thereby allowing gas to escape from a confined space.

Definitions of the input parameters are as follows:

VELO_NORMAL_LS	Boundary condition designation.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<*blk_id*>	*blk_id*, an optional parameter that is the element block number in conjugate problems that identifies the material region where the *VELO_NORMAL_LS* condition will be applied (usually the liquid element block in solid/liquid conjugate problems). For external boundaries, this optional parameter can be set to unity to force the condition to be kept at a corner between two side sets (2D only). This is handy for corner conditions. Please see GTM-004.0 for details.1
<float>	L=interface half-width over which the VELO_NORMAL bc changes.
<float2>	alpha=shift in the VELO_NORMAL change relative to the LS interface. With alpha=0, VELO_NORMAL begins to be enforced when the LS interface reaches a distance L from a wall. With alpha=1, VELO_NORMAL begins to be enforced when the LS inteface reaches the wall.

Examples

The following is a sample input card:

```
BC = VELO_NORMAL_LS SS 10   0.0 {blk_id=1} 0.05 0.4.
```

Technical Discussion

The technical discussion under VELO_NORMAL largely applies here as well.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GTM-004.1: Corners and Outflow Boundary Conditions in Goma, April 24, 2001, P. R. Schunk

VELO_NORM_COLLOC

```
BC = VELO_NORM_COLLOC SS <bc_id> <float>
```

Description / Usage

(PCC/ROTATED MOMENTUM)

This boundary condition allows the user to set the outward velocity component normal to a surface. It is identical in function to the *VELO_NORMAL* boundary condition, but differs in that it is applied as a point collocated condition.

Definitions of the input parameters are as follows:

VELO_NORM_COLLOC	Boundary condition designation.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	V_n, value of normal velocity component. Note that this velocity component is relative to the motion of the underlying mesh.

Examples

Following is a sample card:

```
BC = VELO_NORM_COLLOC SS 20 0.0
```

This boundary condition will enforce an impenetrability constraint over side set 20 as it excludes normal velocity of the fluid relative to the mesh. This is by far the most common context for this boundary condition.

Technical Discussion

- The actual equation that is applied to a node, *j*, on the surface in question is as follows:

$$n \cdot (v_j - v_s) = v_n$$

where v_j is the fluid velocity at the node, *n* the outward-pointing normal to the surface, v_s the velocity of the underlying mesh at the node, and v_n is the normal velocity set by <float> above.

- This constraint is a rotated collocated equation so that it will replace one of the rotated components of the fluid momentum equation. This component should generally always be the normal rotated component. In two dimensions, this replacement is automatic. In three dimensions, this replacement must be specified by a *ROT* condition.
- As noted above this boundary condition applies exactly the same constraint as the *VELO_NORMAL* condition but via a point collocated method instead of as a strongly integrated condition. This might be advantageous at times when it is desirable to enforce a normal velocity component unambiguously at a point in the mesh.

VELO_NORMAL_DISC

```
BC = VELO_NORMAL_DISC SS <bc_id> <float>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition card balances mass loss from one phase to the gain from an adjacent phase. It is the same as the *KINEMATIC_DISC* card but is applied to the fluid momentum equation. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

Definitions of the input parameters are as follows:

VELO_NORMAL_DISC	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. It is important to note that this side set should be shared by both element blocks for internal boundaries.
<float>	Set to zero for internal interfaces; otherwise used to specify the mass average velocity across the interface for external boundaries.

Examples

Following is a sample card:

```
BC = VELO_NORMAL_DISC SS 66 0.0
```

is used at internal side set 10 (note, it is important that this side set include elements from both abutting materials) to enforce the overall conservation of mass exchange.

Technical Discussion

- This boundary condition card applies the following constraint to nodes on the side set:

$$\rho_1 n \cdot (v - v_s)\Big|_1 = \rho_2 n \cdot (v - v_s)\Big|_2$$

where 1 denotes evaluation in phase 1 and 2 denotes evaluation in phase 2. This constraint replaces only one of the momentum equations present at an internal discontinuous boundary between materials. There usually must be another momentum boundary condition applied to this side set. In addition, there must also be a distinguishing condition applied to the mesh equations if mesh motion is part of the problem.

- This boundary condition is typically applied to multicomponent two-phase flows that have rapid mass exchange between phases, rapid enough to induce a diffusion velocity at the interface, and to thermal contact resistance type problems. The best example of this is rapid evaporation of a liquid component into a gas.

References

No References.

VELO_NORMAL_EDGE

```
BC = VELO_NORMAL_EDGE SS <bc_id1> <bc_id2> <float>
```

Description / Usage

(PCC-EDGE/ROTATED MOMENTUM)

This boundary condition card is used to specify the normal velocity component on a dynamic contact line in three-dimensions. The velocity component is normal to the contact line in the plane of the web and is equal to V_n. The free-surface side set should always be <bc_id1>, the primary side set, and the web side set should be <bc_id2>, the secondary side set. Usually, this boundary condition is used to model dynamic contact lines in three dimensions and is usually found in conjunction with a *VELO_TANGENT_EDGE* card, a *VAR_CA_EDGE* or *CA_EDGE* card as explained below.

Definitions of the input parameters are as follows:

VELO_NORMAL_EDGE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This is the primary side set defining the edge and should also be associated with the capillary free surface if used in the context of a dynamic contact line.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. Together with <bc_id1>, this secondary side set defines the edge/curve on which the boundary condition applies as the intersection of the two side sets. In problems involving dynamic contact lines, this side set should correspond to the moving substrate.
<float>	V_n, a parameter supplying the imposed normal velocity component. This component is taken normal to the edge curve parallel to <bc_id2>. See below for a more detailed description.

Examples

The following is a sample input card:

```
BC = VELO_NORMAL_EDGE SS 5 4 0.0
```

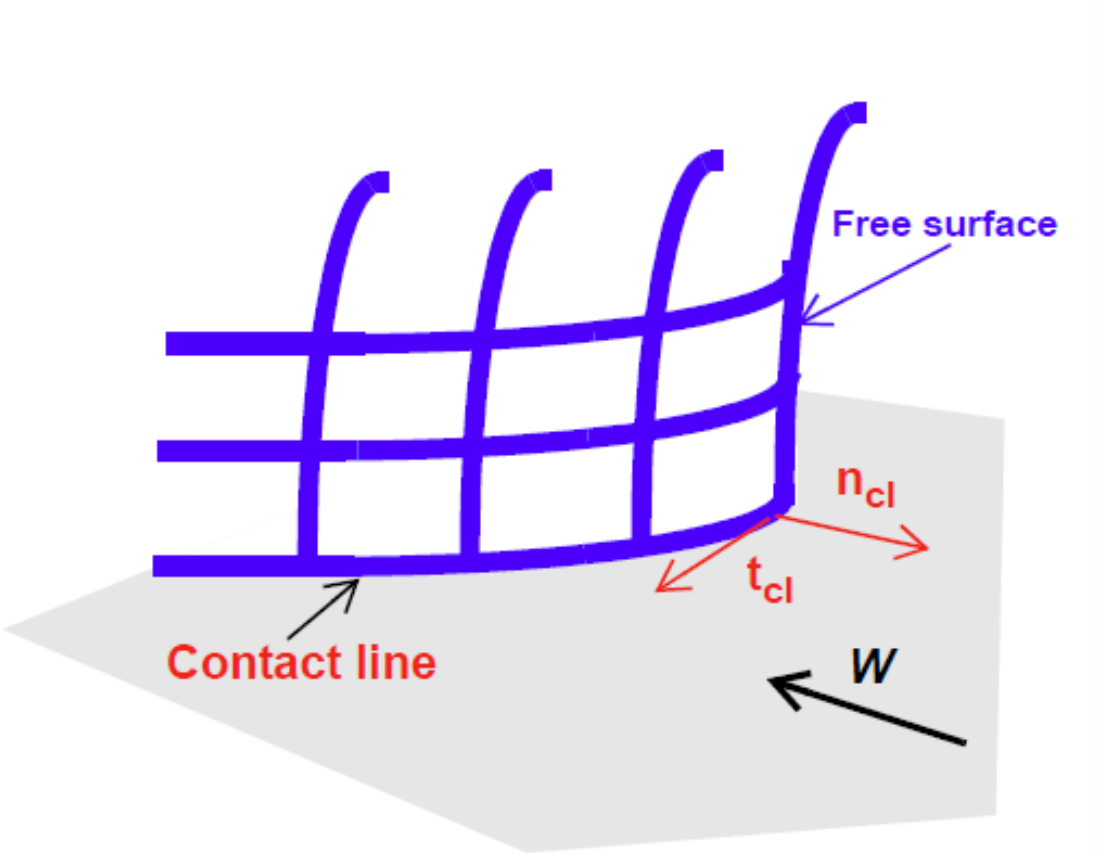
This card sets the normal-to-contact line component of the velocity to zero along the curve defined by the intersections of side set 5 and 4.

Technical Discussion

- This boundary condition imposes a point collocated constraint of the form:

$$n_{cl} \cdot (v - v_m) = V_n$$

where v is the fluid velocity, v_m is the mesh velocity and n_cl is the normal to the contact line in the plane of <bc_id2>. The sketch at right depicts the orientation of this latter vector. Note that the collocation points for this boundary condition only are not the nodes on the edge curve but integration points in each of the edge elements. The reason for this is historical and uninteresting from a user point of view.



- This boundary condition is used almost exclusive in problems involving dynamic contact lines in three dimensions. Imposition of wetting line physics is a difficult problem in modeling situations involving dynamic contact lines. In twodimensions, the assumption is often made that the effect of any wetting line force is to locally produce a condition in which the fluid velocity at the contact line is zero *in the laboratory reference frame*. That is to say, that at the contact line noslip between fluid and moving substrate is not enforced and instead a zero velocity condition

is imposed. In this way, the difficult-to-model wetting line forces are not included directly, but instead are included by their effect on the velocity. One might argue with this model, and many do, but as a practical approach, this has been shown to work well.

Generalizing this notion into three dimensions is the primary motivation for this boundary condition. In the case of a dynamic contact line that is a curve in three dimensions, it is not correct to simply set all velocity components to zero because that would imply that the wetting forces act equally in all three directions. It is more reasonable to say that the wetting forces can act only in a direction normal to the contact line in the plane of the substrate. Therefore, the correct generalization of the wetting line model described in the previous paragraph is to set the velocity component normal to the contact line in the plane of the substrate to zero. This is done by using the *VELO_NORMAL_EDGE* boundary condition with V_n set to zero. In the case of a transient problem, it is necessary to add the qualifier, "relative to the mesh motion." This accounts for the mesh motion velocity in the constraint equation. See Baer, et.al. (2000) for a more complete discussion of this wetting line model.

- Generally, a *VELO_NORMAL_EDGE* card must be accompanied by other boundary conditions for a correct application. Firstly, since *VELO_NORMAL_EDGE* forces the velocity vector to be parallel to the contact line (at least in steady state), the *KINEMATIC* condition on any free surface attached to the contact line will overspecify the problem at the contact line. For this reason, it is generally the case that a *CA_EDGE, VAR_CA_EDGE* or *VAR_CA_USER* (or their variants) should also be present for the contact line. These boundary conditions replace the *KINEMATIC* card on the mesh at the contact line.

  In addition, a *VELO_TANGENT_EDGE* card should be present to enforce no-slip between fluid and substrate *in the tangential direction*. Also it should be recognized that *VELO_NORMAL_EDGE* will not override other Dirichlet conditions on the substrate side set. Typically, the latter are used to apply no slip between fluid and substrate. If such conditions are used over the entirety of the substrate side set, both *VELO_NORMAL_EDGE* and *VELO_TANGENT_EDGE* conditions applied at the contact will be discarded.

  There are two potential solutions to this. First, the substrate region could be divided into two side sets, a narrow band of elements adjacent to the contact line and the remainder of substrate region. In the narrow band of elements, the no slip condition is replaced by a *VELO_SLIP* card with the substrate velocity as parameters. This allows the velocity field to relax over a finite region from the velocity imposed at the contact line to the substrate field. The second method uses only a single side set for the substrate region, but replaces the Dirichlet no slip boundary conditions with a penalized *VELO_SLIP* condition. That is, the slip parameter is set to a small value so that no slip is effectively enforced, but within the context of a weakly integrated condition. Since the *VELO_NORMAL_EDGE* and *VELO_TANGENT_EDGE* cards are strongly enforced on the contact lines, the *VELO_SLIP* card will be overridden in those locations and the velocity field will deviate appropriately from the substrate velocity.

References

Baer, T.A., R.A. Cairncross, P.R.Schunk, R.R. Rao, and P.A. Sackinger, "A finite element method for free surface flows of incompressible fluids in three dimensions. Part II. Dynamic wetting lines." IJNMF, 33, 405-427, (2000).

VELO_NORMAL_EDGE_INT

```
BC = VELO_NORMAL_EDGE_INT SS <bc_id1> <bc_id2> <float>
```

Description / Usage

(SIC-EDGE/ROTATED MOMENTUM)

This boundary condition card is used to specify the normal velocity component on a dynamic contact line in three-dimensions. The velocity component is normal to the contact line in the plane of the web and is equal to V_n. The free-surface side set should always be <bc_id1>, the primary side set, and the web side set should be <bc_id2>, the secondary side set. This boundary condition is identical in function to *VELO_NORMAL_EDGE*. It differs only in that is applied as a strongly integrated condition along the curve defined by <bc_id1> and <bc_id2>.

Definitions of the input parameters are as follows:

VELO_NORMAL_EDGE_INT	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) for the primary side set in the problem domain. This side set should also be the side set associated with the capillary free surface if used in the context of a dynamic contact line.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) for the secondary side set defining the edge in the problem domain. Together with <bc_id1>, this defines the curve on which the boundary condition applies as the intersection of the two side sets. In problems involving dynamic contact lines, this side set should correspond to the moving substrate.
<float>	V_n, a parameter supplying the imposed normal velocity component value. This component is taken normal to the edge curve parallel to <bc_id2>. See below for a more detailed description.

Examples

The following is a sample card:

```
BC = VELO_NORMAL_EDGE_INT SS 5 4   0.0
```

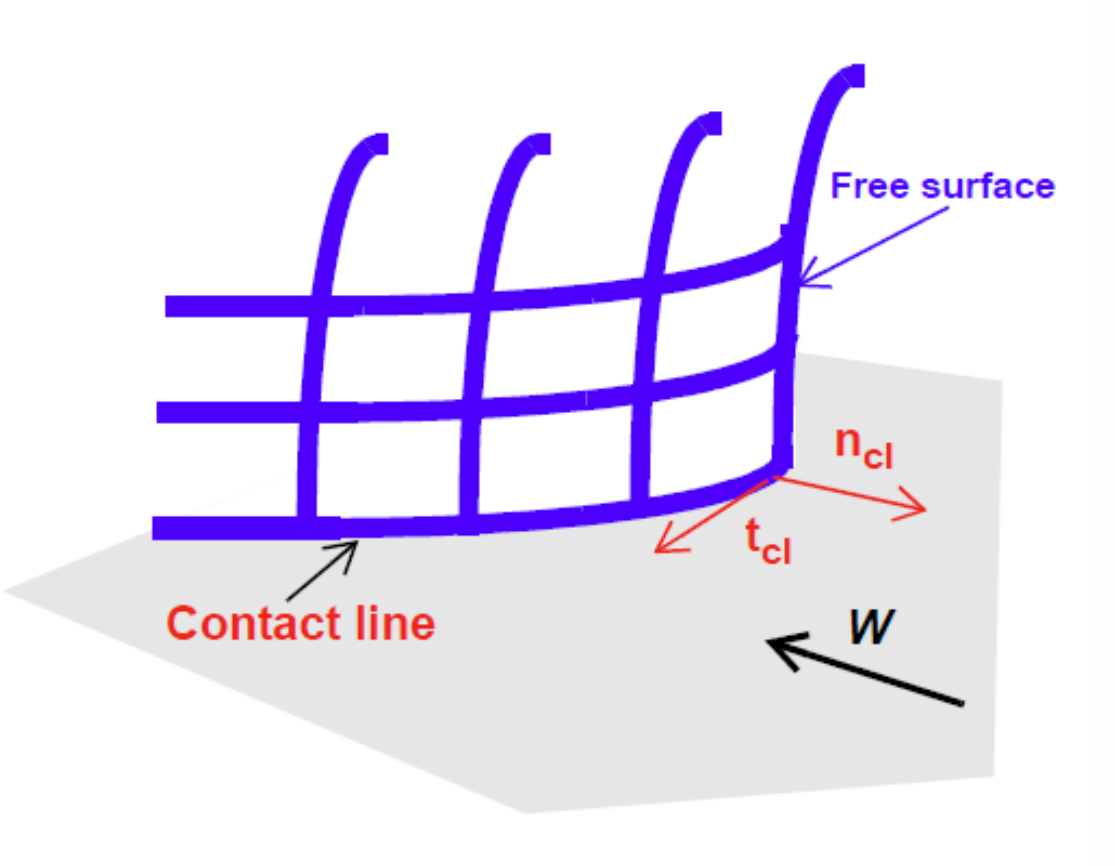
This card sets the normal-to-contact line component of the velocity to zero along the curve defined by the intersections of side set 5 and 4.

Technical Discussion

- This boundary condition imposes a strongly integrated constraint of the form:

$$\int_C \phi_i (\boldsymbol{n}_{cl} \cdot (\boldsymbol{v} - \boldsymbol{v}_{\boldsymbol{m}}) - V_n) dC = 0$$

where ϕ_i is the velocity trial function, v is the fluid velocity, v_m is the mesh velocity and $n_c l$ is the normal to the contact line in the plane of the moving substrate <bc_id2>. The sketch at right depicts the orientation of this latter vector.


Free surface
n_{cl}
t_{cl}
Contact line
W

- As noted above, this boundary condition functions nearly identically to the *VELO_NORMAL_EDGE* condition (except for its manner of application within *Goma*) and all comments appearing for the latter apply equally well for this boundary condition.

References

No References.

VELO_TANGENT

```
BC = VELO_TANGENT SS <bc_id> <integer> <float_list>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition is used to specify strongly the component of velocity tangential to the side set. An added feature is the ability to relax the condition near a point node set according to supplied length scale and slipping parameters. This has application to problems involving moving contact lines. Note that this boundary condition is applicable only to two-dimensional problems and will result in an error if it is used in a three-dimensional context.

The <float_list> has three parameters; definitions for all input parameters is as follows:

VELO_TANGENT	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	N_cl, parameter that identifies a single-node node set that coincides with the location in the model of the moving contact line. Distances in the slipping model are computed relative to the location of this node. When the slipping model is not used, this parameter can safely be set to zero. Another toggle setting can be triggered by setting this integer to -1; with this the *VELO_TANGENT* condition is kept at a rolling motion dynamic contact line. (See FAQ below on rolling motion conditions.)
<float1>	v_t, a parameter specifying the value of the tangent velocity component. The component direction is $n \times k$ where k is the z-component unit vector.
<float2>	β, a parameter specifying the coefficient for slip velocity (see model below); setting β to zero disables the slipping model.
<float3>	α, a parameter specifying the length scale for the position dependent slip (see model below); setting α to zero disables the slipping model.

Examples

The following is a sample input card:

```
BC = VELO_TANGENT SS 10 100 0.0 1.0 0.1
```

Technical Discussion

- Most often this boundary condition is used only to set the tangential speed on a side set because simpler Dirichlet conditions are not appropriate. An example is a sloping fully-developed inlet plane which does coincide with a coordinate axis. In this case, this boundary condition would be used to set the tangential velocity to be zero. The constraint applied at node *i* is as follows:

$$\int_{\Gamma} \phi_i (t \cdot v - v_t) d\Gamma = 0$$

- Alternatively, a dynamic contact line might be present in the problem and it is desirable that this condition be relaxed near the position of this contact line. This can be done by supplying non-zero values for α and β. In this case, the constraint that is applied at the i^{th} node on the boundary is:

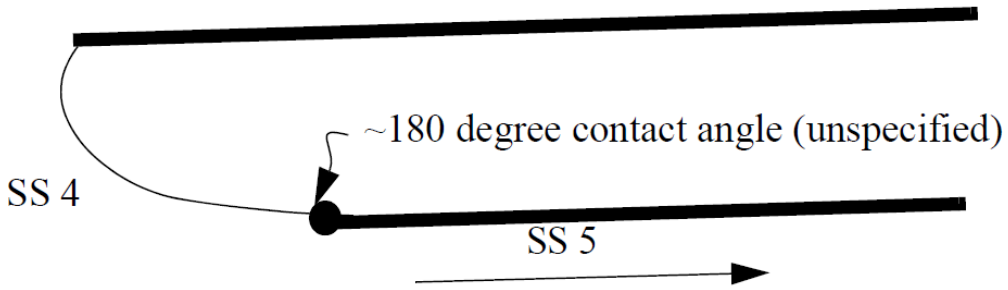
$$\int_{\Gamma} \phi_i (t \cdot v - \beta \dot{x} e^{-\alpha d} - v_t) d\Gamma = 0$$

in which *d* is the straightline distance to the node attached to <N_{cl}> and $\dot{x}$ is the velocity vector of the mesh. It should be recognized that for steady state problems the mesh motion is by definition always zero so this constraint reverts to the previous expression.

FAQs

Rolling Motion Conditions for high Capillary number dynamic wetting. Often times it is desirable to model a case of dynamic wetting for which the conditions result in a high capillary number. At this limit, it is well known that a contact angle specification is in fact an overspecification. *Goma* has always been able to model this case, except recently some changes have been made to allow for the combination of conditions at a dynamic contact line to be controlled. It should be stressed that all finite capillary number cases still work as always. This FAQ addresses the special case in which you desire to specify no-slip right up to the contact line. In most cases a *VELO_SLIP* card or outright setting the velocity components to zero at the moving contact line in order to impart slip will circumvent the issue taken up here.

The figure below diagrams the situation:



Basically the web in this example corresponds to side set 5 and the free surface to side set 4. The conditions we desire in the vicinity of the contact line are as follows:

```
    $web surface
BC = VELO_TANGENT SS 5   0   {web_sp}   0.0   0.0
BC = VELO_NORMAL SS 5   0.0
BC = GD_PARAB   SS 5   R_MESH2   0   MESH_POSITION1   0   0. 0. 1.
BC = GD_PARAB   SS 5   R_MESH2   0 MESH_POSITION2   0 0.   {2*roll_rad} 1.
$   upstream heel
BC = KINEMATIC SS 4 0.
BC = CAPILLARY   SS   4 {inv_cap}   0.0   0.0
```

Notice how there is no contact angle specified and even with the CAPILLARY card, the effect of,

VELO_NORMAL, surface tension is very small. The desired set of conditions that should be applied at the dynamic contact line are as follows:

```
    At node 1:
R_MOMENTUM1   gets VELO_NORMAL   from SS 5, CAPILLARY   from SS 4,
R_MOMENTUM2   gets VELO_TANGENT   from SS 5, CAPILLARY   from SS 4,
R_MESH1   gets KINEMATIC   from SS 4,
R_MESH2   gets GD_PARAB   from SS 5, GD_PARAB   from SS 5,
```

This clearly shows that at the contact line, which happens to be node number 1 as shown by this clip from the BCdup.txt file resulting from the run, both *VELO_NORMAL* and *VELO_TANGENT* cards are applied, which implies no-slip. This is the so-called rolling-motion case (or tank-tread on a moving surface) in which the "kinematic paradox" is no longer a paradox. That is, both the *KINEMATIC* condition on the free surface and the no-slip condition on the substrate can be satisfied without loss or gain of mass through the free surface (see Kistler and Scriven, 1983). In order to make sure that both the combination above is applied, a "-1" must be placed in the first integer input of the *VELO_TANGENT* card, vis.,

```
BC = VELO_TANGENT SS 5   -1 {web_sp}   0.0   0.0
```

This integer input slot is actually reserved for a variable slip coefficient model and is normally used to designate the nodal bc ID of the contact line. In this case of no-slip, it is not needed so we added this special control. If the following card is issued:

```
BC = VELO_TANGENT SS 5   0 {web_sp}   0.0   0.0
```

then the following combination results:

```
    At node 1:
R_MOMENTUM1 gets   VELO_NORMAL   from SS 5, CAPILLARY   from SS 4,
R_MOMENTUM2 gets   CAPILLARY   from SS 4,
R_MESH1 gets   KINEMATIC   from SS 4,
R_MESH2 gets   GD_PARAB   from SS 5, GD_PARAB   from SS 5,
```

which is desired in the case for which a contact angle and liquid slip is applied.

References

Kistler, S. F. and Scriven, L. E. 1983. Coating Flows. In Computational Analysis of Polymer Processing. Eds. J. A. Pearson and S. M. Richardson, Applied Science Publishers, London.

VELO_TANGENT_EDGE

```
BC = VELO_TANGENT_EDGE SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(PCC-EDGE/ROTATED MOMENTUM)

This boundary condition card is used to make the velocity component tangent to the contact line in the plane of the web equal to the component of web velocity (W_x, W_y, W_z) along the contact line. This constraint replaces the tangential component of the *MOMENTUM* equation along the contact line. It is used with the *VELO_NORMAL_EDGE* condition to impose a wetting line model onto dynamic contact lines in three-dimensions. The constraint is a rotated collocated condition.

Definitions of the input parameters are as follows:

VELO_TANGENT_EDGE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) of the primary side set defining the edge geometry in the problem domain. When applied to dynamic contact lines, this side set should correspond to the free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) of the secondary side set defining the edge geometry in the problem domain. The boundary condition is applied to the curve defined as the intersection of this side set with the primary side set When applied to dynamic contact lines, this side set should correspond to the substrate.
<float1>	W_x, x-component of the substrate (or web) velocity.
<float2>	W_y, y-component of the substrate (or web) velocity.
<float3>	W_z, z-component of the substrate (or web) velocity.

Examples

The following is a sample input card:

```
BC = VELO_TANGENT_EDGE SS 5 4 -1.0 0.0 0.0
```

This card imposes a tangent velocity component along the curve formed by the intersection of sidesets 5 and 4. The value of the component is the projection of the substrate velocity (-1.0, 0. ,0.) into the tangent direction. The tangent direction is along the curve itself.

Technical Discussion

- This equation imposes the following constraint as a point collocated condition at the integration points of the elements along the curve:

$$\mathbf{t}_{cl} \cdot \mathbf{v} = \mathbf{t}_{cl} \cdot \mathbf{W}$$

where t_{cl} is a vector tangent to the curve, v is the fluid velocity, and W is the (constant) velocity of the moving substrate. The reader is referred to the sketch appearing with the *VELO_NORMAL_EDGE* card for a depiction of these vectors. It is applied as a point collocated condition at the integration points of the line elements along the curve.

- As noted above this boundary condition is used in concert with the *VELO_NORMAL_EDGE* condition to impose a model of wetting line physics along a dynamic contact line in three dimensions. The reader is referred to the discussion section of this latter boundary condition for a thorough exposition of this model. Suffice it to say that this boundary condition enforces no-slip between substrate and fluid in the tangent direction to the contact line. This is an essential part of the wetting line model because it implies that the wetting line forces related to surface tension etc. do not act tangential to the wetting line. Therefore, there is no agent in this direction which could account for departures from a strictly no-slip boundary condition.
- The astute user might note that the mesh velocity doesn't appear in this expression whereas it does in the expression for *VELO_NORMAL_EDGE*. In the latter expression, the normal motion of the mesh represents the wetting velocity of the contact line normal to itself. It has a physical significance and so it make senses to connect it to the fluid velocity at that point. In the case of the tangential mesh motion velocity, it cannot be attached to any obvious physical part of the wetting model. It makes no sense that the tangential motion of nodes along the contact line should induce velocity in the fluid and vice versa. As a result, mesh motion is left out of the preceding relation.

VELO_TANGENT_EDGE_INT

```
BC = VELO_TANGENT_EDGE_INT SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(SIC-EDGE/ROTATED MOMENTUM)

This boundary condition card is used to make the velocity component tangent to the contact line in the plane of the web equal to the component of web velocity (W_x, W_y, W_z) along the contact line. It imposes the identical constraint as the *VELO_TANGENT_EDGE* card, but applies it as a strongly integrated condition rather than a point collocated condition.

Definitions of the input parameters are as follows:

VELO_TANGENT_EDGE_INT	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) of the primary side set defining the edge geometry in the problem domain. When applied to dynamic contact lines, this side set should correspond to the free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) of the secondary side set defining the edge geometry in the problem domain. The boundary condition is applied to the curve defined as the intersection of this side set with the primary side set When applied to dynamic contact lines, this side set should correspond to the substrate.
<float1>	W_x, x-component of the substrate (or web) velocity.
<float2>	W_y, y-component of the substrate (or web) velocity.
<float3>	W_z, z-component of the substrate (or web) velocity.

Examples

The following is a sample input card:

```
BC = VELO_TANGENT_EDGE_INT SS 5 4   -1.0 0.0 0.0
```

This card imposes a tangent velocity component along the curve formed by the intersection of sidesets 5 and 4. The value of the component is the projection of the substrate velocity (-1.0, 0. ,0.) into the tangent direction. The tangent direction is along the curve itself.

Technical Discussion

- This equation imposes the following constraint as a point collocated condition at the integration points of the elements along the curve:

where t_{cl} is a vector tangent to the curve, *v* is the fluid velocity, *W* is the (constant) velocity of the moving substrate, ϕ_i is the shape function each node along the curve C. This integral condition is imposed strongly at each node. The reader is referred to the sketch appearing with the *VELO_NORMAL_EDGE* card for a depiction of these vectors.

- The reader is referred to the *VELO_TANGENT_EDGE* discussion for information about the context in which this condition is applied. Because it is applied in a different fashion than the former condition, it sometimes is the case that it will allow more flexibility in situations involving many boundary conditions applied in close proximity. There may also be situations where an integrated constraint results in better matrix conditioning that a collocated constraint.

$$\int_C \phi_i(\boldsymbol{t}_{cl} \cdot \boldsymbol{v} - \boldsymbol{t}_{cl} \cdot \boldsymbol{W}) dC = 0$$

VELO_TANGENT_3D

```
BC = VELO_TANGENT_3D SS <bc_id> <float_list>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition is the three dimensional analog of the *VELO_TANGENT* condition. It is used to strongly set the tangential velocity component along a side set in a three-dimensional problem. It is not a completely general condition since it can set only a single tangential velocity component. It can only be applied to flat surfaces or surfaces which have only one radius of curvature such as a cylinder.

The <float_list> requires four values be specified; a description of the input parameters follows:

VELO_TANGENT_3D	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	v_t, the value assigned to the tangential velocity component.
<float2>	t_x, the x-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outward-pointing normal.
<float3>	t_y, the y-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outwardpointing normal.
<float4>	t_z, the z-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outwardpointing normal.

Examples

The following is an example of the card:

```
BC = VELO_TANGENT_3D SS 10   1.0   0.0 0.0 1.0
```

One could use this card to set the tangential velocity on a cylindrically shaped side set 10 provided that the cylinders axis was parallel to the z-axis. In this fashion, the tangential velocity component perpendicular to the z-axis is set to 1.0.

Technical Discussion

- The constraint applied to the velocity vector by this condition on the side set is:

where $\tilde{t}$ = n × t with the components of t supplied on the card. The advantages of introducing the normal vector is that it permits use of this card on curving surfaces provided the curvature occurs in only one direction and a single tangent vector exists that is perpendicular to both the surface normal and the direction of curvature. This of course implies that the tangential component can only be applied in the direction of the curvature.

- Such conditions are of course met by a planar surface, but also a cylindrical surface. In the latter case, the vector t should be parallel to the axis of the cylinder. One application for this condition is in three-dimensional eccentric roll coating in which the roll speed can be set using this condition. The axis vectors of both roll coaters are supplied on the card.

VELO_SLIP

```
BC = VELO_SLIP SS <bc_id> <float_list> [integer1] [float5]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition allows for slip between the fluid and a boundary using an implementation of the Navier slip relation. This relation fixes the amount of slip as a function of the applied shear stress. The scaling between stress and slip is a user parameter. This implementation also permits (in two dimensions only) variable scaling dependent upon distance from a mesh node. The latter can be used in modeling dynamic contact lines. This condition cannot currently be used on connecting surfaces.

There are four required values in <float_list> and two optional values; definitions of the input parameters are as follows:

VELO_SLIP	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	β, the slip coefficient. The inverse of β defines the scaling between stress and slip. Hence, for small values of β, large shear stresses are needed for a given amount of slip, and conversely, for large values of β, the amount of stress needed for the same degree of slip decreases (see below for a more rigorous description).
<float2>	$v_{s,x}$, the x-component of surface velocity vector. This would be the x-component of the fluid velocity if a no slip condition were applied.
<float3>	$v_{s,y}$, the y-component of surface velocity vector. This would be the y-component of the fluid velocity if a no slip condition were applied.
<float4>	$v_{s,z}$, the z-component of surface velocity vector. This would be the z-component of the fluid velocity if a no slip condition were applied.
[integer]	N_{cl}, a single-node node set identification number. When the variable coefficient slip relation is used, distance is measured relative to this node (see discussion below). Normally, this node set represents the location of the dynamic contact line. Note that this option is generally only used in two-dimensional simulations.
[float5]	α, the distance scale in the variable slip model (see the discussion below). Both N_{cl} and α should be present to activate the variable slip model.

Examples

Following is a sample card without the optional parameters:

```
BC = VELO_SLIP SS 10 0.1 0.0 0.0 0.0
```

Technical Discussion

- The general form of this boundary condition is

where τ is the deviatoric portion of the fluid stress tensor, β is the Navier slip coefficient and v_s is the velocity of the solid surface. The velocity of the surface must be specified, as described in the Description/Usage subsection above. It is a weakly integrated vector condition, as noted above, so it will be added to each of the three momentum equation components.

This last point is important to keep in mind, especially when applying this condition to boundaries that are not parallel to any of the principle axes. It is possible under these circumstances that this condition will allow motion through a boundary curve in addition to slip tangential to it. This can be avoided by including a rotated boundary condition like *VELO_NORMAL* on the same sideset. This will cause the momentum equations to be rotated to normal and tangential components and also enforce no normal flow of the material. Whatever slipping that takes place will be in the tangential direction.

$$n \cdot \tau = \frac{1}{\beta}(v - v_s)$$

- The variable slip coefficient model is quite simple: **EQUATION**, where d is the absolute distance from node N_{cl} identified on the card; the coefficients β and α are also supplied on input. This relation is protected against overflowing as d increases. This model can be used to allow slipping to occur in a region close to the node set, but at points further removed, a no slip boundary (β large) is reinstated on the sideset.

VELO_SLIP_ROT

```
BC = VELO_SLIP_ROT SS <bc_id> <float_list> [integer] [float5]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is a variant of the *VELO_SLIP* boundary condition and serves much the same function: to allow the fluid to slip relative to a solid substrate boundary. The difference is that the assumed substrate is a rotating cylindrical surface with axis parallel to the z-direction. Also as in the *VELO_SLIP* case, an optional variable slip coefficient model is available that allows for slip to occur only in a region near to a mesh node. This boundary condition is applicable generally only to two-dimensional problems or very specialized three dimensional problems.

The <float_list> has four values and there are two optional values; definitions of the input parameters are as follows:

VELO_SLIP_ROT	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	β the slip coefficient. The inverse of β defines the scaling between stress and slip. Hence, for small values of β, large shear stresses are needed for a given amount of slip, and conversely, for large values of β, the amount of stress needed for the same degree of slip decreases (see below for a more rigorous description).
<float2>	ω, rotation rate of the cylindrical substrate surface in radians/T. Positive values for this parameter correspond to rotation in the clockwise direction.
<float3>	x_c, the x-position of rotation axis.
<float4>	y_c, the y-position of rotation axis.
[integer]	N_{cl}, a single-node node set identification number. When variable coefficient slip relation is used, distance is measured relative to this node (see discussion below). For problems involving dynamic contact lines, this nodeset coincides with the location of the contact line.
[float5]	α, the distance scale in the variable slip model (see the discussion below). Both N_{cl} and α should be present to activate the variable slip model.

Examples

The following is a sample card without the optional parameters:

```
BC = VELO_SLIP_ROT SS 10 0.1 3.14 0.0 1.0
```

This condition specifies a moderate amount of slip (0.1) on a cylindrical surface rotating at 3.14 rad/sec around the point (0.0,1.0).

Technical Discussion

The comments that appear in the Technical Discussion section of the *VELO_SLIP* card apply equally well here. In particular, the discussion of the variable slip coefficient model applies here as well. The only significant difference is that the velocity of the substrate is not a fixed vector; instead, it is tangent to the cylindrical substrate with a magnitude consistent with the radius of the cylinder and the rotation rate.

References

No References.

VELO_SLIP_FILL

```
BC = VELO_SLIP_FILL SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is applied only in problems involving embedded interface tracking, that is, level set or volume of fluid. As in the case of the *VELO_SLIP* card, it allows for slip to occur between fluid and solid substrate, but in this case slipping is allowed only in a narrow region around the location of the interface where it intercepts the solid boundary. Elsewhere, this boundary condition enforces a no-slip condition between fluid and substrate.

When using the level set tracking, slip is allowed only near the intersection of the zero level set contour and the substrate boundary, and then only in a region twice the level set length scale wide centered on the zero level set. When using volume of fluid, the criterion for slipping is that the absolute value of the color function should be less than 0.25.

This boundary condition is most often used in conjunction with the *FILL_CA* boundary condition. The latter applies forces to contact lines in order to simulate wetting line motion. These forces are applied in a weak sense to the same regions near the interface so it is necessary to use *VELO_SLIP_FILL* with a large slipping coefficient so that effectively no-slip is relaxed completely near the interface.

Definitions of the input parameters are as follows:

VELO_SLIP_FILL	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	β, the slip coefficient. The inverse of β defines the scaling between stress and slip. The parameter supplied on the input deck is used only in the region define above. Elsewhere, the slip coefficient is uniformly set to 10^{-6}.
<float2>	$v_{s,x}$, the x-component of surface velocity vector. This would be the x-component of the fluid velocity if a noslip condition were applied.
<float3>	vs, y, the y-component of surface velocity vector. This would be the y-component of the fluid velocity if a noslip condition were applied.
<float4>	vs, z, the z-component of surface velocity vector. This would be the z-component of the fluid velocity if a noslip condition were applied.

Examples

Following is a sample card without the optional parameters:

```
BC = VELO_SLIP SS 10 100000.0 0.0 0.0 0.0
```

The large value of slip coefficient ensures nearly perfect slip in the region around the interface.

Technical Discussion

- See the documentation under *VELO_SLIP* boundary condition for a description of the nature of this boundary condition.
- An important caveat when using this boundary condition to relax no-slip in the vicinity of the interface is that it relaxes all constraints on the velocities in the region. This includes the constraint to keep fluid from passing through the substrate boundary. For this region, it is usually also necessary to use a impenetrability condition, *VELO_NORMAL* for example, in conjunction with this boundary condition for appropriate results.

References

No References.

VELO_SLIP_ELECTROKINETIC

```
BC = VELO_SLIP_ELECTROKINETIC SS <bc_id> <float1> <float2>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition allows for slip between the fluid and a solid boundary due to electrokinetic effects on the charged solid wall. The user provides the following parameters: zeta potential at the wall and permittivity of the fluid.

VELO_SLIP_ELECTROKINETIC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	ε, absolute permittivity of the fluid.
<float2>	ζ, the surface potential of solid boundary. It is referred to as the zeta potential.

Examples

Following is a sample card:

```
BC = VELO_SLIP_ELECTROKINETIC SS 10 1.e-5 1.e-2
```

Technical Discussion

- The general form of this boundary condition is

where ε is the absolute permittivity of the medium, ζ is the zeta potential, E_t is the electric field tangent to the solid surface, and v_s is the slip velocity.

VELO_SLIP_ELECTROKINETIC3D

```
BC = VELO_SLIP_ELECTROKINETIC3D SS <bc_id> [floatlist]
```

$$v_s = -\frac{\varepsilon \zeta E_t}{\mu}$$

Description / Usage

(SIC/ROTATED MOMENTUM)

This is a 3D generalization of the VELO_SLIP_ELECTROKINETIC boundary condition. It is similar to VELO_TANGENT_3D except the slip velocity is calculated based on Helmholtz-Smulkowski relation. This boundary condition allows for slip between the fluid and a solid boundary due to electrokinetic effects on the charged solid wall. The user provides the following parameters: zeta potential at the wall, permittivity of the fluid and.

VELO_SLIP_ELECTROKINETIC_3D	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	ε, absolute permittivity of the fluid.
<float2>	ζ, the surface potential of solid boundary. It is referred to as the zeta potential.
<float3>	t_x, the x-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outwardpointing normal.
<float4>	t_y, the y-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outwardpointing normal.
<float5>	t_z, the z-component of a unit normal vector tangent to the surface; this vector must be tangent at all points on the surface. The direction of the imposed tangential velocity component is n × t with *n* the outwardpointing normal.

Examples

Following is a sample card:

```
BC = VELO_SLIP_ELECTROKINETIC3D SS 10 1.e-5 1.e-2 0. 0. 1.
```

Technical Discussion

- The general form of this boundary condition is

$$v_s = -\frac{\varepsilon \zeta E_t}{\mu}$$

where ε is the absolute permittivity of the medium, ζ is the zeta potential, E_t is the electric field tangent to the solid surface, and v_s is the slip velocity.

VELO_TANGENT_SOLID

```
BC = VELO_TANGENT_SOLID SS <bc_id> <integer1> <integer2>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition sets the tangential fluid velocity component at a fluid/solid interface to the tangential velocity component of the solid material. The latter includes any motion of the stress-free state. This boundary condition is applicable only to twodimensional problems and is normally used in conjunction with the Total Arbitrary Lagrangian/Eulerian algorithm in Goma (See GT-005.3).

Definitions of the input parameters are as follows:

VELO_TANGENT_SOLID	The name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	The element block id defining the solid phase adjacent to <bc_id>.
<integer2>	The element block id defining the liquid phase adjacent to <bc_id>.

Examples

The following is an example of this card

```
BC = VELO_TANGENT_SOLID SS 10   2 1
```

In this case, sideset 10 is an internal sideset between two separate materials, the solid material in element block 2 and the liquid material in element block 1.

Technical Discussion

The boundary condition being applied is the strong integrated condition:

$$\underset{\sim}{t} \cdot \underset{\sim}{v}_m \Big|_{fluid} = \underset{\sim}{t} \cdot v_{sfs} \cdot \underset{\sim}{F}_m + \underset{\sim}{t} \cdot \frac{\mathrm{d}\underset{\sim}{x}_m}{\mathrm{d}t}$$

where v_m is the fluid velocity, v_{sfs} is the velocity of the solid material stress-free-state (usually solid-body translation, or rotation..see *Advected Langragian Velocity* card) including the motion of the deformed coordinates, and *t* is the vector tangent to the side set. F_m is the deformation gradient tensor and the time derivative term is the motion of the deformed state tangential to the surface in question.

This condition is advocated for use with the TALE algorithm (see GT-005.3).

VELO_SLIP_SOLID

```
BC = VELO_SLIP_SOLID SS <bc_id> <integer_list> <float1> [integer3, float2]
```

Description / Usage

(WIC/ROTATED MOMENTUM)

This boundary condition is similar in function to the *VELO_SLIP* condition in that it permits a tangential velocity in a fluid phase to be proportional to the shear stress at the boundary. This boundary condition allows for this type of slip to occur at the interface between a fluid material and a *LAGRANGIAN* or *TALE* solid material. The velocity of the solid substrate is obtained automatically from the motion of the solid material, including advection of the stress-free state. As in the case of the *VELO_SLIP* condition, this condition also permits the user to vary the slip coefficient depending upon the distance from a specified point in the mesh. The variable slip model can only be used in two-dimensional problems.

The <integer_list> has two values; the definitions of the input parameters and their significance in the boundary condition parameterization is described below:

VELO_SLIP_SOLID	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This should be an internal sideset defined at the interface between solid and liquid material blocks.
<integer1>	The element block id defining the solid material phase.
<integer2>	The element block id defining the liquid material phase.
<float1>	β, the slip coefficient. The inverse of β defines the scaling between stress and slip. Hence, for small values of β, large shear stresses are needed for a given amount of slip, and conversely, for large values of β, the amount of stress needed for the same degree of slip decreases (see below for a more rigorous description).
[integer3]	N_{cl}, a single-node node set identification number. When the variable coefficient slip relation is used, distance is measured relative to this node (see discussion below). Normally, this node set represents the location of the dynamic contact line. Note that this option is generally only used in two-dimensional simulations.
[float2]	α, the distance scale in the variable slip model (see the discussion below). Both N_{cl} and α should be present to activate the variable slip model.

Examples

The following is a sample card:

```
BC = VELO_SLIP_SOLID SS 20    2 1 0.001 0.0 4 0.01
```

This boundary condition sets the slip coefficient between solid material 2 and liquid material 1 to be 0.001 except in the vicinity of the nodeset 4 (a single node) where the variable model is used.

Technical Discussion

- The general form of this boundary condition is

where τ is the deviatoric portion of the fluid stress tensor, β is the Navier slip coefficient and v_{sfs} is the velocity of the solid surface stress-free state, with F_m the deformation gradient tensor; this motion includes any rigid solid body motion and any superimposed deformation velocity.

- It is worthwhile noting that, unlike the *VELO_SLIP* condition, this condition is actually a rotated condition. It is applied to the tangential component of the rotated momentum equations weakly. This means that the normal

$$\left(\underset{\sim}{t} \cdot \underset{\sim}{v} \Big|_{fluid} - \underset{\sim}{t} \cdot v_{sfs} \cdot \underset{\sim}{F}_m - \underset{\sim}{t} \cdot \frac{\mathrm{d}\underset{\sim}{x}_m}{\mathrm{d}t} \right) = \beta_{\underset{\sim}{n}} \cdot \underset{\sim}{t} \cdot \underset{\sim}{T} \Big|_{fluid}$$

component of the momentum equation is not affected by this boundary condition. Normally, some sort of no-penetration condition must accompany this boundary condition for this reason.

- The reader is referred to the documentation of the variable slip coefficient model to apply slip near contact lines under the *VELO_SLIP* boundary condition.

VELO_SLIP_POWER

```
BC = VELO_SLIP_POWER SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)**

This boundary condition allows for slip between the fluid and a boundary using an implementation of the Navier slip relation. This relation fixes the amount of slip as a function of the applied shear stress. The scaling between stress and slip is a user parameter.

The slip velocity is in the tangential direction $t \cdot v_{slip}$ and is raised to a power (user param m)

There is an optional tangent vector to be input to the BC, the BC expects this vector to be a unit tangent vector.

There are five required values in <float_list> and three optional values; definitions of the input parameters are as follows:

VELO_SLIP_POWER Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<float1> β, the slip coefficient. The inverse of β defines the scaling between stress and slip.

<float2> $v_{s,x}$, the x-component of surface velocity vector. This would be the x-component of the fluid velocity if a no slip condition were applied.

<float3> $v_{s,y}$, the y-component of surface velocity vector. This would be the y-component of the fluid velocity if a no slip condition were applied.

<float4> $v_{s,z}$, the z-component of surface velocity vector. This would be the z-component of the fluid velocity if a no slip condition were applied.

<float5> m, the power to raise the slip velocity to

[float6] t_x, x-component of tangent vector

[float7] t_y, y-component of tangent vector

[float8] t_z, z-component of tangent vector

Examples

Following is a sample card without the optional parameters:

```
BC = VELO_SLIP_POWER SS 10 0.1 0.0 0.0 0.0 2.0
```

Technical Discussion

Boundary condition of the form:

$$\mathbf{n} \cdot \tau = -\frac{1}{\beta} \left(\mathbf{t} \cdot (\mathbf{v} - \mathbf{v_s}) \right)^m$$

Theory

No Theory.

FAQs

No FAQs.

References

VELO_SLIP_POWER_CARD

```
BC = VELO_SLIP_POWER_CARD SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)**

This boundary condition allows for slip between the fluid and a boundary using an implementation of the Navier slip relation. This relation fixes the amount of slip as a function of the applied shear stress. The scaling between stress and slip is a user parameter.

The slip velocity is a vector and is raised to a power component-wise.

There are five required values in <float_list> and three optional values; definitions of the input parameters are as follows:

VELO_SLIP_POWER_CARD Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<float1> β, the slip coefficient. The inverse of β defines the scaling between stress and slip.

<float2> $v_{s,x}$, the x-component of surface velocity vector. This would be the x-component of the fluid velocity if a no slip condition were applied.

<float3> $v_{s,y}$, the y-component of surface velocity vector. This would be the y-component of the fluid velocity if a no slip condition were applied.

<float4> $v_{s,z}$, the z-component of surface velocity vector. This would be the z-component of the fluid velocity if a no slip condition were applied.

<float5> m, the power to raise the slip velocity to

Examples

Following is a sample card without the optional parameters:

```
BC = VELO_SLIP_POWER_CARD SS 10 0.1 0.0 0.0 0.0 2.0
```

Technical Discussion

Boundary condition of the form:

$$(\mathbf{n} \cdot \tau)_x = -\frac{1}{\beta} \left((\mathbf{v}_x - (\mathbf{v_s})_x) \right)^m$$

Theory

No Theory.

FAQs

No FAQs.

References

DISCONTINUOUS_VELO

```
BC = DISCONTINUOUS_VELO SS <bc_id> <char_string> <integer1> <integer2>
```

Description / Usage

(SIC/MOMENTUM)

This boundary condition card, used to set the normal component of mass averaged velocity at an interface, specifies that the net flux of the last component in a nondilute mixture across an internal interface, is equal to zero. The condition only applies to interphase mass, heat, and momentum transfer problems applied to nondilute material phases with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

Definitions of the input parameters are as follows:

DISCONTINOUS_VELO	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<char_string>	A character string identifiying the condition to be applied on the liquid phase relative to the gas phase. • **EVAPORATION** • **DISSOLUTION** - not currently valid. Note, this parameter replaces the boundary condition *EVAPORATION_VELO*.
<integer1>	Element block id of liquid or high density phase.
<integer2>	Element block id of gas or low density phase.

Examples

Following is a sample input card that applies this BC on the block 1 side of side set 7, the liquid side; the block 2 side is the gas side.

```
BC = DISCONTINUOUS_VELO SS 7 EVAPORATION 1 2
```

Technical Discussion

The *DISCONTINUOUS_VELO* boundary condition applies the following equation:

$$\int_{\Gamma}\left[n_s \bullet \left[\rho^{+}\left(1-\sum_{i=1}^{N-1} Y_i^{+}\right)(\boldsymbol{u}^{+}-\boldsymbol{u}_s)-\sum_{i=1}^{N-1}\boldsymbol{j}_i^{+}\right]\right]\phi_{\boldsymbol{u}^{+}}^{i}\,d\Gamma = 0$$

It specifies the diffusive flux of the last species in the mechanism, i.e., the one for which no explicit continuity equation exists, to be equal to zero. This is done via a strong integral condition applied to one side of the interface, the "+" side of the interface. This boundary condition, combined with the *KINEMATIC_SPECIES* and *KINEMATIC_DISC* boundary conditions, implies that the diffusive flux of the last species on both sides of the boundary is equal to zero.

The *DISCONTINUOUS_VELO* boundary condition requires an evaluation of the derivative of the species mass fraction at the interface. Thus, the mesh convergence properties of the algorithm are reduced to O(h). Also, discretization error must interfere with the total mass balance across a phase, since the expression for j_i^+ is substituted for in some places, the *YFLUX_SPECIES* boundary condition, but used in the *DISCONTINUOUS_VELO* boundary condition.

HYDROSTATIC_SYMM

```
BC = HYDROSTATIC_SYMM
```

Description / Usage

(WIC/VECTOR MOMENTUM)

No longer supported in GOMA. Do not use.

Examples

No Example.

Technical Discussion

No Discussion.

References

No References.

FLOW_PRESSURE

```
BC = FLOW_PRESSURE SS <bc_id> <float>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card is used to set a constant value of pressure on a boundary. Most often this condition is used to set an upstream or downstream pressure over a fully-developed inflow/outflow boundary.

Definitions of the input parameters are as follows:

FLOW_PRESSURE	Boundary condition name.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	P_{ex}, the applied pressure. Positive values imply compressive forces on the fluid, negative values imply tensile forces.

Examples

The following sample input card will impose a constant compressive pressure force on the boundary defined by sideset 23:

```
BC = FLOW_PRESSURE SS 23   5.0
```

Technical Discussion

- The actual boundary condition that is applied to the fluid is given as follows:

$$n \cdot T\Big|_{fluid} = -nP$$

where n is the outward normal vector to the boundary, T is the total fluid stress tensor, and P is the applied pressure equal to <float1> above. From this the user should be able to deduce the appropriate sign for his/her pressure value.

- This boundary condition is a weak integrated condition implying that it is added to all three components of the fluid momentum equation prior to rotation of equations or application of strongly enforced conditions or Dirichlet conditions.
- The astute user who is also well-versed in finite element formulations and terminology will recognize that this boundary condition is providing a value for the boundary condition term that appears after application of the divergence theorem to the weighted fluid momentum residual equations. Hence, imposing a value of zero for <float1> is exactly equivalent to saying nothing at all about the fluid velocity at a boundary.
- This boundary condition is found predominantly in two applications. First, setting the external pressure imposed on a free surface, and second, providing the driving force for flow by being imposed on an inflow or outflow fully-developed boundary. In this latter role, the usual procedure is to apply the *FLOW_PRESSURE* condition while strongly enforcing a zero condition on the velocity components transverse to the boundary. For boundaries parallel to one of the principle coordinate directions, Dirichlet conditions can be used to set these transverse components. For other inflow or outflow boundaries, it is suggested that the *VELO_TANGENT* and *VELO_TANGENT_3D* cards be employed instead.
- This boundary condition is very useful when working with non-Newtonian models where the inlet velocity field is apt to be complicated and hard to determine *a priori*. By imposing a pressure at the inflow with this card, the non-Newtonian inlet velocity profile will be determined implicitly. Augmenting conditions can then be used to couple the imposed pressure to the average flow rate over the boundary for an even more advanced capability.

FLOW_STRESSNOBC

```
BC = FLOW_STRESSNOBC SS <bc_id> <float> [integer]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card applies the *free* outflow boundary condition developed by Papanastasiou, et.al. (1992) on the fluid momentum with the option of setting the pressure level. It is appropriate only for outflow boundaries where it is inappropriate to use natural boundary conditions or *FLOW_PRESSURE*-class boundary conditions. It is only supported for generalized Newtonian fluid constitutive equations.

Definitions of the input parameters are as follows:

FLOW_STRESSNOBC	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	$P_{applied}$, the applied pressure.
[integer]	An optional parameter. • blank/-1 the pressure in the normal stress is replaced by $P_{applied}$. • $\neq -1$ the pressure in the solution vector is retained in the normal stress.

Examples

Following is a sample card:

```
BC = FLOW_STRESSNOBC SS 10   1.0 -1
```

Here the boundary condition is applied to sideset 10 with a constant pressure of 1.0 required.

Technical Discussion

- The finite element formulation of the fluid momentum equations generates boundary integrals of the form:

where *P* is the isotropic pressure and τ the viscous stress. Often this boundary term is left off entirely on a particular boundary with the result that a zero normal force is applied implicitly. These are referred to as imposing a "natural" boundary conditions. Alternatively, this integral might be included but with the integrand replaced by an known value of force. This is the concept behind the *FLOW_PRESSURE* and *FLOW_HYDROSTATIC* boundary conditions.

However, both types of boundary conditions imply that something is known about the stress and, by association, the velocity field on the boundary. It is often the case that outflow boundaries are present where it is difficult to provide this information. A prime example is the outflow of a fluid jet accelerating downward due to gravity. In this case, the downward velocity field is still developing at this boundary so it is problematic to specify a stress value. Other examples include imposing conditions at a "truncated" outflow where the exiting fluid is still developing.

$$\int_A \phi_i n \cdot (-P\delta + \tau) dA$$

The *FLOW_STRESSNOBC* seeks to remedy this problem. Formulationally, the boundary term as written above is included as just another term dependent upon solution degrees of freedom. This permits the pressure and velocity gradients on the boundary to float as needed so that one does not need to say anything about the stress or pressure on the boundary.

Now strictly speaking, the ellipticity of the viscous flow equations suggests that this operation should result in an ill-posed problems. Elliptic equations by their very nature require that something be said about every boundary in the problem. However, in the case of outflow boundaries it appears that this restriction can be relaxed in certain circumstances with good results. Papanastasiou, et.al., (1997), Renary (1997), Griffiths (1997) and Sani and Gresho (1994) discuss this.

- The boundary condition does permit that the pressure value be fixed while the viscous stress is allowed to float. This is done by setting the optional parameter to -1 and supplying the pressure value as $P_{applied}$. When this is done depends upon circumstance. Note that this is distinctly different from setting a normal stress component using *FLOW_PRESSURE*.
- As noted above, this boundary condition is currently implemented only for generalized Newtonian fluid models. Polymeric fluid models will not work with it.

References

Griffiths, D.F., "The 'no boundary condition' outflow boundary condition," IJNMF, 24, 393-411, (1997)

Papanastasiou, T. C., N. Malamataris, and K. Ellwood, "A New Outflow Boundary Condition", IJNMF, 14, 587-608, (1992).

Renardy, M., "Imposing 'NO' boundary conditions at outflow: Why does this work?" IJNMF, 24, 413-417, (1997).

Sani, R.L., and P.M. Gresho, "Resume and remarks on the open boundary condition minisymposium," IJNMF, 18, 983-1008, (1994).

FLOW_GRADV

```
BC = FLOW_GRADV SS <bc_id> <float> [integer]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card stipulates a vanishing normal velocity gradient on a boundary with the option of setting the pressure level.

Definitions of the input parameters are as follows:

FLOW_GRADV	Name of the boundary condition.	
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.	
<float>	$P_{applied}$, the applied pressure.	
[integer]	An optional parameter.	
	blank/-1	the pressure in the normal stress is replaced by $P_{applied}$.
	$\neq -1$	**the pressure in the solution vector is** retained in the normal stress.

Examples

The sample input card:

```
BC = FLOW_GRADV SS 15   0.0
```

sets the gradient of velocity normal to sideset 15 to zero. A pressure value of zero is used in the boundary condition.

```
BC = FLOW_GRADV SS 15   0.0   1.0
```

In the preceding example, the pressure value used is obtained from the solution itself.

Technical Discussion

- This boundary condition is related in form and formulation to the *FLOW_STRESSNOBC* boundary condition in that it includes terms for the boundary integrals that appear in the momentum equation after application of integration by parts and the divergence theorem. In this boundary condition, the following integral is included with the momentum equation:

where μ is the viscosity of a Newtonian or generalized Newtonian fluid. As in the case of the *FLOW_STRESSNOBC* condition the preceding integral appears as a function of pressure and velocity unknowns as any other term.

- The pressure term in the preceding may be replaced by a fixed, imposed pressure value. This is done by setting the optional input integer to -1 and providing the imposed value in $P_{applied}$; otherwise, the value set in $P_{applied}$ is ignored.

$$\int_A \phi_i n \cdot (-P\delta + \mu \nabla \boldsymbol{v}) dA$$

FLOW_PRESS_USER

```
BC = FLOW_PRESS_USER
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition has been deprecated in favor of the *PRESSURE_USER* boundary condition; use the latter instead.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

FLOW_HYDROSTATIC

```
BC = FLOW_HYDROSTATIC SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition allows the user to impose a pressure force that varies linearly with position over the boundary. It functions in much the same manner as the *FLOW_PRESSURE* boundary condition except that more variability is allowed in the imposed pressure. As the name implies, this boundary condition is most often used to impose hydrostatic pressure profiles in problems in which gravitational forces play a role.

The <float_list> has four values to be specified; definitions of the input parameters are as follows:

FLOW_HYDROSTATIC	Boundary condition name.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	δP_x, the pressure variation in x-direction.
<float2>	δP_y, the pressure variation in y-direction.
<float3>	δP_z, the pressure variation in z-direction.
<float4>	P_0, the pressure value at the coordinate point (0,0,0). This serves as a means of establishing a datum and it is not required that (0,0,0) lie on the sideset.

Examples

Following is a sample card:

```
BC = FLOW_HYDROSTATIC SS 15   0.0 0.0 -1.5 10.0
```

This card will impose a pressure profile on side set 15 so that the pressure decreases by 1.5 as the z coordinate increases by one unit. At the point, (0,0,0) the pressure imposed is 10.0. Note that (0,0,0) does not necessarily have to be on side set 15.

Technical Discussion

- The mathematical form of the boundary condition imposed by this card is as follows:

$$n \cdot T\big|_{fluid} = -n(x\delta P_x + y\delta P_y + z\delta P_z + P_0)$$

where *n* is the outward normal vector to the boundary, *T* is the total fluid stress tensor, and *x, y, z* are the global coordinate positions.

- Like the *FLOW_PRESSURE* conditions, this is a weakly integrated condition and the comments appearing with that card apply equally well here.
- Most often this boundary condition is used in problems in which gravity is present. Under these circumstances, the pressure profile across a fully-developed flow inlet is not constant but varies according to hydrostatic head. Hence, the *FLOW_PRESSURE* condition cannot be used to provide the inlet pressure. Instead, this card is used with the variation in the pressure being imposed according to the direction of gravity. Thus, some if not all of δP_x, δP_y, or δP_z will be functions of gravity and the fluid density.

- It is true that this variation could be determined automatically by *Goma* from its known values for density and gravitational direction. But for a variety of reasons, this may not always be the best option. Instead, the user is allowed to vary the pressure on a boundary independently of the density and gravitational forces set elsewhere in the material file. If consistency is important in the problem at hand, then the user is cautioned to be consistent.
- The input parameter P_0 as noted above serves as a datum to the relationship. In theory, it is the pressure value that would be computed at the point (0,0,0), but in reality it is chosen to impose a known pressure at some point in the domain.

References

No References.

FLOWRATE

```
BC = FLOWRATE SS <bc_id> <float> <float | char_string>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition allows the user to specify a single value for the volumetric flowrate of material across an inflow (or outflow) boundary. The pressure and velocity fields on this boundary are then computed implicitly by *Goma*.

Definitions of the input parameters are as follows:

FLOWRATE	Boundary condition name.	
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.	
<float>	*flowrate*, a parameter fixing the value of volumetric flowrate across this boundary. For two-dimensional *CARTESIAN* simulations, this value should be per unit distance in the out-of-plane coordinate. For *CYLINDRICAL* and *SWIRLING* coordinate systems, this flowrate value should include integration in the azimuthal direction.	
<float|char_string>	This parameter can either be a <float> or a <char_string>.	
	float	P_{guess}, an initial guess for the pressure on the inlet
	char_string	*read*, indicating that the initial guess for the pressure file should be read from the ASCII file identified on the *GUESS file* card.

Examples

Specifying the average velocity on the inlet to a tube of radius 1.0:

```
BC = FLOWRATE SS 10 3.1415 10.0
```

Since the radius is 1.0, the area of the surface is 3.1415 so the volumetric flowrate must be specified as shown. An initial pressure guess of 10.0 is also supplied. Note this does not specify the pressure on the boundary as the final value will generally be different than specified here.

Continuing in the flowrate by reading the last pressure value from *GUESS file*:

```
BC = FLOWRATE SS 10 3.2 read
```

Technical Discussion

- The requirement that is imposed by this boundary condition is the following integral:

$$\int_{\Gamma} (u \cdot n)\, d\Gamma = U$$

where U is the flowrate value supplied on the card. It is imposed by the addition of a Lagrange multiplier unknown on the boundary in question which will be determined as a part of the solution process. For Newtonian and generalized Newtonian models, the value of the multiplier is the inverse of the pressure value on the boundary. Thus, a boundary condition nearly identical to a *FLOW_PRESSURE* condition is applied to the sideset, but it takes as its pressure the value of the inverse of the Lagrange multiplier unknown as it is computed.

The augmenting condition capability in *Goma* is used to impose the above integral. When the boundary condition is invoked, an augmenting condition of the appropriate type is automatically created. Its associated degree of freedom is the Lagrange multiplier. During the iteration sequence, the user will see updates and residuals for this augmenting condition.

- Originally, the initial guessed value for the pressure over the side set is read from the float value specified on this card, or from the *GUESS file* (if the parameter *read* is specified on this card). However, it can also be read from an EXODUS II database file. This is the same file the rest of the solution vector is read from if the problem is being restarted from a previous computation. If a value for the augmenting condition is present in this EXODUS II file, it will be read in. This value will override the float value specified on this card. The initial guess may still be read from the ASCII *GUESS file* by specifying *read* on the *Initial Guess* card and on the *Augmenting Conditions Initial Guess* card.

References

No References.

PRESSURE_USER

```
BC = PRESSURE_USER SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition allows the user to specify an arbitrary functional form for the pressure field on a boundary via a user-defined subroutine. The boundary condition is identical in form to the *FLOW_PRESSURE* and *FLOW_HYDROSTATIC* conditions, but whereas the latter conditions have constant and linear spatial dependencies for the pressure, this boundary condition allows for any dependency, including dependencies on other degrees of freedom and time.

Definitions of the input parameters are as follows:

PRESSURE_USER	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutine so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following is a sample input card:

```
BC = PRESSURE_USER SS 10 0.0 3.0 0.5
```

Technical Discussion

- Frequently, it is desired to be able to set a pressure on a boundary that is more complicated than constant or linear; this boundary condition is used for this purpose. By modifying a function in user_bc.c (fn_dot_T_user), any functional dependence of pressure can be installed. This dependence may entail a more complicated spatial dependence, variability in time, and/or dependence on other degrees of freedom.
- An example is supplied in fn_dot_T_user that illustrates how this boundary condition can be used to set a sinusoidal-type of spatial dependence. A similar function could be used to set a temporal sinusoidal variation. The only caveat is that when inserting a function, it is very important that the sensitivities of the function with respect to position (and other degrees of freedom if they exist) be added to the array d_func. This does not apply to the time variable however.
- Like *FLOW_PRESSURE* and *FLOW_HYDROSTATIC*, this boundary condition is a weakly integrated condition. Therefore, it is additive with other weak conditions, but is superseded by strong conditions or Dirichlet conditions.

CONT_TANG_VEL

```
BC = CONT_TANG_VEL SS <bc_id>
```

Description / Usage

(SIC/MOMENTUM)

This boundary condition card enforces continuity of tangential velocity between two phases with discontinuous velocity treatment. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

Definitions of the input parameters are as follows:

CONT_TANG_VEL	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

Examples

The following is a sample card:

```
BC = CONT_TANG_VEL SS 10
```

Technical Discussion

No Discussion.

CONT_NORM_VEL

```
BC = CONT_NORM_VEL SS <bc_id>
```

Description / Usage

(SIC/MOMENTUM)

This boundary condition card is similar to the *VELO_NORM_DISC* card except that it enforces a continuous normal velocity component in a discontinuous boundary field. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

Definitions of the input parameters are as follows:

CONT_NORM_VEL	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

This boundary condition is typically applied to multicomponent two-phase flows that have rapid mass exchange between phases, rapid enough to induce a diffusion velocity at the interface, and to thermal contact resistance type problems. The best example of this is rapid evaporation of a liquid component into a gas.

Examples

The following is a sample card:

```
BC = CONT_NORM_VEL SS 10
```

Technical Discussion

No Discussion.

References

No References.

VNORM_LEAK

```
BC = VNORM_LEAK SS <bc_id> <float1> <float2>
```

Description / Usage

This boundary condition card is used to specify a normal velocity boundary condition with mass transfer on momentum equations. The flux quantity is specified on a per mass basis so the heat and mass transfer coefficients are in units of L/t.

$$\underline{n} \bullet (\underline{v} - \underline{v}_s) = \sum_i h_i (y_i - y_i^0)$$

Definitions of the input parameters are as follows:

VNORM_LEAK	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	h_i, mass transfer coefficient for bulk fluid (n+ 1^{th} species).
<float2>	y_i^0, driving force concentration in external phase.

Examples

The following is a sample input card:

```
BC = VNORM_LEAK SS 1   1. 0.
```

Technical Discussion

This card is the equivalent of *KIN_LEAK* except it is solved for the normal component of the momentum equation. Similar to *KIN_LEAK*, this flux provides an overall mass transfer balance at an interface. Please refer to the technical discussion of *KIN_LEAK* boundary card.

CAPILLARY

```
BC = CAPILLARY SS <bc_id> <float_list> [integer]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card is used to apply capillary forces (surface tension) to the momentum equation on a free-surface.

Definitions of the input parameters are as follows:

CAPILLARY	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	σ, surface tension or capillary term multiplier. **IMPORTANT NOTE: if no Surface Tension card appears in the material file, this parameter is the surface tension value used here. If Surface Tension is set in the material file, however, this float value will multiply the surface tension value from the material file prior to it's application in this boundary condition. Best practice is to set this parameter to 1.0 and set the surface tension value in the material file.**
<float2>	P_{ex}, the external applied isotropic pressure on the free surface.
<float3>	P_r, deprecated capability. Always set to zero.
[integer]	Optional integer value indicating the element block id from which to apply the boundary condition. This is used to force the capillary stresses to be applied from within a phase where the momentum equations are defined.

Examples

Following is a sample card:

```
BC = CAPILLARY SS 12   1.0 10.0 0.0
```

This card specifies that capillary forces be applied to the free surface on side set 12. If a surface tension material parameter value or model is supplied, this is the surface tension value used. If not, the surface tension value used is 1.0. An external isotropic pressure of 10.0 is applied from the surrounding environment.

Technical Discussion

- One of the primary characteristics of a free-surface is the presence of surface tension-related forces. This boundary condition permits application of such forces. The forces on the fluid at the free-surface are set via the following relation:

$$n \cdot T\big|_{fluid} = -nP_{ex} + 2H\sigma n + \nabla_s \cdot \sigma$$

where n is the outward normal to the surface, T is the fluid stress tensor, P_{ex} is the external applied pressure described above, H is the surface curvature defined as, $H = -\Delta_s \cdot n/2$, σ is the surface tension, and Δ_s is the surface divergence operator defined as $\Delta_s^f = (I - nn) \cdot \Delta f$.

- Typical usage of this boundary condition is in conjunction with a *KINEMATIC* boundary condition. The latter enforces no penetration of fluid through a free surface by deforming the mesh and this boundary condition acts on the fluid momentum equation to enforce the capillary jump condition given above.
- No end of confusion results from use of this card primarily because of overloading the surface tension parameter. To reiterate, the value for surface tension that appears on this card is the actual (constant) value of surface tension that is used *if a surface tension model has NOT been specified explicitly in the material file*. If such a model has been identified, the surface tension parameter in the *CAPILLARY* card is a *multiplier* to the surface tension. The best practice is to simply always use 1.0 for this parameter and set the surface tension in the material file.
- The optional (integer) element block ID corresponds to the material numbers given in the *Problem Description* section of the input file.

CAP_REPULSE

```
BC = CAP_REPULSE SS <bc_id> <float_list> [mat_id]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This card functions much in the same way as the *CAPILLARY* card. It imposes surface tension forces on free-surfaces. The addition to this card, however, is that in the vicinity of a specified planar boundary, an additional repulsive force is added to the surface tension force. This force is directed away from the planar surface and increases in proportion to 1/ r^2 as the free-surface approaches the planar surface. This condition can be used to contend with the difficult problem of fluid/solid contact in an approximate way. This boundary condition is only applicable to two-dimensional problems; trying to apply it in a three-dimensional problem will cause an error.

There are seven values in the <float_list>; definitions of the input parameters are as follows:

CAP_REPULSE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	σ, the surface tension value or a multiplier for the capillary effect. See the important caveat under the *CAPILLARY* card regarding the use of this parameter when a surface tension value is supplied in the material file.
<float2>	P_{ex}, the applied external pressure field on the free surface.
<float3>	P_{rep}, the coefficient on the surface repulsion term. This parameter should have units of (ML/ T^2). See below for an exact description of the surface repulsion term.
<float4>	*a*, the sensitivity with respect to x-coordinate (the *a* coefficient) of the plane surface that is repelling the free surface sideset.
<float5>	*b*, the sensitivity with respect to y-coordinate (the *b* coefficient) of the plane surface that is repelling the free surface sideset.
<float6>	*c*, the sensitivity with respect to z-coordinate (the *c* coefficient) of the plane surface that is repelling the free surface sideset.
<float7>	*d*, the constant *d* coefficient of the plane surface equation that is repelling the free surface sideset.
[mat_id]	In the case of a surface node shared by more than one material, this optional integer parameter allows the user to specify which material the condition will be applied in. This is rarely used.

Examples

Following is a sample card:

```
BC = CAP_REPULSE SS 24 1.0 0.0 0.1 1. 1. 0. 2.
```

applies a standard capillary surface tension pressure jump condition to side set 24, except as the free surface approaches the plane surface defined by a solution to the equation *x* + *y* = *-2.0*.

Technical Discussion

- This boundary condition applies the following force term to the fluid momentum equation:

$$n \cdot T\big|_{fluid} = -nP_{ex} + 2H\sigma n + \nabla_s \cdot \sigma + P_{rep}/d^2$$

which is almost identical to the force applied by the *CAPILLARY* card. The only difference is the last term on the right in which d is the normal distance from a given point on the free-surface side set and the planar surface defined by the equation:

$$ax + by + cz = -d$$

CAP_RECOIL_PRESS

```
BC = CAP_RECOIL_PRESS SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition calculates the surface recoil from an evaporating metal alloy component or water.

There are seven values in the <float_list>; definitions of the input parameters are as follows:

CAP_RECOIL_PRESS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	This float is currently disabled.
<float2>	This float is currently disabled.
<float3>	Temperature at which the metal alloy begins to boil.
<float4>	Liquidus temperature of metal alloy.
<float5>	Reference temperature.
<float6>	Conversion scale for pressure.
<float7>	Conversion scale for temperature.

Examples

The following is a sample input card:

```
BC = CAP_RECOIL_PRESS SS 1   0.0 0.0 3000.0 1623.0 0.0 1.0 1.0
```

Technical Discussion

Currently this boundary condition has coefficients for only iron and water. Several required pieces of information to use this boundary condition are not in final form, and the user can expect future changes and improvements. This boundary condition is designed for use with *Q_LASER_WELD*.

References

No References.

ELEC_TRACTION

```
BC = ELEC_TRACTION SS <bc_id> <integer> <float>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card is used to add to the momentum equation the electric, or Maxwell, stress at a free-surface. Definitions of the input parameters are as follows:

ELEC_TRACTION	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Integer value indicating the element block ID from which to apply the boundary condition.
<float>	A term-multiplier.

Since this boundary condition only adds the electric stress, it is commonly used with one of the *CAPILLARY, CAP_RECOIL_PRESS* or *CAP_REPULSE* boundary conditions, viz. the capillary stress must be added separately.

Examples

For a system consisting of an insulating liquid (element block ID 1) and an insulating, passive gas (element block ID 2) with a free-surface designated by side set 12, the following is a sample usage:

```
BC = ELEC_TRACTION SS 12 1 1.0
```

```
BC = ELEC_TRACTION SS 12 2 1.0
```

```
BC = CAPILLARY SS 12 1.0 0.0 0.0 1
```

The first and second lines adds the electric stress due to the electric field in the liquid and gas phases, respectively. The third line adds the capillary stress due to surface tension. **IMPORTANT NOTE: the optional element block ID argument to the CAPILLARY card is used to make sure that the capillary stress is added from within a phase where the momentum equations are defined. The same holds for the KINEMATIC boundary condition**.

Technical Discussion

This boundary condition adds the electric, or Maxwell, stress contribution to the traction condition. To use this boundary condition there must be a *VOLTAGE* equation present in one or both of the materials neighboring the interface, i.e., one or both of the neighboring materials must be a dielectric. The electrical permittivity of each dielectric material must be supplied via the *Electrical Conductivity* card (yes, this is a kludge) in the material property file.

In its most general form, the traction condition is written

$$\boldsymbol{n} \cdot [\mathbf{T}^{(o)} - \mathbf{T}^{(i)}] = -2H\sigma\boldsymbol{n} - \nabla\sigma$$

where **T** is the stress tensor, the superscripts (*o*) and (*i*) denote the outer and inner phases, *n* is a unit normal pointing into the outer phase, -*H* is the local mean curvature, and σ is the surface tension.

The stress tensor can be written as the sum of the mechanical stress T_m (e.g., the Newtonian stress tensor) and an electrical stress T_e, viz. **T** = T_m + T_e. The electric stress tensor provided through this boundary condition applies to incompressible, polarizable materials:

$$\mathbf{T}_e = \varepsilon \boldsymbol{E}\boldsymbol{E} - \frac{1}{2}\varepsilon \boldsymbol{E} \cdot \boldsymbol{E}\mathbf{I}$$

where ε is the electrical permittivity, **E** = – Δ *V* is the electric field and *V* is the voltage or electric potential.

In expanded form, the traction condition becomes

The *ELEC_TRACTION* boundary condition is responsible for applying either the first or second terms on the right hand side (specified through the element block ID parameter) whereas the *CAPILLARY* (or related boundary condition) is responsible for the third and fourth terms.

The term multiplier supplied by the <float> input is used in the elec_surf_stress() function (*mm_ns_bc.c*) which applies the *ELEC_TRACTION* boundary condition. It is the etm function argument. The normal term multipliers couldn't be

$$\boldsymbol{n} \cdot [\mathbf{T}_m^{(o)} - \mathbf{T}_m^{(i)}] = -\boldsymbol{n} \cdot \mathbf{T}^{(o)}{}_e + \boldsymbol{n} \cdot \mathbf{T}^{(i)}{}_e - 2H\sigma\boldsymbol{n} - \nabla\sigma$$

used because this boundary condition can be applied from within a material that doesn't have the momentum equations defined (or properly set).

CAP_ENDFORCE

```
BC = CAP_ENDFORCE NS <bc_id> <float_list>
```

Description / Usage

(SPECIAL/VECTOR MOMENTUM)

This boundary condition card adds surface tangent forces to the momentum equations at the endpoint of a free-surface. There are four values to be input for the < float_list>; definitions of the input parameters are as follows:

CAP_ENDFORCE	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	X-component of surface tangent vector at end point.
<float2>	Y-component of surface tangent vector at end point.
<float3>	Z-component of surface tangent vector at end point.
<float4>	Equilibrium surface tension value. (See Technical Discussion.)

This condition need only be applied at the intersection of outflow or inflow surfaces and the free-surface. The sign on the tangent vector depends on whether the computed tangent vector is facing inward or outward. This can be figured by t = n × k.

Examples

The following is a sample input card using several APREPRO variables:

```
BC = CAP_ENDFORCE NS 100   {sind(th2)} {-cosd(th2)} 0.0 {surf_tens}
```

Technical Discussion

- The need for this boundary condition appears out of the formulation used to apply capillary forces to surfaces. The surface divergence theorem is used to simplify the curvature term in the capillary stress jump condition. This produces integrals of the form:

$$\int_C \phi_i \sigma \boldsymbol{m} \, dC$$

where C is the bounding curve of the capillary free surface, σ is the surface tension, ϕ_i is a finite element shape function and m is a vector that is at once normal to the capillary surface and also normal to the curve C. It always points outward from the domain in question. While this is completely general for threedimensions, a surface can be reduced to a curve for two-dimensions and the divergence theorem still applies (for this boundary condition).

- This card or the *CAP_ENDFORCE_SCALAR* is used in conjunction with the *CAPILLARY* card to complete (as indicated above) the treatment of capillarity conditions. It is only required when an inflow or outflow boundary intersects a free surface.
- The *CAP_ENDFORCE* boundary condition is applied through function fapply_ST (in file *mm_ns_bc.c*). The boundary term is computed as the product of the surface tension supplied on this card (<float4>) and the value supplied on the *Surface Tension* card in the material file. When the latter card is missing, *Goma* defaults its value to 1.0.
- This card was previously called *SURFTANG* for the surface tangent component of the capillary force. Old input decks can be updated simply by changing the name of the boundary condition without changing the parameters.

SURFTANG_EDGE

```
BC = SURFTANG_EDGE SS <bc_id1> <bc_id2> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card is used to insert surface tension forces on an edge boundary defined by the primary and secondary sidesets. The direction of application of the surface tension (σ) is specified by the vector (defined by (< m_x >, < m_y >, < m_z >)). This card is the three-dimensional analog of the *CAP_ENDFORCE* card. It is often used at free-surface outflow boundaries if the outflow velocity is not set by a strong condition. This condition is an unrotated, weak integrated vector condition.

There are four values to be supplied in the <float_list>; definitions of the input parameters are as follows:

SURFTANG_EDGE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the primary boundary location (side set in EXODUS II) in the problem domain. This side set is usually attached to a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the secondary boundary location (side set in EXODUS II) in the problem domain. The boundary condition is applied on the edge defined by the intersection of this side set with the primary side set.
<float1>	m_x, the x-component of direction of application of surface tension force.
<float2>	m_y, the y-component of direction of application of surface tension force.
<float3>	m_z, the z-component of direction of application of surface tension force.
<float4>	a factor multiplying the surface tension value read from the material file when evaluating the surface integral imposed by this boundary condition.

Examples

The following is a sample input card:

```
BC = SURFTANG_EDGE SS 80 60   0. -1. 0. 1.
```

Technical Discussion

- The need for this boundary condition appears out of the formulation used to apply capillary forces to surfaces in three-dimensions. The surface divergence theorem is used to simplify the curvature term in the capillary stress jump condition. This produces integrals of the form:

where *C* is the bounding curve of the capillary free surface, σ is the surface tension, ϕ_i is a finite element shape function and **m** is a vector that is at once normal to the capillary surface and also normal to the curve *C*. It always points outward from the domain in question.

Most often this boundary condition appears at outflow boundaries of free-surfaces. It is applied along the edge where the free-surface intercepts the outflow plane. In this case, the **m** vector is normal to the outflow plane. If the outflow velocity is not strongly set by a Dirichlet condition or other strongly enforced condition, this boundary condition needs to be present so that a proper inclusion of all relevant surface tension terms is performed.

- The <factor> parameter is provided to allow the user to independently vary the surface tension value associated with this term alone. The value for σ used in the preceding expression is the surface tension value obtained from the model specified in the material file multiplied by the value of <float>. Reasons for doing this are somewhat obscure but important to the practitioners of this art.

$$\int_C \phi_i \sigma \boldsymbol{m} \, dC$$

CAP_ENDFORCE_SCALAR

```
BC = CAP_ENDFORCE_SCALAR NS <bc_id> <float>
```

Description / Usage

(SPECIAL/VECTOR MOMENTUM)

This boundary condition card is very similar to the *CAP_ENDFORCE* card. It adds surface tangent forces to the momentum equations at the endpoint of a free-surface, but does not require specification of the surface tangent vector. The current free-surface tangent vector is used as the surface tangent vector. Definitions of the input parameters are as follows:

CAP_ENDFORCE_SCALAR	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float>	Equilibrium surface tension value. (See Technical Discussion.)

This condition need only be applied at the intersection of outflow or inflow surfaces and the free-surface.

Examples

The following is a sample input card:

```
BC = CAP_ENDFORCE_SCALAR NS 100   60.0
```

Technical Discussion

- The need for this boundary condition appears out of the formulation used to apply capillary forces to surfaces. The surface divergence theorem is used to simplify the curvature term in the capillary stress jump condition. This produces integrals of the form:

$$\int_C \phi_i \sigma \boldsymbol{m} \, dC$$

where C is the bounding curve of the capillary free surface, σ is the surface tension, ϕ_i is a finite element shape function and **m** is a vector that is at once normal to the capillary surface and also normal to the curve C. It always points outward from the domain in question. While this is completely general for threedimensions, a surface can be reduced to a curve for two-dimensions and the divergence theorem still applies (for this boundary condition).

- This card or the *CAP_ENDFORCE* is used in conjunction with the *CAPILLARY* card to complete (as indicated above) the treatment of capillarity conditions. It is only required when an inflow or outflow boundary intersects a free surface.
- The *CAP_ENDFORCE_SCALAR* boundary condition is applied through function fapply_ST_scalar (in file *mm_ns_bc.c*). The boundary term is computed as the product of the surface tension supplied on this card (<float>) and the value supplied on the *Surface Tension* card in the material file. When the latter card is missing, *Goma* defaults its value to 1.0.
- This card was previously called *SURFTANG_SCALAR* for the surface tangent component of the capillary force. Old input decks can be updated simply by changing the name of the boundary condition without changing the parameters.

SURFTANG_SCALAR_EDGE

```
BC = SURFTANG_SCALAR_EDGE SS <bc_id1> <bc_id2> <float>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

Like the *SURFTANG_EDGE* card, this boundary condition card is used to insert surface tension forces on an outflow edge boundary defined by the primary and secondary sidesets. In contrast to the *SURFTANG_EDGE* card, the direction of application of the surface tension (σ) is predetermined automatically as the binormal along the edge with respect to the outward facing normal of the primary sideset. This condition is also an unrotated, weak integrated vector condition. It should be used only in three-dimensional applications.

Definitions of the input parameters are as follows:

SURFTANG_EDGE_SCALAR	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database. Since it is an EDGE condition, it applies to a curve defined as the intersection of the primary and secondary sideset.
<bc_id1>	The boundary flag identifier, an integer associated with <bc_type> that identifies the primary boundary location (side set in EXODUS II) in the problem domain. This side set is used in defining the edge and the local vector basis (normal, tangent, binormal) and is usually attached to a free surface.
<bc_id2>	The boundary flag identifier, an integer associated with <bc_type> that identifies the secondary boundary location (side set in EXODUS II) in the problem domain. It is used in defining the edge and the local vector basis (normal, tangent, binormal).The boundary condition is applied on the edge defined by the intersection of this side set with the primary side set.
<float>	A factor multiplying the surface tension value read from the material file when evaluating the surface integral imposed by this boundary condition.

Examples

The following is a sample input card:

```
BC = SURFTANG_EDGE_SCALAR SS 5 10 1.0
```

applies the boundary integral (see the Technical Discussion) along the curve described by the intersection of side sets 5 and 10. The value for surface tension in the material file is used unmodified since the multiplying factor is 1.0.

Technical Discussion

- The need for this boundary condition appears out of the formulation used to apply capillary forces to surfaces in three-dimensions. The surface divergence theorem is used to simplify the curvature term in the capillary stress jump condition. This produces integrals of the form:

where *C* is the bounding curve of the capillary free surface, σ is the surface tension, ϕ_i is a finite element shape function and **m** is the outward binormal vector to the curve *C* with respect to the normal of the primary side set.

Most often this boundary condition appears at outflow boundaries of free surfaces. It is applied along the edge where the free surface intercepts the outflow plane. If the outflow velocity is not strongly set by a Dirichlet condition or other

$$\int_C \phi_i \sigma \boldsymbol{m} \, dC$$

strongly enforced condition, this boundary condition needs to be present so that a proper inclusion of all relevant surface tension terms is performed.

- The <factor> parameter is provided to allow the user to independently vary the surface tension value associated with this term alone. The value for σ used in the preceding expression is the surface tension value obtained from the model specified in the material file multiplied by the value of <float>. Reasons for doing this are somewhat obscure but important to the practitioners of this art.

FILL_CA

```
BC = FILL_CA SS <bc_id> <float>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to impose a contact angle on a boundary when using *Level Set Interface Tracking*.

A description of the input parameters follows:

FILL_CA	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	θ, the contact angle imposed, in degrees.

Examples

An example:

```
BC = FILL_CA SS 10 30.0
```

Technical Discussion

This boundary condition must be used in conjunction with the *VELO_SLIP_FILL* boundary condition. This latter condition permits the fluid to slip in the vicinity of the contact line. The *FILL_CA* acts by imposing a force on the momentum equation. The size of this force is more or less in proportion between the actual contact angle on the boundary and the value specified on the card. This force is applied as a weakly integrated condition and if the *VELO_SLIP_FILL* condition is not present, the *FILL_CA* will be overwritten and *ipso facto* absent.

References

No References.

MOVING_CA

```
BC = MOVING_CA NS <bc_id> <float_list>
```

Description / Usage

(PCC/ROTATED MOMENTUM)

The intent of this boundary condition is to apply a contact angle at wetting in a twodimensional flow that is a function of the rate of advance or recession of the contact line over the substrate. It is experimental, untested, and unsupported; use it at your own risk.

There are ten values that must be specified in the <float_list>; definitions of the input parameters are as follows:

MOVING_CA	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain. A *KINEMATIC* free surface must terminate at this single-node node set.
<float1>	θ_{stc}, the static contact angle, in degrees.
<float2>	n_x, the x-component of solid surface normal vector.
<float3>	n_y, the y-component of solid surface normal vector.
<float4>	n_z, the z-component of solid surface normal vector.
<float5>	θ_{adv}, the advancing contact angle, in degrees.
<float6>	θ_{rec}, the receding contact angle, in degrees.
<float7>	α, the scale-factor, see below.
<float8>	v_{wx}, the x-component of wall velocity.
<float9>	v_{wy}, the y-component of wall velocity.
<float10>	v_{wz}, the z-component of wall velocity.

Examples

The following is a sample input card:

```
BC = MOVING_CA NS 100 90.0 0. 1. 0. 135.0 45.0 1.0 -1. 0. 0.
```

Technical Discussion

- This boundary condition applies a point collocated constraint on the angle between the solid surface vector and the free-surface normal of the form:

$$n \cdot n_{fs} = \cos(\theta)$$

where n is the solid surface vector specified on the card and n_{fs} is the free-surface normal computed automatically by *Goma*. The contact angle is variable depending upon the relative velocity of the mesh speed, $\dot{x}$, and the substrate speed, v_w specified on the card float_list:

$$\theta = \theta_{stc} + (\theta_{adv} - \theta_{stc}) \tanh(\alpha(\dot{x} - v_w) \cdot n_{fs})$$

- This constraint on the moving contact angle replaces a rotated component of the momentum equation. In effect a wetting force is applied at the contact line whose magnitude depends on the discrepancy between actual contact angle and that computed by the above expressions. Note that other contact angle constraints are applied to rotated components of the mesh equation. A real question exists whether such a formulation is consistent with a *KINEMATIC* boundary condition also applied to this node.
- Not also that since this boundary condition is applied to the momentum equation, care must be taken to relax any Dirichlet on the substrate velocity. Otherwise, this latter constraint will override this constraint.
- Users are again cautioned that this boundary condition is untested and potentially inconsistent. It may not work.

SDC_STEFANFLOW

```
BC = SDC_STEFANFLOW SS <bc_id> <integer> {char_string}
```

Description / Usage

(SIC/MOMENTUM)

This boundary condition represents the specification of the normal component of the interfacial velocity on one side of the interface. These are *DVI_SIDTIE_VD* boundary conditions (Moffat, 2001) that have an additional property. The first time encountered in the formation of the residual, the results of a sub calculation are stored either at the node structure level or at the surface Gauss point level. The surface reaction and surface species are specified as part of a surface domain within Chemkin.

The *SURFDOMAINCHEMKIN_STEFAN_FLOW* (shortened to *SDC_STEFANFLOW* in the *name2* member of the *BC_descriptions* struct in mm_names.h) boundary condition solves the following equation representing Stefan flow at a boundary.

$$n_l \bullet [\rho^l (\boldsymbol{u}^l - \boldsymbol{u}_s)] = \sum_{k=1}^{N} -W_k S_k^l$$

where n_l is the outward facing normal to the liquid material, p^l is the liquid density, u^l is the (mass average) velocity at the current surface quadrature point, and u_s the velocity of the mesh (i.e., the interface if the mesh is fixed at the interface). The summation over *N* species is for the product of molecular weight (W_k) and the source term for creation of species k in the liquid (S_k^l). Note, while it may seem that one side of the interface is getting special treatment, the combination of this boundary condition with the KINEMATIC_CHEM boundary condition actually creates a symmetric treatment of the boundary condition. *SDC_STEFANFLOW* is linked to the SDC_SPECIES_RXN boundary conditions just as the KINEMATIC_CHEM boundary conditions are by the expression for the interface reaction. The sum is over all of the interfacial source terms for species in the phase.

Definitions of the input parameters are as follows:

SDC_STEFANFLOW	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Element Block ID of the phase on whose side of the interface this boundary condition will be applied.
char_string	S_k^l, string indicating where the surface source term information for this boundary condition will be obtained. Three options exist: • **IS_EQUIL_PSEUDORXN** • **VL_EQUIL_PSEUDORXN** • **SDC_SURFRXN** These are boundary conditions that apply to the *Species Equations*. The last boundary condition is not yet implemented, so **SDC_SURFRXN** currently does nothing.

Examples

The following is a sample input card:

```
BC = SDC_STEFANFLOW SS 1    0 VL_EQUIL_PSEUDORXN
```

The above card will create a strongly integrated boundary condition specifying the normal component of the velocity on side set 1 on the element block 0 side of the interface. The source term to be used will be taken from multiple previously specified *VL_EQUIL_PSEUDORXN* cards.

Technical Discussion

- Currently, this card has only been tested on internal interfaces containing discontinuous interfaces using the *VL_EQUIL_PSEUDORXN* source term. The *SDC_SURFRXN* boundary condition has not been implemented yet.
- The *DVI_SIDTIE_VD* variable is a nomenclature adopted by Moffat (2001) in his development of a revised discontinuous variable implementation for *Goma*. It pertains to Discontinuous Variable Interfaces (**DVI**) and the strongly integrated Dirichlet (**SID**) boundary conditions prescribing the discontinuous value of variables on either side of an interface (**TIE** boundary conditions). The user is referred to Moffat (2001) for detailed presentation on discontinuous variables.

References

GTM-015.1: Implementation Plan for Upgrading Boundary Conditions at Discontinuous-Variable Interfaces, January 8, 2001, H. K. Moffat

FLUID_SOLID

```
BC = FLUID_SOLID SS <bc_id> <integer1> <integer2> [float]
```

Description / Usage

(PCC/VECTOR MOMENTUM)

Used for fluid-structure interaction problems, the *FLUID_SOLID* condition equates the normal traction (the tangential and normal force components, per unit area) between adjacent fluid and solid materials. This condition is only to be used on boundaries between regions of *ARBITRARY* mesh motion with fluid momentum equations and of *LAGRANGIAN* or *DYNAMIC_LAGRANGIAN* mesh motion, with solid momentum equations (or mesh equations); see *Mesh Motion* and *EQ* cards. With this boundary condition, the local residual and Jacobian contributions from the fluid mechanics momentum equations (on the *ARBITRARY* side of the boundary) are added into weak form of the residual and Jacobian entries for the solid mechanics equations (on the solid *LAGRANGIAN* side of the boundary). All elements on both sides of the interface must have the same element type, i.e., the same order of interpolation and basis functions, e.g., Q1 or Q2. Also, such interfaces must include element sides from both sides of the interface in the defining side set.

Definitions of the input parameters are as follows:

FLUID_SOLID	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid phase (of *LAGRANGIAN* motion type) from the EXODUS II database.
<integer2>	Element block ID of liquid phase from the EXODUS II database.
[float]	Scale factor for stress balance for non-dimensionalization. This parameter, which multiplies the solid phase contribution, is optional; the default is 1.0.

Examples

The following is a sample input card:

```
BC = FLUID_SOLID SS 5   2   1
```

In this example, side set 5 is a boundary between a solid blade and a liquid; material 2 is the rubber blade, and material 1 is the fluid. Along that blade, a companion boundary condition of the form

```
BC = NO_SLIP SS 5   2 1
```

should also be applied.

Technical Discussion

The functional form of the boundary condition is:

$$\underset{\sim}{n} \cdot \underset{\sim}{T} = (\underset{\sim}{n} \cdot \underset{\sim}{\sigma})\lambda$$

where $\underline{T}$ is the fluid phase stress tensor given by any one of the specified fluid-phase constitutive equations, $\underline{\sigma}$ and is the solid-phase stress tensor, also given by any one of the solid-phase constitutive equation (see material file specifications). λ is a scaling factor that defaults to unity (and is usually best taken as such unless some scaling is invoked).

This balance is applied to the weak form of the solid-phase momentum residuals, from the fluid phase, viz. in the fluid-phase, the fluid-stress at the interface is added to the solid-phase momentum residuals. As mentioned above, this condition usually needs to be supplemented by a statement of mass conservation across the interface, which will depend on whether the solid phase is of *CONTINUOUS* or *POROUS* media (see *Media Type* card).

FAQs

Troubleshooting 1: This boundary condition requires that the side set contain elements from both the fluid and the solid side of the interface. For the FASTQ tool, this is the default case; for CUBIT and possibly other related tools, this can be forced on the side set definition options. Interestingly, the boundary condition does work if the side set is attached to the fluid phase only, but just due to the way in which it is applied.

Troubleshooting 2: This boundary condition does not enforce mass conservation. A combination of *NO_SLIP* or *VELO_NORMAL/VELO_TANGENT* must be invoked to achieve a material surface. For the latter, care must be taken to maintain the application of the *VELO_NORMAL* condition after a remesh. This condition is applied only to one side of the interface and depends on the ss_to_blks connectivity structure; it may be necessary to force its application, especially after remeshes. To be sure that the proper set of conditions is being applied, look at the BC_dup.txt file for nodes along the interface.

References

GT-003.1: Roll coating templates and tutorial for GOMA and SEAMS, February 29, 2000, P. R. Schunk and Matt Stay

GT-006.3: Slot and Roll coating with remeshing templates and tutorial for GOMA and CUBIT/MAPVAR, August 3, 1999, R. R. Lober and P. R. Schunk

FLUID_SOLID_RS

```
BC = FLUID_SOLID_RS SS <bc_id> <integer1> <integer2> [float]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

Please see *SOLID_FLUID_RS*. This boundary condition has not yet been implemented.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

DARCY_CONTINUOUS

```
BC = DARCY_CONTINOUS SS <bc_id> <integer1> <integer2> [float1]
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This condition enforces continuity of mass flux at an interface between a continuous medium and a saturated or partially saturated porous medium. In other words, *DARCY_CONTINUOUS* is a boundary condition that equates the velocity component in the liquid phase normal to the interface with the Darcy velocity in the porous phase, normal to the same interface, with proper accounting for conservation of mass using the liquid phase densities in the material files.

DARCY_CONTINUOUS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of porous phase from the EXODUS II database.
<integer2>	Element block ID of continuous fluid phase from the EXODUS II database.
[float1]	An optional floating point that is used for level-set free surface problems. This floating point represents a length scale over which "contact" of a liquid free surface represented by a level set field and a porous medium. It should be set to some small integer multiple of the smallest element size along the boundary. Note that this length scale is only required in cases where "sharp" interfaces using subelement integration are used. It is not required for diffuse interface representations.

Examples

The boundary condition

```
BC = DARCY_CONTINUOUS SS 5 2 1
```

applies to the interface defined by side set 5 which joins EXODUS II block 2 (porous phase) and block 1 (continuous phase).

Technical Discussion

The *DARCY_CONTINUOUS* boundary condition imposes the following requirement at the interface between a continuous medium and a saturated or partially saturated porous medium:

$$\underset{\sim}{n} \cdot \underset{\sim}{q} = \rho_l \underset{\sim}{n} \cdot \left(\underset{\sim}{v} - \underset{\sim}{v}_s \right)$$

where $\underline{n}$ is the outward-pointing normal to the surface, $\underline{q}$ is the Darcy flux, p_l is the liquid density, presumed to be the same in the adjacent phases, $\underline{v}$ is the fluid velocity and $\underline{v}_s$ is the mesh velocity.

Typically this boundary condition is applied between two blocks, one being of a *LAGRANGIAN* mesh motion type (see *Mesh Motion* card) and the other being of an *ARBITRARY* mesh motion type. Within the *LAGRANGIAN* material the *Media Type* card is set to *POROUS_SATURATED, POROUS_UNSATURATED*, or *POROUS_TWO_PHASE*. The other block is of type *CONTINOUS*.

Refer to the citations below where this boundary condition is discussed in more detail.

FAQs

Important troubleshooting note: Density, as specified in the material files for the continuous and porous phase, MUST be the same for this boundary condition to make sense.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

GT-028.0: Liquid Drop Impact on a Porous Substrate: a level-set tutorial, August 15, 2005.

VN_POROUS

```
BC = VN_POROUS SS <bc_id> <integer_list> <float>
```

Description / Usage

(SIC/ROTATED MOMENTUM)

This boundary condition is used to calculate the normal component of gas phase velocity at the interface between a continuous gas phase and porous phase. The condition is basically the unsaturated equivalent to *DARCY_CONTINUOUS*, and hence is a condition on the normal component of fluid (gas) velocity on the continuous side of the interface (see below). The flux on the porous medium side includes Darcy flux and Fickian diffusive flux in the porous phase. The vapor flux into gas is used to determine gas velocity. The condition is similar to the solid-liquid interface conditions that apply to interfaces between a porous medium and an external gas (in which the energy equation is used to solve for solvent concentration in the gas phase). This boundary condition is still under development and has not been recently tested. Its last use was for evaporation from a porous unsaturated film in a sol-gel application (see references below).

There are three values to be supplied for the <integer_list>; definitions of the input parameters are as follows:

VN_POROUS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid, porous phase from the EXODUS II database.
<integer2>	Element block ID of gas phase from the EXODUS II database.
<integer3>	**Set to zero for now**, indicating that this condition pertains to the primary liquid solvent vapor. At some point this will be generalized to include all vapor components.
<float>	Density of pure solvent vapor.

Examples

The following is a sample input card:

```
BC = VN_POROUS SS 5   1 2 0 1.e-3
```

This condition applies to internal side set 5, which defines the interface between element block 1 (the solid porous phase which has *Media Type* of *POROUS_PART_SAT* or *POROUS_TWO_PHASE*) and element block 2 (the fluid phase which has Media Type *CONTINUOUS*). It is based on the flux of liquid solvent in the porous phase (denoted by the integer 0), the vapor form of which has a density of 1.e-3. The condition results in a blowing or sucking velocity at the interface in the fluid (gas) continuous phase.

Technical Discussion

The functional form of this boundary condition is

$$n \cdot (v_g \rho_{g_i} + v_l \rho_{l_i} + J_{g_i} + J_{l_i}) = \underset{\sim}{n} \cdot (\underset{\sim}{v} - \underset{\sim}{v}_m)$$

Here, the left hand side is the total flux of liquid solvent, in both gas and liquid phases. The first two terms are the Darcy pressure driven contributions, and the second two terms are the Fickian flux contributions.

This condition would be useful for predicting the gas-flow pattern above a drying porous matrix, in which the vapor flux being driven out of the porous skeleton were a mass source to drive flow in the surrounding gas. The condition has not been tested since 1995.

References

GTM-028.0: Modeling Drying of Dip-Coated Films with Strongly-Coupled Gas Phase Natural Convection, R. A. Cairncross, 1999.

CAPILLARY_SHEAR_VISC

```
BC = CAPILLARY_SHEAR_VISC SS <bc_id> <float_list> [integer]
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition card is used to apply additional capillary forces beyond surface tension and surface tension gradients (as applied with use of the CAPILLARY BC) to the momentum equation on a free-surface. These additional forces are caused by surface deformation (surface expansion/contraction/shear) in the presence of surface-active species. Microstructural layers of surfactants in a capillary free surface can lead to significant dissipation of mechanical energy due to an effective surface viscosity. These additional properties are specified as inputs to this boundary condition.

Definitions of the input parameters are as follows:

CAPILLARY_SHEAR_VISC	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	μ_s, surface shear viscosity.
<float2>	κ_s , surface extensional/dilatational viscosity.
[integer]	Optional integer value indicating the element block id from which to apply the boundary condition. This is used to force the capillary stresses to be applied from within a phase where the momentum equations are defined.

Examples

Following is a sample card:

```
BC = CAPILLARY SS 12   1.0 10.0 0.0
```

```
BC = CAPILLARY_SHEAR_VISC SS 12   0.001 0.01
```

These cards specifies that capillary forces be applied to the free surface on side set 12. If a surface tension material parameter value or model is supplied, this is the surface tension value used. If not, the surface tension value used is 1.0. An external isotropic pressure of 10.0 is applied from the surrounding environment. The second card adds a surface viscosity effect. Note that you must solve the shell equation *EQ = n_dot_curl_v* to pick up this term.

Technical Discussion

The functional form of this boundary condition is

$$n \cdot T\big|_{fluid} = -nP_{ex} + 2H\sigma n + \nabla_s \cdot \sigma$$

where n is the outward normal to the surface, T is the fluid stress tensor, ¶$_{ex}$ is the external applied pressure described above, H is the surface curvature defined as, H = – $\Delta_s \cdot$ n⁄2, σ is the surface tension, and Δ_s is the surface divergence operator defined as Δ_s f = (I – nn) $\cdot \Delta$ f .

The Boussinesq-Scriven surface rheological constitutive equation is as follows:

$$-(\underline{n} \cdot \underline{\underline{T}}) = F^w + 2H\sigma\underline{n} + \nabla_s\sigma + (k^s + \mu^s)\nabla_s(\nabla_s \cdot \underline{v}) + 2\mu^s\underline{n}(\underline{\underline{b}} - 2H\underline{\underline{I}}_s) \bullet \nabla_s\underline{v} + 2\mu^s\underline{n}(k^s + \mu^s)\nabla_s \cdot \underline{v}$$
$$+ \mu^s\{\underline{n} \times \nabla_s([\nabla_s \times \underline{v}] \cdot \underline{n}) - 2(\underline{\underline{b}} - 2H\underline{\underline{I}}_s) \cdot (\nabla_s\underline{v}) \cdot \underline{n}\}$$

Here, Δ_s = ($\underline{\underline{I}} - \underline{n}\underline{n}$) $\cdot \Delta$ is the surface gradient operator, I_s = ($\underline{\underline{I}}$ - ($\underline{n}\underline{n}$) unit tensor. μ_s and κ_s is the surface are the surface shear viscosity and surface dilatational viscosity, respectively. The terms beyond the first three on the right are added by this boundary condition card. Note that the first three terms on the right are balance of the stress in the standard goma CAPILLARY condition, with surface tension gradients being accommodated through variable surface tension. The boundary condition CAPILLARY_SHEAR_VISC is used to set the additional terms of this constitutive equation. *As of January 2006 only the 7th term on the right hand side is implemented, as it is the only nonzero term in a flat surface*

shear viscometer. The building blocks for the other terms are available through additional shell equations (specifically you must solve *EQ = n_dot_curl_v* equation on the same shell surface). . These remaining terms actually represent additional dissipation caused by surface active species microstructures flowing in the surface. The best source of discussion of this equation is a book by Edwards et al. (1991. *Interfacial Transport Processes and Rheology*. Butterworth-Heinemann, Boston).

VELO_THETA_COX

```
BC = VELO_THETA_COX NS <bc_id> <float_list> [integer]
```

Description / Usage

(PCC/R_MOM_TANG1)

This boundary condition card applies a dynamic contact angle dependent velocity condition in a similiar fashion to VELO_THETA_TPL, but the functional form of the velocity is different. The functional form stems from the hydrodynamic theory of wetting by Cox.

The <float_list> for this boundary condition has eight values; definitions of the input parameters are as follows:

VELO_THETA_COX	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	θ, equilibrium (static) contact angle subtended by wall normal and free surface normal, in units of degrees.
<float2>	n_x , x-component of normal vector to the geometry boundary (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float3>	n_y , y-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float4>	n_z , z-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float5>	ε_s is the dimensionless slip length, i.e. the ratio of the slip length to the characteristic length scale of the macroscopic flow.
<float6>	σ is the surface tension. This value is multiplied by the surface tension value stipulated by the surface tension material model.
<float7>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float8>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.
[integer]	*blk_id*, an optional parameter that is the element block number in conjugate problems that identifies the material region to which the contact angle applies (usually the liquid element block in solid/liquid conjugate problems).

Examples

Following is a sample card:

```
BC = VELO_THETA_COX NS   100   {45}   0.   1.   0. 0.1   72.0   0   0   2
```

This condition applies a contact angle of 45 degrees between the free surface normal at the 100 nodeset and the vector (0,1,0). The surface tension is 72, the reciprocal of the slip coefficient is 0.1, and the dynamic contact angle is taken from element block 2. Normally, this latter vector is the normal to the solid surface in contact with the free surface at this point.

Technical Discussion

- The constraint that is imposed at the nodeset node is as follows:

$$v = V_{old} + (v_{Cox} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

where v_{Cox} is computed from

$$Ca \equiv \frac{\mu v_{Cox}}{\sigma} = \frac{g(\theta, \lambda) - g(\theta_{eq}, \lambda)}{\left[\ln(\varepsilon_s^{-1}) + \frac{q_{inner}}{f(\theta_{eq}, \lambda)} - \frac{q_{outer}}{f(\theta, \lambda)}\right]}$$

where the Cox functions, f and g, are given by;

$$f(\theta, \lambda) \equiv \frac{2\sin\theta\{\lambda^2(\theta^2 - (\sin\theta)^2) + 2\lambda[\theta(\pi - \theta) + (\sin\theta)^2] + [(\pi - \theta)^2 - (\sin\theta)^2]\}}{\lambda(\theta^2 - (\sin\theta)^2)[\pi - \theta + \sin\theta\cos\theta] + [(\pi - \theta)^2 - (\sin\theta)^2][\theta - \sin\theta\cos\theta]} \quad \textbf{(4-102)}$$

$$g(\theta, \lambda) \equiv \int_0^{\theta} \frac{1}{f(\theta, \lambda)} d\beta \quad \textbf{(4-103)}$$

- The parameters λ, q_{inner}, and q_{outer} are currently not accessible from the input card and are hard-set to zero. λ is the ratio of gas viscosity to liquid viscosity whereas q_{inner} and q_{outer} represent influences from the inner and outer flow regions

When smoothing is not used, i.e. $t_{relax} = 0$, the imposed velocity is equal to that stipulated by the Hoffman correlation. Also see WETTING_SPEED_COX and SHARP_COX_VELOCITY for level-set versions.

$$f(\theta, 0) \equiv \frac{2\sin\theta}{[\theta - \sin\theta\cos\theta]}$$

- For steady problems, the substrate velocity will be extracted from adjoining VELO_TANGENT, VELO_SLIP, or VELO_SLIP_ROT boundary conditions.
- The Cox wetting velocity requires evaluation of integrals for the function g (θ, λ) which is currently done numerically using 10-point Gaussian quadrature. As such the evaluation of the integrals is expected to become inaccurate as either $\theta_e q$ tends toward zero or θ tends toward 180 degrees. Note that the integrand becomes singular as θ tends toward 0 or 180 degrees.
- This condition was motivated by the Cox hydrodynamic theory of wetting (cf. Stephan F. Kistler, "Hydrodynamics of Wetting," in Wettability edited by John Berg, 1993).

VELO_THETA_HOFFMAN

```
BC = VELO_THETA_HOFFMAN NS <bc_id> <float_list> [integer]
```

Description / Usage

(PCC/R_MOM_TANG1)

This boundary condition card applies a dynamic contact angle dependent velocity condition in a similiar fashion to VELO_THETA_TPL, but the functional form of the velocity is different. The functional form stems not from a theory of wetting, but instead, from a correlation of many empirical measurements.

The <float_list> for this boundary condition has eight values; definitions of the input parameters are as follows:

VELO_THETA_HOFFMAN	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	θ, equilibrium (static) contact angle subtended by wall normal and free surface normal, in units of degrees.
<float2>	n_x , x-component of normal vector to the geometry boundary (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float3>	n_y , y-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float4>	n_z , z-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float5>	currently not used.
<float6>	σ is the surface tension. This value is multiplied by the surface tension value stipulated by the surface tension material model.
<float7>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float8>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.
[integer]	*blk_id*, an optional parameter that is the element block number in conjugate problems that identifies the material region to which the contact angle applies (usually the liquid element block in solid/liquid conjugate problems).

Examples

Following is a sample card:

```
BC = VELO_THETA_HOFFMAN NS   100   {45}   0.   1.   0.   0.   72.0   0   0   2
```

This condition applies a contact angle of 45 degrees between the free surface normal at the 100 nodeset and the vector (0,1,0). The surface tension is 72 and the dynamic contact angle is taken from element block 2. Normally, this latter vector is the normal to the solid surface in contact with the free surface at this point.

Technical Discussion

- The constraint that is imposed at the nodeset node is as follows:

$$v = V_{old} + (v_{Hoffman} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

where $v_{Hoffman}$ is computed from the implicit solution of the Hoffman correlation;

or

where the Hoffman functions, fHoff and gHoff, which are inverses of each other are given by;

When smoothing is not used, i.e. t_{relax} = 0 , the imposed velocity is equal to that stipulated by the Hoffman correlation. Also see WETTING_SPEED_HOFFMAN and SHARP_HOFFMAN_VELOCITY for level-set versions.

$$\theta = f_{Hoff}[Ca + f^{-1}_{Hoff}(\theta_{eq})]$$

$$Ca \equiv \frac{\mu v_{Hoffman}}{\sigma} = g_{Hoff}(\theta) - g_{Hoff}(\theta_{eq})$$

$$f_{Hoff}(Ca) = arc\cos\left\{1 - 2\tanh\left[5.16\left(\frac{Ca}{1 + 1.31Ca^{0.99}}\right)^{0.706}\right]\right\}$$

$$g_{Hoff}(\theta) = (1 + 1.31g_{Hoff}(\theta)^{0.99})\left[\frac{1}{2(5.16)}\ln\frac{3 - \cos\theta}{1 + \cos\theta}\right]$$

- For steady problems, the substrate velocity will be extracted from adjoining VELO_TANGENT, VELO_SLIP, or VELO_SLIP_ROT boundary conditions.
- Because the Hoffman functions are implicit, iteration is required in the determination of the wetting velocity. As a result, for very high capillary numbers, i.e. > 10^6, the iteration procedure in Goma may need to be modified.
- This condition was motivated by the Hoffman empirical correlation (cf. Stephan F. Kistler, "Hydrodynamics of Wetting," in Wettability edited by John Berg, 1993).

VELO_THETA_TPL

```
BC = VELO_THETA_TPL NS <bc_id> <float_list> [integer]
```

Description / Usage

(PCC/R_MOM_TANG1)

This boundary condition card applies a dynamic contact angle dependent velocity condition in place of one component of the fluid momentum equation (unlike CA_BC which applies a fixed contact angle condition ot the mesh equation). The functional form of this condition is given below and stems from the Blake-DeConinck psuedomolecular kinetics theory of wetting. It is recommended that this condition or other forms of it (cf. VELO_THETA_HOFFMAN or VELO_THETA_COX) be used for steady and transient ALE problems. If you are deploying level-set technology to track moving capillary surfaces and three-phase wetting lines then the counterpart to this condition is WETTING_SPEED_LINEAR. It is noteworthy that this condition is applied to the fluid momentum equation, so that the velocity of the wetting line and the cosine of the current measured contact angle difference with the specified static value are related in a linear way.

The <float_list> for this boundary condition has eight values; definitions of the input parameters are as follows:

VELO_THETA_TPL	Name of the boundary condition.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	θ, equilibrium (static) contact angle subtended by wall normal and free surface normal, in units of degrees.
<float2>	n_x , x-component of normal vector to the geometry boundary (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float3>	n_y , y-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float4>	n_z , z-component of normal vector to the geometry boundary. (see important note below regarding variable wall normals, viz. non-planar solid walls).
<float5>	V_0 is a pre-exponential velocity factor (see functional form below)
<float6>	*g* is a thermally scaled surface tension, i.e. σ /2nkT. This value is multiplied by the surface tension value stipulated by the surface tension material model.
<float7>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float8>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.
[integer]	*blk_id*, an optional parameter that is the element block number in conjugate problems that identifies the material region to which the contact angle applies (usually the liquid element block in solid/liquid conjugate problems). NOTE/ WARNING: As of 1/13/2013 this option seems not to work with TALE problems.

Examples

Following is a sample card:

```
BC = VELO_THETA_TPL NS   100   {45}   0.   1.  0.  1000.0   5.e-4   0   0
```

This condition applies a contact angle of 45 degrees between the free surface normal at the 100 nodeset and the vector (0,1,0). The velocity scale is 1000 and the sensitivity scale is 5.e-4. Normally, this latter vector is the normal to the solid surface in contact with the free surface at this point.

Technical Discussion

- The constraint that is imposed at the nodeset node is as follows:

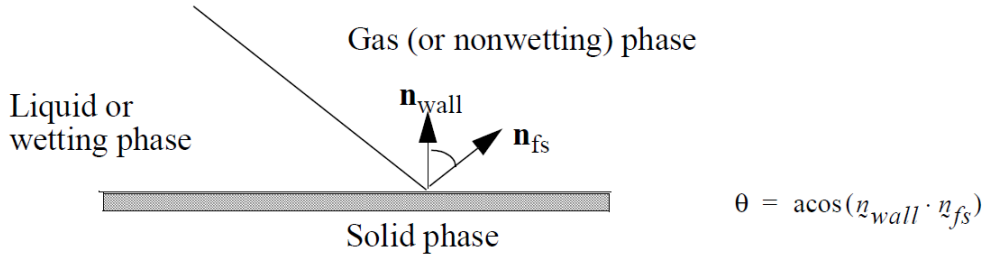
$$v = V_{old} + (v_{Blake} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

$$v_{Blake} = v_0 \sinh\left[g(\cos\theta_{eq} - \cos\theta)\right]$$

When smoothing is not used, i.e. t_{relax} = 0 , the imposed velocity is equal to that stipulated by the Blake-DeConinck equation. Also see WETTING_SPEED_LINEAR and WETTING_SPEED_BLAKE for level-set versions of this and VELO_THETA_HOFFMAN and VELO_THETA_COX for other functional forms.

- We recommend use of this condition over CA_BC for all transient problems. In this case this condition displaces a momentum equation component, with the other component being used to enforce no substrate penetration. The kinematic condition is applied to the mesh motion a this node so as to conserve mass.
- For steady problems, the substrate velocity will be extracted from adjoining VELO_TANGENT, VELO_SLIP, or VELO_SLIP_ROT boundary conditions.
- **Important: Variable Wall Normals**. Situations for which the wall shape is nonplanar, meaning that the normal vector is not invariant as the contact line moves, there is an option to leave all of the normal-vector components zero. In this case *Goma* then seeks to determine the local wall normal vector from the geometry it is currently on, using the element facets. It is recommended that this option not be used unless the geometry is truly nonplanar, as the logic is complex and not 100% reliable. See documentation for CA_BC for an example.
- This condition was motivated by T. D. Blake and is the so-called Blake- DeConinck condition (T. D. Blake, J. De Coninck 2002. “The influence of solidliquid interactions on dynamic wetting”, Advances in Colloid and Interface Science 96, 21-36.). See this article for some options for the form of the preexponential velocity, V_0.
- **Important: Wall Normal convention**. The wall normal vector on an external solid boundary is defined in goma as the inward facing normal to the mesh, and the free surface normal to the liquid (or wetting phase for two-liquid systems) is defined as the outward facing normal to the free surface. Put another way and referring to the picture below, the wall normal is directed from the “solid phase” to the “liquid phase”, and the free surface normal is directed from the “liquid phase” or “wetting phase” to the “vapor phase” or “Non-wetting phase”. Note that

for zero contact angle the liquid is “perfectly wetting”. The air-entrainment limit (viz. the hydrodynamic theory interpretation) would occure at a 180 degree contact angle. Recall that the angle is specified in radians on this card.



SHEET_ENDSLOPE

```
BC = SHEET_ENDSLOPE NS <bc_id> <float_list>
```

Description / Usage

(SPECIAL/VECTOR MOMENTUM)

This boundary condition card is used to enforce a slope of a membrane surface (cf. to be used in conjuction with *BC = TENSION_SHEET*) at its enpoints. . There are two values to be input for the <float_list>; definitions of the input parameters are as follows:

SHEET_ENDFORCE	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	X-component of upstream idler point (see discussion below)
<float2>	Y-component of upstream idler point.

Examples

The following is a sample input card using several APREPRO variables:

```
BC = SHEET_ENDSLOPE NS 100   {sind(th2)}   {-cosd(th2)
```

This condition would enforce a slope equivalent to that defined between the coordinates of the node at NS 100 with the point [sind(th2), -cosd(th2)].

Technical Discussion

Only two dimensional applications, viz. the nodeset is a single-node nodeset.

This is a single point nodeset boundary condition. Its function is to set the slope of the web at the single point nodeset N. It does this by enforcing continuity of the slope of the TENSION_SHEET sideset with the straight line that connects the point (X,Y) with nodeset N. Thus, this boundary condition can be used to model the influence of an upstream idler roller located at the point (X,Y). Indeed, this boundary condition has an alternate name: IDLER_LOC.

This boundary condition exploits the natural boundary conditions associated with the TENSION_SHEET formulation so it really can only beused in conjunction with the latter boundary condition.

One caveat that must be mentioned is that the formulation of these two boundary conditions is not general and therefore they should only be applied to web geometries that are predominantly horizontal. That is, the x component of the normal vector to the web sideset should at each point be less than or equal to the y component.

TENSION_SHEET

```
BC = TENSION_SHEET SS <bc_id> <float>
```

Description / Usage

(SIC/VECTOR MOMENTUM)

This boundary condition card is used to apply a membrane force to the fluid momentum equation in order to model a membrane-like structure, viz. one with no bending stiffness but with significant tension much larger than the fluid viscous stresses. This boundary condition is basically the same mathematically as the capillary condition, with the tension here specified instead of a capillary surface tension. The only difference is the way in which it is applied: it is applied as a strong integrated condition instead of a weak form condition.

Definitions of the input parameters are as follows:

TENSION_SHEET	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	T, Tension of the membrane.

Examples

Following is a sample card:

```
BC = TENSION_SHEET SS 12
```

Technical Discussion

Usage notes:

- Can only be applied in two dimensions.
- One caveat that must be mentioned is that the formulation of these two boundary conditions is not general and therefore they should only be applied to web geometries that are predominantly horizontal. That is, the x component of the normal vector to the web sideset should at each point be less than or equal to the y component.
- To set the slope of the membrane at an endpoint, see the BC = SHEET_ENDSLOPE card.
- This boundary condition that can be used to model the interaction of fluid with a thin sheet under a constant tension load. The sideset to which it is applied must be fully "wetted" by a fluid. Note that this boundary condition arises as a simplification of the tensioned-web shell equations (cf. shell_tension and shell_curvature equations as described in GT-033.0 and GT-027.0) subject to two simplifying assumptions:
  1. The sheet supports no bending moments. That is, it isn't very rigid.
  2. The tension in the sheet is significantly larger than the viscous stresses in the fluid.

Given these assumptions this boundary condition can be used to model tensioned web applications without having to resort to the shell equations. It is a strongly integrated, rotated boundary condition on the mesh equations. It can only be used in twodimensional applications.

References

GT-033.0 and GT-027.0.

Category 5: Energy Equations

The following conditions are applied as boundary conditions to the energy equation. These conditions include strong Dirichlet conditions, such as hard sets on temperature on a boundary as a constant or function of position, weak-form conditions, such as a specified heat flux from a convective heat transfer model or a constant flux, and a host of interfacial conditions for phase change (viz. latent heat effects), etc. The energy equation is of course a scalar equation. Some with a molten metal surface. These conditions will also apply in general to the porous energy equation (see Porous Energy). highly specialized equations are also available, such as a heat flux model for a laser interaction

T

```
BC = T NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/ENERGY)

This Dirichlet boundary condition card is used to set constant temperature. Definitions of the input parameters are as follows:

T	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of temperature.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample input card:

```
BC = T NS 100 273.13
```

Technical Discussion

No Discussion.

T_USER

```
BC = T_USER SS <bc_id> <float_list>
```

Description / Usage

(PCC/ENERGY)

This boundary condition card is used to call a routine for a user-defined temperature. Specification is made via the function tuser in file "user_bc.c." Definitions of the input parameters are as follows:

T_USER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutine so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function in file user_bc.c.

Examples

The following is a sample input card with two parameters passed to function tuser:

```
BC = T_USER SS 100   273.13 100.0
```

Technical Discussion

No Discussion.

References

No References.

QCONV

```
BC = QCONV SS <bc_id> <float1> <float2>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card specifies convective heat flux. Definitions of the input parameters are as follows:

QCONV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	h, heat transfer coefficient.
<float2>	T_s, sink temperature.

Examples

The following is a sample card:

```
BC = QCONV SS 100 10.0 293.0
```

Technical Discussion

The convective heat flux is defined as

where h and T_s are the convective heat transfer coefficient and the sink temperature, respectively.

$$\underset{\sim}{n} \cdot \underset{\sim}{q} = h(T - T_s)$$

QRAD

```
BC = QRAD SS <bc_id> <float_list>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card specifies heat flux using both convective and radiative terms. The <float_list> has four parameters; definitions of the input parameters are as follows:

QRAD	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	h, convective heat transfer coefficient.
<float2>	T_s, sink temperature.
<float3>	ε, total hemispherical emissivity.
<float4>	σ, Stefan-Boltzmann constant.

Examples

The following is a sample card:

```
BC = QRAD SS 100 10.0 273.0 0.3 5.6697e-8
```

Technical Discussion

The heat flux definition for this card is a combined convective and radiative formulation:

$$\underset{\sim}{n} \cdot \underset{\sim}{q} = h(T - T_s) + \varepsilon\sigma(T^4 - T_s^4)$$

where h and T_s are the convective heat transfer coefficient and the sink temperature, and ε and σ are the total hemispherical emissivity and Stefan-Boltzmann constant, respectively. The latter constant has been made an input parameter rather than a code constant so that the user can specify its value in units that are consistent for the problem being modeled.

The *QRAD* boundary condition can be used in place of *QCONV* by simply setting the emissivity value to zero.

QSIDE

```
BC = QSIDE SS <bc_id> <float1>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used to specify a constant heat flux. Definitions of the input parameters are as follows:

QSIDE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Value of heat flux. A positive value implies that energy is being added to system; a negative value implies energy is being taken from the system through the boundary.

Examples

The following is a sample card:

```
BC = QSIDE SS 22   1.50
```

Technical Discussion

The mathematical form of the boundary condition.

$$n \cdot q = q_o$$

T_CONTACT_RESIS, T_CONTACT_RESIS_2

```
BC = T_CONTACT_RESIS SS <bc_id> <int1> <int2> <float1>
```

```
BC = T_CONTACT_RESIS_2 SS <bc_id> <int2> <int1> <float1>
```

Description / Usage

(WIC/ENERGY)

This boundary condition set is used to specify a thermal contact resistance at an interface between two mesh regions defined by a side set. Please see special usage notes below regarding proper side-set specification and the reasons that both BC cards are required for an internal interface. NOTE that the temperature field MUST be interpolated with the discontinous versions of Q1 or Q2, viz. Q1_D and Q2_D. Definitions of the input parameters are as follows:

T_CONTACT_RESIS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database. Note this side set MUST contain elements on both sides of the interface.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<int1>	Material/region ID associated with first material.
<int2>	Material/region ID associated with second material. Note that these material IDs are reversed on the second BC.
<float1>	Value contact resistance in units of thermal conductivity divided by length.

Examples

The following is a sample card:

```
BC = T_CONTACT_RESIS SS 3 1 2 10.0
```

```
BC = T_CONTACT_RESIS_2 SS 3 2 1 10.0
```

Note that both boundary condition cards are required at an internal interface. In this case the interface divides mesh/material ID 1 and 2. Note also how these material IDs are reversed on the second card. These conditions apply a thermal contact resistance of 10. (units of thermal conductivity divided by length) at the interface defined by SS 3.

Technical Discussion

The mathematical form of the boundary condition.

$$\boldsymbol{n} \cdot \boldsymbol{q}\big|_a = R^{-1}(T_a - T_b) = \boldsymbol{n} \cdot q_b$$

The flux into the interface from material "a" is equivalent to that into material "b", both equal to the temperature jump across the interface times the contact resistance R^{-1}.

The side set to which this boundary condition is applied must contain elements on both sides of the interface. Look up any special commands in your mesh generator to make sure this occurs. In CUBIT, for example, you have to add "wrt volume 1 2" like qualifiers on the side set command. The reason for the "double application" of this condition is to pick up the all the terms from both sides of the interface with the proper sign. The nodes at the interface have two temperatures, one from each side, and so two weak form applications of this equation are required, one from each side.

QNOBC

```
BC = QNOBC SS <bc_id>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card applies the open boundary condition equivalent on the energy equation (see momentum open BC FLOW_STRESSNOBC)

QNOBC	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

Examples

Following is a sample card:

```
BC = QNOBC SS 3
```

Here the BC is applied to SS 3

Technical Discussion

The finite element formulation of the energy equation generates boundary integrals of the form:

$$\int_{\Gamma} \mathbf{n} \cdot \mathbf{q} \, dS$$

where **q** is the heat flux.

This boundary condition adds this term back in at the boundary avoiding the natural boundary condition.

References

Griffiths, D.F., "The 'no boundary condition' outflow boundary condition," IJNMF, 24, 393-411, (1997)

Sani, R.L., and P.M. Gresho, "Resume and remarks on the open boundary condition minisymposium," IJNMF, 18, 983-1008, (1994).

QUSER

```
BC = QUSER SS <bc_id> <float_list>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used to call a routine for a user-defined heat flux model. Definitions of the input parameters are as follows:

QUSER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the quser_surf C function in file user_bc.c.

Examples

The following is a sample input card for a heat flux model requiring two parameters:

```
BC = QUSER SS 100   10.0 3.14159
```

Technical Discussion

No Discussion.

Q_VELO_SLIP

```
BC = Q_VELO_SLIP SS <bc_id>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used to calculate the surface integral for viscous heating due to slip in the tangential velocity component on a surface. Definitions of the input parameters are as follows:

Q_VELO_SLIP	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

Examples

The following is a sample input card:

```
BC = Q_VELO_SLIP_BC SS 10
```

Technical Discussion

Use of this boundary condition requires specification of the slip velocity components by using either the *VELO_SLIP* or *VELO_SLIP_ROT* boundary condition.

Q_LASER_WELD

```
BC = Q_LASER_WELD SS <bc_id> <float_list>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card specifies the thermal boundary conditions for laser welding. The <float_list> requires twenty-seven values be specified; definitions of the input parameters are as follows:

Q_LASER_WELD	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Nominal power of laser.
<float2>	Power of laser at base state (simmer).
<float3>	Base value of surface absorptivity.
<float4>	Switch to allow tracking of normal component of liquid surface relative to laser beam axis for surface absorption (0 = OFF, 1 = ON)
<float5>	Cutoff time for laser power.
<float6>	Time at which laser power drops to 1/e.
<float7>	For pulse weld, the laser power overshoot (%) of peak power at time to reach peak laser power.
<float8>	Radius of laser beam.
<float9>	For pulse weld, the time for laser pulse to reach peak power.
<float10>	For pulse weld, the time for laser pulse to reach steady state in power.
<float11>	Switch to either activate laser power distribution from beam center based on absolute distance (0) or based on radial distance in 2D plane (1).
<float12>	Location of laser beam center (x-coordinate).
<float13>	Location of laser beam center (y-coordinate).
<float14>	Location of laser beam center (z-coordinate).
<float15>	Laser beam orientation, normal to x-coordinate of body.
<float16>	Laser beam orientation, normal to y-coordinate of body.
<float17>	Laser beam orientation, normal to z-coordinate of body.
<float18>	For pulse weld, spot frequency.
<float19>	For pulse weld, total number of spots to simulate.
<float20>	Switch to set type of weld simulation. (0=pulse weld, 1=linear continuous weld, -1=pseudo pulse weld, 2=sinusoidal continous weld)
<float21>	For pulse weld, spacing of spots.
<float22>	For radial traverse continuous weld, radius of beam travel.
<float23>	Switch to activate beam shadowing for lap weld (0=OFF, 1=ON). Currently only active for ALE simulations.
<float24>	Not active, should be set to zero.
<float25>	For continuous weld, laser beam travel speed in xdirection (u velocity).
<float26>	For continuous weld, laser beam travel speed in ydirection (v velocity).
<float27>	For continuous weld, laser beam travel speed in zdirection (w velocity).

Examples

The following is a sample input card:

```
BC = Q_LASER_WELD SS 10 4.774648293 0 0.4 1 1 1.01 4.774648293 0.2
0.01 0.01 1 0.005 0 -0.198 -1 0 0 0.025 1 1 0.2032 -1000 0 0 0 0 0.0254
```

Technical Discussion

Several required pieces of information to use this boundary condition are not in final form, and the user can expect future changes and improvements. Below is a listing of some of these parameters:

- This boundary condition requires that node sets 1001 is defined in the EXODUS II file. NS 1001 should include the point at the center of the keyhole on the surface closest to the beam.
- Currently the laser flux distribution is set as a fixed exponential distribution. Plans are to include more options including a user-defined exponential and a TABLE option.
- Correlations are used to specify the evaporation energy loss. Currently only iron and ice correlations exist; the appropriate correlation is selected based on the value set for the *Solidus Temperature* (in *Thermal Properties* portion of the material file).

Q_VAPOR_BC

```
BC = Q_VAPOR SS <bc_id> <float_list>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used to specify heat loss due to evaporation. It is typically used in conjunction with Q_LASER_WELD.

Q_VAPOR	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Temperature scale.
<float2>	Energy unit scale.

Examples

The following is a sample input card:

```
BC = Q_VAPOR SS 10 100. 10.
```

Technical Discussion

This condition is turned on above the boiling point, which is story in the melting point solidus temperature.

VP_EQUIL

```
BC = VP_EQUIL SS <bc_id> <integer_list> <float>
```

Description / Usage

(SIC/ENERGY)

This boundary condition card is used to equate solvent partial pressure in the gas between the porous medium and the external phase. The condition is similar to the solid-liquid interface conditions that apply to interfaces between a porous medium and an external gas phase (in which the energy equation is used to solve for solvent concentration in the gas phase). This boundary condition is still under development.

There are three values to be specified for the <integer_list>; definitions of the input parameters are as follows:

VP_EQUIL	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid, porous phase from the EXODUS II database.
<integer2>	Element block ID of gas phase from the EXODUS II database.
<integer3>	Species number of liquid phase in porous medium.
<float>	Ambient pressure in external gas phase.

Examples

The following is a sample input card:

```
BC = VP_EQUIL SS 100   1 2 0 0.0
```

where the solid/porous phase is present in element block 1 and the gas phase is present in element block 2. The external gas phase pressure has been set to 0.0.

Technical Discussion

No Discussion.

LATENT_HEAT

```
BC = LATENT_HEAT SS <bc_id> <integer> <float_list>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used for latent heat release/adsorption at an external interface. The flux quantity is specified on a per mass basis so the heat and mass transfer coefficients are in units of L/t.

The <float_list> has three values to be specified; definitions of the input parameters are as follows:

LATENT_HEAT	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number.
<float1>	Latent Heat for the pure w+1 species component.
<float2>	Mass transfer coefficient for the w+1 species component.
<float3>	Sink concentration for the w+1 species component.

The float values on this card apply to the bulk species (i.e., the w+1 component) in a multi-species problem and in the case of a pure fluid. Important usage comments are contained in the Technical Discussion below.

Examples

The following is a sample input card:

```
BC = LATENT_HEAT SS 3 0   540. 0.1   0.
```

Two more detailed examples are contained in the Technical Discussion section.

Technical Discussion

The *LATENT_HEAT* boundary condition has the form

where $\underline{n}$ is the outward normal to the surface, $\underline{q}$ is the heat flux vector, $\Delta\ H_\nu$ is the heat of vaporization, ρ is density, h_i is the heat-transfer coefficient for species *i*, and y_i^0 is the reference concentration of species *i* at locations remote from the boundary; the summation is over the number of species in the material block. The manner of usage of this boundary condition depends on the set of conditions characterizing the problem; example conditions are described below.

This card is used for external surfaces for which heat transfer and mass transfer beyond it's surfaces are governed by heat and mass transfer coefficients. The *LATENT_HEAT* BC is applied to the energy equation so a heat flux can be specified for thermal problems alone. The mass transfer portion of the vaporization phenomenon is handled by the *KIN_LEAK* and

$$\underset{\sim}{n} \cdot \underset{\sim}{q} = \sum_{i=1}^{Numspec} \Delta H_0 \, \rho h_i (y_i - y_i^0)$$

YFLUX BC cards; these boundary conditions are applied to the mesh equations. The *LATENT_HEAT_INTERNAL* card should be used for internal surfaces, or interfaces, at which transfer is governed by actual physics being modeled as a part of the problem.

When vaporization of a pure liquid is being modeled, there is only a 'single species', the bulk volatile liquid. In the single species case, the *Species Properties* of the corresponding material file (which includes the *Heat of Vaporization* card) is not even read so the actual value of the latent heat of vaporization must be entered on the*LATENT_HEAT* card (<float1>). If multiple species are present, the latent heat value for each species is entered in the material file and the *LATENT_HEAT* card does for the energy equation the same thing the *KIN_LEAK* card does for the mesh equation (i.e., collects the flux conditions that apply for each species).

For mass transfer in the single species/pure liquid case, the mass transfer coefficient is specified on the *KIN_LEAK* card. When multiple species are present, the mass transfer coefficient and driving concentration on the *KIN_LEAK* card are set to zero and the appropriate coefficient and driving concentration are set for each species on the *YFLUX* card, one for each species. The *KIN_LEAK* card (or the LATENT HEAT for energy flux) must be present to signal *Goma* to look for multiple *YFLUX* cards.

The latent heat quantity is specified on a per mass basis and the transfer coefficients are in units of L/t. Some examples of *LATENT_HEAT* application follow:

Pure Liquid Case

```
BC = LATENT HEAT   SS 3 0   540.   0.1   0.
```

```
BC = KIN_LEAK   SS 3   0.1   0.
```

Two-Species Case

```
BC = LATENT HEAT   SS 3 0   0.   0.1   0.
```

```
BC = KIN_LEAK   SS 3   0.   0.
```

```
BC = YFLUX SS 3 0   0.12   0.
```

```
BC = YFLUX   SS 3 1   0.05   0.
```

plus, in the corresponding material file:

```
---Species Properties
Diffusion Constitutive Equation= FICKIAN
Diffusivity = CONSTANT   0   1.e-8
Latent Heat Vaporization = CONSTANT   0   540.
```

(continues on next page)

(continued from previous page)

```
Latent Heat Fusion = CONSTANT   0   0.
Vapor Pressure = CONSTANT   0   0.
Species Volume Expansion = CONSTANT   0   1.
Reference Concentration = CONSTANT   0   0.
Diffusivity = CONSTANT   1   1.e-6
Latent Heat Vaporization = CONSTANT   1   125.
Latent Heat Fusion = CONSTANT   1   0.
Vapor Pressure = CONSTANT   1   0.
Species Volume Expansion = CONSTANT   1   1.
Reference Concentration = CONSTANT   1   0.
```

References

No References.

LATENT_HEAT_INTERNAL

```
BC = LATENT_HEAT_INTERNAL SS {char_string} <integer_list> <float>
```

Description / Usage

(WIC/ENERGY)

This boundary condition card is used for latent heat release/adsorption at an internal interface. See usage comments in the Technical Discussion.

The <integer_list> requires two values be specified; definitions of the input parameters are as follows:

LATENT_HEAT_INTERNAL	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
{char_string}	Variable name with the following permissible values: • **LIQUID_VAPOR** • **SOLID_LIQUID**
<integer1>	NOT ACTIVE. Any integer will do.
<integer2>	NOT ACTIVE. Any integer will do.
<float3>	Value of latent heat of vaporization/fusion for a pure material case, in units of Energy/mass.

Examples

The following is a sample input card:

```
BC = LATENT_HEAT_INTERNAL SS 40 SOLID_LIQUID 1 2 2.6e5
```

Technical Discussion

The *LATENT_HEAT_INTERNAL* card should be used for internal surfaces, or interfaces, at which transfer is governed by actual physics being modeled as a part of the problem. See *LATENT_HEAT* card for further information.

Category 6: Mass Equations

The collection of boundary conditions in this category are applied to the mass balance equations, specifically the species component balance equations. Most boundary conditions are weakly integrated conditions defining fluxes at internal or external surfaces, although strongly integrated and Dirichlet conditions are also available to control known values of dependent variables or integrated quantities. Boundary conditions are available for chemical species as well as charged species, suspensions and liquid metals. An important capability in *Goma* is represented by the discontinuous variable boundary conditions, for which users are referred to Schunk and Rao (1994) and Moffat (2001). Care must be taken if the species concentration is high enough to be outside of the *dilute species* assumption, in which case transport of species through boundaries will affect the volume of the bounding fluids. In these cases, users are referred to the *VNORM_LEAK* condition for the fluid momentum equations and to *KIN_LEAK* for the solid momentum (mesh) equations. And finally, users are cautioned about different bases for concentration (volume, mass, molar) and several discussions on or references to units.

Y

```
BC = Y NS <bc_id> <integer> <float1> [float2] [integer2]
```

Description / Usage

(DC/MASS)

This card is used to set the Dirichlet boundary condition of constant concentration for a given species. Definitions of the input parameters are as follows:

Y	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<integer1>	Species number of concentration.
<float1>	Value of concentration, in user's choice of units, e.g. moles/ cm^3.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*
[integer2]	Element block ID; only applicable to node sets, this optional parameter specifies the element block on which to impose the boundary condition, if there is a choice, as occurs at discontinuous variable interfaces where there may be more that one unknown corresponding to species 0 at a single node. This parameter allows the user to specify which unknown to set the boundary condition on, and allows for a jump discontinuity in species value across a discontinuous variables interface.

Examples

The following is a sample card with no Dirichlet flag:

```
BC = Y NS 3   0   0.00126
```

Technical Discussion

No Discussion.

YUSER

```
BC = YUSER SS <bc_id> <integer> <float_list>
```

Description / Usage

(SIC/MASS)

This is a user-defined mass concentration boundary. The user must supply the relationship in function *yuser_surf* within *user_bc.c*.

YUSER	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<integer>	Species number.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following sample input card applies a user flux condition to side set 100 for species 0 that requires two input parameters:

```
BC = YUSER SS 100 0 .5 .5
```

Technical Discussion

No Discussion.

References

No References.

Y_DISCONTINUOUS

```
BC = Y_DISCONTINUOUS NS <bc_id> <integer1> <float1> [float2 integer2]
```

Description / Usage

(DC/MASS)

This card is used to set a constant valued Dirichlet boundary condition for the species unknown. The condition only applies to interphase mass, heat, and momentum transfer problems applied to discontinuous (or multivalued) species unknown variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

Definitions of the input parameters are as follows:

Y_DISCONTINUOUS	Name for the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<integer1>	Species subvariable number.
<float1>	Value of the species unknown on the boundary. Note, the units depend on the specification of the type of the species unknown.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*
[integer2]	Element block ID; only applicable to node sets, this optional parameter specifies the element block on which to impose the boundary condition, if there is a choice, as occurs at discontinuous variable interfaces where there may be more that one unknown corresponding to species 0 at a single node. This parameter allows the user to specify which unknown to set the boundary condition on, and allows for a jump discontinuity in species value across a discontinuous variables interface.

Examples

The following is a sample input card with no Dirichlet flag:

```
BC = Y_DISCONTINUOUS SS 3   0   0.00126
```

Technical Discussion

Typically, this boundary condition may be used to set the species unknown variable on one side of a discontinuous variables interface, while the species unknown variable on the other side of the interface is solved for via a *KINEMATIC_SPECIES* boundary condition. Note, this boundary condition is not covered by the test suite, and thus, may or may not work.

YFLUX

```
BC = YFLUX SS <bc_id> <integer1> <float1> <float2>
```

Description / Usage

(WIC/MASS)

This boundary condition card is used to specify the mass flux of a given species normal to the boundary (or interface) using a mass transfer coefficient. When used in conjunction with the *KIN_LEAK* card, the *YFLUX* card also enables the determination of velocity normal to the moving boundary at which the *YFLUX* boundary condition is applied.

Definitions of the input parameters are as follows:

YFLUX	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	*i*, species number of concentration.
<float1>	k_i, value of mass transfer coefficient of species *i*.
<float2>	c_i^∞ value of reference concentration of species *i*.

Examples

Following are two sample cards:

```
BC = YFLUX SS   3 0   0.12   0.
```

```
BC = YFLUX SS   3 1   0.05   0.
```

Technical Discussion

Specifically, the species mass flux is given by

$$\boldsymbol{n} \cdot \boldsymbol{J}_i + \boldsymbol{n} \cdot (\boldsymbol{v} - \boldsymbol{v}_m) c_i = k_i (c_i - c_i^\infty)$$

where *n* is the unit vector normal to the boundary, J_i is mass flux of species *i*, *v* is the fluid velocity, v_m is the mesh displacement velocity, k_i is mass transfer coefficient of species *i*, c_i is concentration of species *i* at the boundary surface, c_i^∞ and is reference concentration of species *i*. The units of J_i, k_i, c_i and c_i^∞ depend on the user's choice. For example, if c_i and c_i^∞ are chosen to have units of moles/ cm^3, then k_i has the unit of cm/ s, and J_i has the units of moles/ cm^2 /s.

For the *KIN_LEAK* and *VNORM_LEAK* cards, the information from *YFLUX* boundary conditions corresponding to each species is needed. *Goma* automatically searches for these boundary conditions and uses an extra variable in the BC data storage to record the boundary condition number of the next *YFLUX* condition in a linked list; when the extra storage value is -1, there are no more *YFLUX* conditions at this boundary.

FAQs

A question was raised regarding the use of volume flux in Goma; the following portion of the question and response elucidate this topic and the subject of units. Note the references in the response are to the Version 2.0 Goma User's Manual.

Question: … I know what you are calling volume flux is mass flux divided by density. The point I am trying to make is that the conservation equations in the books I am familiar with talk about mass, energy, momentum, and heat fluxes. Why do you not write your conservation equations in their naturally occurring form? If density just so happens to be common in all of the terms, then it will be obvious to the user that the problem does not depend on density. You get the same answer no matter whether you input rho=1.0 or rho=6.9834, provided of course this does not impact iterative convergence. This way, you write fluxes in terms of gradients with the transport properties (viscosity, thermal conductivity, diffusion coefficient, etc.) being in familiar units.

Answer: First let me state the only error in the manual that exists with regard to the convection-diffusion equation is the following:

> J_t in the nomenclature table … should be described as a volume flux with units of L/t, i.e., $D \cdot \Delta_y i$, where D is in $L^2/$t units.

Now, … this is actually stated correctly, as it states the J_i is a diffusion flux (without being specific); to be more specific here, we should say it is a "volume flux of species *i*." So, in this case D is in L · L/t units, y_i is dimensionless and it is immaterial that (the mass conservation equation) is multiplied by density or not, *as long as density is constant*.

Now, in *Goma* we actually code it up EXACTLY as in the … (mass conservation equation), i.e., there are no densities anywhere for the *FICKIAN* diffusion model. For the *HYDRO* diffusion model, we actually compute a J_i/ρ in the code, and handle variable density changes through that p . In that case J_i as computed in Goma is a mass flux vector, not a volume flux vector, but by dividing it by p and sending it back up to the mass conservation equation it changes back into a volume flux. i. e., everything is the same.

Concerning the units of the mass transfer coefficient on the *YFLUX* boundary condition, the above discussion now sets those. *Goma* clearly needs the flux in the following form:

$$\underset{\sim}{n} \cdot D\nabla Y = K \cdot (y_i - y_i^{\infty})$$

and dimensionally for the left hand side

$$(L^2/t) \cdot (1/L) = L/t$$

where D is in units L^2 /t , the gradient operator has units of 1/L so K HAS to be in units of L/t (period!) because y_i is a fraction.

So, if you want a formulation as follows:

$$\underset{\sim}{n} \cdot D\nabla Y = \hat{K}(p_i - p_i^{\infty})$$

then K's units will have to accommodate for the relationship between p_i and y_i in the liquid, hopefully a linear one as in Raoult's law, i.e. if $p_i = P_V y_i$ where P_v is the vapor pressure, then

$$\underset{\sim}{n} \cdot D\nabla Y = KP_V(y_i - y_i^{\infty})$$

and so K on the *YFLUX* command has to be KP_v ….and so on.

Finally, you will note, since we do not multiply through by density, you will have to take care of that, i. e., in the Price paper he gives K in units of t/L. So, that must be converted as follows:

$$K_{price}(P_V/\rho) = K_{goma} \quad : \quad (t/L)(M/Lt^2)(L^3/M) = L/t$$

This checks out!

References

Price, P. E., Jr., S. Wang, I. H. Romdhane, 1997. "Extracting Effective Diffusion Parameters from Drying Experiments", AIChE Journal, 43, 8, 1925-1934.

YFLUX_CONST

```
BC = YFLUX_CONST SS <bc_id> <integer> <float>
```

Description / Usage

(WIC/MASS)

This boundary condition card is used to specify a constant diffusive mass flux of a given species. This flux quantity can be specified on a per mass basis (e.g. with units of g/ cm^2 /s) or on a per mole basis (e.g. with units of moles/ cm^2 /s), depending on the user's choice of units in the species unknown.

Definitions of the input parameters are as follows:

YFLUX_CONST	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number.
<float>	Value of diffusive mass flux; the units of this quantity depends on the user's choice of units for species concentration.

Examples

Following is a sample card:

```
BC = YFLUX_CONST SS 1 0 10000.2
```

Technical Discussion

No Discussion.

References

No References.

YFLUX_EQUIL

```
BC = YFLUX_EQUIL SS <bc_id> {char_string} <integer> <float_list>
```

Description / Usage

(WIC/MASS)

This boundary card is used when equilibrium-based mass transfer is occurring at an vapor-liquid external boundary; i.e.,

$$J_i = k_i(w_i^v - w_i^{v,\infty})$$

This is different from an internal boundary since only one phase is represented in the computational domain. This boundary condition then describes the rate of mass entering or leaving the boundary via vapor-liquid equilibria. The w_i^v is the mass fraction of component *i* in *vapor* that is in equilibrium with the liquid phase. The $w_i^{v\infty}$ is the bulk concentration of component *i* in vapor.

The <float_list> requires three input values; definitions of the input parameters are as follows:

YFLUX_EQUIL	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
{char_string}	This refers to the equilibrium model for mass transfer; the options are either **FLORY** or **RAOULT**.
<integer1>	Species id.
<float1>	Total system pressure.
<float2>	Mass transfer coefficient.
<float3>	Bulk concentration in vapor ($w_i^{v\infty}$)

Examples

The following is a sample input card:

```
BC = YFLUX_EQUIL SS 1   FLORY 0   1. 5.4e-3   0.
```

Technical Discussion

This boundary condition is very similar to *VL_EQUIL* and *VL_POLY* except that it is only applied at an external boundary where vapor phase is not modeled in the problem.

References

GTM-007.1: New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

YFLUX_SULFIDATION

```
BC = YFLUX_SULFIDATION SS <bc_id> {char_string} <integer> <float_list>
```

Description / Usage

(WIC/MASS)

The *YFLUX_SULFIDATION* card enables computation of the molar flux of the diffusing species (e.g. copper vacancy) using copper-sulfidation kinetics at the specified boundary (gas/ Cu^2S or Cu/ Cu^2S interface). When used in conjunction with the *KIN_LEAK* card, it also enables the determination of velocity normal to the moving gas/ Cu^2S interface.

The <float_list> contains ten values to be defined; these and all input parameter definitions are as follows:

YFLUX_SULFIDATION	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
{char_string}	Name of sulfidation kinetic models. Allowable names are: • **SOLID_DIFFUSION_SIMPLIFIED** • **SOLID_DIFFUSION** • **SOLID_DIFFUSION_ELECTRONEUTRALITY** • **SOLID_DIFFUSION_ELECTRONEUTRALITY_LINEAR** • **GAS_DIFFUSION** • **FULL** • **ANNIHILATION** • **ANNIHILATION_ELECTRONEUTRALITY** Detailed description of kinetic models with these name key words are presented in the Technical Discussion section below.
<integer>	Species number of concentration.
<float1>	Stoichiometric coefficient.
<float2>	Rate constant for forward copper sulfidation reaction.
<float3>	Activation energy for forward copper sulfidation reaction.
<float4>	Rate constant for backward copper sulfidation reaction.
<float5>	Activation energy for backward copper sulfidation reaction.
<float6>	Temperature.
<float7>	Bulk concentration of H^2S
<float8>	Bulk concentration of O^2
<float9>	Molecular weight of copper sulfide (Cu_2S)

Examples

Examples of this input card follow:

```
BC = YFLUX_SULFIDATION SS 3 SOLID_DIFFUSION_ELECTRONEUTRALITY 0 -2.0
1.46e+7 6300.0 1.2e+14 6300.0 303.0 1.61e-11 8.4e-6 159.14 5.6
```

```
BC = YFLUX_SULFIDATION SS 1 ANNIHILATION_ELECTRONEUTRALITY 0 1.0   10.0
0.0   0.0   0.0   303.0 1.61e-11 8.4e-6 159.14 5.6
```

```
BC = YFLUX_SULFIDATION SS 3 SOLID_DIFFUSION 1 -2.0 1.46e7 6300.0 1.2e+14
6300.0 303.0 1.61e-11 8.4e-6 159.14 5.6
```

```
BC = YFLUX_SULFIDATION SS 1 ANNIHILATION 1 1.0 10.0   0.0   0.0   0.0
303.0 1.61e-11 8.4e-6 159.14 5.6
```

Technical Discussion

Key word **SOLID_DIFFUSION_SIMPLIFIED** refers to the following simplified kinetic model of copper sulfidation in which gas-phase diffusion is neglected and Cu is taken to be the diffusing species:

$$r = ke^{-\frac{E}{RT}} c_{H_2S}{}^c Cu$$

where *r* is molar rate of formation of sulfidation-corrosion product, Cu^2S, per unit area, ${}^cH_2S$ is the molar concentration of H_2S taken to be fixed at its bulk value, ${}^cCu$ is the molar concentration of Cu at the sulfidation surface (Cu_2S /gas interface), *k* is the rate constant, *E* is the activation energy, *R* is the universal gas constant, and *T* is the temperature.

Key word **SOLID_DIFFUSION** refers to the following kinetic model of copper sulfidation in which gas-phase diffusion is neglected and Cu vacancies and electron holes are taken as the diffusing species:

$$r = k_1 e^{-\frac{E_1}{RT}} c_{H_2S} \sqrt{c_{O_2}} - k_{-1} e^{-\frac{E_{-1}}{RT}} c_V^2 c_h^2$$

where *r* is molar rate of formation of Cu_2S per unit area, ${}^cH_2S$ and ${}^cO_2$ are the molar concentrations of H_2S and O_2, respectively, taken to be fixed at their bulk values, $\frac{c}{V}$ and $\frac{c}{h}$ are the molar concentrations of Cu vacancies and electron holes, respectively, at the sulfidation surface, k_1 and k_{-1} are rate constants, respectively, for the forward and backward sulfidation reactions, E_1 and E_{-1} are activation energies, respectively, for the forward and the backward sulfidation reactions.

Key word **SOLID_DIFFUSION_ELECTRONEUTRALITY** refers to the following kinetic model of copper sulfidation in which Cu vacancies and electron holes are taken as the diffusing species and the electroneutrality approximation is applied such that concentrations of Cu vacancies and electron holes are equal to each other:

$$r = k_1 e^{-\frac{E_1}{RT}} c_{H_2S} \sqrt{c_{O_2}} - k_{-1} e^{-\frac{E_{-1}}{RT}} c_V^4$$

Key word **SOLID_DIFFUSION_ELECTRONEUTRALITY_LINEAR** refers to the following kinetic model of copper sulfidation:

$$r = k_1 e^{-\frac{E_1}{RT}} c_{H_2S} \sqrt{c_{O_2}} - k_{-1} e^{-\frac{E_{-1}}{RT}} c_V c_h$$

Key word **GAS_DIFFUSION** refers to the following simplified kinetic model of copper sulfidation in which solid-phase diffusion is neglected, and H_2S and O_2 are taken to be the diffusing species:

$$r = k e^{-\frac{E}{RT}} c_{H_2S} \sqrt{c_{O_2}}$$

Key word **FULL** refers to the following kinetic model in which diffusion in both the gas phase and the solid phase are important, and H_2S, O_2, Cu vacancies, and electron holes are taken as the diffusing species:

$$r = k_1 e^{-\frac{E_1}{RT}} c_{H_2S} \sqrt{c_{O_2}} - k_{-1} e^{-\frac{E_{-1}}{RT}} c_V^2 c_h^2$$

where ${}^c H_2S$ and ${}^c O_2$ are the time-dependent molar concentrations of H_2S and O_2, respectively, at the sulfidation surface.

Key word **ANNIHILATION** refers to the following kinetic model in which diffusion in both the gas phase and the solid phase are important, and H_2S, O_2, Cu vacancies, and electron holes are taken as the diffusing species:

where k_2 are E_2 are the rate constant and activation energy, respectively, for the annihilation reaction.

Key word **ANNIHILATION_ELECTRONEUTRALITY** is similar to **ANNIHILATION** except that, here, the electroneutrality approximation is applied and concentrations of Cu vacancies and electron holes are taken to be equal to each other:

$$r = k_2 e^{-\frac{E_2}{RT}} c_V c_h$$

$$r = k_2 e^{-\frac{E_2}{RT}} c_V^2 \quad .$$

YFLUX_SUS

```
BC = YFLUX_SUS SS <bc_id> <integer>
```

Description / Usage

(WIC/MASS)

This boundary defines a flux of suspension particles at an interface. Definitions of the input parameters are as follows:

YFLUX_SUS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species id; the species number for suspension particles.

Examples

The following is a sample input card:

```
BC = YFLUX_SUS SS 1    0
```

Technical Discussion

This condition is only used in conjunction with the *SUSPENSION* liquid constitutive models, *HYDRODYNAMIC* diffusivity model, and *SUSPENSION* or *SUSPENSION_PM* density models. A theoretical outflux condition associated with suspension particles leaving the domain is tied to the Phillips diffusive-flux model. Please refer to discussions on *HYDRODYNAMIC* diffusivity to gain more understanding of the suspension flux model.

YFLUX_BV

```
BC = YFLUX_BV SS <bc_id> <integer1> <floatlist>
```

Description / Usage

(WIC/MASS)

The *YFLUX_BV* card enables computation of the molar flux of the specified species using Butler-Volmer kinetics at the specified boundary (namely, the electrode surface). When used in conjunction with the *KIN_LEAK* card, it also enables the determination of velocity normal to the moving solid-electrode surface.

The <floatlist> consists of nine values; definitions of the input parameters are as follows:

YFLUX_BV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Species number of concentration.
<float1>	Stoichiometric coefficient.
<float2>	Kinetic rate constant.
<float3>	Reaction order.
<float4>	Anodic direction transfer coefficient.
<float5>	Cathodic direction transfer coefficient.
<float6>	Electrode potential or applied voltage.
<float7>	Theoretical open-circuit potential.
<float8>	Molecular weight of solid deposit.
<float9>	Density of solid deposit.

Examples

The following is a sample input card:

```
BC = YFLUX_BV SS 1 0 -1. 0.00001 1. 0.21 0.21 -0.8 -0.22 58.71 8.9
```

Technical Discussion

No Discussion.

YFLUX_HOR

```
BC = YFLUX_HOR SS <bc_id> <integer> <floatlist>
```

Description / Usage

(WIC/MASS)

The **YFLUX_HOR** card enables computation of the molar flux of the specified species at the specified boundary (i.e., at the electrode surface) using the linearized Butler- Volmer kinetics such as that for the hydrogen-oxidation reaction in polymerelectrolyte- membrane fuel cells.

The <floatlist> consists of 10 values; definitions of the input parameters are as follows:

YFLUX_HOR	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float1>	Product of interfacial area per unit volume by exchange current density, ai_0, in units of A/cm^3.
<float2>	Catalyst layer or catalyzed electrode thickness, *H*, in unit of *cm*.
<float3>	Reference concentration, c_{ref}, in units of *moles*/ cm^3.
<float4>	Anodic direction transfer coefficient, α_a.
<float5>	Cathodic direction transfer coefficient, α_c.
<float6>	Temperature, *T*, in unit of *K*.
<float7>	Theoretical open-circuit potential, U_0, in unit of V.
<float8>	Reaction order, β.
<float9>	Number of electrons involved in the reaction, *n*.
<float10>	Electrode potential, *V*, in unit of *V*.

Examples

The following is a sample input card:

```
BC = YFLUX_HOR SS 14 0 1000. 0.001 4.e-5 1. 1. 353. 0. 0.5 2. 0.
```

Technical Discussion

For electrochemical reactions such as the hydrogen-oxidation reaction (HOR), surface overpotential is relatively small such that the Butler-Volmer kinetic model can be linearized to yield:

$$r = \frac{ai_0 H}{nRT}\left(\frac{c}{c_{ref}}\right)^{\beta}(\alpha_a + \alpha_c)(V - \Phi - U_0)$$

where *r* is the surface reaction rate in units of *moles*/ $cm^2 - s$; ai_0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/cm^3; *H* is the catalyst layer or catalyzed electrode thickness in unit of *cm*; *n* is the number of electrons involved in the electrochemical reaction; *R* is the universal gas constant ($\equiv$ 8.314 J/mole-K); T is temperature in unit of *K*; *c* and c_{ref} are, respectively, species and reference molar concentrations in units of *moles*/ cm^3; β is reaction order; α_a and α_c are, respetively, the anodic and cathodic transfer coefficients; *V* and ϕ are, respectively, the electrode and electrolyte potentials in unit of *V*; U_0 and is the opencircuit potential in unit of *V*.

References

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, NJ (1991).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

YFLUX_ORR

```
BC = YFLUX_ORR SS <bc_id> <integer> <floatlist>
```

Description / Usage

(WIC/MASS)

The **YFLUX_ORR** card enables computation of the molar flux of the specified species at the specified boundary (i.e., at the electrode surface) using the Tafel kinetics such as that for the oxygen-reduction reaction in polymer-electrolyte-membrane fuel cells.

The <floatlist> consists of 9 values; definitions of the input parameters are as follows:

YFLUX_ORR	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float1>	Product of interfacial area per unit volume by exchange current density, ai_0, in units of A/cm^3.
<float2>	Catalyst layer or catalyzed electrode thickness, *H*, in unit of *cm*.
<float3>	Reference concentration, c_{ref}, in units of *moles*/ cm^3.
<float4>	Cathodic direction transfer coefficient, α_c.
<float5>	Temperature, *T*, in unit of *K*.
<float6>	Electrode potential, *V*, in unit of *V*.
<float7>	Theoretical open-circuit potential, U_0, in unit of V.
<float8>	Reaction order, β.
<float9>	Number of electrons involved in the reaction, *n*.

Examples

The following is a sample input card:

```
BC = YFLUX_ORR SS 15 1 0.01 0.001 4.e-5 1. 353. 0.7 1.18 1. 4.
```

Technical Discussion

For electrochemical reactions such as the hydrogen-oxidation reaction (HOR), surface overpotential is relatively small such that the Butler-Volmer kinetic model can be linearized to yield:

$$r = -\frac{ai_0 H}{nF}\left(\frac{c}{c_{ref}}\right)^{\beta} e^{-\frac{\alpha_c F}{RT}(V-\Phi-U_0)}$$

where *r* is the surface reaction rate in units of *moles/* $cm^2 - s$; ai_0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/ cm^3; *H* is the catalyst layer or catalyzed electrode thickness in unit of *cm*; *n* is the number of electrons involved in the electrochemical reaction; *F* is the Faraday's constant ($\equiv$ 96487 *C/mole*); *c* and c_{ref} are, respectively, species and reference molar concentrations in units of *moles/* cm^3; β is reaction order; α_c is the anodic and cathodic transfer coefficient; *R* is the universal gas constant ($\equiv$ 8.314 *J/mole-K*); *T* is temperature in unit of *K*; *V* and ϕ are, respectively, the electrode and electrolyte potentials in unit of *V*; U_0 and is the open-circuit potential in unit of *V*.

References

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, NJ (1991).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

YFLUX_USER

```
BC = YFLUX_USER SS <bc_id> <integer> <float_list>
```

Description / Usage

(WIC/MASS)

This boundary condition card is used to set mass flux to a user-prescribed function and integrate by parts again. The user should provide detailed flux conditions in the *mass_flux_user_surf* routine in *user_bc.c*. The flux quantity is specified on a per mass basis so the heat and mass transfer coefficients are in units of L/t.

Definitions of the input parameters are as follows:

YFLUX_USER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following is a sample input card:

```
BC = YFLUX_USER SS 2   0 .5 .5
```

Technical Discussion

No Discussion.

References

No References.

YFLUX_ALLOY

```
BC = YFLUX_ALLOY SS <bc_id> <integer1> <float_list>
```

Description / Usage

(WIC/MASS)

This boundary condition card calculates the surface integral for a mass flux transfer model for the evaporation rate of molten metal.

The <float_list> requires six values; definitions of the input parameters are as follows:

YFLUX_ALLOY	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Species number.
<float1>	Liquidus temperature of metal alloy, T_m.
<float2>	Base Concentration, y^∞.
<float3>	Coefficient c_0.
<float4>	Coefficient c_1.
<float5>	Coefficient c_2.
<float6>	Coefficient c_3.

Examples

The following is a sample input card:

```
BC = YFLUX_ALLOY SS 10 0 1623.0 0.5 0.01 -1e-3 1e-4 -1e-5
```

Technical Discussion

Basically the difference between this model and the simple convective mass transfer coefficient (say k_i for *YFLUX*) is that the transfer coefficient here (the exponential term) has a cubic dependence on temperature.

$$n \cdot j_i = \exp[c_0 + c_1(T - T_m) - c_2(T - T_m)^2 + c_3(T - T_m)^3] \cdot (y_i - y_i^{\infty})$$

YTOTALFLUX_CONST

```
BC = YTOTALFLUX_CONST SS <bc_id> <integer> <float>
```

Description / Usage

(WIC/MASS)

This boundary condition card is used to specify a constant total mass flux (including contribution from diffusion, migration, and convection) of a given species. This card enables the treatment of the situation in which diffusion, migration and convection fluxes cancel each other such that the total flux vanishes (e.g. is equal to zero). This flux quantity can be specified on a per mass basis (i.e., with units of g/ cm^2/s) or on a per mole basis (e.g. with units of moles/cm^2/s), depending on the user's choice of units in the species concentration unknown.

Definitions of the input parameters are as follows:

YTOTALFLUX_CONST	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float>	Value of total mass flux - the units of this quantity depends on the user's choice of units for species concentration.

Examples

Following is a sample card:

```
BC = YTOTALFLUX_CONST SS 5   0   0.0
```

Technical Discussion

No Discussion.

VL_EQUIL

```
BC = VL_EQUIL SS <bc_id> <integer_list> <float_list>
```

Description / Usage

(SIC/MASS)

This boundary condition card enforces vapor-liquid equilibrium between a gas phase and a liquid phase using Raoult's law. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

The <integer_list> has three values and the <float_list> has five values; definitions of the input parameters are as follows:

VL_EQUIL	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Species number.
<integer2>	Element block ID of liquid phase.
<integer3>	Element block ID of gas phase.
<float1>	Base ambient pressure in gas phase.
<float2>	Molecular weight of first volatile species.
<float3>	Molecular weight of second volatile species.
<float4>	Molecular weight of condensed phase.
<float5>	Molecular weight of insoluble gas phase.

This boundary condition is applied to ternary, two-phase flows that have rapid mass exchange between phases, rapid enough to induce a diffusion velocity at the interface, and to thermal contact resistance type problems. The best example of this is rapid evaporation of a liquid component into a gas. In the current discontinuous mass transfer model, we must require the same number of components on either side of interface. In this particular boundary, two of three components are considered *volatile*, so they participate in both vapor and liquid phases. The third component is considered either non-volatile or non-condensable, so it remains in a single phase.

Examples

A sample input card follows for this boundary condition:

```
BC = VL_EQUIL SS 4 0 1 2 1.e+06 28. 18. 1800. 18.
```

The above card demonstrates these characteristics: species number is “0”; liquid phase block id is 1; gas phase block id is 2; ambient pressure is 1.e6 Pa; the molecular weights of the volatile species are 28 and 18; of the condensed phase and insoluble portion of the gas phase, 1800 and 18, respectively.

Technical Discussion

One of the simplest forms of the equilibrium relation is the Raoult’s law, where the mole fraction of a species is equal to its mole fraction in the liquid multiplied by the ratio of its pure component vapor pressure to the total pressure in the system.

$$y_i P^{total} = x_i P^v \quad \forall \; i$$

where y_i are the mole fraction of species *i* in the gas phase and x_i is the mole fraction in the liquid phase. The molecular weights required in this boundary card are used for converting mass fractions to mole fractions. The temperature dependency in the equilibrium expression comes from a temperature-dependent vapor pressure model. Either Riedel or Antoine temperature-dependent vapor pressure model can be specified in the *VAPOR PRESSURE* material card in order to link temperature to Raoult’s law.

References

GTM-007.1: New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

Schunk, P.R. and Rao, R.R. 1994. “Finite element analysis of multicomponent twophase flows with interphase mass and momentum transport,” IJNMF, 18, 821-842.

VL_POLY

```
BC = VL_POLY SS <bc_id> {char_string} <integer_list> <float>
```

Description / Usage

(SIC/MASS)

This boundary condition card enforces vapor-liquid equilibrium between a gas phase and a liquid phase using Flory-Huggins activity expression to describe polymer-solvent mixtures. The condition only applies to interphase mass, heat, and momentum transfer problems with discontinuous (or multivalued) variables at an interface, and it must be invoked on fields that employ the **Q1_D** or **Q2_D** interpolation functions to "tie" together or constrain the extra degrees of freedom at the interface in question.

There are three input values in the <integer_list>; definitions of the input parameters are as follows:

VL_POLY	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
{char_string}	the concentration basis; two options exist: • **MASS** - the concentration variable in *Goma* is equivalent to mass fractions. • **VOLUME** - the concentration variable in *Goma* is based on volume fractions for all species.
<integer1>	Species number of concentration.
<integer2>	Element block id that identifies the liquid phase.
<integer3>	Element block id that identifies the vapor phase.
<float>	Total pressure of the system.

Examples

This is a sample input card for this boundary condition:

```
BC = VL_POLY SS 7 MASS 0 1 2 1.e+05
```

Technical Discussion

One of the simplest forms of the equilibrium relation is the Raoult's law, where the mole fraction of a species is equal to its mole fraction in the liquid multiplied by the ratio of its pure component vapor pressure to the total pressure in the system.

$$y_i P^{total} = \phi_i \gamma_i P_i^v \qquad \forall i$$

$$\sum_{i=1}^{N} \phi_i = 1$$

γ_i is defined as the activity coefficient of species *i* and is considered a departure function from the Raoult's law. The fugacity in the liquid is reformulated in terms of volume fraction ϕ_i for polymer mixtures to avoid referencing the molecular weight of polymer (Patterson, et. al., 1971).

Based on an energetic analysis of excluded volume imposed by the polymer, the activity coefficient model of Flory-Huggins is widely used for polymer-solvent mixtures (Flory, 1953). The general form of the Flory-Huggins model for multicomponent mixtures is a summation of binary interactions terms; i.e.,

$$\ln\gamma_i = \sum_{k=1}^{N}\left[\left(\frac{\bar{v}_i}{\bar{v}_k}\right)(\delta_{ki}-\phi_i)\right] + \sum_{k=2}^{N}\sum_{j=1}^{k-1}\left[\delta_{ij}\phi_k\chi_{jk} + \left(\frac{\bar{v}_i}{\bar{v}_j}\right)\phi_j(\delta_{ki}-\phi_k)\chi_{jk}\right]$$

v_i is the molar volume of component *i* (or the average-number molar volume if *i* is a polymer). ζ_{ki} is the Dirac delta. χ_{jk} is known as the Flory-Huggins interaction parameter between components *j* and *k*, and is obtainable by fitting the solubility data to the above model. For a simple binary pair (solvent (1)-polymer (2)) and assuming $v_2 \gg v_1$, the above model reduces to a simpler form.

$$\ln\gamma_1 = \phi_2 + \chi_{12}\phi_2^2$$

References

Flory, P., Principles of Polymer Chemistry, Cornell University Press, New York (1953)

Patterson, D., Y.B. Tewari, H.P. Schreiber, and J.E. Guillet, "Application of Gas-Liquid Chromatography to the Thermodynamics of Polymer Solutions,"Macromolecules, 4, 3, 356-358 (1971)

GTM-007.1: New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

VL_EQUIL_PSEUDORXN

```
BC = VL_EQUIL_PSEUDORXN SS <bc_id> <integer_list> <float>
```

Description / Usage

(WIC/MASS)

This boundary condition card enforces vapor-liquid equilibrium between a gas phase and a liquid phase species component using Raoult's law expressed via a finite-rate kinetics formalism. The condition only applies to problems containing internal interfaces with discontinuous (or multilevel) species unknown variables. The species unknown variable must employ the **Q1_D** or **Q2_D** interpolation functions in both adjacent element blocks. This boundary condition constrains the species equations on both sides of the interface (i.e., supplies a boundary condition) by specifying the interfacial mass flux on both sides.

Definitions of the input parameters are as follows:

VL_EQUIL_PSEUDORXN	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Species number.
<integer2>	Element Block ID of the liquid phase.
<integer3>	Element Block ID of the gas phase.
<float>	Rate constant for the forward reaction in units of length divided by time.

This boundary condition is typically applied to multicomponent two-phase flows that have rapid mass exchange between phases. The best example of this is rapid evaporation of a liquid component into a gas.

Examples

The following sample input card

```
BC = VL_EQUIL_PSEUDORXN SS 4 0 1 2 100.
```

demonstrates the following characteristics: species number is "0"; liquid phase element block id is "1"; gas phase element block id is "2"; a forward rate constant of 100.0 cm s^{-1}.

Technical Discussion

The *VL_EQUIL_PSEUDORXN* boundary condition uses the following equations representing a kinetic approach to equilibrium expressed by Raoult's law, relating species *k* on the liquid side to species *k* on the gas side.

$$n_l \bullet [\rho^l Y_k^l (\boldsymbol{u}^l - \boldsymbol{u}_s) + \boldsymbol{j}_k^l] = -W_k^l S_k^l$$

$$n_g \bullet [\rho^g Y_k^g (\boldsymbol{u}^g - \boldsymbol{u}_s) + \boldsymbol{j}_k^g] = -W_k^g S_k^g$$

where

$$S_k^l = k^f [C_k^g - K_k^c C_k^l] \quad \text{and} \quad S_k^g = -S_k^l$$

and where

The usage of the same index, *k*, on either side of the interface is deliberate and represents a stoichiometric limitation to this type of boundary condition. Y_k^l and Y_k^g are the mass fraction of species *k* on the liquid and gas sides of the interface, respectively. W_k^l is the molecular weight of species *k*. S_k^l is the source term for creation of species *k* in the liquid phase at the interface (mol cm^{-2} s^{-1}). is the pseudo reaction rate (cms^{-1}) input from the boundary condition card. K_k^c is the concentration equilibrium constant, which for the restricted stoichiometry cases covered by this boundary condition, is unitless. p_k^v is the vapor pressure of gas species *k* above a liquid entirely consisting of liquid species *k*. It is a function of temperature. $\tilde{C}_l$ is the average concentration in the liquid (mol cm^{-3}). C_k^l and C_k^g are the liquid and gas concentrations of species *k* (mol cm^{-3}).

The choice for the independent variable is arbitrary, although it does change the actual equation formulation for the residual and Jacobian terms arising from the boundary condition. The internal variable Species_Var_Type in the Uniform_Problem_Description structure may be queried to determine what the actual species independent variable is. Also note, if mole fractions or molar concentration are chosen as the independent variable in the problem, the convention has been to formulate terms of the residuals in units of moles, cm, and seconds. Therefore, division of the equilibrium equations by W_k would occur before their inclusion into the residual. j_k^l and j_k^g are the diffusive flux of species k (gm cm^{-2} s^{-1}) relative to the mass averaged velocity. u_s is the velocity of the interface. A typical value of k^f that would lead to good numerical behavior would be 100 cm s^{-1}, equivalent to a reaction with a reactive sticking coefficient of 0.01 at 1 atm and 300 K for a molecule whose molecular weight is near to N_2 or H_2S.

$$K_k^c = \frac{p_k^v}{RT} \frac{1}{\tilde{C}^l}$$

IS_EQUIL_PSEUDORXN

```
BC = IS_EQUIL_PSEUDORXN SS <bc_id> <integer_list> <float>
```

Description / Usage

(WIC/MASS)

This boundary condition card enforces equilibrium between a species component in two ideal solution phases via a finite-rate kinetics formalism. The condition only applies to problems containing internal interfaces with discontinuous (or multilevel) species unknown variables. The species unknown variable must employ the **Q1_D** or **Q2_D** interpolation functions in both adjacent element blocks. This boundary condition constrains the species equations on both sides of the interface (i.e., supplies a boundary condition) by specifying the interfacial mass flux on both sides.

IS_EQUIL_PSEUDORXN is equivalent to the *VL_EQUIL_PSEUDORXN* except for the fact that we do not assume that one side of the interface is a gas and the other is a liquid. Instead, we assume that both materials on either side of the interface are ideal solutions, then proceed to formulate an equilibrium expression consistent with that.

The <integer_list> requires three values; definitions of the input parameters are as follows:

IS_EQUIL_PSEUDORXN	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Species number.
<integer2>	Element Block ID of the first phase, the "+" phase.
<integer3>	Element Block ID of the second phase, the "-" phase.
<float>	Rate constant for the forward reaction in units of length divided by time.

Examples

The sample card:

```
BC = IS_EQUIL_PSEUDORXN SS 4 0 1 2 100.
```

demonstrates the following characteristics: species number is “0”; the “+” phase element block id is “1”; the “-” phase element block id is “2”; a forward rate constant of 100. cm s^{-1}.

Technical Discussion

The *IS_EQUIL_PSEUDORXN* boundary condition uses the following equations representing a kinetic approach to equilibrium expressed by an ideal solution model for thermodynamics on either side of the interface. Initially, we relate species *k* on the + side to species *k* on the - side of the interface via a kinetic formulation, whose rate constant is fast enough to ensure equilibrium in practice. However, later we may extend the capability to more complicated stoichiometric formulations for equilibrium, since the formulation for the equilibrium expression is readily extensible, unlike *Goma's* previous treatment.

$$n^{+} \bullet [\rho^{+} Y_k^{+}(\boldsymbol{u}^{+} - \boldsymbol{u}_s) + \boldsymbol{j}_k^{+}] = -W_k^{+} S_k^{+}$$

$$n^{-} \bullet [\rho^{-} Y_k^{-}(\boldsymbol{u}^{-} - \boldsymbol{u}_s) + \boldsymbol{j}_k^{-}] = -W_k^{-} S_k^{-}$$

where

$$S_k^{+} = k^f \left[C_k^{-} - \frac{C_k^{+}}{K_k^c} \right] \quad \text{and} \quad S_k^{-} = -S_k^{+}$$

The “-” phase is defined as the reactants, while the “+” phase is defined to be the products. The expression for the concentration equilibrium constant, K_k^c , is based on the ideal solution expression for the chemical potentials for species *k* in the two phases [Denbigh, p. 249],

where μ_k^{+*} (T,P) is defined as the chemical potential of species *k* in its pure state (or a hypothetical pure state if a real pure state doesn't exist) at temperature *T* and pressure *P*. μ_k^{+*} (T,P) is related to the standard state of species *k* in phase

$$\mu_k^+ = RT\ln(X_k^+) + \mu_k^{+*}(T,P)$$

+, μ_k^+, $\underline{o}$ + (T) , which is independent of pressure, through specification of the pressure dependence of the pure species k. Two pressure dependencies are initially supported:

PRESSURE_INDEPENDENT $\mu_k^{+*}(T,P) = \mu_{k,o}^+(T)$

IDEAL_GAS $\mu_k^{+*}(T,P) = \mu_{k,o}^+(T) + RT\ln(P/1\text{ atm})$

With these definitions, K_k^c can be seen to be equal to

where

The chemical potential for a species in a phase will be calculated either from CHEMKIN or from the *Chemical Potential, Pure Species Chemical Potential, and Standard State Chemical Potential cards* in the materials database file.

The choice for the independent variable for the species unknown is relatively arbitrary,although it does change the actual equation formulation for the residual and Jacobian terms arising from the boundary condition. The internal variable Species_Var_Type in the Uniform_Problem_Description structure is queried to determine what the actual species independent variable is. A choice of SPECIES_UNDEFINED_FORM is unacceptable. If either mole fractions or molar concentration is chosen as the independent variable in the problem, the convention has been to formulate terms of the residuals in units of moles, cm, and seconds. Therefore, division of the equilibrium equations W_k by occurs before their inclusion into the residual. J_k^l and J_k^g are the diffusive flux of species *k* (gm cm^{-2} s^{-1}) relative to the mass-averaged velocity. u^s is the velocity of the interface. A typical value of k^f that would lead to good numerical behavior would be 100 cm s^{-1}, equivalent to a reaction with a reactive sticking coefficient of 0.01 at 1 atm and 300 K for a molecule whose molecular weight is near to N_2 or H_2S.

$$K_k^c = \frac{C^+}{C^-} \exp\left[\frac{-\Delta G_k^*}{RT}\right]$$

$$\Delta G_k^* = \mu_k^{+*}(T,P) - \mu_k^{-*}(T,P) \quad .$$

References

Denbigh, K., The Principles of Chemical Equilibrium, Cambridge University Press, Cambridge, 1981

SURFACE_CHARGE

```
BC = SURFACE_CHARGE SS <bc_id> <integer> <float>
```

Description / Usage

(SIC/MASS)

The *SURFACE_CHARGE* card specifies the electrostatic nature of a surface: electrically neutral, positively charged or negatively charged.

Definitions of the input parameters are as follows:

SURFACE_CHARGE	Name of the boundary condition (<bc_name>).	
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.	
<integer>	Index of species to which surface charge condition applies.	
<float>	z, value of surface charge.	
	0	electroneutrality
	positive z	positively charged surface
	negative z	negatively charged surface

Examples

The following input card indicates that on side set 1 species 1 is electrically neutral:

```
BC = SURFACE_CHARGE SS 1 1 0.0
```

Category 7: Continuity Equation

The continuity equation rarely requires a boundary condition as it represents an overall mass balance constraint on the velocity field for the fluid, viz. normally it is used to enforce incompressibility. Boundary conditions for pressure are most often put on the fluid-momentum equations as a part of the stress condition at an inflow or outflow plane (see for example boundary condition cards *FLOW_PRESSURE, FLOW_HYDROSTATIC*, etc.). On occasion, however, we can use a pressure condition as a pressure datum, as the Dirichlet pressure condition below allows, though the user must keep in mind that it is a condition on continuity and not momentum. When using pressure stabilization, viz. PSPG techniques, then also there is an occasional need for a boundary condition on this equation.

P

```
BC = P NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/CONTINUITY)

This Dirichlet boundary condition specification is used to set a constant pressure on a node set. It is generally used for specifying a pressure datum on a single-node node set. The pressure datum is useful for setting the absolute value of the pressure, which, for many problems, is indeterminate to a constant. Pressure datums are especially important for closed flow problems, such as the lid driven cavity, where there is no inflow or outflow. Mass conservation problems can arise if this card is used to specify the pressure along a group of nodes, since this equation replaces the continuity equation. To specify pressure for a group of nodes, it is preferable to use the flow pressure boundary condition, which is applied in a weak sense to the momentum equation and does not cause mass conservation problems. Definitions of the input parameters are as follows:

P	One-character boundary condition name (<bc_name>) that defines the pressure.
NS	Type of boundary condition (<bc_type>), where **NS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of pressure.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following are sample cards for specifying a pressure Dirichlet card:

```
BC = P NS 7   0.
```

```
BC = P NS 7   0. 1.0
```

where the second form is an example using the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity for a discussion of the two ways of applying Dirichlet boundary conditions.

PSPG

```
BC = PSPG SS <bc_id>
```

Description / Usage

(WIC/CONTINUITY)

This special type of boundary condition exists for pressure-stabilized incompressible flow simulations only. This card should be used only if the value of the *Pressure Stabilization* card has been set to *yes*. In conjunction with this feature, equal-order interpolation should be used for the velocity and pressure. If *PSPG* is used, a boundary integral will be added to the continuity equation to represent the gradients of velocity in the momentum residual, which has been added onto the continuity equation for stabilization. This term is only needed on inflow and outflow boundaries; in the rest of the domain, it cancels out. For more details about the derivation of this term, see the paper by Droux and Hughes (1994).

This boundary condition card requires no integer or floating point constants. Definitions of the input parameters are as follows:

PSPG	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

Examples

The following is an example of using this card on both the inflow and outflow planes of the domain.

```
BC = PSPG    SS    40
```

```
BC = PSPG    SS    20
```

Technical Discussion

Please see Rao (1996) memo for a more detailed discussion of pressure stabilization and its implementation in *Goma*.

References

GTM-001.0: Pressure Stabilization in Goma using Galerkin Least Squares, July 17, 1996, R. R. Rao

Droux, J. J. and T. J. R. Hughes, "A Boundary Integral Modification of the Galerkin Least Squares Formulation for the Stokes Problem, " Comput. Methods Appl. Mech. Engrg., 113 (1994) 173-182.

PRESSURE DATUM

```
PRESSURE DATUM = <integer> <float>
```

Description / Usage

This card is used to set a hydrodynamic pressure datum on fluid mechanics problems that contain no implicit or explicit boundary conditions on stress or pressure. Definitions of the input parameters are as follows:

<integer>	Element number on which the datum is set. This number should correspond to that shown when viewing the mesh, less one, as the numbering convention in the C language starts at zero rather than at one.
<float>	Value of the hydrodynamic pressure datum.

Noteworthy is that this card is optional, and if used, is placed outside the BC section and just below it.

Examples

Following is a sample card:

```
PRESSURE DATUM = 10 1.
```

Technical Discussion

No Discussion.

Category 8: Porous Equations

The following conditions are applied as boundary conditions to the porous-flow equations. These conditions include strong Dirichlet conditions, such as hard sets on porous phase pressure on a boundary as a constant or function of position, weak-form conditions, such as a specified phase flux from a convective mass transfer model or a constant flux, and a host of interfacial conditions for impregnation, etc. The porous flow equations are actually scalar equations that represent component mass balances. Specifically, there is one component mass balance for the liquid phase, one for the gas phase, and one for the solid phase. The corresponding three dependent variables in these balances are the liquid phase pressure, the gas phase pressure, and the porosity, respectively. These variables are related to the flow through a boundary by their normal gradients (Darcy's law formulation) and to the local inventory of liquid and gas through the saturation function. These implicit terms can often lead to some confusion in setting the boundary conditions so it is recommended that the user consult the supplementary documentation referenced in the following porous boundary condition cards.

POROUS_LIQ_PRESSURE

```
BC = POROUS_LIQ_PRESSURE NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/POR_LIQ_PRES)

This Dirichlet boundary condition is used to set the liquid phase pore pressure at a noden set. It can be applied to a node set on a boundary of a *POROUS_SATURATED, POROUS_UNSATURATED* or *POROUS_TWO_PHASE* medium type (see *Media Type* card).

POROUS_LIQ_PRESSURE	Boundary condition name (bc_name).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of liquid phase pressure.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The boundary condition card

```
BC = POROUS_LIQ_PRESSURE NS 101 {pcmin}
```

sets the porous liquid pressure at the boundary denoted by node set 101 to the value represented by the APREPRO variable {pcmin}.

Technical Discussion

Setting the porous liquid pressure to a value cannot be done independently of the saturation as the two are related through the vapor pressure curve for simulations in partially saturated media (see *Saturation* model card). Keep in mind that when using this card in these situations, you are setting also the saturation level based on the capillary pressure, defined as p_{gas} - p_{liq} = p_c . The convention in *Goma* is that when the capillary pressure p_c is greater than zero, the saturation level is less than unity, viz. the medium is partially saturated. When p_c is less than zero, i.e., when the liquid phase pressure is greater than the gas phase pressure, then the medium is saturated (in this case the capillary pressure is poorly defined, though). Also, for *Media Type* options of *POROUS_UNSATURATED*, the ambient gas pressure is constant within the pore space and is set by the *Porous Gas Constants* card in the material file. This boundary condition, when setting the liquid phase pressure, must be used with consideration of these definitions.

For saturated media (viz. *Media Type* of *POROUS_SATURATED*), this discussion is not relevant. In this case, one must only consider the pressure level as it may effect the isotropic stress in poroelastic problems.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

POROUS_LIQ_FLUX_CONST

```
BC = POROUS_LIQ_FLUX_CONST SS <bc_id> <float1> [float2]
```

Description / Usage

(WIC/POR_LIQ_PRES)

This boundary condition sets the flux of liquid-phase solvent to a constant value in the Galerkin finite element weak sense. Specifically, this flux is applied to a side set as a weak-integrated constant and will set the net flux of liquid phase solvent component (in both gas and liquid phases) to a specified value. It can be applied to material regions of *Media Type POROUS_SATURATED, POROUS_UNSATURATED*, and *POROUS_TWO_PHASE* (see Technical Discussion below).

Definitions of the input parameters are as follows:

POROUS_LIQ_FLUX_CONST	Name of boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Value of the liquid-solvent total flux, in M/ L^2 -t.
[float2]	This optional parameter is not applicable to this boundary condition type, even though it is parsed if present. This parameter is used for boundary conditions of the Dirichlet type.

Examples

The input card

```
BC = POROUS_LIQ_FLUX_CONST SS 102 200.0
```

sets the total liquid-solvent mass flux, in both gas and liquid phases, to 200.0 along the side set 102.

Technical Discussion

This boundary condition is of the mathematical form:

$$\rho_l^T n \cdot (v_s)\phi - \left(-\frac{\rho_l k k_l}{\mu_l}\nabla p_l\right) = const$$

where v_s is the user supplied convection velocity of the stress-free state as defined on the *Convective Lagrangian Velocity* card (this is usually zero except in advanced cases), p_l^T is the total bulk density of liquid phase solvent (in both gas and liquid phase, and hence depends on the local saturation), p_l is the pure liquid density, ϕ is the porosity, p_l is the liquid phase pressure, and the other quantities on the second term help define the Darcy velocity. The *const* quantity is the input parameter identified above (<float1>). Note that this sets the flux relative to the boundary motion to the *const* value, but by virtue of the Galerkin weak form this condition is automatically applied with *const* =0 if no boundary condition is applied at the boundary. In a saturated case, viz. *POROUS_SATURATED* media type, this condition is applied as

$$\rho_l n \cdot (v_s)\phi - \left(-\frac{\rho_l k}{\mu_l}\nabla p_l\right) = const \quad .$$

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

POROUS_GAS_PRESSURE

```
BC = POROUS_GAS_PRESSURE NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/POR_GAS_PRES)

This Dirichlet boundary condition is used to set the gas-phase pore pressure at the boundary of a *POROUS_TWO_PHASE* medium type (see *Media Type* card). This condition makes no sense on other *POROUS Media Types*; the gas pressure in those cases is constant and set using the *Porous Gas Constants* card (*Microstructure Properties*).

POROUS_GAS_PRESSURE	Boundary condition name (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float>	Value of gas phase pressure.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The boundary condition card

```
BC = POROUS_GAS_PRESSURE NS 101 {pgas}
```

sets the porous gas pressure at the boundary denoted by node set 101 to the value represented by the APREPRO variable {pgas}.

Technical Discussion

Setting the porous liquid pressure to a value cannot be done independently of the saturation as the two are related through the vapor pressure curve for simulations in partially saturated media (see *Saturation* model card). Keep in mind that when using this card in these situations, you are setting also the saturation level based on the capillary pressure, defined as p_{gas} - p_{liq} = p_c . The convention in Goma is that when the capillary pressure p_c is greater than zero, the saturation level is less than unity, viz.the medium is partially saturated. When p_c is less than zero, i.e., when the liquid phase pressure is greater than the gas phase pressure, then the medium is saturated (in this case the capillary pressure is poorly defined, though). Also, this pressure sets the datum of pressure for deformable porous media and must be set in a manner compatible with the solid-stress values on the boundaries of the porous matrix.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

POROUS_GAS

```
BC = POROUS_GAS SS <bc_id> <integer_list> <float_list>
```

Description / Usage

(SIC/POR_LIQ_PRES)

This boundary condition card is used to equate flux of solvent in the porous medium and external gas. The condition is similar to the solid-liquid interface conditions that apply to interfaces between a porous medium and an external gas (in which the energy equation is used to solve for solvent concentration in the gas phase). This boundary condition is still in development.

There are three values in the <integer_list> and two values in the <float_list> for which to supply values; definitions of the input parameters are as follows:

POROUS_GAS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid, porous phase from the EXODUS II database.
<integer2>	Element block ID of gas phase from the EXODUS II database.
<integer3>	Species number of liquid phase in porous medium.
<float1>	Vapor density.
<float2>	Factor to allow normal velocity in gas.

Examples

Users are referred to the Cairncross (1999) reference for the best example of card usage.

Technical Discussion

This highly specialized boundary condition is best explained in a paper by Cairncross (1999).

References

GTM-028.0: Modeling Drying of Dip-Coated Films with Strongly-Coupled Gas Phase Natural Convection, R. A. Cairncross, 1999.

POROUS_GAS_FLUX_CONST

```
BC = POROUS_GAS_FLUX_CONST SS <bc_id> <float1> [float2]
```

Description / Usage

(WIC/POR_GAS_PRES)

This boundary condition card is used to set the flux of gas-phase solvent to a constant value in the Galerkin finite element weak sense. Specifically, this flux is applied to a side set as a weak-integrated constant and will set the net flux of gas phase solvent component (in both gas and liquid phases, but because the gas solvent is assumed insoluble in the liquid phase, the liquid phase portion vanishes) to a specified value. This boundary condition can be applied to material regions of *Media Type POROUS_TWO_PHASE* only, as only this type contains a field of gas-phase solvent flux. (See technical discussion below).

Definitions of the input parameters are as follows:

POROUS_GAS_FLUX_CONST	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	Value of the gas-solvent total flux, in M/ L^2 -t.
[float2]	This optional parameter is not applicable to this boundary condition type, even though it is parsed if present. This parameter is used for boundary conditions of the Dirichlet type.

Examples

The input card

```
BC = POROUS_LIQ_FLUX_CONST SS 102 200.0
```

sets the total gas-solvent mass flux, in the gas phase only, to 200.0 along the side set 102.

Technical Discussion

This boundary condition is of the mathematical form:

$$\rho_g^T n \cdot (v_s)\phi - \left(-\frac{\rho_g k k_g}{\mu_g} \nabla p_g \right) = const$$

where v_s is the user supplied convection velocity of the stress-free state as defined on the *Convective Lagrangian Velocity* card (this is usually zero except in advanced cases), p_g^T is the total bulk density of gas phase solvent (in both gas and liquid phase, and hence depends on the local saturation), p_g is the pure gas density, ϕ is the porosity, p_g is the gas-phase pressure, and the other quantities on the second term help define the Darcy velocity. The *const* quantity is the input parameter described above (<float1>). Note that this sets the flux relative to the boundary motion to the *const* value, but by virtue of the Galerkin weak form this condition is automatically applied with *const = 0* if no boundary condition is applied at the boundary.

POROUS_CONV

```
BC = POROUS_CONV SS <bc_id> <integer>
```

Description / Usage

(WIC/POR_LIQ_PRES)

This boundary condition is used to set the total flux of the liquid phase solvent (in both the gas and liquid phase) at the surface of a *POROUS_UNSATURATED* or *POROUS_TWO_PHASE* medium to the net convection of solvent due to a superimposed convective Lagrangian velocity (see *Media Type* card and *Convective Lagrangian Velocity* card). The only input is an integer indicating which component of the liquid phase solvent is to be set (*as of 11/2/01 this component selectability option is not available and as indicated below should be set to zero; this card has not been tested*).

Definitions of the input parameters are as follows:

POROUS_CONV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of transported species. (currently only used for multicomponent species in the phases, which as of 11/2/01 is not active, so set to zero).

Examples

Following is a sample card:

```
BC = POROUS_CONV SS 12 0
```

that applies a convective flux to side set 12 for porous liquid phase species 0. This species number is currently not used and ignored.

Technical Discussion

This boundary condition has the following form

$$(n \cdot (v_g \rho_{g_i} + v_l \rho_{l_i} + J_{g_i} + J_{l_i})) = (n \cdot v_s)\rho_{g_i}$$

where the left hand side is the total flux of the solvent i in the medium, which includes, in order, the flux due to Darcy flow of gas vapor, the Darcy flow of liquid solvent, the diffusive flux of gas vapor in the pore space and the diffusive flux of liquid solvent in the liquid phase. v_s is the user supplied convection velocity of the stress-free state as defined on the *Convective Lagrangian Velocity* card. As of now (11/2/01), this condition is used for a single component liquid solvent and has not been furbished for a single component of that solvent. Also, as of 11/02/01 the condition has not been tested.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

POROUS_FLUX

```
BC = POROUS_FLUX SS <bc_id> <integer> <float_list>
```

Description / Usage

(WIC/POR_LIQ_PRES)

This boundary condition is used to set the total flux of the liquid phase solvent (in both the gas and liquid phase) at the surface of a *POROUS_UNSATURATED* or *POROUS_TWO_PHASE* medium to mass transfer coefficient times driving force (see *Media Type* card). The flux quantity is specified on a per mass basis so the mass transfer coefficient is in units of L/t, and the sink density is in units of M/ L^3.

The <float_list> for this boundary condition has four values; the definitions of the input parameters are as follows:

POROUS_FLUX	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of transported species (currently only used for multicomponent species in the phases, which as of 11/2/01 is not active; so set to zero).
<float1>	Value of mass transfer coefficient, h_1 in units of L/t, consistent with gas phase concentration driving force.
<float2>	Driving force concentration in external phase, i.e., sink density, p^0_{gt} in units of M/ L^3.
<float3>	Value of pressure-driven mass transfer coefficient, h_2 in units of 1/L, for a liquid exiting a partially saturated domain.
<float4>	Driving force concentration in external phase, i.e., sink pressure for liquid extraction, p^0_{liq} in units of M/L/t.

Examples

Following is a sample card:

```
BC = POROUS_FLUX SS 12 0 0.03 0. 0. 0. 0.
```

This card applies the lumped mass transfer model for the liquid phase solvent with a mass transfer coefficient of 0.03 and a sink density of 0.0 for the total flux. The boundary condition is applied to side set 12 and to the species number 0. This species number is currently not used and ignored.

Technical Discussion

The mathematical form for this boundary condition is as follows

$$n \cdot (v_g \rho_{g_i} + v_l \rho_{l_i} + J_{g_i} + J_{l_i}) =$$

$$= h_1 \phi(\rho_{g_i} - \rho^0_{g_i}) + H(p_{liq} - p^0_{liq}) h_2 \phi(p_{liq} - p^0_{liq}) - (n \cdot v_s)\rho_{g_i}$$

where the left hand side is the total flux of the liquid solvent i in the medium, which includes, in order, the flux due to Darcy flow of gas vapor, the Darcy flow of liquid solvent, the diffusive flux of gas vapor in the pore space and the diffusive flux of liquid solvent in the liquid phase. The parameters are h_1, p^0_g , h_2 , $pliq^0$ and as defined on the input card. v_s is the user supplied convection velocity of the stress-free state as defined on the *Convective Lagrangian Velocity* card.

At the present time (11/2/01), this condition is only used for single component liquid phases and has not been furbished for multicomponent capability yet. Note that usually the second term on the right is turned off, as in the example above, unless the liquid pressure at the surface of the sample is greater than the external pressure. This term was added for applications in which liquid is being squeezed out of a medium and then drips off or disappears, as liquid is not allowed to be sucked back in (Heaviside function, *H*), although the condition could be furbished for this.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

POROUS_PRESSURE

```
BC = POROUS_PRESSURE SS <bc_id> <integer1> <integer2>
```

Description / Usage

(PCC/POR_LIQ_PRES)

This condition enforces a continuous fluid-phase pressure between material types, and is applied to a side set between two materials, one of type *POROUS_SATURATED, POROUS_UNSATURATED*, or *POROUS_TWO_PHASE*, and the other of type *CONTINUOUS* (see material card *Media Type*). Basically it sets the continuity of hydrodynamic pressure in the continuous fluid to the liquid Darcy pressure in the porous medium, at the interface. The input data is as follows:

POROUS_PRESSURE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **NS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<integer1>	Element block ID of the porous phase medium.
<integer2>	Element block ID of the continuous fluid phase medium.

Examples

An example input card for this boundary condition follows:

```
BC = POROUS_PRESSURE NS 101   1   2
```

This card sets the Darcy liquid phase pressure (*p_liq* in the output EXODUS II file) in element block 1 equal to the continuous phase hydrodynamic pressure (*P* in the output EXODUS II file) in element block 2.

Technical Discussion

The mathematical form for this boundary condition is as follows

$$p_{pore} = p_{continous}$$

but its implementation is not; a memo describing the details of this boundary condition and how it is applied is cited below. This continuity of pressure is critical for the sensitivity of pressurizing the continuos phase to the penetration rate

of the porous phase. Interestingly, it forces one to set the pore-phase pressure datum to the same datum in the continuous phase, and that effects the level of the Saturation versus capillary pressure curve (see *Saturation* material card).

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

P_LIQ_USER

```
BC = P_LIQ_USER SS <bc_id> <float_list>
```

Description / Usage

(PCC/R_POR_LIQ_PRES)

This boundary condition card is used to call a routine for a user-defined liquid-phase pressure for porous flow problems at an external boundary of a material of one of the following media types: *POROUS_SATURATED, POROUS_UNSATURATED, POROUS_TWO_PHASE.*. Specification is made via the function p_liq_user_surf in file "user_bc.c." Definitions of the input parameters are as follows:

P_LIQ_USER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutine so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function in file user_bc.c.

Examples

The following is a sample input card with two parameters passed to function tuser:

```
BC =P_LIQ_USER SS 100   273.13 100.0
```

Technical Discussion

No Discussion.

References

No References.

POROUS_TEMPERATURE

```
BC = POROUS_TEMPERATURE NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/POR_TEMP)

This Dirichlet boundary condition is used to set the temperature for a nonisothermal porous media problem at a node set. It can be applied to a node set on a boundary of a *POROUS_SATURATED, POROUS_UNSATURATED* or *POROUS_TWO_PHASE* medium type (see *Media Type* card).

POROUS_TEMPERATURE	Boundary condition name (bc_name).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of temperature at the NS in the porous medium.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

An example input card for this boundary condition follows:

```
BC = POROUS_TEMPERATURE NS 101   1.0   1.0
```

This card sets the temperature(*p_temp* in the output EXODUS II file) in element block 1 at the nodes defined by nodeset 101. Also, the second 1.0 float is to instruct goma to apply this condition in a residual form.

Technical Discussion

This condition is used to set a temperature boundary condition for nonisothermal porous media problems, viz. problems that use the R_POR_ENERGY equation (called *EQ = porous_energy*). This energy equation is written in multiphase enthalpy form and hence requires a different equatioin that for continuous media.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Category 9: Stress Equations

The following conditions provide a means to set boundary conditions for the hyperbolic viscoelastic stress equations

S11

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS11)

This Dirichlet boundary condition specification is used to set a constant xx-stress for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S11 | S11_1 | S11_2 | S11_3 | S11_4 | S11_5 | S11_6 | S11_7}	Boundary condition name (<bc_name>) that defines the xx-stress for a given mode, where:	
	S11	xx-component of stress tensor for mode 1
	S11_1	xx-component of stress tensor for mode 2
	S11_2	xx-component of stress tensor for mode 3
	S11_3	xx-component of stress tensor for mode 4
	S11_4	xx-component of stress tensor for mode 5
	S11_5	xx-component of stress tensor for mode 6
	S11_6	xx-component of stress tensor for mode 7
	S11_7	xx-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of xx-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*	

Examples

Following are sample cards for applying a Dirichlet condition on the xx-stress component for mode 2 on node set 7:

```
BC = S11_1 NS 7   4.0
```

```
BC = S11_1 NS 7   4.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

S12

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS12)

This Dirichlet boundary condition specification is used to set a constant xy-stress (also known as the shear stress) for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S12 | S12_1 | S12_2 | S12_3 | S12_4 | S12_5 | S12_6 | S12_7}	Boundary condition name (<bc_name>) that defines the xy-stress for a given mode, where:	
	S12	xy-component of stress tensor for mode 1
	S12_1	xy-component of stress tensor for mode 2
	S12_2	xy-component of stress tensor for mode 3
	S12_3	xy-component of stress tensor for mode 4
	S12_4	xy-component of stress tensor for mode 5
	S12_5	xy-component of stress tensor for mode 6
	S12_6	xy-component of stress tensor for mode 7
	S12_7	xy-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of xy-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.	

Examples

Following are sample cards for applying a Dirichlet condition on the xy-stress component for mode 5 on node set 10:

```
BC = S12_4 NS 10   1.25
```

```
BC = S12_4 NS 10  1.25   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

S13

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS13)

This Dirichlet boundary condition specification is used to set a constant xz-stress for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S13 | S13_1 | S13_2 | S13_3 | S13_4 | S13_5 | S13_6 | S13_7}	Boundary condition name (<bc_name>) that defines the xz-stress for a given mode, where:	
	S13	xz-component of stress tensor for mode 1
	S13_1	xz-component of stress tensor for mode 2
	S13_2	xz-component of stress tensor for mode 3
	S13_3	xz-component of stress tensor for mode 4
	S13_4	xz-component of stress tensor for mode 5
	S13_5	xz-component of stress tensor for mode 6
	S13_6	xz-component of stress tensor for mode 7
	S13_7	xz-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of xz-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.	

Examples

Following is a sample card for applying a Dirichlet condition for the xz-stress component for mode 5 on node set 10:

```
BC = S13_4 NS 10   1.3
```

```
BC = S13_4 NS 10   1.3   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

S22

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS22)

This Dirichlet boundary condition specification is used to set a constant yy-stress (also known as the shear stress) for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S22 | S22_1 | S22_2 | S22_3 | S22_4 | S22_5 | S22_6 | S22_7}	Boundary condition name (<bc_name>) that defines the yy-stress for a given mode, where:	
	S22	yy-component of stress tensor for mode 1
	S22_1	yy-component of stress tensor for mode 2
	S22_2	yy-component of stress tensor for mode 3
	S22_3	yy-component of stress tensor for mode 4
	S22_4	yy-component of stress tensor for mode 5
	S22_5	yy-component of stress tensor for mode 6
	S22_6	yy-component of stress tensor for mode 7
	S22_7	yy-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of yy-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.	

Examples

Following are sample cards for applying a Dirichlet condition on the yy-stress component for mode 8 on node set 20:

```
BC = S22_7 NS 20   5.0
```

```
BC = S22_7 NS 20   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

S23

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS23)

This Dirichlet boundary condition specification is used to set a constant yz-stress for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S23 | S23_1 | S23_2 | S23_3 | S23_4 | S23_5 | S23_6 | S23_7}	Boundary condition name (<bc_name>) that defines the yz-stress for a given mode, where:	
	S23	yz-component of stress tensor for mode 1
	S23_1	yz-component of stress tensor for mode 2
	S23_2	yz-component of stress tensor for mode 3
	S23_3	yz-component of stress tensor for mode 4
	S23_4	yz-component of stress tensor for mode 5
	S23_5	yz-component of stress tensor for mode 6
	S23_6	yz-component of stress tensor for mode 7
	S23_7	yz-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of yz-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.	

Examples

Following are sample cards for applying a Dirichlet condition on the yz-stress component for mode 8 on node set 20:

```
BC = S23_7 NS 20   5.0
```

```
BC = S23_7 NS 20   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

S33

```
BC = {bc_name} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/STRESS33)

This Dirichlet boundary condition specification is used to set a constant zz-stress for any given mode of the stress tensor. Each such specification is made on a separate input card. Definitions of the input parameters are as follows:

{S33 | S33_1 | S33_2 | S33_3 | S33_4 | S33_5 | S33_6 | S33_7}	Boundary condition name (<bc_name>) that defines the zz-stress for a given mode, where:	
	S33	zz-component of stress tensor for mode 1
	S33_1	zz-component of stress tensor for mode 2
	S33_2	zz-component of stress tensor for mode 3
	S33_3	zz-component of stress tensor for mode 4
	S33_4	zz-component of stress tensor for mode 5
	S33_5	zz-component of stress tensor for mode 6
	S33_6	zz-component of stress tensor for mode 7
	S33_7	zz-component of stress tensor for mode 8
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.	
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.	
<float1>	Value of zz-stress.	
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.	

Examples

Following are sample cards for applying a Dirichlet condition on the zz-stress component for mode 1 on node set 100:

```
BC = S33 NS 100   5.0
```

```
BC = S33 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

See the technical discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions.

For details of the stress tensor and its use for solving viscoelastic flow problems, please see the viscoelastic flow tutorial (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

Category 10: Gradient Equations

As companion equations to the viscoelastic stress equations, a continuous velocity gradient is determined through the so-called Velocity Gradient Equations. These boundary conditions are of the Dirichlet type and can be used to put conditions on this class of equations.

G11

```
BC = G11 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT11)

This Dirichlet boundary condition specification is used to set a constant xx-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G11	Boundary condition name (<bc_name>) that defines the xx-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of xx-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample card for applying a Dirichlet condition on the xx-velocity gradient component on node set 100:

```
BC = G11 NS 100   5.0
```

```
BC = G11 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G12

```
BC = G12 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT12)

This Dirichlet boundary condition specification is used to set a constant xy-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G12	Boundary condition name (<bc_name>) that defines the xy-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of xy-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

Following is a sample card for applying a Dirichlet condition on the xy-velocity gradient component on node set 100:

```
BC = G12 NS 100   5.0
```

```
BC = G12 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G13

```
BC = G13 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT13)

This Dirichlet boundary condition specification is used to set a constant xz-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G13	Boundary condition name (<bc_name>) that defines the xz-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of xz-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample card for applying a Dirichlet condition on the xz-velocity gradient component on node set 100:

```
BC = G13 NS 100   5.0
```

```
BC = G13 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G21

```
BC = G21 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT21)

This Dirichlet boundary condition specification is used to set a constant yx-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G21	Boundary condition name (<bc_name>) that defines the yx-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of yx-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the yx-velocity gradient component on node set 100:

```
BC = G21 NS 100   5.0
```

```
BC = G21 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G22

```
BC = G22 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT22)

This Dirichlet boundary condition specification is used to set a constant yy-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G22	Boundary condition name (<bc_name>) that defines the yy-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of yy-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample card for applying a Dirichlet condition on the yy-velocity gradient component on node set 100:

```
BC = G22 NS 100   5.0
```

```
BC = G22 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G23

```
BC = G23 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT23)

This Dirichlet boundary condition specification is used to set a constant yz-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G23	Boundary condition name (<bc_name>) that defines the yz-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of yz-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the yz-velocity gradient component on node set 100:

```
BC = G23 NS 100   5.0
```

```
BC = G23 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G31

```
BC = G31 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT31)

This Dirichlet boundary condition specification is used to set a constant zx-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G31	Boundary condition name (<bc_name>) that defines the zx-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of zx-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the zx-velocity gradient component on node set 100:

```
BC = G31 NS 100   5.0
```

```
BC = G31 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G32

```
BC = G32 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT32)

This Dirichlet boundary condition specification is used to set a constant zy-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G32	Boundary condition name (<bc_name>) that defines the zy-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of zy-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample card for applying a Dirichlet condition on the zy-velocity gradient component on node set 100:

```
BC = G32 NS 100   5.0
```

```
BC = G32 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

G33

```
BC = G33 NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/GRADIENT33)

This Dirichlet boundary condition specification is used to set a constant zz-velocity gradient component of the velocity gradient tensor. Definitions of the input parameters are as follows:

G33	Boundary condition name (<bc_name>) that defines the zz-velocity gradient.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of zz-velocity gradient.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the zz-velocity gradient component on node set 100:

```
BC = G33 NS 100   5.0
```

```
BC = G33 NS 100   5.0   1.0
```

where the second example uses the "residual" method for applying the same Dirichlet condition.

Technical Discussion

We solve a simple least squares equation to determine the continuous velocity gradient G from the velocity field. This is done so that we may have a differentiable field to get estimates of the second derivative of the velocity field for applications in complex rheology. The velocity gradient equation is:

$$G = \nabla v \quad .$$

Note, that boundary conditions are almost never set on the velocity gradient equation since it is just a least squares interpolation of the discontinuous velocity gradient derived from the velocity field.

See the Technical Discussion for the *UVW* velocity boundary condition for a discussion of the two ways of applying Dirichlet boundary conditions. For details of the velocity gradient tensor and its use for solving viscoelastic flow problems, please see Rao (2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

---

Category 11: Shear Rate Equation

The single boundary condition in this category is used to set a Dirichlet condition for the scalar shear rate equation. This differential equation is employed by the Phillips model for a suspension constitutive equation.

SH

```
BC = SH NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/SHEAR_RATE)

This boundary condition is used to set a Dirichlet condition for the scalar shear rate unknown field.

Description of the input parameters is as follows:

SH	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value at which the scalar shear rate unknown will be fixed on node set <bc_id>.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

An example of its used:

```
BC = SH NS 10 0.5
```

This boundary condition sets the scalar shear rate unknown to 0.5 on nodeset 10.

Technical Discussion

The scalar shear rate unknown field is otherwise known as the second invariant of the rate of deformation tensor.

Category 12: Fill Equation

The so-called *Fill* equation is used by the volume-of-fluid and level-set Eulerian interface tracking in *Goma*. Basically it is a statement of Lagrangian invariance and is hence a hyperbolic statement of the so-called kinematic equation. Given a velocity field, this equation advances the fill function as a set of material points; hence material surfaces remain ostensibly intact. The boundary conditions in this section are used to specify the level-of-fill at a boundary at which a fluid of a specific phase is flowing into the problem domain.

F

```
BC = F NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/FILL)

This Dirichlet boundary condition specifies a value of the fill or level set unknown field on a node set.

A description of the input parameters is as follows:

F	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value at which the fill or level set unknown will be fixed on this node set.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

An example:

```
BC = F NS 100 1.0
```

Technical Discussion

This boundary condition finds most of its use in the VOF/FILL interface tracking algorithm where it is used to fix the value of the color function at an inlet or outlet boundary. In the level set formulation, it is used less but is still useful in defining the absolute fixed location of an interface by setting the value assigned to 0 on a node set.

FILL_INLET

```
BC = FILL_INLET SS <bc_id> <float1>
```

Description / Usage

(SPECIAL/FILL)

This boundary condition allows the user to specify a value on a inlet boundary from VOF problems employing discontinuous interpolation of the color function, *F*.

Description of the input parameters is as follows:

FILL_INLET	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	The value of the fill function, *F*, as it flows across <bc_id> into the domain.

Examples

An example:

```
BC = FILL_INLET SS 10 1.0
```

Technical Discussion

- This boundary condition is useful only in problems involving VOF interface tracking in which the fill function is interpolated discontinuously. In this formulation, communication of the fill function value can only be made by finding the value of the fill function in the element upstream of the current position. While this is a stable formulation for the advective VOF method, it does introduce the complexity of determining which element is actually upstream.
- When there is no element upstream, as in the case of an inlet boundary, this boundary condition must be present to establish the value of the fill function that is flowing across the inlet boundary into the domain. Consequently, this boundary condition should be present on all inlet boundaries of the problem. It sometimes is also useful to have it on outflow boundaries as well, just in case a backflow situation arises.

Category 13: Potential Equation

The *Potential* equation is a Laplace equation for the voltage (potential) given a charge distribution in a dielectric medium or a voltage or current boundary condition in an electrically conductive medium. The following boundary conditions allow the current or voltage to be set on a boundary.

CURRENT

```
BC = CURRENT SS <bc_id> <float>
```

Description / Usage

(WIC/POTENTIAL)

This card specifies the electrical current density at a given boundary.

Definitions of the input parameters are as follows:

CURRENT	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	Value of current density (in A/ m^2 or A/ cm^2, depending on units of length scale used in the problem).

Examples

An example input card:

```
BC = CURRENT SS 1    -0.05
```

Technical Discussion

No Discussion.

CURRENT_USER

```
BC = CURRENT_USER SS <bc_id> <float_list>
```

Description / Usage

(WIC/POTENTIAL)

This boundary condition card is used to define a routine for a user-defined electrical current density model. Definitions of the input parameters are as follows:

CURRENT_USER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following is a sample input card:

```
BC = CURRENT_USER SS 100   10.0 3.14159
```

Technical Discussion

No Discussion.

VOLT

```
BC = VOLT NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/POTENTIAL)

This Dirichlet boundary condition card is used to set a constant voltage. Definitions of the input parameters are as follows:

VOLT	Name of the boundary condition (<bc_name>).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of voltage.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

Following is a sample card:

```
BC = VOLT NS 3   -0.22
```

Technical Discussion

No Discussion.

CURRENT_BV

```
BC = CURRENT_BV SS <bc_id> <integer> <floatlist>
```

Description / Usage

(WIC/POTENTIAL)

The *CURRENT_BV* card enables the specification of variable electrical current density as given by Butler-Volmer kinetics and the Faraday's law at the specified boundary (namely, an electrode surface).

The <floatlist> has seven parameters for this boundary condition; definitions of the input parameters are as follows:

CURRENT_BV	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float1>	Stoichiometric coefficient.
<float2>	Kinetic rate constant.
<float3>	Reaction order.
<float4>	Anodic direction transfer coefficient.
<float5>	Cathodic direction transfer coefficient.
<float6>	Electrode potential or applied voltage.
<float7>	Theoretical open-circuit potential.

Examples

An example input card:

```
BC = CURRENT_BV SS 1 0   -1.0 0.000002 1.0 0.21 0.21 -0.65   -0.22
```

Technical Discussion

Users are referred to Chen (2000) for details of the Butler-Volmer model and also Newman (1991), particularly Equations 8.6 and 8.10 and Chapter 8, pp. 188-189 in the latter.

References

GTM-025.0: Modeling diffusion and migration transport of charged species in dilute electrolyte solutions: GOMA implementation and sample computed predictions from a case study of electroplating, K. S. Chen, September 21, 2000

J. S. Newman, "Electrochemical Systems", Second Edition, Prentice-Hall, Inc. (1991).

CURRENT_HOR

```
BC = CURRENT_HOR SS <bc_id> <integer> <floatlist>
```

Description / Usage

(WIC/POTENTIAL)

The **CURRENT_HOR** card enables the specification of the variable current density as given by linearized Butler-Volmer kinetics (such as that for the hydrogen-oxidation reaction in polymer-electrolyte-membrane fuel cells) at the specified boundary (i.e., at the electrode surface).

The <floatlist> consists of 9 values; definitions of the input parameters are as follows:

CURRENT_HOR	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float1>	Product of interfacial area per unit volume by exchange current density, ai_0, in units of A/ cm^3.
<float2>	Catalyst layer or catalyzed electrode thickness, *H*, in unit of *cm*.
<float3>	Reference concentration, c_{ref}, in units of moles/ cm^3.
<float4>	Anodic direction transfer coefficient, α_a.
<float5>	Cathodic direction transfer coefficient, α_c.
<float6>	Temperature, *T*, in unit of *K*.
<float7>	Theoretical open-circuit potential, U_0, in unit of V.
<float8>	Reaction order, β.
<float9>	Electrode potential, *V*, in unit of *V*.

Examples

The following is a sample input card:

```
BC = CURRENT_HOR SS 14 0 1000. 0.001 4.e-5 1. 1. 353. 0. 0.5 0.
```

Technical Discussion

For electrochemical reactions such as the hydrogen-oxidation reaction (HOR), surface overpotential is relatively small such that the Butler-Volmer kinetic model can be linearized to yield a simplified equation for computing current density:

$$j = ai_0 H\left(\frac{c}{c_{ref}}\right)^{\beta}(\alpha_a + \alpha_c)\frac{F(V - \Phi - U_0)}{RT}$$

where j is current density in units of A/ cm^2; ai_0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/ cm^3; H is the catalyst layer or catalyzed electrode thickness in unit of *cm*; c and c_{ref} are, respectively, species and reference molar concentrations in units of moles/ cm^3; β is reaction order; α_a and α_c are, respetively, the anodic and cathodic transfer coefficients; F is the Faraday's constant (96487 *C/mole*); R is the universal gasl constant (8.314 *J/mole-K*); T is temperature in unit of *K*; V and ϕ are, respectively, the electrode and electrolyte potentials in unit of V; U_0 and is the open-circuit potential in unit of V.

References

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, NJ (1991).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

CURRENT_ORR

```
BC = CURRENT_ORR SS <bc_id> <integer> <floatlist>
```

Description / Usage

(WIC/POTENTIAL)

The **CURRENT_ORR** card enables the specification of the variable current density as given by the Tafel kinetics (such as that for the oxygen-reduction reaction in polymerelectrolyte- membrane fuel cells) at the specified boundary (i.e., at the electrode surface).

The <floatlist> consists of 8 values; definitions of the input parameters are as follows:

CURRENT_ORR	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<integer>	Species number of concentration.
<float1>	Product of interfacial area per unit volume by exchange current density, ai_0, in units of A/ cm^3.
<float2>	Catalyst layer or catalyzed electrode thickness, *H*, in unit of *cm*.
<float3>	Reference concentration, c_{ref}, in units of moles/ cm^3.
<float4>	Cathodic direction transfer coefficient, α_c.
<float5>	Temperature, *T*, in unit of *K*.
<float6>	Electrode potential, *V*, in unit of *V*.
<float7>	Theoretical open-circuit potential, U_0, in unit of *V*.
<float8>	Reaction order, β.

Examples

The following is a sample input card:

```
BC = CURRENT_ORR SS 15 1 0.01 0.001 4.e-5 1. 353. 0.7 1.18 1.
```

Technical Discussion

For electrochemical reactions such as the oxygen-reduction reaction (ORR), surface overpotential is large and negative such that the first exponential term in the Butler- Volmer kinetic model is much smaller than the second term and thus can be dropped to yield the Tafel kinetic model for computing current density:

$$j = -ai_0 H\left(\frac{c}{c_{ref}}\right)^{\beta} e^{-\frac{\alpha_c F}{RT}(V-\Phi-U_0)}$$

here *j* is current density in units of A/ cm^2; ai_0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/ cm^3; *H* is the catalyst layer or catalyzed electrode thickness in unit of *cm*; *c* and c_{ref} are, respectively, species and reference molar concentrations in units of moles/ cm^3; β is reaction order; α_c is the anodic and cathodic transfer coefficient; *F* is the Faraday's constant (96487 *C/mole*); *R* is the universal gasl constant (8.314 *J/mole-K*); *T* is temperature in unit of *K*; *V* and ϕ are, respectively, the electrode and electrolyte potentials in unit of *V*; and U_0 is the open-circuit potential in unit of *V*.

References

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, NJ (1991).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

VOLT_USER

```
BC = VOLT_USER SS <bc_id> <float_list>
```

Description / Usage

(WIC/POTENTIAL)

This boundary condition card is used to specify a voltage or potential computed via a user-defined function. Definitions of the input parameters are as follows:

VOLT_USER	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float_list>	A list of float values separated by spaces which will be passed to the user-defined subroutines so the user can vary the parameters of the boundary condition. This list of float values is passed as a one-dimensional double array to the appropriate C function.

Examples

The following is a sample input card:

```
BC = VOLT_USER SS 14 0.33 1000. 0.001 4e-5 1. 1. 353. 0.
```

Technical Discussion

In the **VOLT_USER** model currently implemented in GOMA, the electrolyte potential is given by the linearized Butler-Volmer kinetic model as in the hydrogen-oxidation reaction of a hydrogen-fueled polymer-electrolyte-membrane fuel cell. See the *user_bc.c* routine for details.

References

No References.

Category 14: Fluid-Solid Interaction

This is a special group of boundary conditions for problems in which there are two distinct material phases (fluid and solid) with relative motion between them. These BC's provide a means to apply conditions to a moving boundary with sensitivities to variables in both phases. These problems are formulated in Goma as overset-grid or phase function problems.

LAGRANGE_NO_SLIP

```
BC = LAGRANGE_NO_SLIP SS <bc_id> <integer1> <integer2>
```

Description / Usage

(CONTACT_SURF/R_LAGR_MULT1)

This boundary condition is used to apply a kinematic Lagrange multiplier constraint to a solid/fluid boundary while using Goma's overset grid capability. The condition is used when the complete fluid-structure interaction problem is being solved, viz. stresses between fluid and solid are both accommodated as is the dynamics of the structure and fluid. In contrast, *Goma* allows for a structure to be moved through a fluid under prescribed kinematics, and in that case a different Lagrange multiplier constraint is advocated (see LS_NO_SLIP, for example). Two integer inputs together with a sideset ID integer are required for this boundary condition:

LAGRANGE_NO_SLIP	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<integer1>	Element block ID of solid phase from the EXODUS II database.
<integer2>	Element block ID of liquid phase from the EXODUS II database.

Examples

Following is a sample card:

```
BC = LAGRANGE_NO_SLIP SS 2 1 2
```

In this case the kinematic condition (viz. a velocity match of fluid and solid at the interface) is applied to the interface imprinted by sideset 2. That side set is imprinted on the background fluid mesh. The solid material block ID is 1 in this case and the background fluid material ID is 2.

Technical Discussion

In this work, the governing equations consist of a fluid momentum balance:

$$\int_V \phi_v^i \left[\rho_f \frac{D\underline{v}}{Dt} + \nabla \bullet \tau + \rho_f \underline{F} \right] dV - \int_\Gamma \phi_k^i \underline{\gamma} d\Gamma = 0$$

a mass balance:

$$\int_V \phi_c^i [\nabla \bullet \underline{v}] dV = 0$$

and a solid momentum balance:

$$\int_S \phi_x^i [\rho_s \ddot{\underline{x}} + \nabla \bullet \sigma + \rho_s \underline{F}] dV + \int_\Gamma \phi_k^i \underline{\gamma} d\Gamma = 0$$

The kinematic constraint at the fluid-solid interface is:

and the level set function is evaluated at each fluid mesh node by:

The first four equations are written in a Galerkin/Finite form, with ϕ_i representing the weighting functions at node *i*. The first three equations are enforced at all nodes *i* that contain the appropriate degrees of freedom (viz. solid or fluid dofs). The fourth equation applies at the solid-liquid interface. ρ_f and ρ_s are the fluid and solid material densities, respectively, $\underline{v}$ is the fluid velocity, $\underline{F}$ represents any body forces such as gravity, τ is the fluid stress tensor, $\underline{\gamma}$ is the Lagrange multiplier vector unknown, $\underline{\chi}$ is the solid displacement vector unknown, σ is the solid stress tensor, f is the level set unknown, Θ is a step function which is -1 for points within the region occupied by the solid and +1 outside this region, $\underline{x_i}$ and $\underline{x_s}$ are the position vectors of a fluid node and of the closest point to it on the solid boundary, respectively, *V* is the fluid volume domain, *S* is the solid volume domain, and Γ is the solid boundary (interface) surface domain.

Noteworthy is that this boundary condition applies the second-to-last "kinematic" constraint.

$$\int_{\Gamma} \phi_{\gamma}{}^{i} [\dot{\underline{x}} - \underline{v}] d\Gamma = 0$$

$$f = \Theta \left\| \underline{x}_i - \underline{x}_s \right\|$$

References

GT-026.3: Goma's Overset Mesh Capability. Randy Schunk.

OVERSET_FLUID_SOLID/BAAIJENS_FLUID_SOLID

```
BC = BAAIJENS_FLUID_SOLID PF <pf_id> <integer1> <integer2>
```

Description / Usage

(EMBEDDED_SURF/R_MOMENTUM1)

This boundary condition is used to apply a traction to the fluid that comes from a solid while using *Goma's* overset grid capability. The condition is used when the complete fluid-structure interaction problem is being solved, viz. stresses between fluid and solid are both accommodated as is the dynamics of the structure and fluid. The condition is applied to the fluid phase along a zero-level-set contour, hence the PF BC ID type. In another mode of usage, Goma allows for a structure to be moved through a fluid under prescribed kinematics, and in that case this condition is still applied as a solid traction to the fluid. The value of that traction is dictated by the Lagrange multiplier kinematic constraint (cf. LANGRANGE_NO_SLIP BC and LS_NO_SLIP BC). Note that the condition is applied to a boundary in the fluid defined by a phase-field function (see *phase1* equation type). . Two integer inputs together with a sideset ID integer are required for this boundary condition:

BAAIJENS_FLUID_SOLD	Name of the boundary condition.
PF	Type of boundary condition (<bc_type>), where **PF** denotes a surface defined by a phase function (level-set).
<pf_id>	The boundary flag identifier basically sets the number of the phase field function to which this condition applies. For now you must set this to 1, as this phase-field is hardwired to handle the imprinted fluid solid boundary.
<integer1>	Element block ID of solid phase from the EXODUS II database.
<integer2>	Element block ID of liquid phase from the EXODUS II database.

The peculiar name was derived from a paper by Frank Baaijens, from which Goma's formulation was generated. We are in the process of changing that name to OVERSET_FLUID_SOLID.

Examples

Following is a sample card:

```
BC = BAAIJENS_FLUID_SOLID PF 1 1 2
```

```
BC = LS_NO_SLIP PF 1 1 2
```

This condition set applies a fluid traction condition to a surface defined by phase field 1, which is slaved to a side set that is set in the phase function slave surface capability. (see Phase Function Initialization Method).

Technical Discussion

See discussion on LANGRANGE_NO_SLIP. This condition applies the fluid traction boundary term on the fluid momentum equation.

References

GT-026.3

OVERSET_SOLID_FLUID/BAAIJENS_SOLID_FLUID

```
BC = OVERSET_SOLID_FLUID SS <bc_id> <float_list>
```

Description / Usage

(CONTACT_SURF/ MESH)

This boundary condition is used to apply a traction to a solid that comes from the fluid while using *Goma's* overset grid capability. The condition is used when the complete fluid-structure interaction problem is being solved, viz. stresses between fluid and solid are both accommodated as is the dynamics of the structure and fluid. The condition is applied to the solid phase along a side set that defines the fluid/solid interface. Two integer inputs together with a sideset ID integer are required for this boundary condition:

BAAI-JENS_SOLID_FLUID	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<ss_id>	The boundary flag identifier that sets the side set number.
<integer1>	Element block ID of solid phase from the EXODUS II database.
<integer2>	Element block ID of liquid phase from the EXODUS II database.

The peculiar name was derived from a paper by Frank Baaijens, from which Goma's formulation was generated. We are in the process of changing that name to OVERSET_SOLID_FLUID.

Examples

Following is a sample card set:

```
BC = BAAIJENS_SOLID_FLUID SS 1 2 1
```

```
BC = BAAIJENS_FLUID_SOLID PF 1 2 1
```

```
BC = LAGRANGE_NO_SLIP SS 1 2 1
```

Here, the BAAIJENS_SOLID_FLUID cared applies a boundary fluid traction to a solid phase defined by side set 1. In this case the solid phase material ID is 2 and the fluid phase 1.

Technical Discussion

See discussion on LAGRANGE_NO_SLIP. Basically, this condition results in a boundary traction set by the Lagrange multiplier constraint to be applied to the solid momentum equation (note the weak term that appears on that equation).

References

GT-026.3

F1 F2 F3 F4 F5

```
BC = {F1 | F2 | F3 | F4 | F5} NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/R_PHASE)

This boundary condition format is used to set a constant phase function field values at node sets. Please see "phase#" equation types for a description of the variables. Each such specification (for each field being used) is made on a separate input card. These boundary conditions must be applied to node sets. Definitions of the input parameters are as follows:

{F1 | F2 | F3 | F4 | F5 }	Two-character boundary condition name (<bc_name>) that defines which phase field variable is being set. There are a maximum of five addtional level-set/phase fields.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value at which the phase field unknown will be fixed on this node set.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

Following is a sample card which applies an phase field boundary condition to the nodes in node set 100, specifically an phase field-3 value of 1.0.

```
BC = F3 NS 100 1.0
```

Technical Discussion

This boundary condition finds most of its use in the Phase Function interface tracking algorithm where it is used to fix the value of the color function at an inlet or outlet boundary. The phase function fields were put in to supplement Goma's base level set capability to provide the ability to model multiple (more than two) materials. We don't anticipate that these boundary conditions will be used much. Nonetheless, this condition allows Dirichlet conditions to be applied to each of the five additional level set fields.

References

GT-026.3

PF_CAPILLARY

```
BC = PF_CAPILLARY LS <integer> <float1> <float2> <float3>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition applies an "embedded" surface tension source term when solving capillary hydrodynamics problems with phase function level set interface tracking. Note that its counterpart for the base level set field is LS_CAPILLARY, and this boundary condition is applied the same way to other level-set fields defined by the Phase Function cards. It can be used with only subgrid integration. The surface tension value used in this boundary condition is obtained from the Surface Tension material parameter defined in the mat file. Note that each phase-function field requires a separate PF_CAPILLARY boundary condition.

A description of the input parameters follows:

PF_CAPILLARY	Name of the boundary condition.
PF	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter that is used to specify to which phase function field that the boundary condition is to be applied.
<float1>	Not currently used.
<float2>	Not currently used.
<float3>	Not currently used.

Examples

An example:

```
BC = PF_CAPILLARY PF 1
```

Technical Discussion

Surface tension forces at a level set (phase function) representation of an interfacial boundary are applied solely via this boundary condition. An additional divergence of stress tensor term $\Delta \cdot T_{cap}$ is added to the fluid momentum equation. The form of this tensor is

$$T_{cap} = \sigma(I - nn)\delta_{\alpha}(\phi)/|\nabla\phi|$$

where *s* is the (isotropic) surface tension, *I* is the identity tensor, *n* is the vector normal to the interface and ζ_a (ϕ) is the smoothed Dirac delta function. The surface tension value used in this expression is obtained from the Surface Tension card found in the material file.

The actual implementation in Goma integrates the divergence term by parts so the expression that is added to the weak form of the momentum equation is:

$$\int (\nabla N_j \cdot T_{cap}) d\Omega$$

This fact introduces the issue of integration error into the problem. As obvious above, this source term involves the non-linear Dirac delta function factor. Conventional numerical integration methods often do not offer adequate accuracy in evaluating this integral, especially if if the interface width is a fraction of the average element size. This has led to introduction the level-set-specific integration methods: subelement integration and subgrid integration. In the latter case, more integration points are clustered around the interface (in essence) to improve accuracy. The integer parameter on the card should be set to zero to signify that the surface tension forces are distributed in equal measure on both sides of the interfacial curve.

In the subelement integration case, however, an actual subelement mesh is place on each of the interface-containing elements which is made to conform to the interface curve. That is, the interface curve itself is covered by these subelement boundaries. This allows the volume integral to be collapsed into a line integral and the line integral evaluated along the subelement boundaries. This, however, introduces the problem of identifying which side of the element the surface tension forces should actually be applied to. Applying them to both simultaneously while either result in a cancellation or a doubling of the surface tension effect. For these cases, the integer parameter on this card is set to a -1 or a +1 to signify that the surface tension forces are applied to the negative or positive side of the interface curve, respectively.

References

No References.

Category 15: Level Set Interfaces

These boundary conditions are designed to apply conditions in materials along surfaces whose position is monitored using Level Set Interface Tracking.

LS_ADC

```
BC = LS_ADC SS <bc_id> <float1> <float2> <float3>
```

Description / Usage

(Special/LEVEL SET)

This boundary condition is used exclusively with level set interface tracking. It is used to simulate contact and dewetting events. It employs a probabilistic model applied to elements on a boundary that contain an interface to determine whether contact or dewetting occurs there. It then uses a direct, brute force algortihm to manipulate the level set field to enforce contact or dewetting.

A description of the input parameters follows:

LS_ADC	Name of the boundary condition.
SS	This string indicates that this boundary is applied to a sideset.
<bc_id>	This is a sideset id where contact or dewetting processes are anticipated. Only elements that border this sideset will be considered as possibilities for ADC events.
<float1>	θ_c, the capture angle in degrees.
<float2>	α_c, the capture distance (L)
<float3>	N_c, the capture rate (1/ L^2 -T)

Examples

An example:

```
BC = LS_ADC SS 10 15.0   0.2   100.0
```

Technical Discussion

It has been found that level set interface tracking problems that involve contact or dewetting of the interfacial representation pose special problems for our numerical method. To a certain extent, we can model this type of event by making special modifications to the slipping properties of the boundary in question, however, this does not always work, especially in the case of dewetting events.

What seems to be the trouble is that we are attempting to use continuum-based models to simulate phenomena that essentially are due to molecular forces being expressed over non-molecular length scales. These length scales, while big with respect to molecules, are small with respect to our problem size. Hence, they are difficult to include in the context of reasonable mesh spacing.

The approach this boundary condition takes to inclusion of contact and dewetting phenomena is not attempt to model the finer details, but to simply note that they are due to "molecular weirdness" and thus take place outside of ordinary continuum mechanics. Therefore, there is some justification for, very briefly and in a localized area, dispensing with continuum mechanics assumption and simply imposing a contact or dewetting event. We refer to these as ADC events and will describe them in more detail later.

The parameters supplied with the card are used to determine where and when such an ADC event occurs. We have chosen to introduce a probabalistic model for this purpose. The reasoning for this comes from reflecting on the dewetting problem. If one imagines a thin sheet of fluid on a wetting substrate, it is clear that dewetting will occur eventually at some point on that sheet. Where that event occurs is somewhat random for a detached perspective. Introduction of a probability model for ADC events attempts to capture this.

Whether an ADC event occurs at an element on the sideset is determined by the following requirements:

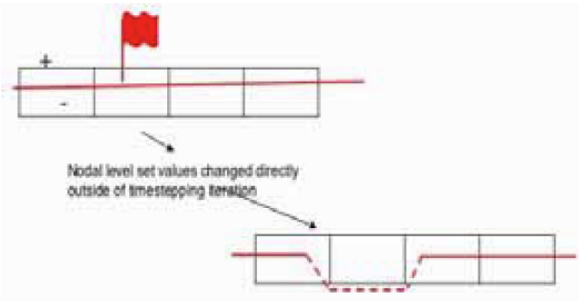
- The interface surface passes through the element.
- There isn't a contact line in the element.
- The angle between the interface normal and the sideset surface normal is less than or equal to the capture angle, θ_c.
- A random number in the range (0,1) determined by the standard C rand() function is less than a probability, P, given by

$$P = \begin{cases} N_c h^2 \Delta t \exp\left(1 - \left(\frac{d}{\alpha_c}\right)^2\right), & d > \alpha_c \\ N_c h^2 \Delta t, & d \le \alpha_c \end{cases}$$

where d is the average distance of the interface to the sideset in that element, Dt is the time step size, and h is the side length of the element (Note for 2D problems h^2 is replaced by h where the other dimension is assumed unity in the z direction).

Interpretation of this probability relation might take the following course. Given that the fluid interface lies within $\alpha)c$ of the surface, the length of time necessary before and ADC event is certain to occur is given by 1/ $N_c h^2$. Hence, the bigger the capture rate parameter the faster this is likely to occur. The functional form for the case of $d > \alpha_c$ is included merely to ensure that the probability drops smoothly to zero as quickly as possible. One might point out that the probability at a specific element tends towards zero as the element size decreases. Of course, in that context, the number of elements should increase in number so that the overall probability of an ADC event should not be a function of the degree of mesh refinement. A second point is that this boundary condition can be made to function as means to initiate contact without delay by simply choosing a capture rate that is large enought with respect to the current time step.

Application of an ADC event in a element that meets the preceding criteria is illustrated in the cartoon below:



It is a simple manipulation of the level set values in that element so that the interface will follow the path indicated by the dashed curve in the lower figure. No effort is made in preservation of volume when this is done. The assumption is that these events will occur infrequently enough that this is not a significant problem. However, the user should be aware of this assumption and be careful that these events do not occur on a regular basis as then the mass loss might be more significant.

References

No References.

LS_CA_H

```
BC = LS_CA_H SS <bc_id> <float>
```

Description / Usage

(WIC/SCALAR CURVATURE)

This boundary condition is used only in conjunction with level set interface tracking and the LS_CAP_CURVE embedded surface tension source term. Its function is impose a contact angle condition on that boundary.

A description of the input parameters follows:

LS_CA_H	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	A float value that is the imposed contact angle in degrees.

Examples

An example:

```
BC = LS_CA_H SS 10 45.0
```

Technical Discussion

The projection equation operator for solving for the curvature degree of freedom from a level set field is a Laplacian. It is standard to integrate these operators by parts but in the process one always generates a boundary integral. In this case the integral takes the form:

$$\int n_w \cdot n_{fs} d\Gamma$$

where n_w is the wall surface normal and n_{fs} is the normal to free surface (zero contour of the level set function). This is a convenient event because it allows us to impose a contact angle condition on a sideset using this boundary integral by making the assignment

$$n_w \cdot n_{fs} = \cos(\theta)$$

where θ is the contact angle specified on the card.

The effect of this boundary condition is impose a disturbance in the curvature field near the boundary that has the effect of accelerating or decelerating the fluid near the wall in response to whether the actual contact angle is greater or less than the imposed value. Thus, over time, given no other outside influences, the contact angle should evolve from its initial value (that presumably is different than the imposed value) to the value imposed on this card. The user should expect that the contact angle will instantaneously jumped to the imposed value.

References

No References.

LS_CAPILLARY

```
BC = LS_CAPILLARY LS <integer>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition applies an "embedded" surface tension source term when solving capillary hydrodynamics problems with level set interface tracking. It can be used both when subgrid or subelement integration is being used. The surface tension value used in this boundary condition is obtained from the Surface Tension material parameter defined in the mat file.

A description of the input parameters follows:

LS_CAPILLARY	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the surface tension forces are applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.

Examples

An example:

```
BC = LS_CAPILLARY LS 0
```

Technical Discussion

First, a warning: If subelement integration is off, make sure there is a nonzero levelset length scale or no surface forces term will be applied.

Surface tension forces at a level set representation of an interfacial boundary are applied solely via this boundary condition. An additional divergence of stress tensor term $\Delta \cdot T_{cap}$ is added to the fluid momentum equation. Following Jacqmin the form of this tensor is

$$T_{cap} = \sigma(I - nn)\delta_{\alpha}(\phi)/|\nabla\phi|$$

where s is the (isotropic) surface tension, I is the identity tensor, *n* is the vector normal to the interface and ζ_a (ϕ) is the smoothed Dirac delta function. The surface tension value used in this expression is obtained from the Surface Tension card found in the material file.

The actual implementation in Goma integrates the divergence term by parts so the expression that is added to the weak form of the momentum equation is:

$$\int (\nabla N_j \cdot T_{cap}) d\Omega$$

This fact introduces the issue of integration error into the problem. As obvious above, this source term involves the non-linear Dirac delta function factor. Conventional numerical integration methods often do not offer adequate accuracy in evaluating this integral, especially if if the interface width is a fraction of the average element size. This has led to introduction the level-set-specific integration methods: subelement integration and subgrid integration. In the latter case, more integration points are clustered around the interface (in essence) to improve accuracy. The integer parameter on the card should be set to zero to signify that the surface tension forces are distributed in equal measure on both sides of the interfacial curve.

In the subelement integration case, however, an actual subelement mesh is place on each of the interface-containing elements which is made to conform to the interface curve. That is, the interface curve itself is covered by these subelement boundaries. This allows the volume integral to be collapsed into a line integral and the line integral evaluated along the subelement boundaries. This, however, introduces the problem of identifying which side of the element the surface tension forces should actually be applied to. Applying them to both simultaneously while either result in a cancellation or a doubling of the surface tension effect. For these cases, the integer parameter on this card is set to a -1 or a +1 to signify that the surface tension forces are applied to the negative or positive side of the interface curve, respectively.

References

No References.

LS_CAP_DENNER_DIFF

```
BC = LS_CAP_DENNER_DIFF LS <double>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition is expected to be paired with another boundary condition that represents the surface tension term.

e.g.

LS_CAPILLARY LS_CAP_DIV_N etc..

And should not be used with LS_CAP_HYSING

Embedded boundary condition for applying a diffusion to the level set capillary boundary condition. A time scaled diffusion term to aid with reducing spurious currents.

Not compatible with subelement integration.

A description of the input parameters follows:

LS_CAP_DENNER_DIFF	Boundary condition name
LS	Indicator that this is a level set boundary condition
<double>	Scaling of the diffusive term

Examples

An example:

```
BC = LS_CAP_DENNER_DIFF LS 1.0
```

Technical Discussion

Adds the following embedded boundary condition to the momentum equations:

$$-\beta \Delta t^{n+1} \int \nabla_s \mathbf{v} : (\sigma \delta_\alpha(\phi) \nabla_s \mathbf{u}^{n+1}) d\mathbf{x}$$

And

$$\nabla_s f = \nabla f - \hat{\mathbf{n}}^n (\hat{\mathbf{n}}^n \cdot \nabla f)$$

Where, β is a scaling term, n represents a timestep, and $\hat{\mathbf{n}}$ represents the normal found from the level set function (or from the normal equations when enabled), and $\mathbf{u}$ is the velocity.

Theory

No Theory.

FAQs

No FAQs.

References

Denner, F., Evrard, F., Serfaty, R. and van Wachem, B.G., 2017. Artificial viscosity model to mitigate numerical artefacts at fluid interfaces with surface tension. Computers & Fluids, 143, pp.59-72.

LS_CAP_HYSING

```
BC = LS_CAP_HYSING LS <double>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

Embedded boundary condition for applying a surface tension source term with level set. Has the added advantage of a time scaled diffusion term to aid with reducing spurious currents.The surface tension value used in this boundary condition is obtained from the Surface Tension material parameter defined in the mat file.

Not compatible with subelement integration.

A description of the input parameters follows:

LS_CAP_HYSING	Boundary condition name
LS	Indicator that this is a level set boundary condition
<double>	Scaling of the diffusive term

Examples

An example:

```
BC = LS_CAP_HYSING LS 1.0
```

Technical Discussion

Adds the following embedded boundary condition to the momentum equations:

$$-\int \nabla \mathbf{v} : (\mathbf{I} - \hat{\mathbf{n}}^n \hat{\mathbf{n}}^n) d\mathbf{x} - \beta \Delta t^{n+1} \int \nabla_s \mathbf{v} : (\sigma \delta_\alpha(\phi) \nabla_s \mathbf{u}^{n+1}) d\mathbf{x}$$

Where, β is a scaling term, n represents a timestep, and $\hat{\mathbf{n}}$ represents the normal found from the level set function, and $\mathbf{u}$ is the velocity.

Theory

No Theory.

FAQs

No FAQs.

References

Hysing, S.R., Turek, S., Kuzmin, D., Parolini, N., Burman, E., Ganesan, S. and Tobiska, L., 2009. Quantitative benchmark computations of two-dimensional bubble dynamics. International Journal for Numerical Methods in Fluids, 60(11), pp.1259-1288.

LS_FLOW_PRESSURE

```
BC = LS_FLOW_PRESSURE LS <integer> <float1>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition applies a scalar pressure value as an "embedded" source term on the fluid momentum equation at the zero level set contour. It can be used both when subgrid or subelement integration is being used.

A description of the input parameters follows:

LS_FLOW_PRESSURE	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the pressure value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float1>	*P*, The constant value of pressure to be applied at the zero level set contour.

Examples

An example:

```
BC = LS_FLOW_PRESSURE LS 0 1013250.0
```

Technical Discussion

This boundary condition is somewhat analogous to the FLOW_PRESSURE boundary condition used quite often in ALE problems. It applies a scalar pressure at the interfacial curve as an embedded boundary condtion. It can be used in by subgrid and subelement methods. In the case of the former, a distributed volume integral of the form:

$$\int_V N_i \vec{n}_{fs} P \delta_\alpha(\phi) dV$$

where $\vec{n}_{fs}$ is the normal to the level set contour δ_α (ϕ) and is the familiar smoothed Dirac delta function with width parameter α. When subelement integration is used this width parameter goes to zero and the volume integral becomes a surface integral along the zero level set contour (Note: as of Oct 2005 subelement integration is not supported for three dimensional problems).

When using this boundary condition concurrent with subgrid integration, the integer parameter that appears on the card should be consistently set to zero. This ensures the volume source will be applied symmetrically. However, when using subelement integration this integer parameter must be entire a +1 or a -1 so that the pressure force will be applied to only on side of the interface and not both which would result in cancellation. This is much the same as was seen for the LS_CAPILLARY boundary condition and the reader is referred to that card for a more detailed discussion.

References

No References.

LS_FLUID_SOLID_CONTACT

```
BC = LS_FLUID_SOLID_CONTACT LS <integer> <integer1>
```

Description / Usage

(EMB/MOMENTUM)

This boundary condition applies a fluid-solid stress balance at a level set interface that is slaved to an overset mesh (see GT-026.3). It is applied as an "embedded" source term on the fluid momentum equations at the zero level set contour. **NOTE**: *This boundary condition has been deprecated in favor of the* **BAAIJENS_SOLID_FLUID** *and* **BAAIJENS_FLUID_SOLID** *boundary conditions, as described in the memo.*

A description of the input parameters follows:

LS_FLUID_SOLID_CONTACT	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the mass flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<integer1>	*Not used. Set to zero.*

Technical Discussion

We discourage use of this experimental boundary condition.

References

No References.

LS_INLET

```
BC = LS_INLET SS <bc_id>
```

Description / Usage

(PCC/LEVEL SET)

This boundary condition is used to set the values of the level set function on a sideset. Most of this is done on an inlet or outlet boundary to elminate the potential for oscillations in the level set field at those points from introducing spurious interfacial (zero contours) .

A description of the input parameters follows:

LS_INLET	Name of the boundary condition.
SS	This string indicates what this boundary is applied to.
<bc_id>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the surface tension forces are applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.

Examples

An example:

```
BC = LS_INLET SS 10
```

Technical Discussion

Surface tension forces at a level set representation of an interfacial boundary are applied solely via this boundary condition. An additional divergence of stress tensor term $\Delta \cdot T_{cap}$ is added to the fluid momentum equation. Following Jacqmin the form of this tensor is

$$T_{cap} = \sigma(I - nn)\delta_{\alpha}(\phi)/|\nabla\phi|$$

where s is the (isotropic) surface tension, I is the identity tensor, n is the vector normal to the interface and δ_{α} (ϕ) is the smoothed Dirac delta function. The surface tension value used in this expression is obtained from the Surface Tension card found in the material file.

The actual implementation in Goma integrates the divergence term by parts so the expression that is added to the weak form of the momentum equation is:

$$\int (\nabla N_j \cdot T_{cap}) d\Omega$$

This fact introduces the issue of integration error into the problem. As obvious above, this source term involves the non-linear Dirac delta function factor. Conventional numerical integration methods often do not offer adequate accuracy in evaluating this integral, especially if if the interface width is a fraction of the average element size. This has led to introduction the level-set-specific integration methods: subelement integration and subgrid integration. In the latter case, more integration points are clustered around the interface (in essence) to improve accuracy. The integer parameter on the card should be set to zero to signify that the surface tension forces are distributed in equal measure on both sides of the interfacial curve.

In the subelement integration case, however, an actual subelement mesh is place on each of the interface-containing elements which is made to conform to the interface curve. That is, the interface curve itself is covered by these subelement boundaries. This allows the volume integral to be collapsed into a line integral and the line integral evaluated along the subelement boundaries. This, however, introduces the problem of identifying which side of the element the surface tension forces should actually be applied to. Applying them to both simultaneously while either result in a cancellation or a doubling of the surface tension effect. For these cases, the integer parameter on this card is set to a -1 or a +1 to signify that the surface tension forces are applied to the negative or positive side of the interface curve, respectively.

References

No References.

LS_NO_SLIP

```
BC = LS_NO_SLIP PF <integer>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition is used to enforce the fluid/solid kinematic constraint for 1) overset grid applications in which 2) the solid material is assumed rigid and therefore has no internal stresses. It requires 3) a slaved phase function be defined along with 4) a vector field of Lagrange multipliers.

A description of the input parameters follows:

LS_NO_SLIP	Name of the boundary condition.
PF	This string indicates that this boundary condition is going to be applied along the zero contour of an embedded phase function (PF) field.
<integer>	This integer identifies the specific phase function field that is defining the contour. At the present time this integer should always be one.

Examples

An example:

```
BC = LS_NO_SLIP PF 1
```

Technical Discussion

This boundary condition is used in the context of Goma's overset grid capability. A thorough treatment of this method is provided in the Goma document (GT-026.3) and the user is directed there. However, a brief discussion of the nature of this boundary condition is in order at this point.

The overset grid capability is used in problems in which a solid material is passing through a fluid material. The solid and fluid materials both have there own meshes. In the general problem, stresses and velocities must be transferred between each phase and therefore there is two-coupling of the respective momentum and continuity equations. This boundary condition, however, is used in the restricted case in which the solid material is assumed to be rigid and having a prescribed motion. Therefore, the coupling only proceeds in one direction : solid to fluid.

This boundary condition concerns itself with enforcing the kinematic constraint:

between the solid material with prescribed motion, $\underline{\dot{x}}$, and the fluid whose velocity is, γ. This kinematic constraint represents a new set of equations in the model for which unknowns must be associated. In this case, we introduce a Lagrange multiplier vector field, $\underline{\gamma}$, at each node in the mesh. For fluid elements that do not intersect the fluid/ solid interface, these Lagrange multipliers are identically zero. They are non zero only for those fluid elements that are crossed by the fluid/solid boundary. These Lagrange multiplier fields couple the influence of the solid material on the fluid through body force terms in the fluid momentum equations of the form:

$$\int_{\Gamma} \phi_{\gamma}^{i} [\dot{\underset{\sim}{x}} - \underset{\sim}{v}] d\Gamma = 0$$

$$\int_{\Gamma} \phi_{v}^{i} \underset{\sim}{\gamma} d\Gamma$$

When applying this boundary condition it is necessary to include Lagrange multiplier equations equal to the number of dimensions in the problem. These are specified in the equation section of the input deck. The shape and weight functions for these fields are generally simple P0 functions. If one were to vector plot the components of the Lagrange multiplier components, you get a general picture of the force interaction field between the liquid and solid. This is sometimes informative.

A slaved phase function field is used to imprint the contour of the solid material on the liquid mesh. The zero contour of this function is then used to evaluate the above line integral. This phase function field is slaved to the solid material and is not evolved in the conventional sense. Nonetheless, a single phase function field equation must be included with the set of equations solved. In the phase function parameters section of the input deck, the user must indicate that this phase function is slaved and also must identify the sideset number of the boundary on the solid material which is the fluid/ solid interface.

References

No References.

LS_Q

```
BC = LS_Q LS <integer> <float1>
```

Description / Usage

(EMB/ENERGY)

This boundary condition applies a scalar heat flux value as an "embedded" source term on the heat conservation equation at the zero level set contour. It can be used both when subgrid or subelement integration is being used.

A description of the input parameters follows:

LS_Q	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the heat flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float1>	, The constant value of heat flux to be applied at the zero level set contour.

Examples

An example:

```
BC = LS_Q LS 0 -1.1e-3
```

Technical Discussion

This boundary condition is somewhat analogous to the QSIDE boundary condition used quite often in non-level set problems. It applies a scalar heat flux at the interfacial curve as an embedded boundary condtion. It can be used in by subgrid and subelement methods. In the case of the former, a distributed volume integral of the form:

$$\int N_i q \delta_\alpha(\phi) dV$$

$$V$$

where $\Delta \cdot T_{cap}$ is the familiar smoothed Dirac delta function with width parameter α. When subelement integration is used this width parameter goes to zero and the volume integral becomes a surface integral along the zero level set contour (Note: as of Oct 2005 subelement integration is not supported for three dimensional problems).

When using this boundary condition concurrent with subgrid integration, the integer parameter that appears on the card should be consistently set to zero. This ensures the volume source will be applied symmetrically. However, when using subelement integration this integer parameter must be entire a +1 or a -1 so that the heat flux will be applied only on side of the interface and not both which would result in cancellation. This is much the same as was seen for the LS_CAPILLARY boundary condition and the reader is referred to that card for a more detailed discussion.

References

No References.

LS_QRAD

```
BC = LS_QRAD LS <integer> <float1> <float2> <float3> <float4>
```

Description / Usage

(EMB/ENERGY)

This boundary condition card specifies heat flux using both convective and radiative terms. This heat flux value is applied as an "embedded" source term on the heat conservation equation at the zero level set contour (cf. *BC = QRAD* for ALE surfaces). It can be used both when subgrid or subelement integration is being used. The <float_list> has four parameters; definitions of the input parameters are as follows:

A description of the input parameters follows:

LS_QRAD	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the heat flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float1>	h, convective heat transfer coefficient.
<float2>	T_s, sink temperature.
<float3>	ε, total hemispherical emissivity.
<float4>	σ, Stefan-Boltzmann constant.

Examples

An example:

```
BC = LS_QRAD LS 0 10.0 273.0 0.3 5.6697e-8
```

Technical Discussion

This is the level-set counterpart to *BC = QRAD* which is the same boundary condition applied to a parameterized mesh surface. Please see the discussion of that input record for the functional form of this boundary condition.

References

No References.

LS_QLASER

```
BC = LS_QLASER LS <integer> <float1> <float2> <float3> <float4>
```

Description / Usage

(EMB/ENERGY)

This boundary condition card specifies heat flux model derived from a laser welding application. This heat flux value is applied as an "embedded" source term on the heat conservation equation at the zero level set contour (cf. *BC = Q_LASER_WELD* for ALE surfaces). It can be used both when subgrid or subelement integration is being used. The <float_list> has twenty-seven parameters; definitions of the input parameters are as follows:

A description of the input parameters follows:

LS_QLASER	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the heat flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float 1>	Nominal power of laser.
<float 2>	Power of laser at base state (simmer).
<float 3>	Base value of surface absorptivity.
<float 4>	Switch to allow tracking of normal component of liquid surface relative to laser beam axis for surface absorption (0 = OFF, 1 = ON)
<float 5>	Cutoff time for laser power.
<float 6>	Time at which laser power drops to 1/e.
<float 7>	For pulse weld, the laser power overshoot (%) of peak power at time to reach peak laser power.
<float 8>	Radius of laser beam.
<float 9>	For pulse weld, the time for laser pulse to reach peak power.
<float 10>	For pulse weld, the time for laser pulse to reach steady state in power.
<float 11>	Switch to either activate laser power distribution from beam center based on absolute distance (0) or based on radial distance in 2D plane (1).
<float 12>	Location of laser beam center (x-coordinate).
<float 13>	Location of laser beam center (y-coordinate).
<float 14>	Location of laser beam center (z-coordinate).
<float 15>	Laser beam orientation, normal to x-coordinate of body.
<float 16>	Laser beam orientation, normal to y-coordinate of body.
<float 17>	Laser beam orientation, normal to z-coordinate of body.
<float 18>	For pulse weld, spot frequency.
<float 19>	For pulse weld, total number of spots to simulate.
<float 20>	Switch to set type of weld simulation. (0=pulse weld, 1=linear continuous weld, -1=pseudo pulse weld, 2=sinusoidal continous weld)
<float 21>	For pulse weld, spacing of spots.
<float 22>	For radial traverse continuous weld, radius of beam travel.
<float 23>	Switch to activate beam shadowing for lap weld (0=OFF, 1=ON). Currently only active for ALE simulations.
<float 24>	Not active, should be set to zero.
<float 25>	For continuous weld, laser beam travel speed in xdirection (u velocity).
<float 26>	For continuous weld, laser beam travel speed in ydirection (v velocity).
<float	For continuous weld, laser beam travel speed in zdirection (w velocity).

Examples

An example:

```
BC = LS_QLASER LS -1 4.774648293 0 0.4 1 1 1.01 4.774648293 0.2
0.01 0.01 1 0.005 0 -0.198 -1 0 0 0.025 1 1 0.2032 -1000 0 0 0 0
0.0254
```

Technical Discussion

This is the level-set counterpart to *BC = Q_LASER* which is the same boundary condition applied to a parameterized mesh surface. Please see the discussion of that input record for the functional form of this boundary condition.

References

No References.

LS_RECOIL_PRESSURE

```
BC = LS_RECOIL_PRESSURE LS <integer> <float1> <float2> <float3> <float4>
```

Description / Usage

(EMB/VECTOR MOMENTUM)

This boundary condition card specifies heat flux model derived from a laser welding application.. This heat flux value is applied as an "embedded" source term on the heat conservation equation at the zero level set contour (cf. *BC = CAP_RECOIL_PRESS* for ALE surfaces). It can be used both when subgrid or subelement integration is being used. The <float_list> has seven parameters; definitions of the input parameters are as follows:

A description of the input parameters follows:

LS_RECOIL_PRESSURE	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the heat flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float1>	This float is currently disabled.
<float2>	This float is currently disabled.
<float3>	This float is currently disabled.
<float4>	Disabled. The boiling temperature is set to the melting point of the solidus. Use the material property "Solidus Temperature" card for this.
<float5>	This float is currently disabled.
<float6>	Conversion scale for pressure.
<float7>	Conversion scale for temperature.

Examples

Technical Discussion

Currently this boundary condition has coefficients for only iron and water. Several required pieces of information to use this boundary condition are not in final form, and the user can expect future changes and improvements. This boundary condition is designed for use with *LS_QLASER*.

References

No References.

LS_VAPOR/LS_QVAPOR

```
BC = LS_VAPOR LS <integer> <float1> <float2> <float3> <float4>
```

Description / Usage

(EMB/ENERGY)

This boundary condition card specifies heat flux model derived from a laser welding application. This particular contribution accounts for the energy lost by vapor flux. This heat flux value is applied as an "embedded" source term on the heat conservation equation at the zero level set contour (cf. *BC = Q_LASER_WELD* for ALE surfaces). It can be used both when subgrid or subelement integration is being used. The <float_list> has four parameters; definitions of the input parameters are as follows:

A description of the input parameters follows:

LS_VAPOR	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the heat flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<float1>	T_scale. Temperature scaling.
<float2>	q_scale. Heat flux scaling.

Examples

An example:

```
BC = LS_VAPOR LS 0 273. 1.
```

Technical Discussion

Currently this BC is hardwired to parameters (viz. heat capacitance, etc.) for iron. The melting point temperature is taken from the material property "Liquidus Temperature". This boundary condition is still in the developmental stage. In using it is advisable to be working with the Sandia Goma code team.

References

No References.

LS_YFLUX

```
BC = LS_YFLUX LS <integer> <integer1> <float1> <float2>
```

Description / Usage

(EMB/ENERGY)

This boundary condition applies a scalar mass flux value as an "embedded" source term on a species conservation equation at the zero level set contour. It can be used both when subgrid or subelement integration is being used.

A description of the input parameters follows:

LS_YFLUX	Name of the boundary condition.
LS	This string is used to indicated that this is a "boundary" condition is applied at an internal phase boundary defined by the zero contour of the level set function.
<integer>	An integer parameter than is permitted to take one of three values -1, 0, or 1. Depending upon the choice of this parameter the mass flux value is applied to the negative phase, both phase, or the positive phase, respectively. Details are given below.
<integer1>	w, This the species equation index to which this boundary condition is applied.
<float1>	h_c, a constant value for the mass transfer coefficient at the interface.
<float2>	Y_c, the "bulk" concentration of species used in conjunction with the mass transfer coefficient to compute the mass flux.

Examples

An example:

```
BC = LS_YFLUX LS 0 0 1.e-2 0.75
```

Technical Discussion

This boundary condition is somewhat analogous to the YFLUX boundary condition used quite often in non-level set problems to apply a scalar species flux at a at boundary defined by a side set. It applies a scalar mass transfer flux at the interfacial curve as an embedded boundary condtion. It can be used in by subgrid and subelement methods. In the case of the former, a distributed volume integral of the form:

$$\int N_i J \delta_\alpha(\phi) dV$$

where δ_α (ϕ) is the familiar smoothed Dirac delta function with width parameter α and the mass flux , J, is given by the typical relation:

$$J = h_c(Y_w - Y_c)$$

When subelement integration is used this width parameter goes to zero and the volume integral becomes a surface integral along the zero level set contour (Note: as of Oct 2005 subelement integration is not supported for three dimensional problems).

When using this boundary condition concurrent with subgrid integration, the integer parameter that appears on the card should be consistently set to zero. This ensures the volume source will be applied symmetrically. However, when using subelement integration this integer parameter must be entire a +1 or a -1 so that the mass flux will be applied only on side of the interface and not both which would result in cancellation. This is much the same as was seen for the LS_CAPILLARY boundary condition and the reader is referred to that card for a more detailed discussion.

References

No References.

SHARP_BLAKE_VELOCITY

```
BC = SHARP_BLAKE_VELOCITY SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. Its formulation is identical to the WETTING_SPEED_BLAKE boundary condition, but it is applied as a single point source on the boundary instead of a distributed stress.

This boundary condition can only be used for two-dimensional problems.

A description of the input parameters follows:

SHARP_BLAKE_VELOCITY	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	V_0 is a pre-exponential velocity factor (see functional form below).
<float3>	g is a thermally scaled surface tension, i.e. σ /2nkT.
<float4>	β, slip coefficient.
<float5>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float6>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.

Examples

An example:

```
BC = SHARP_BLAKE_VELOCITY SS 10 30.0 0.1 8. 0.001 0   0
```

Technical Discussion

The implementation for this wetting condition is identical to that of SHARP_WETLIN_VELOCITY, but the wetting velocity dependence is different. Because the wetting stress is not applied at a point, it is most appropriate for use when using subelement integration which similarly collapses the surface tension sources associated with the interface onto the interfacial curve.

Note also that this boundary condition is strictly for use with two-dimensional problems. Attempting to apply it to a three dimensional problem will result in an error message.

Theory

Derivation of the force condition for this boundary condition starts with a simple relation for wetting line velocity

$$V_{wet} = V_{old} + (V_{Blake} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

$$V_{Blake} = v_0 \sinh[g(\cos\theta_s - \cos\theta)]$$

Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation and has the following form when the velocity smoothing is not used,

$$\tau_w = \frac{V_{wet}}{\beta} = \frac{v_0}{\beta}\sinh[g(\cos\theta_s - \cos\theta)]$$

References

No References.

SHARP_CA_2D

```
BC = SHARP_CA_2D SS <bc_id> <float>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to impose a contact angle on a boundary when using *Level Set Interface Tracking*. It can only be used for two-dimensional problems.

A description of the input parameters follows:

FILL_CA	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	θ, the contact angle imposed, in degrees.

Examples

An example:

```
BC = SHARP_CA_2D SS 10 30.0
```

Technical Discussion

This boundary condition must be used in conjunction with the *VELO_SLIP_FILL* or *VELO_SLIP_LS* boundary condition. These latter conditions permits the fluid to slip in the vicinity of the contact line. The SHARP_CA_2D acts by imposing a force on the momentum equation. The size of this force is more or less in proportion between the actual contact angle on the boundary and the value specified on the card and scales directly with the applied surface tension material parameter. In this manner, it is very similar to the FILL_CA boundary condition.

The manner in which is applied differs. In this case, the applied force is not distributed around the contact line using a smooth delta function weighting in a weak integrated context, but instead the delta function is used to resolve the line integral and the force is applied directly at a point on the sideset set. Hence, this boundary condition is most appropriate for use in conjunction with subelement integration which performs a similar transformation of the volumetric surface tension source terms. Further, the logic use to identify the point of application on the boundary functions only in twodimensions. Hence, this boundary condition is stricly limited to two-dimensional problems.

References

No References.

SHARP_COX_VELOCITY

```
BC = SHARP_COX_VELOCITY SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. Its formulation is identical to the WETTING_SPEED_COX boundary condition, but it is applied as a single point source on the boundary instead of a distributed stress.

This boundary condition can only be used for two-dimensional problems.

A description of the input parameters follows:

SHARP_COX_VELOCITY	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	σ is the surface tension.
<float3>	ε_s is the dimensionless slip length, i.e. the ratio of the slip length to the characteristic length scale of the macroscopic flow.
<float4>	β, slip coefficient.
<float5>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float6>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.

Examples

An example:

```
BC = SHARP_COX_VELOCITY SS 10 30.0 72.0 0.01 0.1   0   0
```

Technical Discussion

The implementation for this wetting condition is identical to that of SHARP_WETLIN_VELOCITY, but the wetting velocity dependence is different. Because the wetting stress is not applied at a point, it is most appropriate for use when using subelement integration which similarly collapses the surface tension sources associated with the interface onto the interfacial curve.

Note also that this boundary condition is strictly for use with two-dimensional problems. Attempting to apply it to a three dimensional problem will result in an error message.

Theory

Derivation of the force condition for this boundary condition starts with a relation for wetting line velocity

$$V_{wet} = V_{old} + (V_{Cox} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

where V_{Cox} is computed from the Cox hydrodynamic wetting theory;

See VELO_THETA_COX for details of the Cox functions f and g. Note that the parameters λ, q_{inner}, and q_{outer} are currently not accessible from the input card and are hard-set to zero. λ is the ratio of gas viscosity to liquid viscosity whereas q_{inner} and q_{outer} represent influences from the inner and outer flow regions.

Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation and has the following form when the velocity smoothing is not used,

$$Ca \equiv \frac{\mu V_{Cox}}{\sigma} = \frac{g(\theta, \lambda) - g(\theta_s, \lambda)}{\left[\ln(\varepsilon_s^{-1}) + \frac{q_{inner}}{f(\theta_s, \lambda)} - \frac{q_{outer}}{f(\theta, \lambda)}\right]}$$

$$\tau_w = \frac{V_{Cox}}{\beta}$$

The Cox wetting velocity requires evaluation of integrals for the function g(θ, λ) which is currently done numerically using 10-point Gaussian quadrature. As such the evaluation of the integrals is expected to become inaccurate as either θ_s tends toward zero or θ tends toward 180 degrees. Note that the integrand becomes singular as θ tends toward 0 or 180 degrees.

References

No References.

SHARP_HOFFMAN_VELOCITY

```
BC = SHARP_HOFFMAN_VELOCITY SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. Its formulation is identical to the WETTING_SPEED_HOFFMAN boundary condition, but it is applied as a single point source on the boundary instead of a distributed stress.

This boundary condition can only be used for two-dimensional problems.

A description of the input parameters follows:

SHARP_HOFFMAN_VELOCITY	Label of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	σ is the surface tension.
<float3>	β, slip coefficient.
<float4>	t_{relax} is a relaxation time which can be used to smooth the imposed contact point velocity for transient problems. Set to zero for no smoothing.
<float5>	V_{old} is an initial velocity used in velocity smoothing for transient problems. Set to zero when smoothing is not used.

Examples

An example:

```
BC = SHARP_HOFFMAN_VELOCITY SS 10 30.0 72.0 0.1   0   0
```

Technical Discussion

The implementation for this wetting condition is identical to that of SHARP_WETLIN_VELOCITY, but the wetting velocity dependence is different. Because the wetting stress is not applied at a point, it is most appropriate for use when using subelement integration which similarly collapses the surface tension sources associated with the interface onto the interfacial curve.

Note also that this boundary condition is strictly for use with two-dimensional problems. Attempting to apply it to a three dimensional problem will result in an error message.

Theory

Derivation of the force condition for this boundary condition starts with a relation for wetting line velocity

$$V_{wet} = V_{old} + (V_{Hoffman} - V_{old})\left[1 - \exp\left(-\frac{t}{t_{relax}}\right)\right]$$

where $V_{Hoffman}$ is computed from the Hoffman correlation;

$$Ca \equiv \frac{\mu V_{Hoffman}}{\sigma} = g_{Hoff}(\theta) - g_{Hoff}(\theta_s)$$

See VELO_THETA_HOFFMAN for details of the Hoffman function g. Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the

$$\tau_w = \frac{V_{Hoffman}}{\beta}$$

opposite. This is mapped to a stress value by analogy with Navier's slip relation and has the following form when the velocity smoothing is not used,

Because the Hoffman functions are implicit, iteration is required in the determination of the wetting velocity. As a result, for very high Capillary numbers, i.e. > 10^6, the iteration procedure in Goma may need to be modified.

References

No References.

SHARP_WETLIN_VELOCITY

```
BC = SHARP_WETLIN_VELOCITY SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. Its formulation is identical to the WETTING_SPEED_LINEAR boundary condition, but it is applied as a single point source on the boundary instead of a distributed stress.

This boundary condition can only be used for two-dimensional problems.

A description of the input parameters follows:

SHARP_WETLIN_VELOCITY	Label for the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	c_T, proportionality constant as defined below
<float3>	Currently not used.
<float4>	β, slip coefficient.

Examples

An example:

```
BC = SHARP_WETLIN_VELOCITY SS 10 30.0 0.1 0. 0.001
```

Technical Discussion

As noted above, this boundary condition imposes the same wetting stress dependence as the WETTING_SPEED_LINEAR boundary condition. However, its application in the FEM context is different. Instead of the wetting stress, τ_w, being applied according to the formula:

$$\int N_j \tau_w \vec{t} \delta_w(\phi) d\Gamma \ ,$$

as is the case for the WETTING_SPEED_LINEAR condition, the Dirac function is used to remove the integral and replace it with a point stress at the location where $\phi = 0$ on the boundary. Designating this point as X_{cl}, the vector applied to the momentum equation is given by

$$N_j(X_{cl}) \tau_w(X_{cl}) \vec{t}(X_{cl})$$

Because the wetting stress is not applied at a point, it is most appropriate for use when using subelement integration which similarly collapses the surface tension sources associated with the interface onto the interfacial curve. Note that this method of application is identical to the SHARP_CA_2D boundary condition discussed elsewhere.

Note also that this boundary condition is strictly for use with two-dimensional problems. Attempting to apply it to a three dimensional problem will result in an error message.

Theory

Derivation of the force condition for this boundary condition starts with a simple relation for wetting line velocity

$$V_{wet} = \frac{1}{c_T}(\cos((\theta) - \cos(\theta_s)))$$

Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation,

$$\tau_w = \frac{V_{wet}}{\beta} = \frac{1}{\beta c_T}(\cos((\theta) - \cos(\theta_s)))$$

It should be noted that there is no distinction for this model in the function of β or c_T. The two parameters are interchangeable. In non-linear models, (see WETTING_SPEED_BLAKE) this is no longer true.

References

No References.

WETTING_SPEED_BLAKE

```
BC = WETTING_SPEED_BLAKE SS <bc_id> <floatlist>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It implements a version of the Blake-DeConinck molecular-kinetic theory wetting model.

A description of the input parameters follows:

WETTING_SPEED_BLAKE	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle, degrees .
<float2>	V_0 is a pre-exponential velocity factor (see functional form below).
<float3>	g is a thermally scaled surface tension, i.e. σ/2nkT.
<float4>	w, width of interfacial region near contact line. Defaults to level set length scale if zero or less.
<float5>	β, slip coefficient.
<float6>	currently not used.
<float7>	currently not used.
<float8>	currently not used.

Examples

An example:

```
BC = WETTING_SPEED_BLAKE SS 10 30.0 20.1 7.0 0.   0.001 0. 0. 0.
```

Technical Discussion

The implementation for this wetting condition is identical to that of WETTING_SPEED_LINEAR, but the wetting velocity dependence is different.

Note that it is a requirement that when using this boundary condition that slip to some extent be allowed on this boundary. This is most often done by applying a VELO_SLIP_LS boundary condition in conjunction with this boundary condition. In addition, a no penetration condition on the velocity is need in either the form of a Dirichlet condition or a VELO_NORMAL condition. It is important to note that the slipping condition need not relax the no slip requirement completely. In fact, its parameters should be set so that no slip is for the most part satisfied on the boundary in regions away from the contact line. Near the contact line however the parameters in the slip condition and the WETTING_SPEED_BLAKE condition need to be fixed so that appreciable fluid velocity is induced. This is a trial and error process at the current time.

Theory

Derivation of this boundary condition starts with a relation propose by Blake and DeConinck for wetting line motion

$$V_{wet} = V_0 \sinh(g(\cos\theta - \cos\theta_s))$$

This is mapped to a stress value by analogy with Navier's slip relation,

$$\tau_w = \frac{V_{wet}}{\beta} = \frac{v_0}{\beta}\sinh[g(\cos\theta_s - \cos\theta)]$$

This relation contrasts with the "linear" relation applied by the WETTING_SPEED_LINEAR relation in that the rate of change of the wetting velocity with the contact angle decreases as the wetting angle deviates more and more from its static value. This is more consisten with physical behaviors that the linear model.

In point of fact this condition is a vector condition so this scalar stress value multiplies the unit vector tangent to the surface and normal to the contact line, $\vec{t}$. This stress is then weighted by smooth Dirac function to restrict its location to being near the interface, weighted by a FEM shape function, integrated over the boundary sideset and added to the fluid momentum equation for the corresponding node j, vis:

$$\int N_j \tau_w \hat{t} \delta_w(\phi) d\Gamma$$

References

No References.

WETTING_SPEED_COX

```
BC = WETTING_SPEED_COX SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It implements a version of the Cox hydrodynamic model of wetting.

A description of the input parameters follows:

WETTING_SPEED_COX	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle, degrees.
<float2>	ε_s is the dimensionless slip length, i.e. the ratio of the slip length to the characteristic length scale of the macroscopic flow.
<float3>	σ is the surface tension.
<float4>	w ,width of interfacial region near contact line. Defaults to level set length scale if zero or less.
<float5>	β, slip coefficient.
<float6>	currently not used.
<float7>	currently not used.
<float8>	currently not used.

Examples

An example:

```
BC = WETTING_SPEED_COX SS 10 30.0 0.01 72.0 0.   0.001 0. 0. 0.
```

Technical Discussion

The implementation for this wetting condition is identical to that of WETTING_SPEED_LINEAR, but the wetting velocity dependence is different.

Note that it is a requirement that when using this boundary condition that slip to some extent be allowed on this boundary. This is most often done by applying a VELO_SLIP_LS boundary condition in conjunction with this boundary condition. In addition, a no penetration condition on the velocity is need in either the form of a Dirichlet condition or a VELO_NORMAL condition. It is important to note that the slipping condition need not relax the no slip requirement completely. In fact, its parameters should be set so that no slip is for the most part satisfied on the boundary in regions away from the contact line. Near the contact line however the parameters in the slip condition and the WETTING_SPEED_BLAKE condition need to be fixed so that appreciable fluid velocity is induced. This is a trial and error process at the current time.

Theory

Derivation of this boundary condition starts with a relation that represents the Cox hydrodynamic wetting model

$$Ca \equiv \frac{\mu V_{Cox}}{\sigma} = \frac{g(\theta, \lambda) - g(\theta_s, \lambda)}{\left[\ln(\varepsilon_s^{-1}) + \frac{q_{inner}}{f(\theta_s, \lambda)} - \frac{q_{outer}}{f(\theta, \lambda)}\right]}$$

See VELO_THETA_HOFFMAN for details of the Hoffman function g. Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation,

$$\tau_w = \frac{V_{Cox}}{\beta}$$

This relation contrasts with the "linear" relation applied by the WETTING_SPEED_LINEAR relation in that more consistent physical behavior should result.

In point of fact this condition is a vector condition so this scalar stress value multiplies the unit vector tangent to the surface and normal to the contact line, $\vec{t}$. This stress is then weighted by smooth Dirac function to restrict its location to being near the interface, weighted by a FEM shape function, integrated over the boundary sideset and added to the fluid momentum equation for the corresponding node j, vis:

$$\int N_j \tau_w \hat{t} \delta_w(\phi) d\Gamma$$

References

No References.

WETTING_SPEED_HOFFMAN

```
BC = WETTING_SPEED_HOFFMAN SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It implements a version of the Hoffman wetting correlation.

A description of the input parameters follows:

WETTING_SPEED_HOFFMAN	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle, degrees .
<float2>	currently not used.
<float3>	σ is the surface tension.
<float4>	w, width of interfacial region near contact line. Defaults to level set length scale if zero or less.
<float5>	β, slip coefficient.
<float6>	currently not used.
<float7>	currently not used.
<float8>	currently not used.

Examples

An example:

```
BC = WETTING_SPEED_HOFFMAN SS 10 30.0 0 72.0   0. 0.001 0. 0. 0.
```

Technical Discussion

The implementation for this wetting condition is identical to that of WETTING_SPEED_LINEAR, but the wetting velocity dependence is different.

Note that it is a requirement that when using this boundary condition that slip to some extent be allowed on this boundary. This is most often done by applying a VELO_SLIP_LS boundary condition in conjunction with this boundary condition. In addition, a no penetration condition on the velocity is need in either the form of a Dirichlet condition or a VELO_NORMAL condition. It is important to note that the slipping condition need not relax the no slip requirement completely. In fact, its parameters should be set so that no slip is for the most part satisfied on the boundary in regions away from the contact line. Near the contact line however the parameters in the slip condition and the WETTING_SPEED_BLAKE condition need to be fixed so that appreciable fluid velocity is induced. This is a trial and error process at the current time.

Theory

Derivation of this boundary condition starts with a relation that represents the Hoffman wetting correlation

$$Ca \equiv \frac{\mu V_{Hoffman}}{\sigma} = g_{Hoff}(\theta) - g_{Hoff}(\theta_s)$$

See VELO_THETA_HOFFMAN for details of the Hoffman function g. Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation,

$$\tau_w = \frac{V_{Hoffman}}{\beta}$$

This relation contrasts with the "linear" relation applied by the WETTING_SPEED_LINEAR relation in that more consistent physical behavior should result.

In point of fact this condition is a vector condition so this scalar stress value multiplies the unit vector tangent to the surface and normal to the contact line, $\vec{t}$. This stress is then weighted by smooth Dirac function to restrict its location to being near the interface, weighted by a FEM shape function, integrated over the boundary sideset and added to the fluid momentum equation for the corresponding node j, vis:

$$\int N_j \tau_w \hat{t} \delta_w(\phi) d\Gamma$$

References

Stephan F. Kistler 1993. “Hydrodynamics of Wetting” in Wettability, edited by John Berg, Surfactant Science Series, 49, Marcel Dekker, NewYork, NY, pp. 311-429.

WETTING_SPEED_LINEAR

```
BC = WETTING_SPEED_LINEAR SS <bc_id> <floatlist
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*.

A description of the input parameters follows:

WETTING_SPEED_LINEAR	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degrees.
<float2>	c_T, proportionality constant as defined below
<float3>	w ,width of interfacial region near contact line. Defaults to level set length scale if zero or less (L).
<float4>	β, slip coefficient.
<float5>	currently not used.
<float6>	currently not used.
<float7>	currently not used.

Examples

An example:

```
BC = WETTING_SPEED_LINEAR SS 10 30.0 0.1 0. 0.001 0. 0. 0.
```

Technical Discussion

The prescence of wetting or contact lines in problems using level set interface tracking introduces the problem of modeling the motion of the wetting line. This boundary condition presents one potential means for doing this. It adds a wall stress value at the boundary in a region near to the wetting line (this is set with the Level Set Length Scale discussed previously). This wall stress value depends upon the deviation of the apparent contact angle determined from the level set function and a set static contact angle. The bigger the deviation in principle the bigger the induced stress. The stress is modeled by analogy with Navier's slip relation (with slip coefficient β). The stress will induce a fluid velocity at the boundary which it is hoped will move the contact line at a velocity that is consistent with the rest of the flow.

An important note is that it is a requirement that when using this boundary condition that slip to some extent be allowed on this boundary. This is most often done by applying a VELO_SLIP_LS boundary condition in conjunction with this boundary condition. In addition, a no penetration condition on the velocity is need in either the form of a Dirichlet condition or a VELO_NORMAL condition. It is important to note that the slipping condition need not relax the no slip requirement completely. In fact, its parameters should be set so that no slip is for the most part satisfied on the boundary in regions away from the contact line. Near the contact line however the parameters in the slip condition and the WETTING_SPEED_LINEAR condition need to be fixed so that appreciable fluid velocity is induced. This is a trial and error process at the current time.

Theory

Derivation of this boundary condition starts with a simple relation for wetting line velocity

$$V_{wet} = \frac{1}{c_T}(\cos((\theta) - \cos(\theta_s)))$$

Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation

$$\tau_w = \frac{V_{wet}}{\beta} = \frac{1}{\beta c_T}(\cos((\theta) - \cos(\theta_s)))$$

It should be noted that there is no distinction for this model in the function of β or c_T. The two parameters are interchangeable. In non-linear models, (see WETTING_SPEED_BLAKE) this is no longer true.

In point of fact this condition is a vector condition so this scalar stress value multiplies the unit vector tangent to the surface and normal to the contact line, $\vec{t}$. This stress is then weighted by smooth Dirac function to restrict its location to being near the interface, weighted by a FEM shape function, integrated over the boundary sideset and added to the fluid momentum equation for the corresponding node j, vis:

$$\int N_j \tau_w \hat{t} \delta_w(\phi) d\Gamma$$

References

No References.

LINEAR_WETTING_SIC

```
BC = LINEAR_WETTING_SIC SS <bc_id> <floatlist>
```

Description / Usage

(SIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It is an alternative to the WETTING_SPEED_LINEAR BC which does not require the VELO_SLIP_LS or VELO_SLIP_FILL BC's on the wetting boundary.

A description of the input parameters follows:

LINEAR_WETTING_SIC	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	c_T, proportionality constant as defined below.
<float3>	w, width of interfacial region near contact line. Defaults to level set length scale if zero or less (L).
<float4>	β, slip coefficient.
<float5>	v_{sx}, x-component of substrate velocity.
<float6>	v_{sy}, y-component of substrate velocity.
<float7>	v_{sz}, z-component of substrate velocity.
<float8>	τ, stability parameter.

Examples

Here is an example card:

```
BC = LINEAR_WETTING_SIC SS 10 30.0 0.1 0. 0.001 0. 0. 0. 0.
```

Technical Discussion

This boundary condition is an additional means to impose a wetting line velocity at the contact line for level set interface tracking problems. The boundary condition uses a form of the Navier-Stokes slip condition to impose a boundary shear stress term to the momentum equation:

$$\vec{n}\vec{t}:T = -\frac{1}{\beta}\left(-\tau\frac{\partial\vec{v}}{\partial t} + \vec{v}_s - \vec{v} + f(F)V_{wet}\vec{t}\right)\cdot\vec{t}$$

where $\vec{n}$ and $\vec{t}$ are the normal and tangent boundary vectors, respectively, β is the "slipping" parameter which in this context is used actually as a penalty parameter, $\vec{v}_s$ is the substrate velocity, τ is a stabilization parameter, V_{wet} is the wetting velocity given by the following relation

$$V_{wet} = \frac{1}{c_T}(\cos(\theta) - \cos(\theta_s))$$

The masking function f(F) is given by the following relation as well:

$$f(F) = \delta_\alpha(F) = \frac{1}{2}\left(1 + \cos\left(\frac{\pi F}{\alpha}\right)\right)$$

where α is the width of the interfacial region near the contact line itself. It has the effect of "turning off" the wetting velocity at points on the boundary away from the interface.

This constraint is then introduced into the fluid momentum equation via the weak natural boundary condition term:

When applying this boundary condition, the user should choose a value for β which is relatively small. Its size is dictated by the requirement that away from the interface this boundary condition should be imposing a no-slip condition on the fluid velocity. Conversely, in the vicinity of the wetting line this boundary condition will impose the wetting velocity as computed from the preceding equation.

This boundary condition probably should be used in conjunction with a no penetration boundary condition, for example, a VELO_NORMAL condition on the same sideset or potentially a Dirichlet condition on velocity if the geometry permits this. In theory, this boundary condition can be used to impose no penetration as well, but this will require a very small value for β. The user should experiment with this.

The stability parameter, τ, as requires commentary. It is helpful to imagine that this parameter introduces a certain amount of inertia to motion of the contact line. With this term active (non-zero value for τ), large changes of the contact line velocity with time are restricted. This can be quite helpful during startup when the intial contact angle is often very different from its equilibrium value and there can be very large velocities generated as a result. These may in turn lead to low time step size and other numerical problems.

Although every situation is different, one should choose values for τ which are on the order of 1 to 10 times the starting time step size of the simulation. One should also recognize that this term is not consistent from a physical standpoint and

$$\int N_j \vec{\vec{n}}\vec{\vec{t}} : T d\Gamma$$

therefore one should endeavor to keep τ as small as possible if not in fact equal to zero.

References

No References.

BLAKE_DIRICHLET

```
BC = BLAKE_DIRICHLET SS <bc_id> <floatlist>
```

Description / Usage

(SIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It is an alternative to the WETTING_SPEED_BLAKE BC which does not require the VELO_SLIP_LS or VELO_SLIP_FILL BC's on the wetting boundary. It uses a Blake-DeConninck relationship between apparent contact angle and wetting velocity. As the name implies, this boundary condition differs from WETTING_SPEED_BLAKE in that wetting velocity is in a strong fashion on the wetting boundary.

A description of the input parameters follows:

BLAKE_DIRICHLET	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degrees.
<float2>	V_0 is a pre-exponential velocity factor (see functional form below). (L/T)
<float3>	g is a thermally scaled surface tension, i.e. σ/2nkT. Note that this parameter will be multiplied by the surface tension supplied in the material file when its used in the wetting velocity relation.
<float4>	w, is the width of the interface wetting region. It defaults to the level set length scale if zero of less.
<float5>	τ, stability parameter (T).
<float6>	v_{sx}, x-component of substrate velocity.
<float7>	v_{sy}, y-component of substrate velocity (L/T).
<float8>	v_{sz}, z-component of substrate velocity.

Examples

Here is an example card:

```
BC = BLAKE_DIRICHLET SS 10 30.0 20.1 7.0 0.0 0.001 0. 0. 0.
```

Technical Discussion

This boundary condition is an additional means to impose a wetting line velocity at the contact line for level set interface tracking problems. It is related to the WETTING_SPEED_BLAKE condition in that it uses the same Blake-DeConninck relationship between contact angle and wetting speed, but it applies this relation to the computational setting in a different way.

In this case, the following vector constraint is added to fluid momentum equation on the sideset to which this boundary condition is applied:

$$P\left(-\tau\frac{\partial\hat{\mathbf{v}}}{\partial t}+V_w f(\phi;w)\hat{\mathbf{t}}+\hat{\mathbf{v}}_s-\hat{\mathbf{v}}\right)$$

The factor P is a large penalty parameter which swamps any contributions from the volumetric momentum equations. Thus, the velocity, sec v , on this boundary will be set solely by the preceding constraint. In this sense, it is a Dirichlet condition (strictly speaking, Dirichlet conditions involve direct substitution of nodal degrees-of-freedom with corresponding elimination of its equation from the matrix which this boundary condition does NOT do).

In the preceding, the vector sec t, is a tangent vector to the surface and always points in the same direction of the level set gradient on the boundary (that is, from negative to positive). In three dimensions, sec t will also be normal to the contact line curve as it intersects the surface itself.

The masking function, f(ϕ;w) , is used to limit the application of the wetting line velocity to only that region of the boundary that is the in immediate vicinity of the contact line. We use a simple "hat" function:

$$f(\phi;w)=\begin{cases}1+\left(\dfrac{\phi}{w}\right); & (-w<\phi\le 0)\\ 1-\left(\dfrac{\phi}{w}\right); & (0<\phi\le w)\end{cases}$$

Needless to say, f(ϕ;w) is identically zero for level set values outside the interval (-w,*w*).

The stabilization term, -τ $\frac{\partial v}{\partial t}$, is intended to introduce something like inertia to the wetting line. That is to say, it's primary effect is to limit the rate of change of the wetting line velocity to "reasonable" values. The τ parameter should be chosen to be on the order of the smallest anticipated time step size in the problem. Setting it at zero, of course, will remove this term entirely.

In general, this boundary condition can be used to exclusively to set both the wetting speed velocity and the no slip requirement on the indicated sideset set. This would also include the no penetration requirement. The user may, however, find it advantageous to apply this constraint directly with the VELO_NORMAL condition on the same side set.

An additional note is that the "scaled viscosity" parameter g will be multiplied by the surface tension value supplied with Surface Tension card in the material file.

Theory

The wetting speed model for this boundary condition is the same used by the WETTING_SPEED_BLAKE card:

$$V_w = V_0 \sinh(g(\cos\theta_s - \cos\theta))$$

References

T. D. Blake and J. De Coninck 2002. "The Influence of Solid-Liquid Interactions on Dynamic Wetting", Advances in Colloid and Interface Science, 96, 21-36.

COX_DIRICHLET

```
BC = COX_DIRICHLET SS <bc_id> <floatlist>
```

Description / Usage

(SIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It is an alternative to the WETTING_SPEED_COX which does not require the VELO_SLIP_LS or VELO_SLIP_FILL BC's on the wetting boundary. It implements a version of the Cox hydrodynamic model of wetting (see below). As the name implies, this boundary condition differs from WETTING_SPEED_COX in that wetting velocity is applied in a strong fashion on the wetting boundary.

A description of the input parameters follows:

COX_DIRICHLET	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	ε_s is the dimensionless slip length, i.e. the ratio of the slip length to the characteristic length scale of the macroscopic flow. (L)
<float3>	σ is the surface tension. Note this value will be scaled by the surface tension value supplied in the material file.(F/L)
<float4>	w, is the width of the interface wetting region. It defaults to the level set length scale if zero of less (L).
<float5>	τ, stability parameter (T).
<float6>	v_{sx}, x-component of substrate velocity.
<float7>	v_{sy}, y-component of substrate velocity (L/T).
<float8>	v_{sz}, z-component of substrate velocity.

Examples

Here is an example card:

```
BC = COX_DIRICHLET SS 10 30.0 0.01 72.0   0. 0.001 0. 0. 0.
```

Technical Discussion

This boundary condition is applied in exactly the same manner as the BLAKE_DIRICHLET boundary condition. The only substantial difference is the model used to derive the wetting speeds relation to the local apparent contact angle. The reader is referred to the BLAKE_DIRICHLET section of the manual for further reference.

Theory

This boundary condition uses this relation that represents the Cox hydrodynamic wetting model

$$Ca \equiv \frac{\mu V_w}{\sigma} = \frac{g(\theta, \lambda) - g(\theta_s, \lambda)}{\left[\ln(\varepsilon_s^{-1}) + \frac{q_{inner}}{f(\theta_s, \lambda)} - \frac{q_{outer}}{f(\theta, \lambda)}\right]}$$

See VELO_THETA_COX for details of the Cox functions f and g. Note that the parameters λ, q_{inner}, and q_{outer} are currently not accessible from the input card and are hard-set to zero. λ is the ratio of gas viscosity to liquid viscosity whereas q_{inner} and q_{outer} represent influences from the inner and outer flow regions.

References

Stephan F. Kistler 1993. “Hydrodynamics of Wetting” in Wettability, edited by John Berg, Surfactant Science Series, 49, Marcel Dekker, NewYork, NY, pp. 311-429.

HOFFMAN_DIRICHLET

```
BC = HOFFMAN_DIRICHLET SS <bc_id> <floatlist>
```

Description / Usage

(SIC/VECTOR MOMENTUM)

This boundary condition is used to induce fluid velocity at a wetting line when using *Level Set Interface Tracking*. It is an alternative to the WETTING_SPEED_HOFFMAN boundary condition which does not require the VELO_SLIP_LS or VELO_SLIP_FILL BC's on the wetting boundary. As the name implies, this boundary condition differs from WETTING_SPEED_HOFFMAN in that wetting velocity is in a strong fashion on the wetting boundary.

A description of the input parameters follows:

HOFFMAN_DIRICHLET	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	θ_s, the static contact angle in degress.
<float2>	not used.
<float3>	σ, the surface tension. Note that this parameter will be scaled by the surface tension value supplied in the material file. (F/L)
<float4>	w, is the width of the interface wetting region. It defaults to the level set length scale if zero of less.
<float5>	τ, stability parameter (T).
<float6>	v_{sx}, x-component of substrate velocity.
<float7>	v_{sy}, y-component of substrate velocity (L/T).
<float8>	v_{sz}, z-component of substrate velocity.

Examples

Here is an example card:

```
BC = HOFFMAN_DIRICHLET SS 10 30.0 0 72.0 0.0 0.001 0. 0. 0.
```

Technical Discussion

The technical details of the application of this boundary differ not at all from those described for the BLAKE_DIRICHLET boundary condition. The user is referred to that section for further details. This boundary condition differs only in the model used to determine the wetting velocity. This is described below and in the VELO_THETA_HOFFMAN card.

Theory

Derivation of this boundary condition starts with a relation that represents the Hoffman wetting correlation

$$Ca \equiv \frac{\mu V_w}{\sigma} = g_{Hoff}(\theta) - g_{Hoff}(\theta_s)$$

See VELO_THETA_HOFFMAN for details of the Hoffman function g. Note that the convention for contact angles in this relation is that values of θ near to zero indicate a high degree of wetting and values of θ near 180 ° indicate the opposite. This is mapped to a stress value by analogy with Navier's slip relation,

$$\tau_w = \frac{V_{Hoffman}}{\beta}$$

References

Stephan F. Kistler 1993. "Hydrodynamics of Wetting" in Wettability, edited by John Berg, Surfactant Science Series, 49, Marcel Dekker, NewYork, NY, pp. 311-429.

VELO_SLIP_LS

```
BC = VELO_SLIP_LS SS <bc_id> <float_list>
```

Description / Usage

(WIC/VECTOR MOMENTUM)

This boundary condition is applied only in problems involving embedded interface tracking, that is, level set or volume of fluid. The boundary condition serves two major purposes: first to allow for slip in the vicinity of a moving contact line and second to facilitate impact of a dense fluid on a substrate, displacing a less dense fluid (e.g. water drop and air displacement). . Elsewhere, this boundary condition enforces a no-slip condition between fluid and substrate. A more detailed description is given below.

This boundary condition is most often used in conjunction with the FILL_CA, WETTING_SPEED_LINEAR, or WETTING_SPEED_BLAKE boundary conditions. These apply forces to contact lines in order to simulate wetting line motion. These forces are applied in a weak sense to the same regions near the interface so it is necessary to use VELO_SLIP_LS with a large slipping coefficient so that effectively no-slip is relaxed completely near the interface.

Definitions of the input parameters are as follows:

VELO_SLIP_LS	Name of the boundary condition.
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float1>	*alpha*, or *slip_width*, a characteristic length scale around the contact line that will be used to apply Navier Slip Condition with β 0 coefficient. This length scale is also used to detect the thickness of light-phase (gas) between the substrate denoted by the sideset, and the zero-level-set contour (or the boundary between liquid and gas). If this distance is less than 8*slip_width, then perfect slip in the gas phase is allowed to help facilitate contact. See discussion below.
<float2>	β 0, the slip coefficient near the contact line. The inverse of β 0 defines the scaling between stress and slip. The parameter supplied on the input deck is used only within a lengthscale *slip_width* setting around the contact line.. Elsewhere, the slip coefficient is uniformly set to β 1. Hence, this parameter is usually set to a large value to allow for perfect slip.
<float3>	$v_{s,x}$, the x-component of surface velocity vector. This would be the x-component of the fluid velocity if a noslip condition were applied.
<float4>	vs, y, the y-component of surface velocity vector. This would be the y-component of the fluid velocity if a noslip condition were applied.
<float5>	$v_{s,z}$, the z-component of surface velocity vector. This would be the z-component of the fluid velocity if a noslip condition were applied.
<float6>	β 1, the slip coefficient away from the contact line. The inverse of β 1 defines the scaling between stress and slip. Hence, this parameter is usually set to a small value (like 1e-6) to allow for no-slip.

Examples

Following is a sample card without the optional parameters:

```
BC = VELO_SLIP_LS SS 10 0.05 100000.0 0.0 0.0 1.e-6
```

The large value of slip coefficient ensures nearly perfect slip in the region around the interface, a region that has a half-width of 0.05 centered about the contact line. Away from the contact line (outside the hat function of width 0.05), the slip coefficient is 1.e-6, which corresponds to significantly less slip. Note also that if the substrate defined by SS 10 is in contact with gas (light phase), and a liquid front (zero-level set) is nearby, or within a distance 8 times 0.05, then the light phase is allowed to slip along the wall with a coefficient of 100000.0. This is to help facilitate contact.

Technical Discussion

This boundary condition was originally developed to allow for fluid slip near a dynamic contact line, a necessary condition for dynamic wetting line motion when the contact angle is not 180 degrees (viz. rolling motion condition). The slippage mechanism was deployed through the use of Navier's slip condition, which basically goes as

Here β is the slip coefficient, which is taken to be variable depending on its proximity to the contact line (through the "slip_width" parameter). Note that the smaller the β, the more no-slip is enforced. The left hand side of this condition is the fluid traction on the substrate. $\underline{v}_s$ is the velocity of the substrate, specified component-wise with {vx} {vy} {vz}. This base functionality of applying the Navier slip condition still exists in this condition, but in addition it was furbished to allow for complete slip on the boundary if a gas film is being displaced by liquid. In this latter case, complete slip is a

$$\underset{\sim}{n} \cdot \underset{\sim}{T} = \frac{\underset{\sim}{v} - \underset{\sim}{v}_s}{\beta}$$

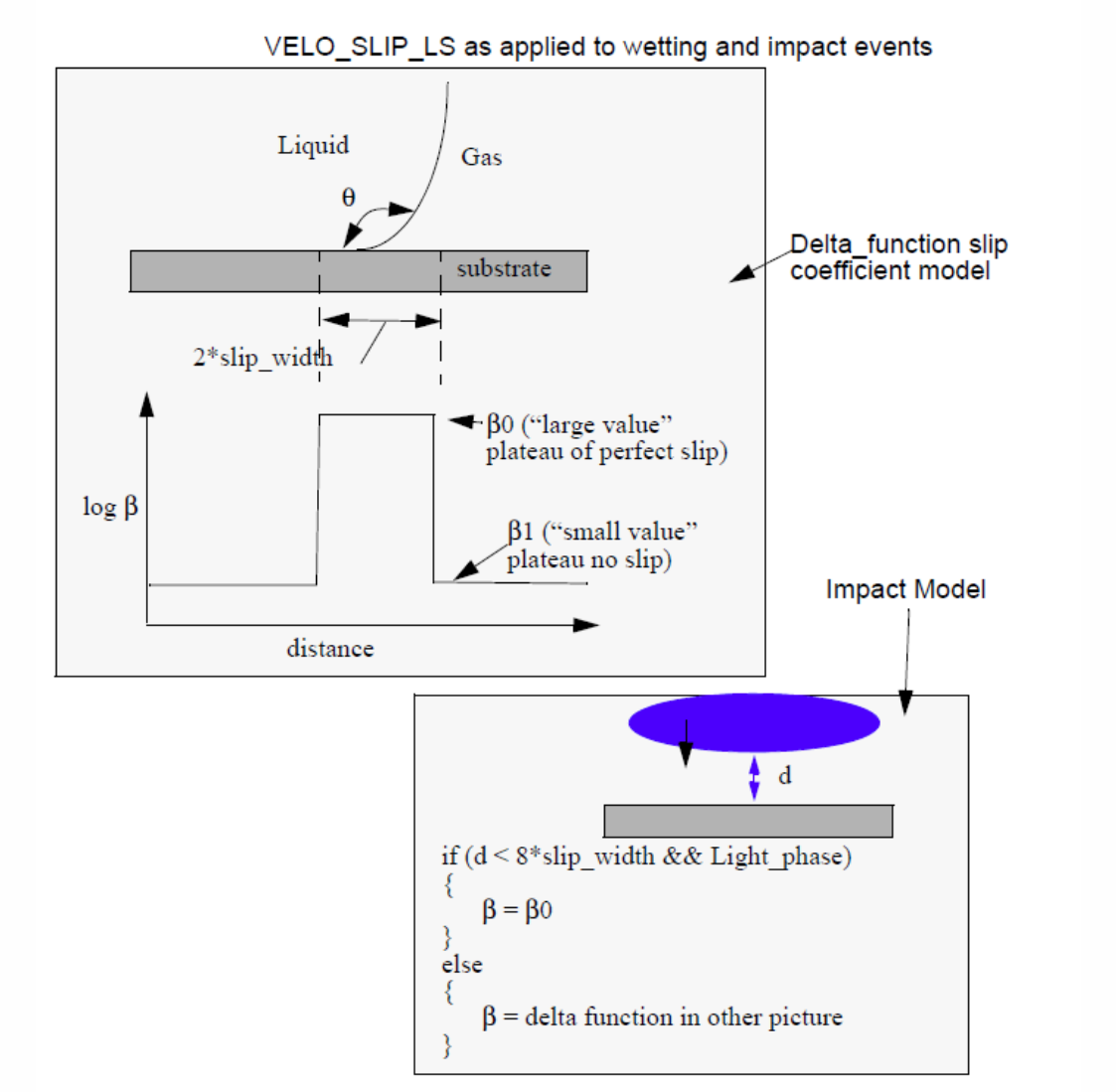
mechanism (subgrid event) that allows for the otherwise infinite stress to be relieved so that the liquid can make contact with the solid. The perfect slip condition at the substrate/gas surface is activated by just setting the slip coefficient to the large value, as this condition does anyway in the vicinity of a contact line. The "gas phase" is determined by determining which phase is the lighter one based on the density specification. The figure below details more on how this condition works for wetting/dewetting and for incipient liquid/solid impact.

Some more usage notes as follows:

- The slip coefficient function is computed as $\beta = \beta\ 0\ \delta\ (\ \phi\ ) + \beta_\infty$, where the delta function is a level-set hat function centered around the zero level set contour where it intersects the boundary. It has a length sacale associated with it which is called "alpha", and that basically sets the length over which the $\beta\ 0$ is applied as the slip parameter and it is large, leading to a shear-stress-free or slippery region in both the gas and liquid phases. β_{INF} is taken as real small (typically 1.e-6 or less) and is applied away from the contact line, and hence forces a true "no-slip" condition.
- The most recent addition to this condition is the functionality that adds perfect slip to a wall in the gas phase as it is displaced during near contact state by a liquid phase. This capability is of course applicable only to level-set capillary hydrodynamics problems. Level-set methods have been plagued by the fact that it is hard to break down the displaced phase (e.g. gas phase) as a liquid phase surface flows towards a solid boundary. Theoretically this event requires an infinite stress, in the continuum. To relieve this stress and promote a collapse and wetting, we add perfect slip in the gas phase at near contact conditions, which reduces the lubrication pressure in the gas film and promotes breakdown. This of course introduces more length scales. First, the length scale over which slip is applied (this is the alpha parameter described above) and seond is the length scale over which "nearness" of the liquid phase to the substrate is considered to be "close enough" to allow for perfect slip. Right now this "nearness" length scale is arbitrarily set to 8*alpha. A third length scale is that which we use to declare contact. We currently have that set to 1.e-6*alpha. After contact is declared, VELO_SLIP_LS reverts to the form under the first bullet. The figure below hopefully clarifies the condition a little better.

Category 16: Shell Equations

These boundary conditions are applied to shell equations, a special category of equations applied on 1D boundaries of 2D surfaces in Goma.


VELO_SLIP_LS as applied to wetting and impact events
Liquid
Gas
θ
substrate
Delta_function slip coefficient model
2*slip_width
β0 ("large value" plateau of perfect slip)
log β
β1 ("small value" plateau no slip)
distance
Impact Model
d
if (d < 8*slip_width && Light_phase)
{
β = β0
}
else
{
β = delta function in other picture
}

SHELL_SURFACE_CHARGE

```
BC = SHELL_SURFACE_CHARGE SS <bc_id> <integer>
```

Description / Usage

(WIC/POTENTIAL)

This boundary condition card is used to add to the potential equation the surface charge term at a shell surface. Definitions of the input parameters are as follows:

SHELL_SURFACE_CHARGE	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This boundary must coincide with the shell element block on which the surface charge equation is applied.
<integer>	Integer value indicating the bulk element block ID from which to apply the boundary condition (not currently implemented).

This boundary condition is currently inoperative.

Examples

For a system consisting of a solid material (element block ID 1) with a conducting shell surface (element block ID 2) whose location coincides with side set 20, the following is a sample usage:

```
BC = SHELL_SURFACE_CHARGE SS 20 1
```

Technical Discussion

This boundary condition was originally developed to allow for fluid slip near a dynamic contact line, a necessary condition for dynamic wetting line motion when the contact angle is not 180 degrees (viz. rolling motion condition). The slippage mechanism was deployed through the use of Navier's slip condition, which basically goes as

$$(\varepsilon \underline{n} \cdot [\underline{\underline{E}}^{(o)} - \underline{\underline{E}}^{(i)}]) = -\frac{\sigma}{2}$$

where **E** is the electric field vector, the superscripts (o) and (i) denote the outer and inner phases, n is a unit normal pointing into the outer phase, ε is the electrical permittivity, $E = -\Delta V$ is the electric field and V is the voltage or electric potential.

SHELL_SURFACE_CHARGE_SIC

```
BC = SHELL_SURFACE_CHARGE_SIC SS <bc_id> <integer>
```

Description / Usage

(WIC/POTENTIAL)

This boundary condition card is used to add to the potential equation the surface charge term at a shell surface. Physically it is the same as the SHELL_SURFACE_CHARGE boundary condition, but is applied as a strongly-integrated condition. Definitions of the input parameters are as follows:

SHELL_SURFACE_CHARGE_SIC	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This boundary must coincide with the shell element block on which the surface charge equation is applied.
<integer>	Integer value indicating the bulk element block ID from which to apply the boundary condition (not currently implemented).

This boundary condition is currently inoperative…

Examples

For a system consisting of a solid material (element block ID 1) with a conducting shell surface (element block ID 2) whose location coincides with side set 20, the following is a sample usage:

```
BC = SHELL_SURFACE_CHARGE_SIC SS 20 1
```

Technical Discussion

This boundary condition applies a surface charge balance along the shell surface.. In its most general form, this balance is written

$$(\varepsilon \underline{n} \cdot [\underline{\underline{E}}^{(o)} - \underline{\underline{E}}^{(i)}]) = -\frac{\sigma}{2}$$

where **E** is the electric field vector, the superscripts (o) and (i) denote the outer and inner phases, n is a unit normal pointing into the outer phase, ε is the electrical permittivity, $\underline{E} = -\Delta V$ is the electric field and V is the voltage or electric potential.

SURFACE_ELECTRIC_FIELD

```
BC = SURFACE_ELECTRIC_FIELD SS <bc_id> <integer> <integer> <integer>
```

Description / Usage

(WSG/SURFACE CHARGE)

This boundary condition card is used to apply a part of the shell surface charge equation which includes the electric field, the negative gradient of the potential variable which is applied on a neighboring bulk block. It is actually an integral part of the surface charge equation. Definitions of the input parameters are as follows:

SURFACE_ELECTRIC_FIELD	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain. This boundary must coincide with the shell element block on which the surface charge equation is applied.
<integer>	Bulk element block ID (from ExodusII database) for neighboring bulk element block on which the potential equation is applied.
<integer>	Shell element block ID (from ExodusII database) for shell block on which the surface charge equation is applied.

This boundary condition is currently inoperative…

Examples

For a system consisting of a solid material (element block ID 1) with a conducting shell surface (element block ID 2) whose location coincides with side set 20, the following is a sample usage:

```
BC = SURFACE_ELECTRIC_FIELD SS 20 1 2
```

Technical Discussion

This is a special type of boundary condition, WEAK_SHELL_GRAD, which is a portion of a shell equation which involves spatial gradients of bulk variables. Since the values of bulk variable gradients depend on all of the degrees of freedom of that variable in the bulk element, and sensitivities to the off-shell degrees of freedom must be applied, a portion of the equation must be evaluated from the bulk side. This is done in Goma by means of a WEAK_SHELL_GRAD boundary condition which evaluates these terms and all bulk sensitivities from the bulk side, the saves these values for later recall when the rest of the surface charge equation is assembled.

In this case, the term n • Δ V and its potential sensitivities are evaluated within the bulk element for inclusion in the surface charge balance along the shell surface.. I

References

No References.

SH_TENS

```
BC = SH_TENS NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/shell_tension)

This Dirichlet boundary condition specification is used to set a tension (in stress per unit length) to the inextensible shell equations (see EQ = shell_tension and EQ = shell_curvature) at an endpoint. This boundary condition can be applied in two dimensions only, and only to the endpoint of a bar-type element. In put is as follows :

SH_TENS	Boundary condition name (<bc_name>) that defines the shell tension (compressive or expansion depending on the sign).
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database. Note that this must be a single-node node set representing and endpoint to a bar element type.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of tension.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the tension for a shell equation:

```
BC = SH_TENS NS 100 10.
```

This condition sets a tension of 10.0 at Nodeset 100.

Technical Discussion

No Discussion.

References

GT-027.1: GOMA's Shell Structure Capability: User Tutorial (GT-027.1)

SH_K

```
BC = SH_K NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/shell_curvature)

This Dirichlet boundary condition specification is used to set a curvature to the inextensible shell equations (see EQ = shell_tension and EQ = shell_curvature) at an endpoint. This boundary condition can be applied in two dimensions only, and only to the endpoint of a bar-type element:

SH_K	Boundary condition name (<bc_name>) that defines the shell curvature.
NS	Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database. Note that this must be a single-node node set representing and endpoint to a bar element type.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Value of curvature.
[float2]	An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample card for applying a Dirichlet condition on the curvature for a shell equation:

```
BC = SH_K NS 100 0.
```

This condition sets a curvature of zero at Nodeset 100.

Technical Discussion

No Discussion.

References

GT-027.1: GOMA's Shell Structure Capability: User Tutorial (GT-027.1)

SH_FLUID_STRESS

```
BC = SH_FLUID_STRESS SS <bc_id> <float>
```

Description / Usage

(PCC/VECTOR MOMENTUM)

Used for fluid-structure interaction problems with structural shell elements, the *SH_FLUID_STRESS* condition equates the normal traction (the tangential and normal force components, per unit area) between adjacent fluid and solid structure. This condition is only to be used on boundaries between regions of ARBITRARY mesh motion with fluid momentum equations: see *Mesh Motion* and *EQ* cards. With this boundary condition, the local residual and Jacobian contributions from the fluid mechanics momentum equations (on the *ARBITRARY* side of the boundary) are added into weak form of the residual and Jacobian entries for the solid structural equations (*see EQ = shell_curvature and EQ = shell_tension*). All elements on both sides of the interface must have the same element type, i.e., the same order of interpolation and basis functions, e.g., Q1 fluid and Q1 (bar element) for shell. Q2 fluid momentum and Q2 (bar element) for the shell equations. Also, such interfaces must be defined as a mesh side set attached to the bulk fluid elements (most mesh generators will not allow for side sets in bar or sheet elements).

Definitions of the input parameters are as follows:

SH_FLUID_STRESS	Name of the boundary condition (<bc_name>).
SS	Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<float>	Scale factor for stress balance for non-dimensionalization. This parameter, which multiplies the liquid phase contribution and should be set to 1.0 if there is no nondimensional treatment.

Examples

The following is a sample input card:

```
BC = SH_FLUID_STRESS SS 5 1.0
```

In this example, side set 5 is a boundary between a solid blade and a liquid; material 2 is the rubber blade, and material 1 is the fluid. Along that blade, a companion boundary condition of the form

```
BC = NO_SLIP SS 5 2 1
```

should also be applied.

Technical Discussion

The functional form of the boundary condition is:

$$\lambda(\underset{\sim}{n} \cdot \underset{\sim}{\boldsymbol{T}}) = \underset{\sim}{n} \cdot \underset{\sim}{\sigma}$$

where $\underline{T}$ is the fluid phase stress tensor given by any one of the specified fluid-phase constitutive equations, and $\underline{\sigma}$ is the solid-phase stress tensor, also given by any one of the solid-phase constitutive equation (see material file specifications). λ is a scaling factor that defaults to unity (and is usually best taken as such unless some scaling is invoked).

This balance is applied to the weak form of the solid-phase momentum residuals, from the fluid phase, viz. in the fluid-phase, the fluid-stress at the interface is added to the solid-phase momentum residuals. As mentioned above, this condition usually needs to be supplemented by a statement of mass conservation across the interface, which will depend on whether the solid phase is of *CONTINUOUS* or *POROUS* media (see *Media Type* card).

FAQs

- **Troubleshooting 1:** This boundary condition requires that the side set contain elements from both the fluid and the solid side of the interface. For the FASTQ tool, this is the default case; for CUBIT and possibly other related tools, this can be forced on the side set definition options. Interestingly, the boundary condition does work if the side set is attached to the fluid phase only, but just due to the way in which it is applied.
- **Troubleshooting 2:** This boundary condition does not enforce mass conservation. A combination of NO_SLIP or VELO_NORMAL/VELO_TANGENT must be invoked to achieve a material surface. For the latter, care must be taken to maintain the application of the VELO_NORMAL condition after a remesh. This condition is applied only to one side of the interface and depends on the ss_to_blks connectivity structure; it may be necessary to force its application, especially after remeshes. To be sure that the proper set of conditions is being applied, look at the BC_dup.txt file for nodes along the interface.

References

GT-003.1: Roll coating templates and tutorial for GOMA and SEAMS, February 29, 2000, P. R. Schunk and Matt Stay

GT-006.3: Slot and Roll coating with remeshing templates and tutorial for GOMA and CUBIT/MAPVAR, August 3, 1999, R. R. Lober and P. R. Schunk

LUB_PRESS

```
BC = LUB_PRESS NS <bc_id> <float_list>
```

Description / Usage

This boundary condition card applies a lubrication pressure to the boundary of a shellelement sheet. The corresponding equation is EQ=lubp. The boundary condition is applied to a node set.

LUB_PRESS	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	lub_p, the value of lubrication pressure at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminate from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = LUB_PRESS NS   100 100.
```

This condition applies a lubrication pressure of 100.0 at nodeset 100.

Technical Discussion

The equation applied at the specified nodeset in place of Reynold's lubrication equation for confined flow. Note that it is not to be used for the film-flow lubrication equations.

GRAD_LUB_PRESS

```
BC = GRAD_LUB_PRESS SS <bc_id> <float1>
```

Description / Usage

(WIC/R_LUBP)

This boundary condition card applies *free* boundary condition, akin to Papanastasiou et al. (1992) for the fluid momentum, at the boundary of a shell-element sheet. The boundary condition is applied to a sideset.

GRAD_LUB_PRESS	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	Flowrate in L^2/t. Usually set for NOBC effect.

Examples

Following is a sample card:

```
BC = GRAD_LUB_PRESS SS   100 0.
```

This condition applied at sideset 100.

Technical Discussion

No Discussion.

SHELL_FILMP

```
BC = SHELL_FILMP NS <bc_id> <float_list>
```

Description / Usage

This boundary condition card applies a film pressure to the boundary of a shell-element sheet. The corresponding equation is EQ=shell_filmp. The boundary condition is applied to a node set.

SHELL_FILMP	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	shell_filmp, the value of lubrication pressure at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminate from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = SHELL_FILMP NS   100 100.
```

This condition applies a film pressure of 100.0 at nodeset 100.

Technical Discussion

The equation applied at the specified nodeset in place of the film-flow lubrication equation.

SHELL_FILMH

```
BC = SHELL_FILMH NS <bc_id> <float_list>
```

Description / Usage

(DC/R_SHELL_FILMH)

This boundary condition card applies a film height to the boundary of a shell-element sheet. The corresponding equation is EQ=shell_filmh. The boundary condition is applied to a node set.

SHELL_FILMH	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	shell_filmh, the value of film thickness at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminate from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = SHELL_FILMH NS   100 1.
```

This condition applies a film height of 1.0 at nodeset 100.

Technical Discussion

The equation applied at the specified nodeset in place of the film-flow height equation.

SHELL_PARTC

```
BC = SHELL_PARTC NS <bc_id> <float_list>
```

Description / Usage

(DC/R_SHELL_PARTC)

This boundary condition card applies a particle volume fraction to the boundary of a shell-element sheet. The corresponding equation is EQ=shell_filmh. The boundary condition is applied to a node set.

SHELL_PARTC	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	shell_partc, the value of film thickness at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminate from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = SHELL_PARTC NS   100 0.
```

This condition applies a particles volume fractioin of 0.0 at nodeset 100.

Technical Discussion

The equation applied at the specified nodeset in place of the particles conservation equation.

References

No References.

SHELL_GRAD_FP

```
BC = BC = SHELL_GRAD_FP SS <bc_id> <float_list>
```

Description / Usage

(SIC/R_SHELL_GRAD_FP)

This boundary condition card applies a volumetric flux of liquid film to the boundary of a shell-element sheet. The corresponding equation is EQ=shell_filmp. The boundary condition is applied to a node set.

SHELL_GRAD_FP	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	volumetric flux

Examples

Following is a sample card:

```
BC = SSHELL_GRAD_FP SS    100 0.0
```

This condition applies a particles volume flux of 0.0 at nodeset 100.

Technical Discussion

The actual weighted residual equation that is applied to node on the surface is

$$\int \phi_i \left\{ \mathbf{n} \cdot \left[\frac{h^3}{3\mu} \left(-\nabla_{II} p \right) + \mathbf{U}_B h \right] - q \right\} \mathrm{d}\Gamma = 0$$

where ϕ_i is the finite element trial function, **n** is the outward-pointing normal to the surface, and q is the volumetric flux specified in the <float1>. Careful attention should be given for the sign of q. The **sign convention** is that q is **positive** when the flow is **exiting** the boundary and **negative** when **entering** the boundary.

The condition replaces the residual equation shell_filmp at the boundary.

SHELL_GRAD_FP_NOBC

```
BC = BC = SHELL_GRAD_FP_NOBC SS <bc_id>
```

Description / Usage

(WIC/R_SHELL_GRAD_FP_NOBC)

This boundary condition card applies *free* boundary condition, akin to Papanastasiou et al. (1992) for the fluid momentum, at the boundary of a shell-element sheet. The boundary condition is applied to a sideset.

SHELL_GRAD_FP_NOBC	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.

Examples

Following is a sample card:

```
BC = SSHELL_GRAD_FP_NOBC SS   100
```

This condition applied at sideset 100.

Technical Discussion

- The finite element formulation of the first equation of the film profile equation boundary integral in the form of

$$\int \phi_i \mathbf{n} \left[\frac{h^3}{3\mu} \left(-\nabla_{II} p \right) + \mathbf{U}_B h \right] d\Gamma = 0$$

- This condition is similar to the SHELL_GRAD_FP boundary condition, except that the condition is now a weak integrated condition that is *added* to the residual equations, instead of replacing them and the flux is no longer specified.

SHELL_GRAD_FH

```
BC = BC = SHELL_GRAD_FH SS <bc_id> <float_list>
```

Description / Usage

(SIC/R_SHELL_GRAD_FH)

This boundary condition card sets a slope to the liquid film at the boundary of a shellelement sheet. The corresponding equation is EQ=shell_filmh. The boundary condition is applied to a node set.

SHELL_GRAD_FH	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	slope

Examples

Following is a sample card:

```
BC = SSHELL_GRAD_FH SS   100 0.0
```

This condition applies a film slope of 0.0 at nodeset 100.

Technical Discussion

- he actual weighted residual equation that is applied to node on the surface is

$$\int \phi_i \left[\mathbf{n} \cdot \nabla_{II} h - \Sigma \right] d\Gamma = 0$$

where ϕ_i is the finite element trial function, **n** is the outward-pointing normal to the surface, Σ and is the slope specified in the <float1>.

- The condition replaces the residual equation shell_filmh at the boundary.

SHELL_GRAD_FH_NOBC

```
BC = BC = SHELL_GRAD_FH_NOBC SS <bc_id>
```

Description / Usage

(WIC/R_SHELL_GRAD_FH_NOBC)

This boundary condition card applies *free* boundary condition, akin to Papanastasiou et al. (1992) for the fluid momentum, at the boundary of a shell-element sheet, in terms of the slope of a thin Reynolds film. The boundary condition is applied to a sideset.

SHELL_GRAD_FH_NOBC	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.

Examples

Following is a sample card:

```
BC = SSHELL_GRAD_FH_NOBC SS   100
```

This condition applied at sideset 100.

Technical Discussion

- The finite element formulation of the second equation of film profile equation generates boundary integral in the form of

$$\int \phi_i \mathbf{n} \cdot \nabla_{II} h \, d\Gamma = 0$$

- This condition is similar to the SHELL_GRAD_FH boundary condition, except that the condition is now a weak integrated condition that is *added* to the residual equations, instead of replacing them and the flux is no longer specified.

SHELL_GRAD_PC

```
BC = BC = SHELL_GRAD_PC SS <bc_id> <float_list>
```

Description / Usage

(WIC/R_SHELL_GRAD_PC)

This boundary condition card allows the user to set volumetric flux of particles inside liquid film at the boundary of a shell-element sheet. The corresponding equation is EQ=shell_partc. The boundary condition is applied to a side set.

SHELL_GRAD_PC	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes side set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	particle flux

Examples

Following is a sample card:

```
BC = SHELL_GRAD_PC SS   100 1.
```

This condition applied at sideset 100. and sets a particle flux to 1.0

Technical Discussion

- The actual weighted residual equation that is applied to node on the surface is

$$\int \phi_i \left\{ \left[\mathbf{n} \cdot Dh \nabla_{II} \varphi - J_p \right] \right\} d\Gamma = 0$$

where ϕ_i is the finite element trial function, **n** is the outward-pointing normal to the surface, J_p and is the particles flux specified in the <float1>.

- The condition replaces the residual equation shell_partc at the boundary.

SHELL_LUBP_SOLID

```
BC = SH_LUBP_SOLID SS <bc_id> <float1>
```

Description / Usage

(WIC/R_MESH1/R_MESH2/RMESH3)

This vector boundary condition card balances the stress in an abutting continuum elastic solid with the lubrication forces (pressure and shear) in a surface shell. The boundary condition is applied to a sideset. Please see notes below on the sideset features which must be specified.

SH_LUBP_SOLID	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes sideset in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (nodeset in EXODUS II) in the problem domain.
<float1>	Scaling factor. Normally set this to 1.0, unless a stressbalance scale is required due to nondimensionalization.

Examples

Following is a sample card:

```
BC = SH_LUBP_SOLID SS   100 1.0
```

This boundary condition is applied at sideset 100.

Technical Discussion

- The mathematical form of the boundary condition is

$$\underline{n} \cdot \underline{\underline{\sigma}} = \underline{k} P_{\text{lub}} + \underline{i}\left(\frac{h}{12}\frac{\partial P}{\partial x} + \frac{u\,\mu}{h}\right) + \underline{j}\left(\frac{h}{12}\frac{\partial P}{\partial x} + \frac{v\,\mu}{h}\right)$$

- This condition is similar to FLUID_SOLID and SOLID_FLUID boundary conditions for the case of fluid-structure interaction between two continuum regions, one fluid and one solid.
- Note that the sideset as generated in CUBIT or related software is actually attached to the continuum domain and not the shell face, as those faces (top and bottom of sheet and not the edges) are not true finite element sides. Most mesh generators will not allow sidesets to be include shell element faces. GOMA figures out the right thing to do.

SHELL_TEMP

```
BC = SHELL_TEMP NS <bc_id> <float_list>
```

Description / Usage

(DC/R_SHELL_ENERGY)

This boundary condition card applies a shell temperature to the boundary of a shellelement sheet. The corresponding equation is EQ=shell_energy. The boundary condition is applied to a node set.

SHELL_TEMP	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	SHELL_TEMP, the value of temperature at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminated from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = SHELL_TEMP 100 1.0
```

This boundary condition is applied at nodeset 100.

Technical Discussion

- The equation applied at the specified nodeset in place of the shell-temperature equation.

References

No References.

SHELL_OPEN_PRESS, SHELL_OPEN_PRESS_2

```
BC = SHELL_OPEN_PRESS NS <bc_id> <float_list>
```

```
BC = SHELL_OPEN_PRESS_2 NS <bc_id> <float_list>
```

Description / Usage

(DC/R_SHELL_SAT_OPEN or SHELL_SAT_OPEN_2)

This Dirichlet boundary condition card applies a shell liquid phase pressure to the boundary of a shell-element sheet. The corresponding equation is EQ=shell_sat_open or correspondingly shell_sat_open_2, depending on which layer. The boundary condition is applied to a node set.

SHELL_OPEN_PRESS	Name of boundary condition.
NS	Type of boundary condition (<bc_type>), where NS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.
<float1>	SHELL_OPEN_PRESSURE, the value of the liquid phase pressure at the boundary.
[float2]	Optional floating point number set between 0.0 and 1.0 which serves as a flag to the code for a Dirichlet boundary condition. If this value is present, and is not 1.0, the condition is applied as a residual equation. Otherwise, it is "hard-set" condition and is eliminated from the matrix. The residual method must beused when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.

Examples

Following is a sample card:

```
BC = SHELL_OPEN_PRESS 100 1.0
```

This boundary condition is applied at nodeset 100.

Technical Discussion

- The equation applied at the specified nodeset in place of the shell-sat-open equation.

References

No References.

LUBP_SH_FP_FLUX

```
C = LUBP_SH_FP_FLUX SS <bc_id> <int1> <int2>
```

Description / Usage

(COLLOC/R_SHELL_FILMP)

This boundary condition card matches the mass flux in one region of confined flow (lubp) to the mass flux from a second region of film flow (shell_filmp). The flux matching is handled as a sideset between two shell regions. In this way both equations can be coupled for exit or entrance flows. The boundary condition is applied in collocated form, and replaces the R_SHELL_FILMP equation.

LUBP_SH_FP_FLUX	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<int1>	Block id of mesh material which invokes the lubp equation.
<int2>	Block id of mesh material which invokes the shell_filmp equation.

Examples

Following is a sample card:

```
BC = LUBP_SH_FP_FLUX SS   100 2 1
```

This condition applies the matching tie condition at a side set boundary between block 2 (which invokes the EQ = lubp equation) and block 1 (which invokes the EQ=shell_filmp equation).

Technical Discussion

The best example of the use of this equation is the exit of a metered coating flow. It must be used together with a pressure-matching condition LUBP_SH_FP_MATCH.

LUBP_SH_FP_MATCH

```
C = LUBP_SH_FP_MATCH SS <bc_id> <int1> <int2>
```

Description / Usage

(STRONG_INT_SURF/R_LUBP)

This boundary condition card matches the pressure in one region of confined flow (lubp) to the pressure from a second region of film flow (shell_filmp). The pressure matching is handled as a sideset between two shell regions. In this way both equations can be coupled for exit or entrance flows. The boundary condition is applied in collocated form, and replaces the R_LUBP equation.

LUBP_SH_FP_FLUX	Name of boundary condition.
SS	Type of boundary condition (<bc_type>), where SS denotes node set in the EXODUS II database.
<bc_id>	The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.
<int1>	Block id of mesh material which invokes the lubp equation.
<int2>	Block id of mesh material which invokes the shell_filmp equation.

Examples

Following is a sample card:

```
BC = LUBP_SH_FP_MATCH SS   100 2 1
```

This condition applies the matching tie condition at a side set boundary between block 2 (which invokes the EQ = lubp equation) and block 1 (which invokes the EQ=lubp equation).

Technical Discussion

The best example of the use of this equation is the exit of a metered coating flow. It must be used together with a flux-matching condition LUBP_SH_FP_FLUX.

Category 17: Acoustic Equations

These boundary conditions are applied to acoustic equations (Preal, Pimag, Reyn_stress).

APR

```
BC = APR NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/ACOUS_PREAL)

This Dirichlet boundary condition card is used to set constant amplitude of the real part of the acoustic pressure. Definitions of the input parameters are as follows:

APR Name of the boundary condition (<bc_name>).

NS Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.

<float1> Value of the real part of the acoustic pressure amplitude.

[float2] An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences*.

Examples

The following is a sample input card:

```
BC = APR NS 100 1000.0
```

APR_PLANE_TRAN

```
BC = APR_PLANE_TRAN SS <bc_id> <float1>
```

Description / Usage

(WIC/SCALAR ACOUS_PREAL)

This boundary condition card applies the plane wave transmission conditions to the acoustic wave equations. This card concerns the real part while API_PLANE_TRAN concerns the imaginary component. This condition is used to set reflection/ transmission conditions for a surrounded material that is not being meshed. Definitions of the input parameters are as follows:

APR_PLANE_TRAN Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set.

<bc_id> The boundary flag identifier, or a side set number which is an integer that identifies the boundary location (side set in EXODUS II) in the problem domain.

<float1> R_2, the acoustic impedance (i.e. product of density and wave speed) in the surrounded material.

Examples

The following is a sample input card:

```
BC = APR_PLANE_TRAN SS 10 0.1
```

Technical Discussion

This condition should be used to account for transmission/reflection conditions for the external boundaries when the acoustic wave equation is used. It reflects characteristics for an acoustic wave encountering a planar interface between two materials;

$$n \bullet \nabla P = -i\frac{k_1 R_1}{R_2} P$$

where k is the acoustic wavenumber and R is the acoustic impedance. The subscript 1 refers to the material inside the external boundary and is the material which is meshed. Subscript 2 refers to the material outside of the external boundary. If R_2 is set equal to R_1, then this condition mimics an infinite boundary condition, i.e. no reflection at the external boundary.

API

```
BC = API NS <bc_id> <float1> [float2]
```

Description / Usage

(DC/ACOUS_PIMAG)

This Dirichlet boundary condition card is used to set constant amplitude of the imaginary part of the acoustic pressure. Definitions of the input parameters are as follows:

API Name of the boundary condition (<bc_name>).

NS Type of boundary condition (<bc_type>), where **NS** denotes node set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (node set in EXODUS II) in the problem domain.

<float1> Value of the imaginary part of the acoustic pressure amplitude.

[float2] An optional parameter (that serves as a flag to the code for a Dirichlet boundary condition). If a value is present, and is not -1.0, the condition is applied as a residual equation. Otherwise, it is a "hard set" condition and is eliminated from the matrix. *The residual method must be used when this Dirichlet boundary condition is used as a parameter in automatic continuation sequences.*

Examples

The following is a sample input card:

```
BC = API NS 100 1000.0
```

API_PLANE_TRAN

```
BC = API_PLANE_TRAN SS <bc_id> <float1>
```

Description / Usage

(WIC/SCALAR ACOUS_PIMAG)

This boundary condition card applies the plane wave transmission conditions to the acoustic wave equations. This card concerns the imaginary part while APR_PLANE_TRAN concerns the real component. This condition is used to set reflection/transmission conditions for a surrounded material that is not being meshed. Definitions of the input parameters are as follows:

API_PLANE_TRAN Name of the boundary condition (<bc_name>).

SS Type of boundary condition (<bc_type>), where **SS** denotes side set in the EXODUS II database.

<bc_id> The boundary flag identifier, an integer associated with <bc_type> that identifies the boundary location (side set in EXODUS II) in the problem domain.

<float1> R_2, the acoustic impedance (i.e. product of density and wave speed) in the surrounded material.

Examples

Following is a sample card:

```
BC = API_PLANE_TRAN SS 10 0.1
```

Technical Discussion

This condition should be used to account for transmission/reflection conditions for the external boundaries when the acoustic wave equation is used. It reflects characteristics for an acoustic wave encountering a planar interface between two materials;

$$n \bullet \nabla P = -i\frac{k_1 R_1}{R_2} P$$

where k is the acoustic wavenumber and R is the acoustic impedance. The subscript 1 refers to the material inside the external boundary and is the material which is meshed. Subscript 2 refers to the material outside of the external boundary. If R_2 is set equal to R_1, then this condition mimics an infinite boundary condition, i.e. no reflection at the external boundary.

END OF BC

```
END OF BC
```

Description / Usage

This card specifies the end of the list of boundary conditions (BCs), and is only used when automatic BC counting is used, as described in the *Number of BC* card. If the value of <integer> in that card is set to -1, all BC cards below the *END of BC* card are ignored, and *Goma* counts the number of BC cards between the *Number of BC* card and the *END of BC* card.

Examples

There are no input parameters for this card, which always appears as follows:

```
END OF BC
```

1.4.12 Rotation Specifications

This section descibes special input for controlling boundary condition implementation of vector equations in 3D problems. For 2D problems, the information in this section of the *Goma* input file is not read or used. It is also optional in 3D problems when none of the boundary conditions are rotated (see discussion below). However, these specifications are mandatory in all 3D problems which require equation rotation at the boundaries (e.g. *PLANE, KINEMATIC, VELO_NORMAL*), a condition especially prevalent in free-surface problems. The goal of this input section is to specify the exact implementation of the equations at any boundary with rotated conditions (called *rotated boundaries* throughout this discussion). But first, consider the necessary background.

Rotation of Vector Equations : The fluid momentum and psuedo-solid or Lagrangian solid momentum equations are vector equations (i.e., they have x, y, and z components). The boundary conditions applied to these equations can either be vector conditions (applying in the x, y, and z directions) or scalar conditions (a single function of the solution and x, y, and z). Scalar conditions applied to vector equations represent a special challenge, because it is often unclear which of the vector equations should be replaced by the scalar conditions. For many scalar conditions, e.g. Dirichlet conditions, the user specifies which component of the momentum equation gets replaced by the scalar condition; however, not replacing all the components of the vector equation at a boundary results in applying a shear-stress-free or normal-stress-free condition there (because the BOUNDARY term of the equations needs to be computed or else the normal traction is implicitly zero).

In some cases, a better way to apply a scalar condition is to use it to replace the normal or tangential contribution of the vector equations, while retaining the other portions of the equation (e.g., a no penetration condition could constrain the normal component of velocity but still allow the stress along the boundary to be shear-free). In *Goma*, this is done by rotating the vector equation into a normal-tangential form:

n and *t* are the unit normal and tangent vectors at the boundary (evaluated at the centroid along the boundary of an element) and R_i^f is the vector form of the weighted residual equation. This rotation is performed after all weak boundary conditions have been applied, but prior to application of strong boundary conditions. Thus any weak contributions to the vector equation are retained throughout the rotation. R_i^{fn} is the normal

$$R_i^{f,n} = \boldsymbol{n} \bullet \boldsymbol{R}_i^f$$

$$R_i^{f,t} = \boldsymbol{t} \bullet \boldsymbol{R}_i^f$$

component of the vector equation and R_i^{ft} is the tangent component of the vector equation. Note that the equations are rotated *after* they have been integrated rather than before; thus, the new residual equations are only strictly in normal-tangential form along straight boundaries (along curved boundaries there may be some error which becomes small as the element size decreases).

In *Goma*, rotated boundary conditions cause rotation of the vector equation on an element side if there are no Dirichlet conditions applied to that vector equation and if the total number of independent rotated conditions is less than the number of dimensions of the physical problem (i.e., in a 2D problem, the vector equation is rotated only when one independent rotated condition exists at that node).

Thus along any rotated boundary, the three vector equations (e.g. x, y, and z mesh equations) are replaced by three new equations as specified in this section. The user can decide to replace the component equations by rotated forms of the equations (or even unrotated forms of the equations), or to replace the component equations by boundary conditions. These specifications also dictate how to calculate the tangent vectors which are sometimes ill-defined in 3D. This section is designed to accommodate an arbitrary number of rotation specifications listed in the *Goma* input between *Rotation Specifications* = and *END OF ROT*.

All of this behavior is implemented through the overloaded *ROT* input card. There are three types of *ROT* cards depending on whether the condition applies on a surface, an edge or a vertex. *Goma* makes no assumptions about the topology of the mesh surfaces; all the topology is defined through the *ROT* card. In this implementation, a *surface* is defined as a side-set, an *edge* is defined as the intersection of two side-sets, and a *vertex* is defined as the intersection of three side-sets at a single node. Although all three types of input cards start with *ROT* =, we list them as three independent cards to make the discussion more straightforward. As nodes that are contained on edges must also be contained on the adjacent surfaces, these rotation specifications have a hierarchy – vertex, edge, surface – such that vertex conditions override edge conditions which override surface conditions.

Note: it is possible to solve a 3D problem with rotated boundaries by only creating rotation specifications for those boundaries, and letting *Goma* determine the behavior at the remaining boundaries. However, this is a dangerous practice; it is much better to explicitly tell *Goma* how to treat all boundaries so that the behavior is well defined. An important example is the intersection of a rotated boundary and an unrotated boundary, it is still a rotated boundary and requires an edge *ROT* specification.

Rotation Specifications

```
Rotation Specifications =
```

Description / Usage

This card denotes the start of the rotation specification cards. All rotation specification cards between this card and the END OF ROT card will read and processed. If this card is not present, no rotation cards will be read.

Examples

There are no input parameters for this card. It should appear on its on line exactly as follows:

```
Rotation Specifications =
```

Technical Discussion

No Discussion.

References

No References.

ROT SURFACE

```
    ROT = {MESH | MOM} SURFACE <bc_id> <string_x> <int_x> <string_y>
<int_y> <string_z> <int_z> {seed_method} <float1> <float2> <float3>
```

Description / Usage

This rotation specification card identifies a specific surface that requires rotation of equations for proper application of boundary conditions in three dimensional problems. It identifies the boundary conditions that are to be applied on that surface. It also identifies which equation components are to be replaced by boundary conditions, which are to be replaced by rotated equation components, and which are to be left alone. Equation components refer, currently, to only rotation of mesh and momentum equations. This card also identifies the manner in which two independent tangent vectors are to be determined on the surface.

Definitions of the first three input parameters are as follows:

{MESH | MOM}	Equation type (<eq_type>) to which this rotation condition applies: • **MESH** - Applies to mesh equations. • **MOM** - Applies to fluid momentum equations.
SURFACE	Type of rotation specification.
<bc_id>	An integer identifying the side set designation of the surface to which this rotation condition applies.

The next six parameters dictate how the x, y, and z components of the vector equation are replaced by boundary conditions or rotated equations using pairs of specifiers, e.g., <string_x> and <int_x> for the x-component of the equation.

<string_x>	A character string that specifies what will replace the x-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 1 (Valid Equation Rotation Strings).
<int_x>	This is an integer parameter specified as follows: • If <string_x> is a boundary condition name, then <int_x> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_x> is a rotation string from Table 1, <int_x> should be specified as 0.
<string_y>	A character string that specifies what will replace the y-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 1.
<int_y>	This is an integer parameter specified as follows: • If <string_y> is a boundary condition name, then <int_y> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_y> is a rotation string from Table 1, <int_y> should be specified as 0.
<string_z>	A character string that specifies what will replace the z-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 1.
<int_z>	This is an integer parameter specified as follows: • If <string_z> is a boundary condition name, then <int_z> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_z> is a rotation string from Table 1, <int_z> should be specified as 0.

Table 1. Valid Equation Rotation Strings

{string_x|y|z}	Description of Equation Rotation Selections
NONE, NA, or **NO**	No rotation is performed for this equation component.
N	This equation component is replaced by normal component of the residual tn • R
T	This equation component is replaced by the tangential component of the residual: t • R (EDGE and VERTEX only)
T1	This equation component is replaced by the first tangential component of the residual: T1 • R
T2	This equation component is replaced by the second tangential component of the residual: T2 • R
X	This equation is replaced by the x-component of the residual.
Y	This equation is replaced by the y-component of the residual.
Z	This equation is replaced by the z-component of the residual.
S	The equation component is replaced by the projection of the equations in the direction of the seed vector: S • R
B	The equation component is replaced by the projection of the equations in the direction of the binormal vector.

In most cases, only one of the three equations on a surface will be replaced by boundary conditions, and the remaining two equations will be rotated in the two tangent directions. Such a form constrains the normal motion of the solid or fluid while allowing tangential motions to occur stress-free.

The last four parameters in the card specify how to calculate the tangent vectors on the surface. In 3D, an infinite number of equally valid tangent pairs exist, so this card enables specifying how to choose those pairs. More specifically it identifies how to identify the first tangent vector (T1) since the second tangent vector is always be obtained via the cross product of the normal vector with the first tangent vector (T1).

{seed_method}	A character string that defines the method of tangent calculation. Valid options are listed in the Surface Tangent Calculation Method (Table 2).
<float1>	x-component of the seed vector, s. This parameter is only needed if {seed_method} is **SEED**.
<float2>	y-component of the seed vector, s. This parameter is only needed if {seed_method} is **SEED**.
<float3>	z-component of the seed vector, s. This parameter is only needed if {seed_method} is **SEED**.

Note that the seed vector specified does not have to be a unit vector.

Table 2. Surface Tangent Calculation Method

{seed_method}	Description of Tangent Calculation Methods
NONE	Tangent vectors should not be calculated. This is the usual choice for EDGE and VERTEX rotation types.
SEED	The first tangent vector (T1) is calculated from a surface projection of a seed vector, s: T1 = (I – nn) • s
BASIS	The first tangent is the direction of the first basis vector in the surface using a weighted average for adjacent elements.
BASIS_FIRST	The first tangent is the direction of the first finite element basis vector in the first element containing a given node.
BASIS_RESEED	The tangent resulting from BASIS_FIRST is used to reseed tangent calculation in the adjacent elements. (This method is the most reliable.)

Examples

The following are several examples of useful rotation specifications for surfaces:

```
ROT = MESH   SURFACE   99   KINEMATIC   99   T2   0   T1   0   BASIS_RESEED
```

```
ROT = MESH   SURFACE   16   T1   0   T2   0   PLANE   16   SEED   1.  0.  0.
```

```
ROT = MOM   SURFACE   5   VELO_NORMAL 5   T1  0  T2   0   BASIS
```

The first example applies to the mesh equations at side set 99, the second to mesh equations at side set 16, and the third to the fluid momentum equations at side set 5. As described previously, the <string_x>, <string_y> and <string_z> parameters can be any boundary condition name or rotation string. Thus for the first example above, the x-component of the mesh equation is replaced by a KINEMATIC boundary condition on side set 99, the y-component of the mesh equation is replaced by the second tangential component (T2) of the mesh equation, and the z-component of the mesh equation is replaced by the first tangential component (T1) of the mesh equation. Since the rotation selections in the first example (T2 and T1) are rotated components instead of boundary conditions, a value of zero for the <int_y> and <int_z> parameters is appropriate. Finally, for the first example, BASIS_RESEED was chosen as the {seed_method}, and thus no subsequent parameters were required. The second example, however, uses SEED as the {seed_method} and thus is followed by the x, y, and z components of the tangent vector, respectively, as <float1> of 1., <float2> of 0., and <float3> of 0.

Technical Discussion

The necessary background discussing the nature and need for rotation procedures and rotation specifications is supplied in several of the references listed below. Briefly, however, in order to apply certain boundary conditions accurately it is necessary that the vector components of the solid mesh or fluid momentum equations be replaced by components that are tangent and normal to the surface in question. This procedure is referred to in this context as "rotation of equations." It should be noted that explicitly specifying rotation conditions is really only necessary for three dimensional problems. Rotation also occurs in two-dimensional problems, but is sufficiently simpler that it can be automated and is therefore transparent to the user.

Not every boundary condition needs an accompanying rotation specification card and those that do are identified in the description of each boundary condition. Each rotated boundary condition will require at least one SURFACE rotation card be included for the boundary condition's side set. Failure to do so is an error. The boundary conditions most often encountered that will require rotation cards are the *VELO_NORMAL* card applied to the fluid momentum equations and the *KINEMATIC, PLANE*, and *SPLINE* cards applied to the solid mesh equations.

In almost every case the boundary condition constraint will replace the normal rotated component so only the two tangential components of the rotated equation remain. All three examples shown above are just this situation. This has the effect of constraining the normal motion of the solid or fluid and imposing zero tangential forces due to the natural boundary conditions present in both fluid and solid momentum equations.

Specification of a seed vector method is needed so that a unique pair of tangent vectors may be determined at each point on the surface. The BASIS, BASIS_FIRST and BASIS_RESEED use the finite element grid in the surface as a means of defining the first tangent vector. They can employ averaging over elements that share a node. They should be employed on surfaces for which it is difficult to find a single consistent seed vector for every point on the surface. The SEED method finds the projection of the vector supplied in the surface at the point of interest. This projection vector is normalized to obtain the first tangent vector. It should be clear that only vectors that are never normal to any point on the surface will be suitable. In practice, this condition can sometimes be hard to meet for some surfaces. In these cases, the other seeding methods should be used.

References

GT-007.2: Tutorial on droplet on incline problem, July 30, 1999, T. A. Baer

GT-012.0: 3D Roll coating template and tutorial for GOMA, February 21, 2000, P.R. Schunk

GT-018.1: ROT card tutorial, January 22, 2001, T. A. Baer

ROT EDGE

```
    ROT = {MESH | MOM} EDGE <bc_id1> <bc_id2> <string_x> <int_x> <string_y>
<int_y> <string_z> <int_z> {seed_method} <float1> <float2> <float3>
```

Description / Usage

This rotation specification card deals with rotation specification along edges. In this context, an edge is the intersection of two side sets. It identifies the boundary conditions that will be applied at nodes on that edge and which equation components are to be associated with them. It also identifies which components of the rotated equations will be used. Currently, only rotation of mesh and momentum equations is allowed on edges. This card can also be used for specifying a seed vector if needed.

Definitions of the input parameters are as follows:

{MESH | MOM}	Equation type to which this rotation condition applies: • **MESH** - Applies to mesh equations. • **MOM** - Applies to fluid momentum equations.
EDGE	Type of rotation specification.
<bc_id1>	Side set ID number of the primary side set.
<bc_id2>	Side set ID number of the secondary side set.

The edge is defined as the intersection of the primary and secondary side sets.

The next six parameters dictate how the x, y, and z components of the vector equation are replaced by boundary conditions or rotated components using pairs of specifiers, e.g., <string_x> and <int_x> for the x-component of the equation.

<string_x>	A character string that specifies what will replace the x-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 3 (Valid EDGE Tangent Equation Rotation Strings).
<int_x>	This is an integer parameter specified as follows: • If <string_x> is a boundary condition name, then <int_x> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_x> is a rotation string from Table 3, <int_x> should be specified as 0.
<string_y>	A character string that specifies what will replace the y-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 3.
<int_y>	This is an integer parameter specified as follows: • If <string_y> is a boundary condition name, then <int_y> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_y> is a rotation string from Table 3, <int_y> should be specified as 0.
<string_z>	A character string that specifies what will replace the z-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 3.
<int_z>	This is an integer parameter specified as follows: • If <string_z> is a boundary condition name, then <int_z> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_z> is a rotation string from Table 3, <int_z> should be specified as 0.

Table 3. Valid EDGE Tangent Equation Rotation Strings

{string_x|y|z}	Description of Equation Rotation Selections
NONE, NA, or NO	No rotation is performed for this equation component.
N	This equation component is replaced by the normal component of the residual: n • R where n is the outwardpointing normal to <bc_id1>
T	This equation component is replaced by the tangential component of the residual: T • R where the tangent is a line tangent along the edge defined by <bc_id1> and <bc_id2>.
B	This equation component is replaced by the outwardpointing binormal component of the residual: B • R where the binormal is perpendicular to both the line tangent T and the outward-pointing normal to <bc_id1>.
S	The equation component is replaced by the projection of the equations in the direction of the seed vector: S • R
X	This equation is replaced by the x-component of the residual.
Y	This equation is replaced by the y-component of the residual.
Z	This equation is replaced by the z-component of the residual.

In most cases, seeding of the tangent vectors is not needed along edges, although it is possible to specify a seed method as defined in the *ROT SURFACE* card via the parameters {seed_method}, <float1>, <float2>, and <float3>. Note also that a seed vector must be specified to use the S rotation option.

Examples

The following is an example of an edge rotation specification:

```
ROT = MESH   EDGE 4 5   PLANE   4   PLANE 5   T   0   NONE
```

This card specifies rotation of the mesh equations along the edge of intersection of side sets 4 and 5. The x and y mesh equations are replaced by PLANE conditions on side sets 4 and 5, respectively. The z mesh equation is replaced by the mesh residuals rotated into the direction of the line tangent along the edge. This enables the mesh to slide freely (i.e., stress-free) along the edge.

Technical Discussion

- The direction of the line tangent is chosen such that the binormal (b = n × t) with *n*, the outward-pointing normal to the primary surface <bc_id1>, is outwardpointing from the edge.
- Along edges, two of the equations are normally replaced by boundary conditions and one equation is replaced by this tangential component. However several options are available for replacing the mesh equations by other forms of the rotated equations as listed in Table 3. (Valid EDGE Tangent Equation Rotation Strings) above.
- It is very rare to require a seed vector be specified on an edge. The SEED vector choice is almost always NONE.
- A precedence rule has been established for the case when more than one *Rotation Specification* could be applied at a point. The rule is as follows:

  The Rotation condition that will be applied is:

  A>The first VERTEX condition in the input deck that could apply. If there is no contravening VERTEX condition then,

  B>The first EDGE condition in the input deck that could apply. If there is no contravening EDGE condition then,

  C>The first SURFACE condition that could apply.

- A very important restriction exists for EDGE and VERTEX rotation conditions. It is a necessary requirement that all elements that are present on an edge have only a single segment present on the edge curve. An element may therefore never contribute more than two corner vertex nodes to the set of nodes on an edge curve. If there are more than two such nodes for a given element, *Goma* will terminate with a *"Side not connected to edge"* error. If such a situation exists, the only solution is to remesh the geometry to eliminate such elements.

References

GT-007.2: Tutorial on droplet on incline problem, July 30, 1999, T. A. Baer

GT-012.0: 3D Roll coating template and tutorial for GOMA, February 21, 2000, P.R. Schunk

GT-018.1: ROT card tutorial, January 22, 2001, T. A. Baer

ROT VERTEX

```
    ROT = {MESH | MOM} VERTEX <bc_id1> <bc_id2> <bc_id3> <string_x>
<int_x> <string_y> <int_y> <string_z> <int_z> {seed_method} <float1> <float2> <float3>
```

Description / Usage

This rotation specification card deals with rotation specification at vertices. In this context, a vertex is the intersection point of three side sets. It identifies the boundary conditions that will be applied at the vertex node and which equation components are to be associated with them. It also identifies which components of the rotated equations will be used. Currently, only rotation of mesh and momentum equations is allowed at a vertex. This card can also be used for specifying a seed vector if needed.

Definitions of the input parameters are as follows:

{MESH | MOM}	Type of equation to which this specification applies,where • **MESH** - Applies to mesh displacement equations. • **MOM** - Applies to fluid momentum equations.
VERTEX	Type of rotation specification.
<bc_id1>	Side set ID number of the primary side set.
<bc_id2>	Side set ID number of the secondary side set.
<bc_id3>	Side set ID number of the tertiary side set.

The vertex is defined as the point at the intersection of the primary, secondary, tertiary side set. Note that it is possible for these three side sets to intersect at more than one discrete point. The VERTEX condition is applied to all such points.

The next six parameters dictate how the x, y, and z components of the vector equation are replaced by boundary conditions or rotated components using pairs of specifiers, e.g., <string_x> and <int_x> for the x-component of the equation.

<string_x>	A character string that specifies what will replace the x-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 4 (Valid VERTEX Tangent Equation Rotation Strings).
<int_x>	This is an integer parameter specified as follows: • If <string_x> is a boundary condition name, then <int_x> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_x> is a rotation string from Table 4, <int_x> should be specified as 0.
<string_y>	A character string that specifies what will replace the y-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 4.
<int_y>	This is an integer parameter specified as follows: • If <string_y> is a boundary condition name, then <int_y> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_y> is a rotation string from Table 4, <int_y> should be specified as 0.
<string_z>	A character string that specifies what will replace the z-component of the vector equation (MESH or MOMENTUM). This string may be the name of a boundary condition already specified in the boundary condition specification section or one of the rotation strings listed in Table 4.
<int_z>	This is an integer parameter specified as follows: • If <string_z> is a boundary condition name, then <int_z> is the side set or node set designation to which the appropriate boundary condition applies. This provides a means of distinguishing between boundary conditions possessing the same string name but applied to different side sets or node sets. • If <string_z> is a rotation string from Table 4, <int_z> should be specified as 0.

Table 4. Valid VERTEX Tangent Equation Rotation Strings

{string_x|y|z}	Description of Equation Rotation Selections
NONE, NA, or NO	No rotation is performed for this equation component.
N	This equation component is replaced by the normal component of the residual: n • R where n is the outwardpointing normal to <bc_id1>
T	This equation component is replaced by the tangential component of the residual: T • R where the tangent is a line tangent along the edge defined by <bc_id1> and <bc_id2>.
B	This equation component is replaced by the outwardpointing binormal component of the residual: B • R where the binormal is perpendicular to both the line tangent T and the outward-pointing normal to <bc_id1>.
S	The equation component is replaced by the projection of the equations in the direction of the seed vector: S • R
X	This equation is replaced by the x-component of the residual.
Y	This equation is replaced by the y-component of the residual.
Z	This equation is replaced by the z-component of the residual.

In most cases, seeding of the tangent vectors is not needed along vertices, although it is possible to specify a seed method as defined in the *ROT SURFACE* card via the parameters {seed_method}, <float1>, <float2>, and <float3>. Note also that a seed vector must be specified to use the S rotation option.

Examples

The following is an example of a vertex rotation specification:

```
ROT = MESH   VERTEX   3 4 6   PLANE   4 PLANE 3   PLANE   6   NONE
```

```
ROT = MESH   VERTEX   5 4 6   PLANE   4 KINEMATIC 5   PLANE   6   NONE
```

In the first example, the vertex is at the intersection of side sets 3, 4 and 6, and the three mesh equations at this vertex are replaced by *PLANE* conditions from side sets 4, 3, and 6, respectively. In the second example, the vertex is at the intersection of side sets 4, 5, and 6, respectively. Since it is conceivable that side set 5 might represent a free surface that curves in three dimensions, the last *VERTEX* card might apply to more than one point.

Technical Discussion

- Despite the fact that *VERTEX* cards apply only at single points, definitions of the normal, tangent and binormal vectors are still operative. The normal vector, **N**, is the outward-pointing normal to the primary side set, the tangent vector, **T**, is defined to lie along the curve defined by the intersection of the primary and secondary side set, and the binormal vector, **B**, is defined simply as the cross product of the normal vector with the tangent vector. Note that the sense of the tangent vector is chosen so that the binormal vector will always point outwards from the domain.
- At a vertex, it is normally the case that all three rotated components will be replaced by boundary conditions as suggested by the examples. However, it is not a rarity that a rotated component, usually **N** or **T**, will also appear.
- The same hierarchy of precedence is used to determine which rotation specification will be applied when more than one could apply to a node. The rule is as follows:

  The Rotation condition that will be applied is:

  > *A>The first VERTEX condition in the input deck that could apply. If there is no contravening VERTEX condition then,*

B>The first EDGE condition in the input deck that could apply. If there is no contravening EDGE condition then,

C>The first SURFACE condition that could apply.

- Very often *VERTEX* cards are used to resolve ambiguities that arise at points where multiple *SURFACE* or *EDGE* cards could apply.

References

GT-007.2: Tutorial on droplet on incline problem, July 30, 1999, T. A. Baer

GT-012.0: 3D Roll coating template and tutorial for GOMA, February 21, 2000, P.R. Schunk

GT-018.1: ROT card tutorial, January 22, 2001, T. A. Baer

END OF ROT

```
END OF ROT
```

Description / Usage

This card is used to end the section of rotation specifications in the input deck. Any ROT conditions listed after this card are ignored. It should always accompany the *"Rotation Specifications* =" card.

Examples

There are no input parameters for this card, which always appears as follows:

```
END OF ROT
```

Technical Discussion

No Discussion.

1.4.13 Problem Description

This section directs all input specifications required for differential equations, material type, mesh motion type, coordinate system, finite element basis function type, and several other input tasks. This section of input records, with the exception of the *Number of Materials* card (the first one below), must be repeated for each material region in the problem. Within that region of the problem domain (and the corresponding section of the input file) there are no restrictions as to which differential or constraint equations can be specified, which is a unique capability of *Goma*.

However, some combinations or specifications do not make much sense, e.g., a cylindrical coordinate region combined with a cartesian one. It is recommended that the user consult the usage tutorials and example problems to get a feel for how this section is constructed.

Number of Materials

```
Number of Materials = <integer>
```

Description / Usage

This required card denotes how many material sections are contained in the *Problem Description File*. Each material section will have its own problem description, consisting of the following: *MAT* card, *Coordinate System* card, *Mesh Motion* card, *Number of bulk species* card, *Number of EQ* card, and zero or more equation cards. The input parameter is defined as

<integer>	The number of MAT cards (i.e., material sections) that follow; this number must be greater than zero.

If there are more MAT cards than specified by <integer>, *Goma* ignores all extras (i.e., the first *Number of Materials* material sections are read). If <integer> is set to -1, *Goma* automatically counts the MAT cards between the *Number of Materials* card and the *END OF MAT* card.

Examples

Following is a sample card, indicating that there are two materials:

```
Number of Materials = 2
```

Technical Discussion

No Discussion.

References

No References.

MAT

```
MAT = <char_string> <integer_list>
```

Description / Usage

This card represents the start of each material section in the *Problem Description File*. Thus, one MAT card is required for each material section. Definition of the input parameters are as follows:

<char_string>	Filename of the material file from which all material properties for the current material will be read. The material file's name plus extension is char_string.mat, and if the file is not present in the current working directory, the code will exit with the error message "Not all Material Files found in current directory."
<integer_list>	This is a list of space delimited integers that define the set of element blocks for which this material file is applicable; the integers are the element block ids defined when the domain was meshed.

Examples

The following specifies material file "sample.mat" applies to element blocks 1, 2, 3, 7, and 9:

```
MAT = sample 1 2 3 7 9
```

Note, the ".mat" extension is not specified explicitly, but appended to the character string by the code.

Technical Discussion

No Discussion.

References

No References.

Coordinate System

```
Coordinate System = {char_string}
```

Description / Usage

This card is required for each material section in the *Problem Description File*. It is used to specify formulation of the equations to be solved. Valid options for {char_string} are as follows:

CARTESIAN	For a two (x-y) or three (x-y-z) dimensional Cartesian formulation.
CYLINDRICAL	For an axisymmetric (z-r) or three-dimensional cylindrical (z-r-θ) formulation; the three-dimensional option has not been tested.
SPHERICAL	For a spherical (r-θ-ϕ) formulation.
SWIRLING	For a two-dimensional formulation (z-r-θ) with a swirling velocity component that is independent of azimuthal coordinate.
PROJECTED_CARTESIAN	For use in the analysis of the three-dimensional stability of a two-dimensional flow field. The formulation (x-yz) has a z-velocity component that is independent of the z-direction.

Examples

The following is a sample card that sets the coordinate system to Cartesian:

```
Coordinate System = CARTESIAN
```

Technical Discussion

Note the coordinate ordering for the **CYLINDRICAL** and **SWIRLING** options where the z-direction is first followed by the r-component (which in lay terms means the modeled region/part will appear to be "lying down.") If the **SWIRLING** option is activated, *Goma* expects a third momentum equation for the θ-direction, i.e. *EQ* = *momentum3*, as explained in the equation section. The third component is basically the azimuthal θ-velocity component, and the appropriate boundary conditions must be applied, e.g., on the w-component as described in the *Category 4* boundary conditions for *Fluid Momentum Equations*.

References

No References.

Element Mapping

```
Element Mapping = <char_string>
```

Description / Usage

This card allows the user to set the order of the finite element shape mapping between the canonical element and each physical element. Valid options for {char_string} are:

isoparametric	This choice sets the element order mapping to the highest order present in the problem. *However, if a mesh displacement field is present, the element mapping order is the interpolation order of the mesh displacement field.*
Q1	This choice sets the element mapping order to bilinear.
Q2	This choice sets the element mapping order to biquadratic.
SP	This choice sets the element mapping to order to subparametric.

Examples

Some text like this:

```
Element Mapping = isoparametric
```

Technical Discussion

No Discussion.

References

No References.

Mesh Motion

```
Mesh Motion = {char_string}
```

Description / Usage

This card is required for each material section in the *Problem Description File* even if a moving mesh problem is not being solved. It is used to specify the method which prescribes the movement of nodes within the mesh. Valid options for {char_string} are:

ARBITRARY	This option triggers the implicit pseudo-solid domainmapping technique using the constitutive equation designated in the corresponding file.mat (see *Material File* description); with this technique, the boundaries of the domain are controlled by distinguishing conditions coupled with the problem physics, and the interior nodes move independently of the problem physics.
LAGRANGIAN	This option triggers coupling the motion of nodes on the interior of the domain to the deformation of an elastic solid, *neglecting inertia*. If the solid is incompressible, this technique uses a pressure (Lagrange multiplier) to couple the solid deformation and the local solvent concentration.
DYNAMIC_LAGRANGIAN	This option triggers coupling the motion of nodes on the interior of the domain to the deformation of an elastic solid, *including inertia*. If the solid is incompressible, this technique uses a pressure (Lagrange multiplier) to couple the solid deformation and the local solvent concentration. Together with the equation term multiplier on the mass matrix (see *EQ* card) and a "transient" specification on the *Time Integration* Card, this option will invoke a Newmark- Beta time integration scheme for the inertia term in the *R_MESH* equations.
TOTAL_ALE	This option allows motion of nodes on the interior of the domain of a solid region to be independent of the material motion. TALE is an acronym for "Total Arbitrary Lagrangian Eulerian" mesh motion. This is typically used in elastic solids in which large scale deformation makes motions under the LAGRANGIAN option unmanageable. If the solid is incompressible, this technique uses a pressure (Lagrange multiplier) to couple the solid deformation and the local solvent concentration. Invoking this option requires mesh equations and real solid equations, as described on the EQ card. Other relevant cards that are often used with this option are *KINEMATIC_DISPLACEMENT* boundary condition, *DX_RS, DY_RS, DZ_RS* boundary conditions, *FORCE_RS, FLUID_SOLID_RS*, and others. See references for more detailed usage procedures.

Examples

The following is a sample card that sets the mesh motion scheme to be arbitrary:

```
Mesh Motion = ARBITRARY
```

Technical Discussion

For the *TOTAL_ALE* mesh motion option we must supply elastic properties and solid constitutive equations for both the mesh and the real solid. It is best to consult the example tutorials cited below for details.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

GT-006.3: Slot and Roll coating with remeshing templates and tutorial for GOMA and CUBIT/MAPVAR, August 3, 1999, R. R. Lober and P. R. Schunk

Number of Bulk Species

```
Number of bulk species = <integer>
```

Description / Usage

This card is required for each material section in the *Problem Description File*. It is used to specify the number of species in a phase. The word, bulk, here, refers to its being distributed throughout the domain, not just at a surface. All loops over property evaluations use this value to specify the length of the loop. The single input parameter is defined as:

<integer>	The number of species. If the value of <integer> is 0, then no species equations are solved for.

In the absence of any further cards specifying the number of species equations, the number of species equations is set equal to the integer value supplied by this card, and there is an implied additional species, i.e., the solute, which is not part of species loops, but which fills out the specification of the phase.

Examples

Following is a sample card:

```
Number of bulk species = 1
```

Technical Discussion

Unfortunately, in the past, this card has specified the number of species equations instead of the number of species, as its name would imply! Now, the preferred treatment is to specify unequivocally both the number of bulk species and the number of bulk species equations using two separate input cards. If the two values are the same, then the system is semantically referred to as being “dilute” (even though it might not be!), and there is an inferred solute which is not part of the loop over species unknowns in property evaluations or even in the specification of properties in the .mat file. If the number of species is one greater than the number of species equations, then the system is deemed “nondilute” and the length of loops over property evaluations is one greater than the number of species equations. For nondilute systems, an equation of state must be implicitly used within *Goma* to solve for the value of the species unknown variable for the last species.

References

No References.

Material is Nondilute

```
Material is nondilute = {yes | no}
```

Description / Usage

This card is a optional for each material section in the *Problem Description File*. It is used to specify the number of species equations in a phase. The single string parameter is a boolean, yes or no.

yes	the number of species equations is set equal to one less than the number of species.
no	the number of species equations is set equal to the number of species.

When the number of species is equal to the number of species equations, there is an implied additional species, i.e., the solute, which is not part of species loops, which fills out the specification of the phase.

Examples

Following is a sample card:

```
Material is nondilute = yes
```

Technical Discussion

See the discussion for the "*Number of bulk species*" card.

References

No References.

Number of Bulk Species Equations

```
Number of bulk species equations = <integer>
```

Description / Usage

This card is optional but strongly recommended for each material section in the *Problem Description File*. It is used to specify the number of species equations in a phase. The word, bulk, here, refers to its being distributed throughout the domain, not just at a surface. The single input parameter is defined as:

<integer>	The number of species equations; if the value of <integer> is 0, then no conservation equations for species are solved for.

When the number of species is equal to the number of species equations, there is an implied additional species, i.e., the solute, which is not part of species loops, which fills out the specification of the phase.

Examples

Following is a sample card:

```
Number of bulk species equations = 1
```

Technical Discussion

See the discussion for the "*Number of bulk species*" card.

References

No References.

Default Material Species Type

```
Default Material Species Type = {species_type_string}
```

Description / Usage

This optional parameter sets the form of the species variable type within *Goma*. Valid options for {species_type_string} are given below by the *SPECIES_** names (along with a description and variable (prefix) name:

SPECIES_MASS_FRACTION	Mass Fractions	*Yk_*
SPECIES_MOLE_FRACTION	Mole Fractions	*Xk_*
SPECIES_VOL_FRACTION	Volume Fractions	*Vk_*
SPECIES_DENSITY	Species Densities	*Dk_*
SPECIES_CONCENTRATION	Species Concentration	*Ck_*
SPECIES_UNDEFINED_FORM	Undefined form	*Y*

The default is to assume **SPECIES_UNDEFINED_FORM**. Please refer to the Technical Discussion for important details.

Examples

Following is a sample card:

```
Default Material Species type = SPECIES_MASS_FRACTION
```

Technical Discussion

For nondilute systems the *SPECIES_* quantities above are not just simply interchangeable via a multiplicative constant. Their values are distinct, and their interrelationship evaluated via a potentially nontrivial equation of state. Prior to the implementation of this card/capability, *Goma* hadn't handled many nondilute cases, and where it had, this issue was finessed by special casing property evaluations.

This card both sets the type of the species variables and establishes a convention for the units of equations within *Goma*. For settings of **SPECIES_MASS_FRACTION** and **SPECIES_DENSITY_FRACTION**, equations generally have a mass unit attached to them. Equations have concentration units attached to them for settings of **SPECIES_MOLE_FRACTION**, **SPECIES_VOL_FRACTION**, and **SPECIES_CONCENTRATION**. For example, given a setting of **SPECIES_MASS_FRACTION**, each volumetric term in the species conservation equation has units of mass per time, i.e., the time derivative term is written as

$$\int \frac{\mathrm{d}}{\mathrm{d}t}(\rho Y_k)\phi_i d\Omega \quad .$$

For a setting of **SPECIES_MOLE_FRACTION**, each volumetric term would have units of moles per time, i.e., the time derivative term is written out as

$$\int \frac{\mathrm{d}}{\mathrm{d}t}(cX_k)\phi_i d\Omega \quad .$$

All this is necessary in order to handle cases where the total density or total concentration of a phase is spatially variable. In that case, it can't just be divided out as in earlier versions of *Goma* but must be included in the conservation equations, and therefore the units of the conservation equation must reflect this.

The species variable type affects the units and thus values of quantities returned from certain boundary conditions. For example, the **IS_EQUIL_PSEUDORXN** boundary condition returns units of moles per time per $length^2$ if the species variable type is defined to be **SPECIES_CONCENTRATION**, but will multiply by molecular weights and thus return units of mass per time per $length^2$ if the species variable type is defined to be **SPECIES_MASS_FRACTION**. This change conforms to the expected units of the overall species conservation equation for the two values of the species variable type variable used as examples above.

The last column in the table above contains a three letter string. This string is used as a prefix for the name of the species variable in the EXODUS output file. If no names are specified in the material file and Chemkin is not used (which provides names for the species variables itself), then integers are used for names. For example, the first species unknown in *Goma* problem employing Mass Fractions as the independent species variables will be called **Yk_1**. If Chemkin is used in the same problem and the first chemkin species is named **H2O**, then the name in the EXODUS output file will be **Yk_H2O**. If a *Goma* problem is solved with no specification of the type of the species variable, then the first unknown in the EXODUS file will be named **Y1**.

Additionally, some boundary conditions and inputs from the material file section will set the species variable type on their own without the benefit of this card, if the species variable type is the default undefined form. Some internal checks are done; if an inconsistency is caught, *Goma* will abort with an informative error message.

Number of Viscoelastic Modes

```
Number of viscoelastic modes = <integer>
```

Description / Usage

This card is required only if you are performing a viscoelastic simulation and have included stress equations in the equation section and chosen a viscoelastic constitutive equation in the material file. The integer value denotes how many viscoelastic tensor stress equations are to be used. The number of modes can vary from a minimum of 1 to a maximum of 8. The input parameter is defined as

<integer>	The number of viscoelastic modes, which must be greater than zero, but less than nine.

Examples

The following is a sample card, indicating that a calculation with two viscoelastic stress modes is being undertaken:

```
Number of viscoelastic modes = 2
```

Technical Discussion

No Discussion.

References

Please see the viscoelastic tutorial memo for a discussion of multimode viscoelastic equations:

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

Number of Matrices

```
Number of Matrices = <integer>
```

Description / Usage

This card specifies the Number of Matrices for the current material. Note when running in parallel all matrices should be on all materials

<integer> The number of matrices on this material

Examples

```
...
  Number of bulk species = 0

Number of Matrices = 2

MATRIX = 1
  Number of EQ   = 3
  ...

MATRIX = 2
  Number of EQ   = 2
  ...
```

Technical Discussion

No Discussion.

MATRIX

```
MATRIX = <integer>
```

Description / Usage

This card indicates which matrix the current equation set corresponds to

<integer> The matrix for the set of equations that follows

Examples

```
...
  Number of bulk species = 0

Number of Matrices = 2

MATRIX = 1
  Number of EQ   = 3
  ...

MATRIX = 2
  Number of EQ   = 2
  ...
```

Technical Discussion

No Discussion.

Disable time step control

```
Disable time step control = [yes|no]
```

Description / Usage

This card specifies whether goma's time step control should be used on this matrix this is useful for matrices that don't have time step control indicators like velocity gradients

<bool> Yes or No to disable time step control for this matrix

Examples

```
MATRIX = 1
  Disable time step control = yes
  Number of EQ   = 3
```

Technical Discussion

No Discussion.

Normalized Residual Tolerance

```
Normalized Residual Tolerance = <float>
```

Description / Usage

This is the per matrix version of this card. This overrides Normalized Residual Tolerance for each matrix

This required card indicates the value of the L_2 norm of the global nonlinear residual vector that indicates termination of Newton's method (i.e., convergence). The input parameter is defined as

<float> tol, a non-negative floating point number (tol $\geq$ 0.0) specifying the L_2 convergence tolerance for the global nonlinear residual vector.

The *Normalized Residual Tolerance* card is required; there is no default.

Examples

Following is a sample card:

```
MATRIX = 1
    Normalized Residual Tolerance = 1.0e-8
    Number of EQ = 3
```

Technical Discussion

Newton's method is terminated when the global nonlinear residual falls below tol, or the maximum number of iterations specified in the *Number of Newton Iterations* is reached.

Number of EQ

```
Number of EQ = <integer>
```

Description / Usage

This card is required for each material section in the *Problem Description File*. It specifies how many equations (i.e., equation cards, *[EQ =]*) follow for this material section, including the mesh motion equations if appropriate. This number of equations is only for the current material, since each material has its own equation section.

The single input parameter is defined as

<integer>	The number of EQ cards following this card. Only the first Number of EQ equations are read; if there are more EQ cards than specified by <integer>, *Goma* ignores the extras. If <integer> is set to -1, *Goma* will automatically count the number of EQ cards between the *Number of EQ* card and the *END OF EQ* card.

Examples

The following is a sample card that sets the number of equations to 5:

```
Number of EQ = 5
```

Technical Discussion

For equation specification in *Goma*, it is important to remember that a scalar equation has a single equation entry (e.g. fill, species, voltage, shear rate, etc.), while a vector equation (e.g. momentum, mesh, mom_solid, etc.) has an entry for each component of the vector. Thus, if you were solving a two-dimension flow problem, you would need to specify both U1 and U2 components of the momentum equation explicitly. The same holds true for tensor equations (e.g. stress and velocity gradient); each term of the tensor is specified explicitly. The one exception to this rule is for multimode viscoelasticity where the first mode equations are specified through the equation card and then the auxiliary modes are set by the *Number of viscoelastic modes* card. Please see the viscoelastic tutorial memo (Rao, 2000) for a detailed discussion of multimode viscoelasticity.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

---

Equation Cards

Following the *Number of EQ card, the equation cards, or records, are racked as intended up to the *END OF EQ* card or to the number specified, with one equation record per line. Each card begins with the "*EQ* =" string, followed by the equation name, e.g., *energy*, some basis function and trial function information, and finally a series of term multipliers. These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. However, one can use these multipliers as a way of adjusting the scaling of individual terms. Exercise caution in using these factors as expedients for transport coefficients; for instance the equation term multiplier for the momentum diffusion term affects both the isotropic stress term (pressure) and the deviatoric stress. It is recommended that you consult the example tutorial menus and problems to get a feel for the structure of this section. A sample input file structure including the EQ section is shown in the figure at the beginning of this chapter.

energy

```
EQ = energy {Galerkin_wt} T {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a conservation of energy differential equation. Definitions of the input parameters are defined below. Note that <floatlist> contains five constants for the Energy equation defining the constant multipliers for each term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

energy	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interface. • **Q1_XV**-Linear interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces.Only used for level-set problems. • **Q2_XV**-Quadratic interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. Only used for level-set problems. • **Q1_GN**-Linear interpolation for capturing variables defined on the negative side of the level-set interface. Similar to Q1_XV. • **Q2_GN**-Quadratic interpolation for capturing variables defined on the negative side of the level-set interface. Similar to Q1_XV
T	Name of the variable associated with this equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable T, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q1_XV**-Linear interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. • **Q2_XV**-Quadratic interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. • **Q1_GN**-Linear interpolation for capturing variables defined on the negative side of the level-set interface. Similar to Q1_XV. • **Q2_GN**-Quadratic interpolation for capturing variables defined on the negative side of the level-set interface. Similar to Q1_XV.
1.4. Problem Description (Input File)	• **Q1_GP**-Linear interpolation for capturing variables defined on the positive side of the level-set interface. Similar to Q1_XV. • **Q2_GNP**-Quadratic interpolation for capturing

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function and has all the term multipliers on except the mass matrix term for time derivatives:

```
EQ = energy   Q1 T Q1   0. 1. 1. 1. 1.
```

Technical Discussion

Some discussion on the XFEM-type enriched basis functions Q1_XV, Q1_GN, Q1_GP, Q2_GN, Q2_GP and Q2_XV is in order. *First of all, these basis functions are to be use with the level-set front tracking capability only*. First of all, these basis functions are typically only used for the continuity equation to capture pressure jumps due to surface tension. However, for phase change problems some experimentation has been pursued with the energy equation.

XFEM Value Enrichment

Enrichment:

$$T(x)=\sum_i N_i(x)T_i+\sum_i N_i(x)g_i(x)a_i \quad , \quad g_i(x)=H\left[\phi(x)\right]-H\left[\phi_i(x)\right]$$

Related "Ghost" Enrichment:

$$T(x)=\sum_i N_i(x)T_i+\sum_i N_i(x)g_i(x)a_i \quad , \quad g_i(x)=H\left[-\phi(x)\phi(x_i)\right]$$

,

$$T(x)=\sum_i N_i(x)\left(1-g_i(x)\right)T_i+\sum_i N_i(x)g_i(x)\hat{T}_i \quad , \quad g_i(x)=H\left[-\phi(x)\phi(x_i)\right]$$

Advantages:

This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. Appears to be method of choice for Pressure discontinuity. Produces interface integral for terms integrated by parts that allows for specifying a weak integrated conditions. This is needed in the laser welding heat transfer problem.

momentum

```
EQ = momentum{1|2|3} {Galerkin_wt} {U1|U2|U3} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for one component of a vector momentum equation. Definitions of the input parameters are defined below. Note that <floatlist> contains six constants for the Momentum equation defining the constant multipliers for each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

momentum1 | momentum2 | momentum3	Name of the equation to be solved, where the 1,2 and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces.
U1 | U2 | U3	Name of the variable associated with the 1, 2 or 3 principal coordinate direction for this component equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **U1**, **U2** or **U3** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces.
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.
<float6>	Multiplier on porous term (linear source).

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous velocity interpolation and weight function and turns on all equation term multipliers except for the mass matrix and the porous term:

```
EQ = momentum1 Q2 U1 Q2 0. 1. 1. 1. 1. 0.
```

Technical Discussion

No Discussion.

References

No References.

pmomentum

```
EQ = pmomentum{1|2|3} {Galerkin_wt} {PU1|PU2|PU3} {Interpol_fnc}<floatlist>
```

Description / Usage

This card provides information for solving a differential equation for one component of a vector particle momentum equation. Definitions of the input parameters are defined below. Note that <floatlist> contains six constants for the Pmomentum equation defining the constant multipliers for each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

pmomentum1 | pmomentum2 | pmomentum3	Name of the equation to be solved, where the 1, 2 and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces.
PU1 | PU2 | PU3	Name of the variable associated with the 1, 2 or 3 principal coordinate direction for this component equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **PU1**, **PU2** or **PU3** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces.
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.
<float6>	Multiplier on porous term (linear source).

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous velocity interpolation and weight function and turns on all equation term multipliers except for the mass matrix and the porous term:

```
EQ = momentum1 Q2 PU1 Q2 0. 1. 1. 1. 1. 0.
```

Technical Discussion

The particle momentum equations have been added to *Goma* as part of a research project and are not currently in use for production computing.

References

No References.

stress

```
EQ = {eqname} {Galerkin_wt} {varname} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation. Definitions of the input parameters are defined below. Note that <floatlist> contains five constants for the Stress equation defining the constant multipliers for each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

{eqname}	The name of the component of the stress equation to be solved, one of the following: **stress11, stress12, stress13, stress22, stress23, stress33**.
{Galerkin_wt}	Two-character or three-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **PQ**-Q1 Discontinuous • **PQ**-Q2 Discontinuous
{varname}	The name of the variable associated with the respective components (11, 12, 13, 22, 23, and 33) of the symmetric Stress tensor, which are **S11, S12, S13, S22, S23, S33**.
{Interpol_fnc}	Two-character or three-character value that defines the interpolation function used to represent the variable **S11, S12, S13, S22, S23** or **S33**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **PQ**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation for stress and turns on all the term multipliers:

```
EQ = stress11 Q1 S11 Q1 1. 1. 1. 1. 1.
```

Technical Discussion

The interpolation/weight functions that are discontinuous, e.g. have the prefix "P", invoke the discontinuous Galerkin method for solving the stress equations where the interpolation is discontinuous and flux continuity is maintained by performing surface integrals. For details of the implementation of the discontinuous Galerkin method in *Goma* please see the viscoelastic tutorial memo (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

species_bulk

```
EQ = species_bulk {Galerkin_wt} Y {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation. Definitions of the input parameters are defined below. Note that <floatlist> contains five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

species_bulk	Name of the equation to be solved. This equation type should only be listed once regardless of the number of species (the *Number of bulk species* card specifies the number of species_bulk equations to be solved). Differences in diffusion coefficients between species should be accounted for in the materials properties section of *Goma*.
{Galerkin_wt}	Two- to four-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous • **Q1_XV, Q1_GN, Q1_GP**-Linear interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. • **Q2_XV, Q2_GN, Q1_GP**-Quadratic interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces.
Y	Name of the variable associated with this equation.
{Interpol_fnc}	Two- to four-character value that defines the interpolation function used to represent the variable **Y**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous • **Q1_XV, Q1_GN, Q1_GP**-Linear interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. *See energy equation for more discussion*. • **Q2_XV, Q2_GN, Q1_GP**-Quadratic interpolation with enrichment in elements of material interfaces. This enrichment function allows discontinuity in value and gradient along interface but maintains continuity at element edges/faces. *See energy equation for more discussion.*

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous interpolation for the species equation and turns on all the term multipliers:

```
EQ = species_bulk Q2 Y Q2 1. 1. 1. 1. 1.
```

Technical Discussion

The interpolation/weight functions that are discontinuous, e.g. have the prefix "P", invoke the discontinuous Galerkin (DG) method for solving the species equations where the interpolation is discontinuous and flux continuity is maintained by performing surface integrals. For details of the implementation of the DG method in *Goma* please see the viscoelastic tutorial memo. Note, the DG implementation for the species equation is only for advection dominated problems; DG methods have not yet been completely developed for diffusion operators.

Also, please see EQ=energy input for more detailed description of the Q1_GN, Q2_GN, Q1_GP, Q2_GP, Q1_XV and Q2_XV enriched basis functions.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

mesh

```
EQ = mesh{1|2|3} {Galerkin_wt} {D1|D2|D3} {Interpol_fnc} <floatlist}
```

Description / Usage

This card provides information for solving a differential equation for one component of mesh motion. Definitions of the input parameters are defined below. Note that <floatlist> contains five constants for the Mesh equation defining the constant multipliers for each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

mesh1 | mesh2 | mesh3	Name of the equation to be solved, where the 1, 2 and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.
{Galerkin_wt}	Two-character value that defines the weighting function type for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
D1 | D2 | D3	Name of the variable associated with the 1, 2 or 3 principal coordinate direction for this component equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **D1, D2** or **D3** where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card for the first mesh equation that uses linear continuous interpolation and turns on all term multipliers except for the mass matrix:

```
EQ = mesh1 Q1 D1 Q1 0. 1. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

No References.

mom_solid

```
EQ = mom_solid{1|2|3} {Galerkin_wt} {D1|D2|D3}_RS {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for one component of a solid momentum equation. Definitions of the input parameters are defined below. Note that <floatlist> contains five constants for the Solid Momentum equation defining the constant multipliers for each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

mom_solid1 | mom_solid2 | mom_solid3	Name of the equation to be solved, where the 1, 2 and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
D1_RS | D2_RS | D3_RS	Name of the variable associated with the 1, 2 or 3 principal coordinate direction for this component equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **D1_RS, D2_RS** or **D3_RS** where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card for the first solid mesh equation that uses linear continuous interpolation and turns on all term multipliers except for the mass matrix:

```
EQ = mom_solid1 Q1 D1_RS Q1 0. 1. 1. 1. 1.
```

Technical Discussion

The solid momentum equations are used as a second set of displacement equations when the ALE (arbitrary-Lagrangian Eulerian) technique is used in the solid phase as well as the liquid phase. We have termed this capability TALE for "total arbitraryd Lagrangian Eulerian" and details of implementation and usage for Goma can be found in Schunk (2000).

References

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

continuity

```
EQ = continuity {Galerkin_wt} P {Interpol_fnc} <float1> <float2>
```

Description / Usage

This card provides information for solving a differential equation. Definitions of the input parameters are defined below. Note that <float1> and <float2> define the constant multipliers for each type of term in the Continuity equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

continuity	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous • **P0_XV**-Constant, discontinuous, enriched (level-set only) • **P1_XV**-Linear, discontinuous, enriched (level-set only) • **Q1_XV**-Linear, continuous, enriched (level-set only) • **Q2_XV**-Linear, continuous, enriched (level-set only)
P	Name of the variable (pressure) associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous • **P0_XV**-Constant, discontinuous, enriched (level-set only) • **P1_XV**-Linear, discontinuous, enriched (level-set only) • **Q1_XV**-Linear, continuous, enriched (level-set only) • **Q2_XV**-Linear, continuous, enriched (level-set only)
<float1>	Multiplier on divergence term.
<float2>	Multiplier on source term. This multiplier is equal to the initial volume fraction of solvents for Lagrangian mesh motion with swelling.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a constant discontinuous pressure interpolation and weight function and turns on both the divergence and source terms.

```
EQ = continuity P0 P P0 1. 1.
```

Technical Discussion

Please see the EQ=energy equation card for a more detailed description of **P0_XV, P1_XV, Q1_XV, Q2_XV** interpolations. These are MOST COMMONLY used for the continuity equation for better accuracy of representing pressure across level-set interfaces with surface tension.

References

No References.

fill

```
EQ = fill {Galerkin_wt} F {Interpol_fnc} <float1> <float2> <float3>
```

Description / Usage

This card provides information for solving a differential equation for the fill equation. Definitions of the input parameters are defined below. Note that <float1> through <float3> define the constant multipliers for each term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

fill	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous
F	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **F**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on source term.

Examples

The following is a sample card that uses continuous linear interpolation for the fill equation:

```
EQ = fill Q1 F Q1 1. 1. 1.
```

Technical Discussion

The fill equation is used in the calculation of volume of fluid interface tracking. It solves an advection equation of a color function that takes on a different integer value depending on which fluid phase you are in. For most applications this capability has been superseded by the level set method of interface tracking.

The interpolation/weight functions that are discontinuous, e.g. have the prefix "P" invoke the discontinuous Galerkin (DG) method for solving the fill equations where the interpolation is discontinuous and flux continuity is maintained by evaluating surface integrals. For details of the implementation of the DG method in *Goma* please see the viscoelastic tutorial memo (Rao, 2000).

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

lagr_mult_1, lagr_mult_2, lagr_mult_3

```
EQ = lagr_mult_{1|2|3} {Galerkin_wt} LM{1|2|3} {Interpol_fnc}
```

Description / Usage

This card provides information for solving a Langrange multiplier vector equation for imposition of the kinematic boundary condition at a fluid/solid interface. It is used soley for the overset grid capability in Goma (cf. GT-026.2). Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be the same for the code to work properly.

lagr_mult_1 | lagr_mult_2 | lagr_mult_3	Name of the equation to be solved. The appended number indexes with the dimension of the problem, viz. lagr_mult_1 and lagr_mult_2 equations are required for a two dimensional problem.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
LM1 | LM2 | LM3	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous

Basically when the level-set field (actually phase field 1, cf. F1 equation) that corresponds to solid/fluid boundary defined by an overset grid (using the Slave Level Set Card) intersects an element, the equations associated with that element will get the kinematic boundary condition for the fluid-structure interaction, which basically equates the fluid velocity to

the solid velocity. In elements that don't contain the solid/ fluid boundary, the equations are trivialized so that they are condensed out of the system to be solved.

Examples

The following is a sample cards are required for the overset grid capability for twodimensional problems. It is recommended that P0 (element constant) interpolation functions be used.

```
EQ = lagr_mult_1 P0 LM1 P0 1
```

```
EQ = lagr_mult_2 P0 LM2 P0 1
```

Technical Discussion

No Discussion.

References

GT-026.3 GOMA's Overset Mesh Method: User Tutorial. November 19, 2003. P. R. Schunk and E. D. Wilkes

level set

```
EQ = level set {Galerkin_wt} F {Interpol_fnc} <float1> <float2> <float3>
```

Description / Usage

This card provides information for solving a differential equation for the level set equation. Definitions of the input parameters are defined below. Note that <float1> through <float3> define the constant multipliers for each term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired, we can set this through a Petrov-Galerkin weight function in the level set section of the input file (*Time Integration Specifications*).

level set	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous
F	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **F**, where: • **P0**-Constant Discontinuous • **P1**-Linear Discontinuous • **Q1**-Bilinear/Trilinear Continuous • **Q2**-Biquadratic/Triquadratic Continuous • **Q1_D**-Standard linear interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **Q2_D**-Standard quadratic interpolation with special allowance for discontinuous degrees of freedom at interfaces. • **PQ1**-Q1 Discontinuous • **PQ2**-Q2 Discontinuous

Examples

The following is a sample card that uses continuous linear interpolation for the level set equation and turns on all term multipliers:

```
EQ = level_set Q1 F Q1 1. 1. 1.
```

Technical Discussion

The interpolation/weight functions that are discontinuous, e.g. have the prefix "P", invoke the discontinuous Galerkin (DG) method for solving the level set equations where the interpolation is discontinuous and flux continuity is maintained by evaluating surface integrals. For details of the implementation of the DG method in *Goma* please see the viscoelastic tutorial memo (Rao, 2000). Note that DG methods are not necessarily recommended for the level set equation since it is inherently smooth.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

voltage

```
EQ = voltage {Galerkin_wt} V {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for the voltage. Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

voltage	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
V	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **V**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear interpolation function for voltage:

```
EQ = voltage Q1 V Q1 0. 1. 1. 1. 1.
```

Technical Discussion

The voltage equation has no mass term, viz. it is quasistatic. So it won't matter whether that multiplier is 1 or 0.

References

No References.

efield

```
EQ = efield{1 | 2 | 3} {Galerkin_wt} {E1 | E2 | E3} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a definition equation for the vector electric field, which is the gradient of the voltage or potential field (see voltage equation). Hence, these equations (two components in two dimensions, and three components in three dimensions) must be solved together with the voltage equation.

efield1 | efield2 | efield3	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
E1 | E2 | E3	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **E1, E2,** or **E3,** where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on advective term.
<float2>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is an example of efield-equation specification using linear elements in two dimensions. Notice the companion voltage equation.

```
EQ = efield1 Q1 E1 Q1 1. 1.
```

```
EQ = efield2 Q1 E1 Q1 1. 1.
```

```
EQ = voltage Q1 V Q1 1. 1. 1. 1. 1. 1.
```

This set of equations is required for applying an electrohydrodynamic force to the fluid momentum equations (see *Navier-Stokes Source* card.)

Technical Discussion

The electric field is defined by $\underline{E} = -\Delta\ \phi$. In some cases it may be more convenient to solve equations for the potential field and the electric field.

enorm

```
EQ = enorm {Galerkin_wt} ENORM {Interpol_fnc} <float1> <float2>
```

Description / Usage

This card provides information for solving a "dependency" equation for the norm of the electric field. Definitions of the input parameters are defined below. Note that <float1> and <float2> define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

enorm	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **P0**-Piecewise constant • **P1**-Piecewise linear • **Q1**-Linear • **Q2**-Quadratic
ENORM	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **ENORM**, where: • **P0**-Piecewise constant • **P1**-Piecewise linear • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on advection term.
<float2>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped. See below for important information regarding this.

Examples

The following is a sample card that uses quadratic continuous interpolation for the enorm equation and turns on all the term multipliers (the usual usage):

```
EQ = enorm Q2 ENORM Q2 1.0 1.0
```

Technical Discussion

This equation allows the user to use the variable ENORM, the norm of the electric field, which is equal to $|\underline{E}|$, or $|\underline{\Delta}$ V |, with V being the voltage potential. As such, the VOLTAGE equation must be present. We refer to this as a "dependent" equation or "auxiliary" equation because, although it's value can technically be derived from the V variable directly, we would lose derivative information by doing so. This equation is introduced solely so one can access higher derivatives of V than its interpolation would normally allow. For example, V if were interpolated with a linear basis, then $\underline{\Delta}$ V would have a constant interpolant. If we wanted access to $\underline{\Delta}$ ($\underline{\Delta}$ V) , it would be zero! (In reality, we would use bilinear or trilinear basis functions, so this isn't precisely true but it expresses the essential problem.) By introducing this primitive variable, we can retrieve useful values for $\underline{\Delta}$.

The two term multipliers refer to the multiple on the assembled value of enorm (stored in the "advection" term–it has nothing to do with advection), and the multiple on the assembled value derived from the voltage equation (stored in the "source" term–again the name of the term is somewhat artificial).

References

No References.

shear_rate

```
EQ = shear_rate {Galerkin_wt} SH {Interpol_fnc} <float1> <float2> <float3>
```

Description / Usage

This card provides information for solving a differential equation for the scalar shear rate invariant. Definitions of the input parameters are defined below. Note that <float1> through <float3> define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shear_rate	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
SH	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **SH**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on advective term.
<float2>	Multiplier on diffusion term.
<float3>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous interpolation for the species equation and turns on all the term multipliers:

```
EQ = shear_rate Q2 SH Q2 1. 1. 1.
```

Technical Discussion

No Discussion.

References

No References.

vort_dir

```
EQ = vort_dir{1|2|3} {Galerkin_wt} {VD1|VD2|VD3} {Interpol_fnc}
```

Description / Usage

This card provides information for solving a differential equation for one component of the vorticity equation. Definitions of the input parameters are defined below; there is no <float> input for this equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

vort_dir1 | vort_dir2 | vort_dir3	Name of the equation to be solved, where the 1, 2 and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.
{Galerkin_wt}	Two–character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
VD1 | VD2 | VD3	Name of the variable associated with the 1, 2 or 3 principal coordinate direction for this component equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **VD1, VD2** or **VD3**, where: • **Q1**-Linear • **Q2**-Quadratic

Examples

The following is a sample card that uses a linear continuous interpolation and weight function:

```
EQ = vort_dir1 Q1 VD1 Q1
```

Technical Discussion

This equation type is used for a research capability involving the flows of suspensions in curvilinear coordinates and is not currently being used for production computations.

vort_lambda

```
EQ = vort_lambda {Galerkin_wt} VLAMBDA {Interpol_fnc}
```

Description / Usage

This card provides information for solving a differential equation for the vorticity direction. Definitions of the input parameters are defined below; there are no <float> input parameters for this equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

vort_lamda	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
VLAMBDA	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **VLAMBDA**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous

Examples

No Examples.

Technical Discussion

This equation type is used for a research capability involving the flows of suspensions in curvilinear coordinates and is not currently being used for production computations.

porous_sat

```
EQ = porous_sat {Galerkin_wt} P_liq {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for saturated porous flow. Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation.The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

porous_sat	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
P_SAT	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P_SAT**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function for the saturated porous equation and has all the term multipliers on except for the mass matrix for time derivatives:

```
EQ = porous_sat Q1 P_liq Q1 0. 1. 1. 1. 1.
```

Technical Discussion

This card is not yet in use.

References

No References.

porous_unsat

```
EQ = porous_unsat {Galerkin_wt} P_LIQ {Interpol_fnc} <floatlist>
```

Description / Usage

This equation cannot be invoked in an element block in which the media type is set to POROUS_TWO_PHASE (cf. *Microstructure Properties, Media Type* card). Otherwise, it is used exactly as the *porous_liq* equation card; please consult that section for a detailed discussion.

See *porous_liq* card for description of input requirements.

Examples

See *porous_liq* card.

Technical Discussion

This card is used for single phase (viz. constant gas pressure) simulations of partially saturated flow, as described by the *Media Type* material property card. The equation it invokes is one of Darcy flow in a partially saturated medium in which the gas phase pressure is taken as constant. The dependent variable here is the liquid phase pressure.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

porous_liq

```
EQ = porous_liq {Galerkin_wt} P_LIQ {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for porous liquid phase pressure. This equation is of the exact same form as *porous_unsat*, but is required if two-phase simulations are to be made (cf. *Miscrostructure Properties, Media Type* card). This equation can be used for media types *POROUS_UNSATURATED, POROUS_SATURATED, and POROUS_TWO_PHASE*. Definitions of the input parameters are defined below. Note that <floatlist> contains five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

porous_liq	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
P_LIQ	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P_LIQ**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function for the porous liquid phase pressure equation and has all the term multipliers on except for the mass matrix for time derivatives:

```
EQ = porous_liq Q1 P_LIQ Q1 0. 1. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

porous_gas

```
EQ = porous_gas {Galerkin_wt} P_GAS {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for porous gas phase pressure. Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation.The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

porous_gas	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
P_GAS	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P_GAS**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function for the porous gas phase pressure equation and has all the term multipliers on except for the mass matrix for time derivatives:

```
EQ = porous_gas Q1 P_GAS Q1 0. 1. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

porous_deform

```
EQ = porous_deform {Galerkin_wt} P_POR {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for porous solid phase porosity. Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly. If upwinding is desired for advection dominated problems, we can set this through a Petrov-Galerkin weight function in the material file.

porous_deform	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
P_POR	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **P_POR**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function for the deforming porous porosity equation and has all the term multipliers on except for the mass matrix for time derivatives:

```
EQ = porous_deform Q1 P_POR Q1 0. 1. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

porous_energy

```
EQ = porous_energy {Galerkin_wt} P_TEMP {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a conservation of energy differential equation for porous media, deploying a multiphase formulation.. Definitions of the input parameters are defined below. Note that <floatlist> contains six constants for the porous energy equation defining the constant multipliers for each term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

porous_energy	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
T	Name of the variable associated with this equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **T**, where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a linear continuous interpolation and weight function and has all the term multipliers on:

```
porous_energy Q1 P_TEMP Q1     1.   1.   1.   1.   1.
```

Technical Discussion

Usage of this equation is discussed extensively in GT-009.3 Output variables in the ExodusII database are POR_TEMP

References

GT-009

surf_charge

```
EQ =surf_charge {Galerkin_wt} QS {Interpol_fnc} <float1> <float2> <float3>
<float4> <float5>
```

Description / Usage

This card provides information for solving a conservation equation for the total surface charge in a 2-dimensional bar (or shell) element.. Note that this equation is not yet available in three dimensions and is in fact untested at this time. The card entries are as follows:

surf_charge	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
QS	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **QS** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier for mass terms. Set to 1.0.
<float2>	Multiple for advection terms. Set to 1.0.
<float3>	Multiplier for boundary terms. Set to 1.0.
<float4>	Multiplier for diffusion terms - required but not currently implemented.
<float5>	Multiplier for source terms - required but not currently implemented.

Examples

The following is a sample card that uses bilinear surface charge interpolation and weight function:

```
EQ = surf_charge Q2 QS Q2 1.0 1.0 1.0 0.0 0.0
```

Technical Discussion

The surface charge conservation equation implemented is:

$$\int_s \left(\frac{\partial \sigma}{\partial t} - D_s \nabla_s{}^2 \sigma + \varepsilon \underline{n} \bullet \underline{E} \right) ds = 0$$

where σ is the surface charge unknown, D_s is the surface diffusion coefficient, e is the electrical permittivity, $\underline{n}$ is the unit normal vector to the surface, and $\underline{E} = -\Delta$ V is the electric field vector. Here, advection contributions are not considered.

This is a special type of shell equation which depends on the gradient of a bulk variable (here, electric potential V). Since values of these variables away from the surface are normally not accessible during assembly of shell equations, this term has to be applied as a special type of boundary condition (WEAK_SHELL_GRAD) which is set up to evaluate sensitivities to interior bulk variable degrees of freedom . This term, though physically an integral part of the surface charge equation, is applied through the SURFACE_ELECTRIC_FIELD_BC boundary condition.

References

Notz, Patrick K. Ph.D. thesis. Purdue University, 2000.

shell_tension

```
EQ = shell_tension {Galerkin_wt} {TENS} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving an equation for tension using the structural shell capability in Goma. The capability is based on inextensible cylindrical shells. One material property is associated with this equation and that is the bending stiffness. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_tens	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
TENS	Name of the variable associated with the shell tension equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **TENS** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous tension interpolation and weight function:

```
EQ = momentum1 Q1 TENS Q1 1.0
```

Technical Discussion

Complete tutorial on the use of this equation exists. See GT-27.1.

Theory

The structural shell equation capability in Goma builds on the shell-element capability built by Pat Notz and Ed Wilkes in FY03. Basically we are solving the following equations for the shell tension (this card) and shell curvature (see shell_curvature equation):

$$-D\frac{\mathrm{d}^2}{\mathrm{d}S}K + KT + \underset{\sim}{n}\cdot\underset{\sim}{n}\cdot\underset{\sim}{\sigma} = 0 \qquad \frac{dT}{dS} - DK\frac{dK}{dS} + \underset{\sim}{t}\cdot\underset{\sim}{n}\cdot\underset{\sim}{\sigma} = 0$$

References

GT-27

shell_curvature

```
EQ = shell_curvature {Galerkin_wt} {K} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a definition equation for total curvature of a two-dimensional shell element. Note that this equation is not yet available in three dimensions. The curvature is required by the inextensible cylindrical shell capability in Goma. See references cited below. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_curvature	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
K	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **K** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = momentum1 Q1 K Q1 1.0
```

Technical Discussion

Complete tutorial on the use of this equation exists. See GT-27.

Theory

The structural shell equation capability in Goma builds on the shell-element capability built by Pat Notz and Ed Wilkes in FY03. Basically we are solving the following equations for the shell tension and shell curvature (this card):

$$-D\frac{\mathrm{d}^2}{\mathrm{d}S}K + KT + \underset{\sim}{n}\cdot\underset{\sim}{n}\cdot\underset{\sim}{\sigma} = 0 \qquad \frac{dT}{dS} - DK\frac{dK}{dS} + \underset{\sim}{t}\cdot\underset{\sim}{n}\cdot\underset{\sim}{\sigma} = 0$$

References

GT-27

shell_angle

```
EQ =shell_angle{1 | 2} {Galerkin_wt} {SH_ANG1 | SH_ANG2} {Interpol_fnc}
```

Description / Usage

This card provides information for solving a definition equation for the surface orientation angle in a 2-dimensional bar element. It applies only to shell element blocks. Note that this equation is available in three-dimensional problems but is in fact untested at this time.. The shell angle equation(s) determine the components of the normal vector to the shell surface; since its magnitude is 1 by definition, one less degree of freedom is required tha the number of coordinates. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_angle{1|2}	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
SH_ANG{1|2}	SH_ANG{1|2} Name of the variable associated with the shell angle equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **SH_ANG** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous

This equation requires no equation term multiplier entries.

Examples

The following are sample cards that use linear continuous curvature interpolation and weight function:

```
EQ = shell_angle1 Q1 SH_ANG1 Q1
```

```
EQ = shell_angle2 Q1 SH_ANG2 Q2
```

The second card applies only to 3D problems.

Technical Discussion

For 2D problems, the defining equation is: Θ = atan[n_x, n_y] where Q is shell_angle1 and n_x and n_y are the components of the normal vector to the shell surface. There is an analogous definition for shell_angle2.

References

No References.

shell_diff_flux

```
EQ =shell_diff_flux {Galerkin_wt} SH_J {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a conservation equation for the total surface diffusive flux in a 2-dimensional bar (or shell) element. Note that this equation is not yet available in three dimensions and is in fact untested at this time. The card entries are as follows:

shell_diff_flux	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
SH_J	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **SH_J** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier for diffusion terms (in this case, the whole equation).

Examples

The following is a sample card that uses bilinear shell diffusive flux interpolation and weight function:

```
EQ = shell_diff_flux Q2 SH_J Q2 1.0
```

Technical Discussion

This is only a preliminary implementation of a shell quantity conservation equation. It is not currently operational. When it is fully implemented, the number of required equation term multiplier entries will be adjusted acordingly.

References

No References.

shell_diff_curv

```
EQ =shell_diff_curv {Galerkin_wt} SH_KD {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a definition equation for the total surface curvature in a 2-dimensional bar element, intended for use with shell diffusive flux problems. Note that this equation is not yet available in three dimensions and is in fact untested at this time. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_diff_curv	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
SH_KD	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **SH_KD** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on diffusion term (i.e. the whole equation). Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = shell_diff_curv Q1 SH_KD Q1 1.0
```

Technical Discussion

The equation solved is the surface curvature definition $\kappa = \Delta_s \underline{n} = (I - \underline{n}\ \underline{n}) \bullet \Delta$ n. See discussion for EQ = shell_surf_div_v.

References

Edwards, D. A., Brenner, H., Wasan, D. T., 1991. Interfacial Transport Processes and Rheology. Butterworth-Heinemann, Boston.

shell_normal

```
EQ =shell_normal{1|2} {Galerkin_wt} {SH_N1|SH_N2} {Interpol_fnc} <float1>
```

Description / Usage

This card specifies a vector of shell normal vector component unknowns in a 2- dimensional bar element, intended for use with shell diffusive flux problems. Note that this equation is not yet available in three dimensions and is in fact untested at this time. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_normal1 | shell_normal2	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
SH_N1 | SH_N2	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **SH_N1(2)** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on diffusion term (i.e. the whole equation). Set to 1.0.

Examples

The following is a pair of sample cards that use linear continuous normal interpolation and weight function:

```
EQ = shell_normal1 Q1 SH_N1 Q1 1.0
```

```
EQ = shell_normal2 Q1 SH_N2 Q1 1.0
```

Note that since this equation applies only to 2D problem domains at this time, two cards are needed as shown above (one for each component).

Technical Discussion

This equation merely sets the components of the shell normal vector equal to those in fv->snormal, which are calculated rigorously in surface_determinant_and_normal(). Consideration is being given to replacing these with a single unknown for shell normal angle, which contains the same information in a single scalar unknown.

References

No References.

shell_surf_curv

```
EQ =shell_surf_curv {Galerkin_wt} gamma2 {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a definition equation for the total surface curvature in a 2-dimensional bar element. Note that this equation is not yet available in three dimensions and is in fact untested at this time. These building blocks are required by the non-Newtonian surface rheology capability in Goma. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

shell_surf_curv	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
gamma2	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	{Interpol_fnc} Two- or four-character value that defines the interpolation function used to represent the variable **K** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = shell_surf_curv Q1 gamma2 Q1 1.0
```

Technical Discussion

See discussion for EQ = shell_surf_div_v

References

Edwards, D. A., Brenner, H., Wasan, D. T., 1991. Interfacial Transport Processes and Rheology. Butterworth-Heinemann, Boston.

shell_surf_div_v

```
EQ = shell_surf_div_v {Galerkin_wt} gamma1 {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a definition equation for surface divergence of the fluid velocity field on a 2-dimensional bar element. Note that this equation is not yet available in three dimensions. This term is required by the non-Newtonian surface rheology capability in Goma. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly. Also note that this term is not currently active in Goma, and the developers should be consulted.

shell_surf_div_v	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
gamma1	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **K** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = shell_surf_div_v Q1 gamma1 Q1 1.0
```

Technical Discussion

This shell equation is required for proper computation of the Boussinesq-Scriven surface rheological constitutive equation (namely the surface divergence of the velocity field in the 4th and 6th terms on the right of the following equation). The functional form of this equation is as follows:

$$-(\underset{\sim}{n}\cdot\underset{\sim}{T}) = F^{w} + 2H\sigma\underset{\sim}{n} + \nabla_s\sigma + (k^s+\mu^s)\nabla_s(\nabla_s\cdot\underset{\sim}{v}) + 2\mu^s\underset{\sim}{n}(\underset{\sim}{b}-2H\underset{\sim}{I}_s)\bullet\nabla_s\underset{\sim}{v} + 2\mu^s\underset{\sim}{n}(k^s+\mu^s)\nabla_s\cdot\underset{\sim}{v}$$
$$+\mu^s\{\underset{\sim}{n}\times\ \nabla_s([\nabla_s\times\ \underset{\sim}{v}]\cdot\underset{\sim}{n}) - 2(\underset{\sim}{b}-2H\underset{\sim}{I}_s)\cdot(\nabla_s\underset{\sim}{v})\cdot\underset{\sim}{n}\}$$

Here, $\Delta_s \equiv (\underline{I} - \underline{n}\,\underline{n})\cdot\Delta$ is the surface gradient operator, and $I_s \equiv (\underline{I} - \underline{n}\,\underline{n})$ is the surface unit tensor. μ_s and κ_s are the surface shear viscosity and surface extensional viscosity, respectively. Note that the first three terms on the right are balance of the stress in the standard Goma CAPILLARY condition, with surface tension gradients being accommodated through variable surface tension. The boundary condition CAPILLARY_SHEAR_VISC is used to set the additional terms of this constitutive equation. *As of January 2006 only the 7th term on the right hand side is implemented, as it is the only nonzero term in a flat surface shear viscometer*. The building blocks for the other terms are available through additional shell equations. These remaining terms actually represent additional dissipation caused by surface active species microstructures flowing in the surface. The best source of discussion of this equation is a book by Edwards et al. (1991. *Interfacial Transport Processes and Rheology*. Butterworth-Heinemann, Boston).

References

Edwards, D. A., Brenner, H., Wasan, D. T., 1991. Interfacial Transport Processes and Rheology. Butterworth-Heinemann, Boston.

grad_v_dot_n1, grad_v_dot_n2, grad_v_dot_n3

```
EQ = grad_v_dot_n[1|2|3] {Galerkin_wt} gamma3_[1|2|3] {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a definition equation for the normal components of the velocity gradient tensor in a 2-dimensional bar element. Note that this equation is not yet available in three dimensions and is in fact untested at this time.. These building blocks are required by the non-Newtonian surface rheology capability in Goma. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

grad_v_dot_n1, grad_v_dot_n2, grad_v_dot_n3	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
gamma3_[1|2|3]	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	Two- or four-character value that defines the interpolation function used to represent the variable **K** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = grad_v_dot_n1 Q1 gamma3_1 Q1 1.0
```

Technical Discussion

See discussion for EQ=shell_surf_div_v

References

Edwards, D. A., Brenner, H., Wasan, D. T., 1991. Interfacial Transport Processes and Rheology. Butterworth-Heinemann, Boston.

n_dot_curl_v

```
EQ = n_dot_curl_v {Galerkin_wt} gamma4 {Interpol_fnc} <float1>
```

Description / Usage

This card provides information for solving a definition equation for the normal component of the surface curl of the velocity field on a 2-dimensional bar element. Note that this equation is not yet available in three dimensions. This term is required by the non-Newtonian surface rheology capability in Goma. Note that <floatlist> contains one constant and it should always be set to one. The Galerkin weight and the interpolation function must be the same for the code to work properly.

n_dot_curl_v	Name of the equation to be solved.
{Galerkin_wt}	Two- or four-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
gamma4	Name of the variable associated with the shell curvature equation.
{Interpol_fnc}	{Interpol_fnc} Two- or four-character value that defines the interpolation function used to represent the variable **K** where: • **Q1**-Linear Continuous • **Q2**-Quadratic Continuous
<float1>	Multiplier on whole equation. Set to 1.0.

Examples

The following is a sample card that uses linear continuous curvature interpolation and weight function:

```
EQ = n_dot_curl_v Q1 gamma4 Q1 1.0
```

Technical Discussion

This shell equation is required for proper computation of the Boussinesq-Scriven surface rheological constitutive equation elements (namely the surface curl of the velocity field, normal component) in the 7th term on the right of the following equation):

$$-(\underset{\sim}{n}\cdot\underset{\sim}{T}) = F^{w} + 2H\sigma\underset{\sim}{n} + \nabla_{s}\sigma + (k^{s}+\mu^{s})\nabla_{s}(\nabla_{s}\cdot\underset{\sim}{v}) + 2\mu^{s}\underset{\sim}{n}(\underset{\sim}{b}-2H\underset{\sim}{I}_{s})\bullet\nabla_{s}\underset{\sim}{v} + 2\mu^{s}\underset{\sim}{n}(k^{s}+\mu^{s})\nabla_{s}\cdot\underset{\sim}{v}$$
$$+\mu^{s}\{\underset{\sim}{n}\times\ \nabla_{s}([\nabla_{s}\times\ \underset{\sim}{v}]\cdot\underset{\sim}{n}) - 2(\underset{\sim}{b}-2H\underset{\sim}{I}_{s})\cdot(\nabla_{s}\underset{\sim}{v})\cdot\underset{\sim}{n}\}$$

Here, $\Delta_s \equiv (\underline{I} - \underline{n}\,\underline{n})\cdot\Delta$ is the surface gradient operator, and $I_s \equiv (\underline{I} - \underline{n}\,\underline{n})$ is the surface unit tensor. μ_s and κ_s are the surface shear viscosity and surface extensional viscosity, respectively. Note that the first three terms on the right are balance of the stress in the standard Goma CAPILLARY condition, with surface tension gradients being accommodated through variable surface tension. The boundary condition CAPILLARY_SHEAR_VISC is used to set the additional terms of this constitutive equation. *As of January 2006 only the 7th term on the right hand side is implemented, as it is the only nonzero term in a flat surface shear viscometer*. The building blocks for the other terms are available through additional shell equations. These remaining terms actually represent additional dissipation caused by surface active species microstructures flowing in the surface. The best source of discussion of this equation is a book by Edwards et al. (1991. *Interfacial Transport Processes and Rheology*. Butterworth-Heinemann, Boston).

References

Edwards, D. A., Brenner, H., Wasan, D. T., 1991. Interfacial Transport Processes and Rheology. Butterworth-Heinemann, Boston.

acous_preal

```
EQ = acous_preal {Galerkin_wt} APR {Interpol_fnc} <float list>
```

Description / Usage

This card provides information for solving a differential equation for the real part of the harmonic acoustic wave equation. Definitions of the input parameters are defined below. Note that <float1> through <float5> define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

acous_preal	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
APR	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation • **Q1**-Linear • **Q2**-Quadratic
<float1>	currently not used.
<float2>	Multiplier on acoustic absorption term.
<float3>	Multiplier on boundary terms.
<float4>	Multiplier on Laplacian term.
<float5>	Multiplier on pressure term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous interpolation for the species equation and turns on all the term multipliers:

```
EQ = acous_preal Q2 APR Q2 0. 1. 1. 1. 1.
```

Technical Discussion

Harmonic form of the wave equation with absorption (attenuation) included. P is the amplitude of the acoustic pressure (complex), k is the wavenumber, α is the absorption coefficient, and ω is the frequency (rad/sec).

$$\nabla^2 P + k^2\left(1 - \frac{2\alpha i}{\omega}\right)P = 0$$

acous_pimag

```
EQ = acous_pimag {Galerkin_wt} API {Interpol_fnc} <float list>
```

Description / Usage

This card provides information for solving a differential equation for the imaginary part of the harmonic acoustic wave equation. Definitions of the input parameters are defined below. Note that <float1> through <float5> define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

acous_pimag	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
API	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **SH**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	currently not used.
<float2>	Multiplier on acoustic absorption term.
<float3>	Multiplier on boundary terms.
<float4>	Multiplier on Laplacian term.
<float5>	Multiplier on pressure term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous interpolation for the species equation and turns on all the term multipliers:

```
EQ = acous_pimag Q2 API Q2 0. 1. 1. 1. 1.
```

Technical Discussion

Harmonic form of the wave equation with absorption (attenuation) included. P is the amplitude of the acoustic pressure (complex), k is the wavenumber, α is the absorption coefficient, and ω is the frequency (rad/sec).

$$\nabla^2 P + k^2\left(1 - \frac{2\alpha i}{\omega}\right)P = 0$$

acous_reyn_stress

```
EQ = acous_reyn_stress {Galerkin_wt} ARS {Interpol_fnc} <float list>
```

Description / Usage

This card provides information for solving a differential equation for the Reynolds stress that results from time averaging of the acoustic pressure and velocity fields. Interactions of the fluid momentum equations with the acoustic wave equations are then afforded through gradients of the scalar acoustic Reynolds stress with the use of the ACOUSTIC Navier-Stokes source. Definitions of the input parameters are defined below. Note that <float1> through <float3> define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

acous_reyn_stress	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
ARS	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **SH**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier for the Reynolds stress variable.
<float2>	Multiplier for the kinetic energy term.
<float3>	Multiplier for the compressional energy term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses quadratic continuous interpolation for the acoustic Reynolds stress equation and turns on all the term multipliers:

```
EQ = acous_reyn_stress Q2 ARS Q2 . 1. 1. 1.
```

Technical Discussion

The Reynolds stress due to acoustic fields reduces to a combination of compressional and kinetic energy terms which can be expressed in terms of the magnitude of the acoustic pressure and its gradient. P is the amplitude of the acoustic pressure (complex), k is the wavenumber, R is the acoustic impedance, and ω is the frequency (rad/sec).

$$ARS = \langle \rho \underset{\sim}{u} \underset{\sim}{u} \rangle = \langle \frac{P^2}{2\rho_0 c^2} \rangle - \langle \frac{\rho_0}{2} \underset{\sim}{u} \bullet \underset{\sim}{u} \rangle = \frac{k}{4R\omega} P^2 + \frac{\nabla P \bullet \nabla P}{4kR\omega}$$

potential1

```
EQ = potential1 {Galerkin_wt} PHI1 {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for the solid-phase electrode potential. This electrode-potential equation is solved together with the liquidphase electrolyte-potential equation (see the potential2 card) for simulating electrochemical processes (such as thermal batteries and proton-exchange-membrane fuel cells) involving simultaneous charge transport in both the liquid-electrolyte and solid-electrode phases (as in the porous anode and cathode). Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

potential1	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
PHI1	Name of the variable associated with this equation.
{Interpol_fnc}	{Interpol_fnc} Two-character value that defines the interpolation function used to represent the variable **PHI1**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a quadratic interpolation and weight function and turns off the mass (or transient) and advection terms but turns on the boundary, diffusion, and source terms:

```
EQ = potential1 Q2 PHI1 Q2 0. 0. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

No References.

potential2

```
EQ = potential2 {Galerkin_wt} PHI2 {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for the liquid-phase electrolyte potential. This electrolyte-potential equation is solved together with the solid-phase electrode-potential equation (see the potential1 card) for simulating electrochemical processes (such as thermal batteries and proton-exchange-membrane fuel cells) involving simultaneous charge transport in both the liquid-electrolyte and solid-electrode phases (as in the porous anode and cathode). Definitions of the input parameters are defined below. Note that <floatlist> has five parameters to define the constant multipliers in front of each type of term in the equation. The Galerkin weight and the interpolation function must be the same for the code to work properly.

potential2	Name of the equation to be solved.
{Galerkin_wt}	Two-character value that defines the type of weighting function for this equation, where: • **Q1**-Linear • **Q2**-Quadratic
PHI2	Name of the variable associated with this equation.
{Interpol_fnc}	Two-character value that defines the interpolation function used to represent the variable **PHI2**, where: • **Q1**-Linear • **Q2**-Quadratic
<float1>	Multiplier on mass matrix term (d⁄dt).
<float2>	Multiplier on advective term.
<float3>	Multiplier on boundary term ($\underline{n}$ • flux).
<float4>	Multiplier on diffusion term.
<float5>	Multiplier on source term.

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

The following is a sample card that uses a quadratic interpolation and weight function and turns off the mass (or transient) and advection terms but turns on the boundary, diffusion, and source terms:

```
EQ = potential2 Q2 PHI2 Q2 0. 0. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

No References.

lubp

```
EQ = lubp {Galerkin_wt} LUBP {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving the Reynold's lubrication equation for confined flow. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly. Counterparts to this equation for lubrication flow of capillary films (filmequations) are shell_filmp and shell_filmh equations.

lubp	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
LUBP	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **LUBP**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term (not used yet as of 3/4/2010).
<float2>	Multiplier for the diffusion term.
<float3>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = lubp Q1 LUBP Q1 1. 1. 1.
```

This applies the confined flow lubrication equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$\frac{\partial(\rho h)}{\partial t}+\nabla_{II}\bullet\left(\frac{\rho h}{2}(U_A+U_B)-\frac{\rho h^3}{12\mu}\nabla_{II}p-\frac{\rho h^3}{12\mu}\sigma\kappa\delta(\phi)\mathbf{n}-\rho(\phi)\mathbf{g}\right)-(j_A+j_B)=0$$

- The first term multiplier, activating the mass (time-derivative) term is not currently activated as the gap-height is user-prescribed.
- The second term multiplier affects the third and fourth terms (grad_p and surface tension terms).

The third term multiplier activates the Couette flow terms.

lubp_2

```
EQ = lubp_2 {Galerkin_wt} LUBP {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a second-layer Reynold's lubrication equation for confined flow. The second layer is solved on an adjacent shell as lubp equation but shares the same nodes. Please consult tutorials for proper usuage. This equation can be used to model transport between alternating stacks of porous materials. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly. Counterparts to this equation for lubrication flow of capillary films (film-equations) are shell_filmp and shell_filmh equations.

lubp_2	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
LUBP_2	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **LUBP_2** where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term (not used yet as of 3/4/2010).
<float2>	Multiplier for the diffusion term.
<float3>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = lubp_2 Q1 LUBP_2 Q1 1. 1. 1.
```

This applies the confined flow lubrication equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$\frac{\partial(\rho h)}{\partial t}+\nabla_{II}\bullet\left(\frac{\rho h}{2}(U_A+U_B)-\frac{\rho h^3}{12\mu}\nabla_{II}p-\frac{\rho h^3}{12\mu}\sigma\kappa\delta(\phi)\mathbf{n}-\rho(\phi)\mathbf{g}\right)-(j_A+j_B)=0$$

- The first term multiplier, activating the mass (time-derivative) term is not currently activated as the gap-height is user-prescribed.
- The second term multiplier affects the third and fourth terms (grad_p and surface tension terms).

The third term multiplier activates the Couette flow terms.

shell_energy

```
EQ = shell_energy {Galerkin_wt} SH_T {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a shell thermal energy equation. Use of this equation can be made for any shell, including those which involve Reynold's film or confined flow lubrication flow. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly.

shell_energy	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_T	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_T**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the advection term.
<float3>	Multiplier for the boundary term (not used).
<float4>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_energy Q1 SH_T Q1 1. 1. 1. 1. 1.
```

This applies the shell energy equation with all terms activated on a SHELL4 or BAR2 mesh.

Technical Discussion

The equation solved is as follows:

$$h\rho C_p \frac{\partial T}{\partial t} + h\rho C_p \underset{\sim}{u}_{II} \cdot \nabla_{II} T - hK_{eff} \nabla_{II} \cdot \nabla_{II} T + Q_{surf} + Q_{VD} + Q_{Joule} = 0$$

- Clearly this equation looks similar to the standard energy equation for continuum formulations, but the presence of the gap/film thickness h indications that the assumption of a constant shell temperature across the thickness is assumed, and hence all the terms are constant in that integrated direction. The source terms are all invoked in the material files, and there are many types and many submodels.
- Special NOTE: This equation can be up-winded for high Peclet number flows. If the Energy Weight Function card in the companion material file is set to SUPG, then the advection term is stabilized with standard streamwise-upwinding-Petrov- Galerkin approach.

shell_filmp

```
EQ = shell_filmp {Galerkin_wt} SHELL_FILMP {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving the film lubrication equation for free surface flow. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly. Counterparts to this equation for lubrication flow of capillary films (film-equations) are lup_p equation.

shell_filmp	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SHELL_FILMP	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SHELL_FILMP**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the advection term. It is not activated.
<float3>	Multiplier for the boundary term. It is not activated.
<float4>	Multiplier for the diffusion term.
<float5>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_filmp Q1 SHELL_FILMP Q1 1. 0. 0. 1. 1.
```

This applies the film flow equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$\frac{\partial h}{\partial t} + \nabla_{II} \left[\frac{h^3}{3\mu} (-\nabla_{II} p + \nabla_{II} \Pi + \mathbf{f}) + \mathbf{U}_B h \right] + \dot{E} = 0$$

- The mass matrix multiplier activates the time-derivative term.
- The diffusion multiplier activates the terms inside the divergence operator and represents the flux or the flow rate of the liquid film.
- The source (or sink, in this case,) activates the last term, rate of evaporation.
- This equation has to be used with the equation describing SHELL_FILMH.

shell_filmh

```
EQ = shell_filmh {Galerkin_wt} SHELL_FILMH {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving the film lubrication equation for free surface flow. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly.

Counterparts to this equation for lubrication flow of capillary films (film-equations) are lup_p equation.

shell_filmh	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SHELL_FILMH	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SHELL_FILMH**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term. It is not activated.
<float2>	Multiplier for the advection term. It is not activated.
<float3>	Multiplier for the boundary term. It is not activated.
<float4>	Multiplier for the diffusion term.
<float5>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_filmh Q1 SHELL_FILMH Q1 0. 0. 0. 1. 1.
```

This applies the film flow equation with all terms activated.

Technical Discussion

The equation solved is as follows:

- The diffusion multiplier activates the capillary pressure term.
- The source activates the first term.
- This equation does not fit the general prototype of conservation equation where the diffusion and source terms really apply. In all cases, both diffusion and source terms need to be activated.
- This equation has to be used with the equation describing SHELL_FILMP.

$$p + \sigma \nabla_{II}^2 h = 0$$

shell_partc

```
EQ = shell_partc {Galerkin_wt} SHELL_PARTC {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving the z-averaged concentration of particles inside film flow. Definitions of the input parameters are defined below. The Galerkin weight and the interpolation function must be set the same for the code to work properly.

shell_partc	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SHELL_PARTC	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SHELL_PARTC**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the advection term.
<float3>	Multiplier for the boundary term. It is not activated.
<float4>	Multiplier for the diffusion term.
<float5>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_partc Q1 SHELL_PARTC Q1 1. 0. 0. 1. 0.
```

This applies the film flow equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$h\frac{\partial \varphi}{\partial t}+\left[\frac{h^3}{3\mu}(-\nabla_{II}p)+\mathbf{U}_B h\right]\bullet\nabla_{II}\varphi-\nabla_{II}\bullet[Dh\nabla_{II}\varphi]-\varphi\dot{E}=0$$

- The mass matrix multiplier activates the time-derivative term.
- The advection multiplier activates the second term, where the flow rate is dotted onto the gradient of particles concentration and it represents advection of particles due to the liquid film flow.
- The diffusion multiplier activates the terms inside the divergence operator and represents the Fickian diffusion of particles.
- The source activates the last term, rate of evaporation of liquid that contributes to the increase of the particles conentration.
- This equation has to be used with the film profile equation describing SHELL_FILMP and SHELL_FILMH.

shell_sat_closed

```
EQ = shell_sat_closed {Galerkin_wt} SH_SAT_CLOSED {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides the capability to solve the porous shell equations for closed (noninterconnected) structured pores. The Galerkin weight and the interpolation function must be set the same for the code to work properly. The counterpart to this equation is porous_sat_open, which solves for interconnected pores.

shell_sat_closed	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_SAT_CLOSED	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_SAT_CLOSED**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_sat_closed   Q1 SH_SAT_CLOSED Q1   1.0 1.0
```

This applies the equation with all terms activated.

Technical Discussion

The equation solved is as follows:

- The mass matrix multiplier activates the time-derivative term.
- The source matrix multiplier activates the remaining term.
- This equation is required to couple with LUBP to solve for the lubrication forces.
- Currently, this equation assumes that the porous shell is located in the +z direction of the lubrication shell, and the coupling is set up to draw liquid from the lubrication layer by adding a sink term into the lubrication equations.

Beyond the standard porous media material cards for continuum element regions, one needs in the thin-shell material inputs in the following section:

```
Porous Shell Closed Porosity = CONSTANT 0.1
```

```
Porous Shell Height = CONSTANT 1.0
```

```
Porous Shell Radius = CONSTANT 0.01
```

```
Porous Shell Atmospheric Pressure = CONSTANT 1.e6
```

```
Porous Shell Reference Pressure = CONSTANT 0.
```

```
Porous Shell Cross Permeability = CONSTANT 0.2
```

$$\frac{dS}{dt} = -\frac{1}{\varphi}\frac{\left(-\frac{\phi r^2}{8}\right)}{H\mu}\frac{dP}{dz}$$

$$\frac{dP}{dz} = \frac{\frac{P_0}{\left(1-\frac{S}{\phi}\right)^n} - P_{lub} - 2\sigma/r}{H\ S}$$

```
Porous Shell Initial Pore Pressure = CONSTANT 0.
```

Please read the associated material property cards sections for details.

shell_sat_gasn

```
EQ = shell_sat_gasn {Galerkin_wt} SH_SAT_GASN {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides the capability to solve the porous shell equation for the inventory of trapped gas in a closed pore shell simulation, viz. the EQ=shell_sat_closed. The equation tracks the inventory of trapped gas and accounts for the compression (ideal gas law) and dissolution into the invading liquid. Two terms are required in this equation:

shell_sat_gasn	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_SAT_GASN	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_SAT_GASN**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_sat_gasn Q1 SH_SAT_GASN Q1   1.0 1.0
```

This applies the equation with all terms activated.

Technical Discussion

No Discussion.

References

S. A. Roberts and P. R. Schunk 2012. In preparation.

shell_sat_open

```
EQ = shell_sat_open {Galerkin_wt} SH_P_OPEN {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides the capability to solve the porous shell equations for open (interconnected) structured pores. The Galerkin weight and the interpolation function must be set the same for the code to work properly. The counterpart to this equation is shell_sat_closed, which solves for non-interconnected pores.

shell_sat_open	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_SAT_OPEN	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_P_OPEN**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_sat_open Q1 SH_P_OPEN Q1   1.0 1.0
```

This applies the equation with all terms activated.

Technical Discussion

The equation solved is as follows:

- The mass matrix multiplier activates the time-derivative term.
- The source matrix multiplier activates the remaining term.
- This equation is required to couple with LUBP to solve for the lubrication forces.
- Currently, this equation assumes that the porous shell is located in the +z direction of the lubrication shell, and the coupling is set up to draw liquid from the lubrication layer by adding a sink term into the lubrication equations.
- NOT FULLY IMPLEMENTED.

Note that this equation requires the Media Type to be set to POROUS_SHELL_UNSATURATED. With this media type the porous properties for the most part are extracted from the regular (non-shell) porous media property cards, e.g. Permeability, Porosity, Saturation, etc. There are a few exceptions, however. Beyond the standard porous media material cards for continuum element regions, one needs in the thin-shell material inputs the following section:

```
Porous Shell Closed Porosity = CONSTANT 0.1
```

```
Porous Shell Height = CONSTANT 1.0
```

```
Porous Shell Radius = CONSTANT 0.01
```

```
Porous Shell Atmospheric Pressure = CONSTANT 1.e6
```

```
Porous Shell Reference Pressure = CONSTANT 0.
```

$$\frac{dS}{dt} = \frac{1}{\varphi} \nabla \cdot \left(-\frac{\kappa}{\mu} \nabla P \right),$$

$$\frac{dP}{dz} = \frac{P_0 - P_{lub} - \frac{2\sigma}{r}}{HS}$$

$$\nabla_{II} P = \nabla_{II} P_{lub},$$

```
Porous Shell Cross Permeability = CONSTANT 0.2
```

```
Porous Shell Initial Pore Pressure = CONSTANT 0.
```

Please read the associated material property cards sections for details.

shell_sat_open_2

```
EQ = shell_sat_open_2 {Galerkin_wt} SH_P_OPEN_2 {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides the capability to solve a second porous shell equation for open (interconnected) structured pores. The use of this equation requires that the shell material share the same nodes but be a distinct material from that which shell_sat_open resides. Please see the associated tutorials. The Galerkin weight and the interpolation function must be set the same for the code to work properly. The counterpart to this equation is shell_sat_closed, which solves for non-interconnected pores.

shell_sat_open_2	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_SAT_OPEN_2	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_P_OPEN_2**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_sat_open_2 Q1 SH_P_OPEN_2 Q1   1.0 1.0
```

This applies the equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$\frac{dS}{dt} = \frac{1}{\varphi} \nabla \cdot \left(-\frac{\kappa}{\mu} \nabla P \right),$$

$$\frac{dP}{dz} = \frac{P_0 - P_{lub} - \frac{2\sigma}{r}}{HS}$$

$$\nabla_{II} P = \nabla_{II} P_{lub},$$

- The mass matrix multiplier activates the time-derivative term.
- The source matrix multiplier activates the remaining term.
- This equation is required to couple with LUBP to solve for the lubrication forces.
- Currently, this equation assumes that the porous shell is located in the +z direction of the lubrication shell, and the coupling is set up to draw liquid from the lubrication layer by adding a sink term into the lubrication equations.
- NOT FULLY IMPLEMENTED.

shell_deltah

```
EQ = shell_deltah {Galerkin_wt} SH_DH {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides the capability to solve an evolution equation for a changing lubrication gap. The most common example of this would be a melting slider, as in the substrate of a snow ski or during high-energy sliding contact. Melting would change the lubrication gap. The Galerkin weight and the interpolation function must be set the same for the code to work properly. This equation could be furbished or advanced to handle other moving boundary problems which would lead to a changing gap. It should be noted, that gap changes due to a bounding flexible solid structure are already accommodated and fully compatible with this condition.

shell_deltah	Name of equation to be solved.
{Galerkin_wt}	Two-or four-character value that defines the type of weighting function for this equation, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
SH_DH	Name of the variable associated with this equation.
{Interpol_fnc}	Two-or four-character value that defines the interpolation function for the variable **SH_DH**, where: • **Q1**–Linear • **Q2**–Quadratic (not recommended at this time)
<float1>	Multiplier for the mass matrix term.
<float2>	Multiplier for the source term.

Examples

Following is a sample card:

```
EQ = shell_deltah Q1 SH_DH Q1   1.0 1.0
```

This applies the equation with all terms activated.

Technical Discussion

The equation solved is as follows:

$$\rho E_0 \frac{d\delta h}{dt} = H_{trans}(T - T_0)$$

where E_0 is the enthalpy, including the effect of phase change through the latent heat material property specified in the material file. H_{trans} is a heat transfer coefficient and is set in the material file as that due to melting/sliding contact (see material file section on MELTING_CONTACT). dh is the unknown.

- The mass matrix multiplier activates the time-derivative term.

- The source matrix multiplier activates the remaining term.
- This equation is required to couple with SH_TEMP to solve for the local temperature. f

moment

```
EQ = moment{0|1|2|3} {Galerkin_wt} {MOM0|MOM1|MOM2|MOM3} {Interpol_fnc} <floatlist>
```

Description / Usage

This card provides information for solving a differential equation for one component of a population balance equation

All four moments are expected to be enabled at the same time

moment0 | moment1 | moment2 | moment3 Name of the equation to be solved, where the and 3 components correspond to one of the principal coordinate directions, e.g. X, Y and Z for Cartesian geometry.

{Galerkin_wt}

Two- or four-character value that defines the type of weighting function for this equation, where:

- **Q1**-Linear
- **Q2**-Quadratic

MOM0 | MOM1 | MOM2 | MOM3 Name of the variable associated with the moment equation

{Interpol_fnc}

Two- or four-character value that defines the interpolation function used

- **Q1**-Linear Continuous
- **Q2**-Quadratic Continuous

<float1> Multiplier on mass matrix term (d⁄dt).

<float2> Multiplier on advective term.

<float4> Multiplier on diffusion term.

<float5> Multiplier on source term.

<float6> Multiplier on divergence term

Note: These multipliers are intended to provide a means of activating or deactivating terms of an equation, and hence should be set to zero or one. If a multiplier is zero, the section of code that evaluates the corresponding term will be skipped.

Examples

```
EQ = moment0 Q1 MOM0 Q1 1. 1. 1e-6  1. 1.
EQ = moment1 Q1 MOM1 Q1 1. 1. 1e-6  1. 1.
EQ = moment2 Q1 MOM2 Q1 1. 1. 1e-6  1. 1.
EQ = moment3 Q1 MOM3 Q1 1. 1. 1e-6  1. 1.
```

Technical Discussion

No Discussion.

References

Ortiz, Weston, Lisa Mondy, Christine Roberts, and Rekha Rao. "Population balance modeling of polyurethane foam formation with pressure-dependent growth kernel." AIChE Journal 68, no. 3 (2022): e17529.

END OF EQ

```
END OF EQ
```

Description / Usage

This card specifies the end of the list of equations in a material section of the *Problem Description File*. It is only used when automatic equation counting is used, as described and activated in the *Number of EQ* card. If the value of <integer> in that card is set to -1, all EQ cards below this card are ignored, and *Goma* counts the number of EQ cards between the *Number of EQ* card and the *END OF EQ* card.

Note that the *END of EQ* card should appear in every material section for which automatic equation counting is being used.

Examples

There are no input parameters for this card, which always appears as follows:

```
END OF EQ
```

Technical Discussion

No Discussion.

END OF MAT

```
END OF MAT
```

Description / Usage

This card specifies the end of the list of materials. It is only used when automatic material counting is used, as described and activated in the *Number of Materials* card. If the value of <integer> in the *Number of Materials* card is set to -1, all MAT cards below the *END OF MAT* card are ignored, and *Goma* counts the number of MAT cards between these two cards.

Examples

There are no input parameters for this card, which always appears as follows:

```
END OF MAT
```

Technical Discussion

No Discussion.

1.4.14 Post Processing Specifications

This section lists the post-processing options that are accessible within *Goma*. Each card below triggers calculations of the nodal values of a given function, which are then written to the EXODUS II output file. Normally these values are smoothed before writing them to the output file. For most of these cards a keyword is the only input; if the keyword is **yes**, the post-processing variable is calculated and written to the file; if the keyword is **no**, no output is generated for that variable. All of these cards are optional and can appear in any order.

The sections below list the post-processing options and a brief description of each. *Users are cautioned - for large, time-dependent runs, the output of many post-processing variables may lead to excessively large EXODUS II output files.*

Stream Function

```
Stream Function = {yes | no}
```

Description / Usage

The stream function provides a visual representation of the flow field in incompressible fluids and is derived from the fluid velocity components identified in the *Output Exodus II file* card.

This auxiliary field triggered by “yes” on this card results in a nodal variable that is called **STREAM** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the stream function.
no	Do not calculate the stream function.

Examples

Following is a sample card:

```
Stream Function = no
```

Technical Discussion

This function is computed with an element-by-element volumetric flow calculation routine. Poor element quality can result in "kinks" in the stream function field when contoured.

It is important to construct a mesh whose elements are contiguously ordered in such a way that there are no isolated clusters as the elements are swept, i.e., element n+1 must be in contact with one of the previous n elements. NOTE: as of 4/2001 an automatic element reordering scheme based on Reverse Cuthill-McKee algorithm has been implemented in *Goma*. Automatic ordering can be assured by issuing the OPtimize command to the FASTQ meshing module (cf. Blacker 1988). Most other mesh generators do not provide this service, viz. they do not put out an element order-map field in the EXODUS II file.

NOTE: THIS FUNCTION IS NOT AVAILABLE IN THREE DIMENSIONS, but pathlines, which are equivalent to streamlines for steady flows can be computed in many graphics packages, like Mustafa (Glass, 1995).

References

SAND88-1326: FASTQ Users Manual: Version 1.2, Sandia Technical Report, Blacker, T. D. 1988.

Mustafa, Glass, M. W., Personal Communication, 1995

Streamwise Normal Stress

```
Streamwise normal stress = {yes | no}
```

Description / Usage

The stream-wise normal stress, T_{tt}, is defined as tt: τ, where t is the unit tangent vector to the streamlines computed as v$/|$ v $|$ and τ is the deviatoric part of the dissipative stress tensor,

$$\tau \equiv \mu[\nabla \mathbf{v} + (\nabla \mathbf{v})^T]$$

associated with the Navier-Stokes equations. This variable is called SNS in the output EXODUS II file.

The permissible values for this postprocessing option are

yes	Calculate the stream-wise normal stress.
no	Do not calculate the stream-wise normal stress.

Examples

Following is a sample card:

```
Streamwise normal stress = yes
```

Technical Discussion

As of 2/9/02 this function is computed with the based viscosity, and not the strain-rate dependent viscosity as might be the case for viscosity models other than *NEWTONIAN* (see *Viscosity* card).

Cross-Stream Shear Rate

```
Cross-stream shear rate = {yes | no}
```

Description / Usage

As of 2/9/02, it is recommended that this card not be used.

The quantity as computed in *Goma* is only applicable in two-dimensions and it is not clear what this quantity is, as it is computed. (PRS)

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

Mean Shear Rate

```
Mean shear rate = {yes | no}
```

Description / Usage

The mean shear rate is defined as

$$4\sqrt{II_D} \quad ,$$

where II_D is the second invariant of **D**, the strain-rate tensor,

$$\mathbf{D} \equiv [\nabla \mathbf{v} + (\nabla \mathbf{v})^T]$$

associated with the Navier-Stokes equations. This variable is called **SHEAR** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the mean shear rate.
no	Do not calculate the mean shear rate.

Examples

Following is a sample card:

```
Mean shear rate = yes
```

Technical Discussion

No Discussion.

Pressure Contours

```
Pressure contours = {yes | no}
```

Description / Usage

The hydrodynamic pressure is normally a field variable within *Goma*; however, it is often interpolated in finite element space with discontinuous basis functions (in order to satisfy the well-known LBB stability criterion, cf. Schunk, et al. 2002). This option enables interpolating and smoothing the hydrodynamic pressure to nodal values that most post-processors can deal with (e.g. BLOT, Mustafa). This variable is called **PRESSURE** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the pressure contours.
no	Do not calculate the pressure contours.

Examples

Following is a sample card:

```
Pressure contours = no
```

Technical Discussion

No Discussion.

References

SAND2001-3512J: Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, P. R. Schunk, M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun. (March 2002)

Fill Contours

```
Fill contours = {yes | no}
```

Description / Usage

This card triggers the inclusion of the level set or VOF fill function as a nodal variable in the output EXODUS II file.

The nodal variable appears as **FILL** in the output EXODUS II file. This function is computed with the FILL equation (see EQ card).

The permissible values for this postprocessing option are:

yes	Calculate the fill contours.
no	Do not calculate the fill contours.

Examples

An example card requesting **FILL** contours be written to the EXODUS II file is:

```
Fill contours = yes
```

Technical Discussion

No Discussion.

References

GT-020.1: Tutorial on Level Set Interface Tracking in GOMA, February 27, 2001, T.A. Baer

Concentration Contours

```
Concentration contours = {yes | no}
```

Description / Usage

As of 2/9/02 this card is not necessary. If *EQ* = *species_bulk* card is active in any material, than the concentration contours are including as post-processing nodal variables in the output EXODUS II file.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

Stress Contours

```
Stress contours = {yes | no}
```

Description / Usage

This card allows the user to invoke the components of the stress tensor for all viscoelastic modes be included as nodal post-processing variables. Often times this is not desirable on long time-dependent runs because of the voluminous data that will appear in the output EXODUS II file. This variable is called *csij_mode* in the output EXODUS II file, where *i* and *j* indicate components of the stress tensor and *mode* indicates the desired viscoelastic mode; for example, **cs23_4** represents the stress contour for the fifth mode of polymer stress component *yz*.

The permissible values for this postprocessing option are:

yes	Calculate and include the stress-tensor components for all modes of viscoelasticity.
no	Do not calculate and include the stress-tensor components.

These stresses become dependent variables if the *Polymer Constitutive Equation* card is given any model but the *NOPOLYMER* model.

Examples

An example card requesting viscoelastic stress components be written:

```
Stress contours = yes
```

Technical Discussion

No Discussion.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

First Invariant of Strain

```
First Invariant of Strain = {yes | no}
```

Description / Usage

The strain tensor is associated with the deformation of the mesh. Its first invariant is its trace and represents the volume change in the small strain limit. This variable is called **IE** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the first invariant.
no	Do not calculate the first invariant.

Examples

Following is a sample card:

```
First Invariant of strain = yes
```

Technical Discussion

Computation of the strain tensor $\underline{\underline{E}}$ in *Goma* is discussed on the *Solid Constitutive Equation* card. The trace is related to the divergence of the tensor, and hence related to a measure of volume change in a material.

It should be noted that the mesh strain is equivalent to the material strain for *LAGRANGIAN* mesh motion types. For *ARBITRARY* or *TOTAL_ALE* mesh motion types (see *Mesh Motion* card), the strain is strictly related to mesh and not the material.

References

No References.

Second Invariant of Strain

```
Second Invariant of Strain = {yes | no}
```

Description / Usage

The strain tensor is associated with the deformation of the mesh. Its second invariant indicates the level of shear strain of the mesh. This variable is called **IIE** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the second invariant.
no	Do not calculate the second invariant.

Examples

Following is a sample card:

```
Second Invariant of strain = yes
```

Technical Discussion

The second invariant is computed in *Goma* as

$$II_E = \frac{1}{2}(E_{ij}E_{ij} - E_{ii}E_{jj}) \quad .$$

Here Einstein's summation convention applies, viz.

$$E_{ij}E_{ij} = \sum_i \sum_j E_{ij}E_{ij} \quad .$$

Third Invariant of Strain

```
Third Invariant of strain = {yes | no}
```

Description / Usage

The strain tensor is associated with the deformation of the mesh. Its third invariant indicates the volume change from the stress-free state (**IIIE** = 1.0 indicates no volume change). This variable is called **IIIE** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the third invariant.
no	Do not calculate the third invariant.

Examples

Following is a sample card:

```
Third Invariant of strain = yes
```

Technical Discussion

The mathematical definition of the third invariant is related to the determinant of the strain tensor, which is defined for the various constitutive equations in the manual entry for the *Solid Constitutive Equation* card.

References

No References.

Velocity Divergence

```
Velocity Divergence = {yes | no}
```

Description / Usage

The divergence of velocity is associated with local mass conservation or how well the solenoidal character of the velocity field in *ARBITRARY* mesh motion regions is being maintained. (Fluid momentum equations are only applied for this *Mesh Motion* option.) Here we calculate the L_2 norm of the divergence of velocity so that it is always zero or positive. This variable is called **DIVV** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the velocity divergence.
no	Do not calculate the velocity divergence.

Examples

A sample input specification for this card is:

```
Velocity Divergence = no
```

Technical Discussion

The divergence of the fluid velocity field is defined as the scalar $\Delta \bullet \underline{v}$.

References

No References.

Particle Velocity Divergence

```
Particle Velocity Divergence = {yes | no}
```

Description / Usage

This option is currently disabled. As of 2/16/2002, the multiphase particle momentum equation is deactivated. It is not recommended that this option be selected.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

Total Velocity Divergence

```
Not activated (PRS 2/16/02)
```

Description / Usage

Not currently activated. (2/16/02)

Examples

No Examples.

Technical Discussion

No Discussion.

Electric Field

```
Electric Field = {yes | no}
```

Description / Usage

The electric field vector components are written to the output EXODUS II file. The electric field is calculated as the negative gradient of the VOLTAGE field variable.

The permissible values for this postprocessing option are

yes	Calculate the electric field vectors.
no	Do not calculate the electric field vectors.

The vector components are called **EX**, **EY**, and (for three dimensional problems) **EZ** in the output EXODUS II file.

Examples

The following is a sample input card to calculate the Electric Field vector components:

```
Electric Field = yes
```

Technical Discussion

See also the *Electric Field Magnitude* post processing option.

References

No References.

Electric Field Magnitude

```
Electric Field Magnitude = {yes | no}
```

Description / Usage

The magnitude of the electric field is written to the output EXODUS II file. The electric field is calculated as the negative gradient of the *VOLTAGE* field variable.

The permissible values for this postprocessing option are:

yes	Calculate the electric field magnitude.
no	Do not calculate the electric field magnitude.

The electric field magnitude is called EE in the output EXODUS II file.

Examples

The following is a sample input card to calculate the Electric Field Magnitude:

```
Electric Field Magnitude = yes
```

Technical Discussion

See also the *Electric Field* post processing option.

References

No References.

Enormsq Field

```
Enormsq Field = {yes | no}
```

Description / Usage

This norm is based on the *ENORM* field variable (which, in turn, is derived from the *VOLTAGE* field variable).

The permissible values for this postprocessing option are:

yes	Calculate the norm.
no	Do not calculate the norm.

The field is stored in **GENS0**, **GENS1**, and **GENS2** (if 3D) in the output EXODUS II file.

Examples

The following is a sample input card to calculate the field:

```
Enormsq Field = yes
```

Technical Discussion

This post-processing variable is equal to $\underline{\Delta}\ enorm^2$. This, in turn, should approximate $\underline{\Delta} \mid (\underline{\Delta} V \mid^2)$.

See also the *Enormsq Field Norm* post processing option.

References

No References.

Enormsq Field Norm

```
Enormsq Field Norm = {yes | no}
```

Description / Usage

This norm is based on the *ENORM* field variable (which, in turn, is derived from the *VOLTAGE* field variable).

The permissible values for this postprocessing option are:

yes	Calculate the norm.
no	Do not calculate the norm.

The norm is called **GENSNORM** in the output EXODUS II file.

Examples

The following is a sample input card to calculate the norm:

```
Enormsq Field Norm = yes
```

Technical Discussion

This post-processing variable is equal to $\underline{\Delta}\ enorm^2$. This, in turn, should approximate $\underline{\Delta} \mid (\underline{\Delta} V \mid^2)$.

See also the *Enormsq Field* post processing option.

References

No References.

Viscosity

```
Viscosity = {yes | no}
```

Description / Usage

This option allows you to plot the viscosity, which is written to the *Output EXODUS II* file as the variable **MU**. This is a useful feature for non-Newtonian fluids such as Phillip's model for suspensions, Bingham plastic models, polymerizing solutions and other materials for which the viscosity may change orders of magnitude, greatly affecting the velocity and pressure fields. Contouring this variable **MU** over the domain can be useful in explaining some physical phenomena.

he permissible values for this postprocessing option are:

yes	Calculate the viscosity and output as a nodal variable in the Output EXODUS II file.
no	Do not calculate the viscosity.

Examples

The following sample card requests **MU** be written to the EXODUS II file:

```
Viscosity = yes
```

Technical Discussion

See the material file *Viscosity* card for an explanation of the models for which the viscosity is variable and dependent on the flow field and other variables.

Density

```
Density = {yes | no}
```

Description / Usage

This card is used to trigger the thermophysical or mechanical property of density (see *Density* card) to be computed and output as an EXODUS II nodal variable in the Ouput EXODUS II file with the variable name **RHO**.

The permissible values for this postprocessing option are:

yes	Calculate the density and store it as a nodal variable in the output EXODUS II file.
no	Do not calculate density.

Examples

This is an example of the input to request density be written to the EXODUS II file.

```
Density = yes
```

Technical Discussion

No Discussion.

References

No References.

Lame MU

```
Lame MU = {yes | no}
```

Description / Usage

This option allows you to plot the Lame MU mechanical property, which is written to the *Output EXODUS II* file as the variable **LAME_MU**. This is a useful feature for temperature dependent mechanical properties and the like. Contouring this variable **LAME_MU** over the domain can be useful in explaining some physical phenomena.

The permissible values for this postprocessing option are:

yes	Calculate the Lame MU and output as a nodal variable in the Output EXODUS II file.
no	Do not calculate the the coefficient (default).

Examples

The following sample card requests **LAME_MU** be written to the EXODUS II file:

```
Lame MU = yes
```

Technical Discussion

No Discussion.

References

No References.

Lame LAMBDA

```
Lame LAMBDA = {yes | no}
```

Description / Usage

This option allows you to plot the Lame LAMDA mechanical property, which is written to the *Output EXODUS II* file as the variable **LAMBDA**. This is a useful feature for temperature dependent mechanical properties and the like. Contouring this variable **LAMBDA** over the domain can be useful in explaining some physical phenomena.

The permissible values for this postprocessing option are:

yes	Calculate the Lame LAMBDA and output as a nodal variable in the Output EXODUS II file.
no	Do not calculate the the coefficient (default).

Examples

The following sample card requests **LAMBDA** be written to the EXODUS II file:

```
Lame LAMBDA = yes
```

Technical Discussion

No Discussion.

References

No References.

Von Mises Strain

```
Von Mises Strain = {yes | no}
```

Description / Usage

This option allows you to plot the Von Mises strain invarients of the strain tensor, for use with the FAUX_PLASTICITY model of the modulus. These quantities are written to the *Output EXODUS II* file.

The permissible values for this postprocessing option are:

yes	Calculate the von Mises strain invariants and output as a nodal variable in the Output EXODUS II file.
no	Do not calculate the invariants (default).

Examples

The following sample card requests **LAMBDA** be written to the EXODUS II file:

```
Von Mises Strain = yes
```

Technical Discussion

No Discussion.

References

No References.

Von Mises Stress

```
Von Mises Stress = {yes | no}
```

Description / Usage

This option allows you to plot the Von Mises stress tensor invarients, for use with the FAUX_PLASTICITY model of the modulus. These quantities are written to the *Output EXODUS II* file.

The permissible values for this postprocessing option are:

yes	Calculate the von Mises stress invariants and output as a nodal variable in the Output EXODUS II file.
no	Do not calculate the invariants (default).

Examples

The following sample card requests **LAMBDA** be written to the EXODUS II file:

```
Von Mises Stress = yes
```

Technical Discussion

No Discussion.

Navier Stokes Residuals

```
Navier Stokes Residuals = {yes | no}
```

Description / Usage

These post-processing nodal variables are constructed from the corresponding weighted residual function of the fluid (e.g. Navier-Stokes) momentum equations, using a Galerkin finite-element formulation. When activated with this card, variables named **RMX**, **RMY**, and **RMZ** appear in the output EXODUS II file, corresponding to each of the independent components of the fluid momentum balance.

The permissible values for this postprocessing option are:

yes	Calculate the Navier-Stokes residuals and store as nodal variables in the output EXODUS II file.
no	Do not calculate Navier-Stokes residuals.

Examples

Following is a sample input card:

```
Navier Stokes Residuals = no
```

Technical Discussion

This option can be used to help understand convergence behavior of a particular problem, as it allows the user to visualize the pattern of residuals over the computational domain during a Newton iteration process. The intermediate solutions of a Newton iteration process can be activated with the *Write Intermediate Results* card.

Moving Mesh Residuals

```
Moving Mesh Residuals = {yes | no}
```

Description / Usage

These nodal variables are constructed from the corresponding weighted residual functions of the solid momentum equations (activated with the *EQ = mesh** cards). The weighted residuals are formed using a Galerkin finite-element formulation. In the output EXODUS II file they appear as nodal variables **RDX**, **RDY**, and **RDZ**, corresponding to each of the independent components of the solid momentum balance (both pseudo and real).

The permissible values for this postprocessing option are:

yes	Include the moving mesh residuals as nodal variables in the ouput EXODUS II file.
no	Do not include moving mesh residuals.

Examples

Following is a sample card which does not activate writing of mesh residuals:

```
Moving Mesh Residuals = no
```

Technical Discussion

This option can be used to help understand convergence behavior of a particular problem, as it allows the user to visualize the pattern of residuals over the computational domain during a Newton iteration process. The intermediate solutions of a Newton iteration process can be activated with the *Write Intermediate Results* card. Contouring these residuals can indicate where the convergence of a problem is being delayed, and give the user/developer some clues as to the boundary condition or local region of the mesh which is responsible.

Mass Diffusion Vectors

```
Mass Diffusion Vectors = {yes | no}
```

Description / Usage

Activating this post-processing option allows the user to visualize the diffusive mass flux directions of all species components in a problem. Species components result from the *EQ = species_bulk* equation card. With this option selected, the output EXODUS II file will contain nodal variables called **Y0dif0** (diffusion of first species in x direction), **Y0dif1** (diffusion of first species in y direction), **Y0dif2** (diffusion of first species in z direction), **Y1dif0** (diffusion of second species in x direction), **Y2dif1** (diffusion of second species in y direction), … and so on, depending on the number of species components in the problem.

The permissible values for this postprocessing option are:

yes	Calculate the mass diffusion vectors and include in the output EXODUS II file.
no	Do not calculate the mass diffusion vectors.

Examples

Following is a sample card:

```
Mass Diffusion Vectors = yes
```

Technical Discussion

Currently this option is available only for *FICKIAN* and *HYDRODYNAMIC* mass flux types (see *Diffusion Constitutive Equation* card). In the *FICKIAN* case, the flux is computed with the base, constant diffusivity.

References

No References.

Diffusive Mass Flux Vectors

```
Diffusive Mass Flux Vectors = {yes | no}
```

Description / Usage

Please see description for *Mass Diffusion Vectors* card; this card performs exactly the same function.

Examples

This card turns on writing of diffusive mass flux vectors to the EXODUS II file:

```
Diffusive Mass Flux Vectors = yes
```

Technical Discussion

Please see description for *Mass Diffusion Vectors* card.

References

No References.

Mass Fluxlines

```
Mass Fluxlines = {yes | no}
```

Description / Usage

With this post-processing option mass-diffusion pathlines are calculated and stored as post-processing nodal variables in the output EXODUS II file. This variables are called **Y0FLUX**, **Y1FLUX**, …(by species number) in the file and can be contoured in the visualization program. These flux lines are analogous to the stream function, viz. contours of the flux function represent pathlines for each species in solution.

The permissible values for this postprocessing option are:

yes	Calculate the mass fluxlines and include in the output EXODUS II file.
no	Do not calculate the mass fluxlines.

Examples

The following sample card requests that mass-diffusion pathlines be written to the EXODUS II file:

```
Mass Fluxlines = yes
```

Technical Discussion

Currently this option is available only for *FICKIAN* and *HYDRODYNAMIC* mass flux types (see *Diffusion Constitutive Equation* card). In the *FICKIAN* case, the flux is computed with the base, constant diffusivity. Also, the *Mass Diffusion Vectors* post processing option must also be activated for this option to work.

References

No References.

Energy Conduction Vectors

```
Energy Conduction Vectors = {yes | no}
```

Description / Usage

Activation of this option can be used to visualize the energy conduction paths in a solution. The resulting nodal variables are called **TCOND0** (conduction in x direction), **TCOND1** (conduction in y direction), and **TCOND2** (conduction in z direction) in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the energy conduction vectors and store in the output EXODUS II file.
no	Do not calculate the energy conduction vectors.

Examples

This example card requests that energy conduction vectors be written to the EXODUS II file:

```
Energy Conduction Vectors = yes
```

Technical Discussion

These quantities can be employed in a hedge-hog or vector plot to visualize the energy conduction pathways across a domain (cf. the vector option in BLOT, or the hedge-hog option in Mustafa, for example).

References

No References.

Energy Fluxlines

```
Energy Fluxlines= {yes | no}
```

Description / Usage

This post-processing option triggers the energy fluxlines to be calculated. The energy flux function is analogous to the stream function, its contours representing paths of energy flow through the domain. This variable is called **TFLUX** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate and write the energy fluxlines to the output EXODUS II file.
no	Do not calculate the energy fluxlines.

Examples

The energy fluxlines are calculated in this sample input card:

```
Energy Fluxlines = yes
```

Technical Discussion

The *Energy Conduction Vectors* must also be activated for this post processing option to work.

References

No References.

Time Derivatives

```
Time Derivatives = {yes | no}
```

Description / Usage

This option enables writing the time derivative of all the field variables as nodal variables to the output EXODUS II file. These variables are labeled **XDOT0** (mesh velocity in x direction), **XDOT1** (mesh velocity in y direction), **XDOT2** (mesh velocity in z direction), **VDOT0** (fluid acceleration in x direction), **VDOT1** (fluid acceleration in y direction), **VDOT2** (fluid acceleration in z direction), **TDOT** (rate of temperature change), **Y0DOT** (rate of 1st species concentration change), **Y1D0T** (rate of second species concentration change), and so on. The quantities can then be contoured or displayed by some other means with a visualization or graphics package.

The permissible values for this postprocessing option are:

yes	Calculate the time derivatives and write them as nodal variables in the output EXODUS II file.
no	Do not calculate the time derivatives.

Examples

The following sample card requests that time derivatives be written to the EXODUS II file:

```
Time Derivatives = yes
```

Technical Discussion

Currently, this routine uses the values in the global vector *xdot* to report this data. During the first time step, all the *xdot* values are zero; by the second time step, these data should be realistic.

References

No References.

Mesh Stress Tensor

```
Mesh Stress Tensor = {yes | no}
```

Description / Usage

The mesh stress tensor is associated with the equations of elasticity. The stress tensor has six entries (in three dimensions, because it is symmetric) called **T11, T22, T33, T12, T13,** and **T23** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the mesh stress tensor and write to output EXODUS II file.
no	Do not calculate the mesh stress tensor.

Examples

The following sample card turns on the writing of the stress tensor to the EXODUS II file:

```
Mesh Stress Tensor = yes
```

Technical Discussion

The defining constitutive equations for these stresses can be found in the description for the *Solid Constitutive Equation* card. This option applies to all solid-material types (see *Mesh Motion* card), viz. *TOTAL_ALE, LAGRANGIAN, ARBITRARY, DYNAMIC_LAGRANGIAN*. In the *TOTAL_ALE* and *ARBITRARY* mesh motion types, the mesh stress is exactly that and not the true stress of the material. For *TOTAL_ALE* mesh motion types, use Real Solid Stress Tensor option to get the true solid material stresses.

References

No References.

Real Solid Stress Tensor

```
Real Solid Stress Tensor = {yes | no}
```

Description / Usage

The real solid stress tensor is associated with the equations of elasticity. If the mesh motion is of *LAGRANGIAN* type, then these quantities are not available; if is of *TOTAL_ALE* type, they are available. The stress tensor has six entries (in three dimensions because it is symmetric) called **T11_RS, T22_RS, T33_RS, T12_RS, T13_RS**, and **T23_RS** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the real solid stress tensor and write to the output EXODUS II file.
no	Do not calculate the real solid stress tensor.

Examples

No stress tensor is written for the following sample input card:

```
Real Solid Stress Tensor = no
```

Technical Discussion

This option is applicable only to *TOTAL_ALE* mesh motion types (see *Mesh Motion* card). Compare this with *Mesh Stress Tensor* post processing option for other types of mesh motion.

References

No References.

Mesh Strain Tensor

```
Mesh Strain Tensor = {yes | no}
```

Description / Usage

The mesh strain tensor is associated with the equations of elasticity. The strain tensor has six entries (in three dimensions, because it is symmetric) called **E11, E22, E33, E12, E13,** and **E23** in the output EXODUS II file, corresponding to the six independent components (the numbers **1, 2,** and **3** indicate the basis direction, e.g. 1 means xdirection for a Cartesian system).

The permissible values for this postprocessing option are:

yes	Calculate the mesh strain tensor and write the components as nodal variables to the output EXODUS II file.
no	Do not calculate the mesh strain tensor.

Examples

The following example input card does not request output of the stain tensor:

```
Mesh Strain Tensor = no
```

Technical Discussion

Definitions of the strain tensor depend on the solid constitutive equation type (see description for *Solid Constitutive Equation* card).

References

No References.

Viscoplastic Def_Grad Tensor

```
Viscoplastic Def_Grad Tensor = {yes | no}
```

Description / Usage

The components of this tensor are associated with the elasto-viscoplasticity model, (described in detail in Schunk, et. al., 2001). If the mesh motion is of *LAGRANGIAN* type, then this card activates the components of this tensor to be available in the postprocessing EXODUS II file (see *Mesh Motion* card). The components are called **FVP11, FVP12, FVP21, FVP22,** and **FVP33**. This tensor is the identity tensor in regions that have not yielded, and so the diagonal components are unity; in regions that have yielded, these components deviate from the identity. Contouring them can reveal regions of plastic flow.

The permissible values for this postprocessing option are:

yes	Calculate the viscoplastic Def_Grad tensor and write components in the output EXODUS II file.
no	Do not calculate the viscoplastic Def_Grad tensor.

Examples

This sample input card does not activate Def_Grad output to the EXODUS II file:

```
Viscoplastic Def_Grad Tensor = no
```

Technical Discussion

Please see complete discussion in Schunk, et. el. (2001).

References

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

Lagrangian Convection

```
Lagrangian Convection = {yes | no}
```

Description / Usage

In deformable solids with a Lagrangian mesh, convection in the stress-free state can be mapped to the deformed configuration; this variable stores the velocity vectors of this solid motion (see *Convective Lagrangian Velocity* card). This variable is called **VL1, VL2, VL3** in the output EXODUS II file; the integer values **1, 2** and **3** denote coordinate directions.

The permissible values for this postprocessing option are:

yes	Calculate the Lagrangian convection and store as nodal variable velocity fields in the output EXODUS II file.
no	Do not calculate the Lagrangian convection.

Examples

Following is a sample card:

```
Lagrangian Convection = no
```

Technical Discussion

This option only applies to *Mesh Motion* type of *LAGRANGIAN*.

References

No References.

Normal and Tangent Vectors

```
Normal and Tangent Vectors = {yes | no}
```

Description / Usage

This option allows one to write the values of the normal and tangent vectors used in rotating the mesh and momentum equations as nodal variables to the output EXODUS II file. In two-dimensional problems, the normal and tangent vectors are saved as **N1, N2, N3** and **T1, T2, T3** in the output EXODUS II file; in two dimensions these vectors are calculated at all the nodes. In three-dimensional problems, the normal and tangent vectors are saved as **N1, N2, N3, TA1, TA2, TA3,** and **TB1, TB2, TB3**; in three dimensions, these vectors only exist at nodes with rotation specifications, and the vectors correspond to the rotation vectors chosen by the *ROT Specifications* for the given node (see description for *ROT* cards). Thus in three-dimensional problems, vectors are not necessarily saved for every node, nor do the vectors necessarily correspond to the normal, first tangent, and second tangent, respectively.

The permissible values for this postprocessing option are:

yes	Calculate the vectors and store as nodal variables in the output EXODUS II file.
no	Do not calculate the vectors.

Examples

The following sample card produces no output to the EXODUS II file:

```
Normal and Tangent vectors = no
```

Technical Discussion

This option is mostly used to debug three-dimensional meshes for full threedimensional ALE mesh motion. The tangent fields in 3D should be smooth across the surfaces, and *Goma* takes many steps to make them so. The surface normal crossed into any vector that is different will produce one tangent vector. Then the normal crossed (viz. cross product of two vectors) with the first tangent will produce a second tangent vector. Because the surface tangent basis fields are not unique, they must be uniform over a surface when the rotated Galerkin weighted residuals are formed (see description for *ROT* cards). Imperfections or defects in the mesh can lead to nonsmooth fields.

References

GT-018.1: ROT card tutorial, January 22, 2001, T. A. Baer

Error ZZ Velocity

```
Error ZZ velocity = {yes | no}
```

Description / Usage

This option has been disabled for lack of use, but originally allowed the user to compute a posteriori error estimates in the velocity field from the Zienkiewicz-Zhu energy norm error measure.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

Error ZZ Heat Flux

```
Error ZZ heat flux = {yes | no}
```

Description / Usage

This option has been disabled for lack of use, but originally allowed the user to compute a posteriori error estimates in the computed temperature/energy flux field from the Zienkiewicz-Zhu energy norm error measure.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

Error ZZ Pressure

```
Error ZZ pressure = {yes | no}
```

Description / Usage

This option has been disabled for lack of use, but originally allowed the user to compute a posteriori error estimates in the pressure field from the Zienkiewicz-Zhu energy norm error measure.

Examples

No Examples.

Technical Discussion

No Discussion.

References

No References.

User-Defined Post Processing

```
User-Defined Post Processing = {yes | no} <float_list>
```

Description / Usage

This option enables user-defined postprocessing options in *Goma*. An arbitrary number of floating point constants can be loaded to use in the user-defined subroutine user_post (user_post.c). This variable is called **USER** in the output EXODUS II file and can be contoured or processed just like any other nodal variable in a postprocessing visualization package.

yes	Calculate and write the user-defined postprocessing variable to the output EXODUSII file.
no	Do not calculate the user-defined postprocessing.
<float_list>	An arbitrary number (including zero) of floating point numbers, which can be accessed in file user_post

Examples

Consider the following sample input card:

```
User-Defined Post Processing = yes 100.
```

Suppose you would like to contour the speed of a fluid in a two-dimensional problem using this card, with your intent being to multiply the calculated value by a factor of 100.0 for some unit conversion or something. You must add

```
post_value = param[0]*sqrt(fv->v[0]*fv->v[0] + fv->v[1]*fv->*v[1]) ;
```

to user_post.c. Note also that you have to comment out the error handler line just above the location you enter the post_value code. The comments in the routine help guide you through the process.

Technical Discussion

See the function user_post in user_post.c.

Porous Saturation

```
Porous Saturation = {yes | no}
```

Description / Usage

In partially saturated porous media, the saturation represents the volume fraction of the pore space that is filled with liquid. If this option is selected, then the saturation level (an auxiliary variable) is included in the output EXODUS II file. This variable is called **SAT** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the porous saturation and write to the output EXODUS II file.
no	Do not calculate the porous saturation.

Examples

This sample input card turns off writing of saturation to the EXODUS II file:

```
Porous Saturation = no
```

Technical Discussion

No Discussion.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Total Density of Solvents in Porous Media

```
Total density of solvents in porous media = {yes | no}
```

Description / Usage

This post processing option can be used to trigger the computation and output of the total density of solvents as a nodal field variable to the output EXODUS II file. Three nodal variables are written, **Rho_Total_Liq, Rho_Total_air** and **Rho_Total_solid**. The mathematical details are given below in the technical discussion. This option applies to media types of *POROUS_SATURATED, POROUS_UNSATURATED*, and *POROUS_TWO_PHASE* (see *Media Type* card). The options are:

yes	Calculate and write the total solvent densities as a postprocessing variable to the output EXODUSII file.
no	Do not calculate the total solvent densities.

Examples

```
Total density of solvents in porous media = yes
```

This card will result in the calculation and output of the mixture density of solvent (viz., phase mixture of liquid solvent in vapor form, liquid form, and the form adsorbed in the solid skeleton for partially saturated porous flows). The form of that mixture density is given in the technical discussion.

Technical Discussion

In saturated flow cases, viz. for *Media Type* selection *POROUS_SATURATED*, the total solvent density is

$$\rho_T = \rho_l \phi$$

where ρ_1 is the pure liquid density and ϕ is the porosity. Here we have assumed that no liquid solvent is adsorbed into the solid struts (currently the assumption used throughout *Goma*).

For partially saturated flows, viz. for *Media Type* selection *POROUS_UNSATURATED* or *POROUS_TWO_PHASE*, the total density is given by

$$\rho_T = \rho_l \phi S X_{ls} + \rho_{gv}(1 - S)\phi$$

where ρ_{gv} is the density of solvent vapor in the total gas-solvent vapor mixture (see *Density of solvents in gas phase in porous media card*), S is the saturation (see *Porous Saturation* card), and χ_{ls} is the volume fraction of solvent in liquid phase (including any condensed species component). When calculating the total density of the liquid (**Rho_Total_liq**), the liquid vapor density comes from a Kelvin vapor-liquid equilibrium relation. The total density of the gas phase (**Rho_Total_gas**) will use a vapor density fro air and a volume fraction of zero (0) since air is insoluble. The total density of the solid in the gas (**Rho_Total_solid**) is zero (0).

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Density of Solvents in Gas Phase in Porous Media

```
Density of solvents in gas phase in porous media = {yes | no}
```

Description / Usage

This post processing option can be used to trigger the computation and output of the density of solvents in the gas phase only, including the volume occupied by the assumed insoluble gas (e.g. air), as a nodal field variable to the output EXODUS II file. The nodal variables are called **RhoSolv_g_liq, RhoSolv_g_air** and **RhoSolv_g_solid**. The mathematical details are given below in the technical discussion. This option applies to media types of *POROUS_UNSATURATED*, and *POROUS_TWO_PHASE* (see *Media Type* card). The options are:

yes	Calculate and write the gas phase solvent density as a postprocessing variable to the output EXODUSII file.
no	Do not calculate the total solvent density.

Examples

The following input card turns off writing solvent densities to the EXODUS II file:

```
Density of solvents in gas phase in porous media = no
```

Technical Discussion

The air and solid components are insoluble in the gas phase so the **RhoSolv_g_air** and **RhoSolv_g_solid** variables will be zero. The gas-density of liquid solvents **(RhoSolv_g_liq)** is determined from the vapor-liquid equilibrium relationship at a liquid-vapor meniscus. Specifically,

$$\rho_{gv} = \frac{M_w p_v}{RT}$$

where M_w is the average molecular weight of solvents in the mixture, R is the ideal gas constant, T is the temperature, and ρ_v is the equilibrium vapor pressure. Note that this vapor pressure can be affected by local meniscus curvature through the Kelvin equation (cf. Schunk, 2002 and *Porous Vapor Pressure* card).

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Density of Liquid Phase in Porous Media

```
Density of liquid phase in porous media = {yes | no}
```

Description / Usage

This post processing option can be used to trigger the computation and output of the density of solvents in the liquid phase only, averaged over the mixture, as a nodal field variable to the output EXODUS II file. The nodal variable is called **Rho_Liq_Phase**. The mathematical details are given below in the technical discussion. This option applies to media types of *POROUS_SATURATED, POROUS_UNSATURATED*, and *POROUS_TWO_PHASE* (see *Media Type* card). The options are:

yes	Calculate and write the density of solvent in the liquid phase as a postprocessing variable to the output EXODUSII file.
no	Do not calculate the liquid solvent density.

Examples

An example of an input card which activates writing of the density to the EXODUS II file is:

```
Density of liquid phase in porous media = yes
```

Technical Discussion

In liquid-saturated flow cases, viz. for *Media Type* selection *POROUS_SATURATED*, the total solvent density in the liquid phase is

$$\rho_T = \rho_l \phi$$

where ρ_l is the pure liquid density and ϕ is the porosity. Here we have assumed that no liquid solvent is adsorbed into the solid struts (currently the assumption used throughout *Goma*).

For partially saturated flows, viz. for *Media Type* selection *POROUS_UNSATURATED* or *POROUS_TWO_PHASE*, the density of solvent in the liquid phase only is given by

$$\rho_T = \rho_l \phi S$$

where S is the saturation (see *Porous Saturation* card). Compare this with the quantity computed with the *Total density of solvents in porous media* card.

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Gas Phase Darcy Velocity in Porous Media

```
Gas phase Darcy velocity in porous media = {yes | no}
```

Description / Usage

This post-processing option will lead to the explicit calculation and storage of the Darcy velocity components in the gas phase, viz. the velocity of the gas phase due to gas-phase pressure gradients. This option is only available for *POROUS_TWO_PHASE* media types (cf. *Media Type* card). The velocity components appear in the output EXODUS II file as the nodal variables **Darcy_Vel_g_0, Darcy_Vel_g_1** and **Darcy_Vel_g_2**.

The permissible values for this postprocessing option are:

yes	Calculate the gas-phase Darcy velocity components and write to the output EXODUSII file.
no	Do not calculate the gas phase velocity components.

Examples

This input example turns on calculation of the gas phase velocity components:

```
Gas phase Darcy velocity in porous media =yes
```

Technical Discussion

The gas-phase Darcy velocity is given by the extended Darcy law, which accounts for the relative reduced flow due to the presence of another phase, viz.

$$\boldsymbol{F}_g = \rho_g \boldsymbol{v}_g = -\frac{\rho_g k k_g}{\mu_g}(\nabla p_g - \rho_g \boldsymbol{g})$$

Here ν_g represents the Darcy flux, or Darcy velocity, in the gas phase, k is the permeability of the porous medium, k_g is the relative permeabilities for the gas and liquid phases respectively, μ_g are the gas viscosity, p_g is the pressure in the gas phase, and g is the gravitational force vector. ρ_g is the density of the gas phase and is equal to the sum of the partial densities of air and solvent vapor,

$$\rho_g = \rho_{gv} + \rho_{ga} \quad .$$

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Liquid Phase Darcy Velocity in Porous Media

```
Liquid phase Darcy velocity in porous media = {yes | no}
```

Description / Usage

This post-processing option will lead to the explicit calculation and storage of the Darcy velocity components in the liquid phase, viz. the velocity of the liquid phase due to liquid phase pressure gradients. This option is available for all porous media types (cf. *Media Type* card). The velocity components appear in the output EXODUS II file as the nodal variables **Darcy_Vel_l_0, Darcy_Vel_l_1** and **Darcy_Vel_l_2**.

The permissible values for this postprocessing option are:

yes	Calculate the liquid-phase Darcy velocity components and write to the output EXODUSII file.
no	Do not calculate the liquid phase velocity components.

Examples

This input example turns on calculation of the liquid phase velocity components:

```
Liquid phase Darcy velocity in porous media = yes
```

Technical Discussion

The liquid-phase Darcy velocity is given by the extended Darcy law, which accounts for the relative reduced flow due to the presence of another phase, viz.

$$\boldsymbol{F}_l = \rho_{lw}\boldsymbol{v}_l = -\frac{\rho_l k k_l}{\mu_l}(\nabla p_l - \rho_l \boldsymbol{g})$$

Here v_l represents the Darcy flux, or Darcy velocity, in the gas phase, k is the permeability of the porous medium, k_l is the relative permeabilities for the liquid and liquid phases respectively, μ_l are the liquid viscosity, p_l is the pressure in the liquid phase, and g is the gravitational force vector. ρ_l is the density of the liquid phase.

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Capillary Pressure in Porous Media

```
Capillary pressure in porous media = {yes | no}
```

Description / Usage

In partially saturated porous media, the capillary pressure is the difference between the gas and liquid pressures. This option only takes affect for *POROUS_TWO_PHASE* and *POROUS_UNSATURATED* media types (see *Media Type* card). This variable is called **PC** in the output EXODUS II file.

The permissible values for this postprocessing option are:

yes	Calculate the capillary pressure and write to output EXODUS II file.
no	Do not calculate the capillary pressure.

Examples

This is a sample input card to activate calculation of capillary pressure:

```
Capillary pressure in porous media = yes
```

Technical Discussion

The capillary pressure is a critical variable for partially saturated porous media, and is in fact the dependent variable for unsaturated (not two-phase) flows for which the gasphase pressure is taken as constant. It is simply defined as

$$p_c = p_g - p_l$$

As such, positive capillary pressures imply liquid phase pressure being greater than gas phase pressure. Because liquid phase saturation strongly correlates to capillary pressure, this current quantity is a good indicator of the level of liquid inventory in smaller pores in the skeleton relative to large pores. Contouring this quantity can give some indication of the level of suction exerted on the porous-skeleton, which is relevant when the skeleton is taken as deformable.

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

Grid Peclet Number in Porous Media

```
Grid Peclet Number in porous media = {yes | no}
```

Description / Usage

This option triggers the computation and output of the so-called grid-level Peclet number as a nodal variable in the output EXODUS II file. It appears as a nodal variable called **Por_Grid_Peclet**. This quantity gives the user a measure of advective transport relative to diffusive transport in a porous medium, and is strongly correlated to the steepness of a saturation front. This quantity is actually used to scale the formulation which employs the streamline upwind Petrov-Galerkin method for stabilizing the equations for partially saturated flow. This option only applies for unsaturated media and only for the *SUPG* option on the *Porous Weight Function* card.

The permissible values for this postprocessing option are:

yes	Compute the grid-level Peclet Number and write to output EXODUS II file.
no	Do not calculate the grid-level Peclet Number.

Examples

This is a sample input card to activate calculation of the Peclet Number:

```
Grid Peclet Number in porous media = yes
```

Technical Discussion

See discussion for the *Porous Weight Function* card.

References

GTM-029.0: SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

SUPG Velocity in Porous Media

```
SUPG Velocity in porous media = {yes | no}
```

Description / Usage

Used to specify use of effective velocities in SUPG formulations for porous media. It is written to the output EXODUS II file as nodal variable **U_supg_porous**.

The permissible values for this postprocessing option are:

yes	Calculate and write the effective velocity components as a postprocessing variable to the output EXODUSII file.
no	Do not calculate the effective velocity components.

Examples

This is a sample input card to activate calculation of SUPG Velocity:

```
SUPG Velocity in porous media = yes
```

Technical Discussion

No Discussion.

References

GTM-029.0: SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

Vorticity Vector

```
Vorticity Vector = {yes | no}
```

Description / Usage

This option allows the user to output the vorticity vector to the output EXODUS II file. It applies to problems with the fluid momentum equations (see *EQ = momentum** cards). The output nodal variables are named **VORTX, VORTY, VORTZ**.

The permissible values for this postprocessing option are:

yes	Calculate the vorticity vectors and store in the output EXODUS II file.
no	Do not calculate the vorticity vectors.

Examples

This example card requests that vorticity vectors be written to the EXODUS II file:

```
Vorticity Vector = yes
```

Technical Discussion

The vorticity vector function, $\underset{\sim}{\omega}$, is defined in terms of the velocity $\underset{\sim}{v}$ as:

$$\underset{\sim}{\omega} = \nabla \times \underset{\sim}{v}$$

1.4.15 Post Processing Fluxes and Data

By *Post-processing Fluxes* we mean area-integrated fluxes that can be calculated for any flux quantity across any surface demarcated by a side set. The area-integrated flux is in fact a total flow rate across the boundary. Examples include heat-flow, total force of a liquid on a surface, and species flow (both diffusive and convective). The integrated flux quantities are output to a specified file at each time step, together with the time stamp and the convective and diffusive components. This capability is useful for extracting engineering results from an analysis, and can further be used to as an objective function evaluator for engineering optimization problems (cf. *Post Processing Flux Sensitivities* card below).

Post Processing Data output can be used to produce spatial {*value, x, y, z*} sets on a specified side set of any primitive variable in the problem, viz. pressure, x-component of velocity, etc. The quantity *value* is the value of the variable at a node in the side set, and x, y, z are the coordinates of the node.

Post Processing Fluxes

```
Post Processing Fluxes =
```

Description / Usage

This card indicates that the cards between this and an *END OF FLUX* card are to be read and processed. If this card (*Post Processing Fluxes*) is not present, *FLUX* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Fluxes =
```

Technical Discussion

No Discussion.

References

No References.

FLUX

```
FLUX = {flux_type} <bc_id> <blk_id> <species_id> <file_name> [profile]
```

Description / Usage

FLUX cards are used to calculate the integrated fluxes of momentum, mass, energy, etc. on a specified side set during post processing. As many of these FLUX cards as desired can be input to *Goma* to direct the calculations. For example, multiple cards may be used to compute a particular flux, e.g. FORCE_NORMAL, on different side sets or different fluxes on the same side set. Cards with identical fluxes and identical side sets could be used to output the flux calculations to different files. Definitions of the input parameters are:

{flux_type}	A keyword that can have any one of the following values: • **FORCE_NORMAL** • **FORCE_TANGENT1** • **FORCE_TANGENT2** • **FORCE_X** • **FORCE_Y** • **FORCE_Z** • **VOLUME_FLUX** • **SPECIES_FLUX** • **HEAT_FLUX** • **TORQUE** • **AVERAGE_CONC** • **SURF_DISSIP** • **AREA** • **VOL_REVOLUTION** • **PORE_LIQ_FLUX** • **CHARGED_SPECIES_FLUX** • **CURRENT_FICKIAN** • **CURRENT** • **ELEC_FORCE_NORMAL** • **ELEC_FORCE_TANGENT1** • **ELEC_FORCE_TANGENT2** • **ELEC_FORCE_X** • **ELEC_FORCE_Y** • **ELEC_FORCE_Z** • **NET_SURF_CHARGE** • **ACOUSTIC_FLUX_NORMAL** • **ACOUSTIC_FLUX_TANGENT1** • **ACOUSTIC_FLUX_TANGENT2** • **ACOUSTIC_FLUX_X** • **ACOUSTIC_FLUX_Y** • **ACOUSTIC_FLUX_Z** For every request, the integral of the diffusive portion followed by that of the convective portion over the requested boundary will be appended to the specified file. If the convective flux is not applicable (i.e.for flux_types **VOLUME_FLUX, TORQUE, AVERAGE_CONC** and **AREA**), the second quantity will be zero. In all cases the area of the face (covered by the entire side set) and the time value are also output.
<bc_id>	The boundary flag identifier, an integer associated with the boundary location (side set in EXODUS II) in the problem domain on which the integrated flux is desired.
<blk_id>	An integer that designates the mesh block (material) from which the flux integral should be performed. This has implications on internal boundaries.
<species_id>	An integer that identifies the species number if an integrated species flux is requested.
<file_name>	A character string corresponding to a file name into which these fluxes should be printed.
[profile]	Inclusion of the optional string "profile' to this card will cause the coordinates (x,y,z), the diffusive integrand, and the convective integrand at each integration point to be printed to the file designated above. You can, for example, print out a pressure distribution used to compute a force.

Examples

The following example shows a sample input deck section that requests five such integrated fluxes:

```
Post Processing Fluxes =
```

```
FLUX = FORCE_X 5 1 0 side5.out
```

```
FLUX = FORCE_Y 5 1 0 side5prof.out   profile
```

```
FLUX = FORCE_NORMAL 8 1 0 side8.out
```

```
FLUX = FORCE_TANGENT1 8 1 0 side8.out
```

```
FLUX = VOLUME_FLUX 8 1 0 side8.out
```

```
END OF FLUX
```

Technical Discussion

The permissible flux types are those listed in file mm_post_def.h for struct *Post_Processing_Flux_Names, pp_flux_names* being one variable of this struct type.

The flux integrations are carried out as follows:

FLUX	DIFFUSIVE FLUX	CONVECTIVE FLUX
FORCE_NORMAL	$\int$ n • $\underline{T}$ • ndA	$\int \rho$ n • (v - v_m) v • ndA
FORCE_TANGENT1	$\int_1$ • $\underline{T}$ • ndA	$\int \rho\, t_1$ (v - v_m) v • ndA
FORCE_TANGENT2	$\int_2$ • $\underline{T}$ • ndA	$\int \rho\, t_2$ (v - v_m) v • ndA
FORCE_X	$\int$ i • $\underline{T}$ • ndA	$\int \rho$ i (v - v_m) v • ndA
FORCE_Y	$\int$ j • $\underline{T}$ • ndA	$\int \rho$ j (v - v_m) v • ndA
FORCE_Z	$\int$ k • $\underline{T}$ • ndA	$\int \rho$ k (v - v_m) v • ndA
VOLUME_FLUX	$\int$ n • (v - v_m) dA	for ARBITRARY mesh motion.
	$\int$ n • ddA	for LAGRANGIAN mesh motion.
SPECIES_FLUX	$\int$ (-$D_j n$ • Δ cj) dA	$\int \rho$ n • (v - v_m) cjdA
HEAT_FLUX	$\int$ (-kn • Δ T) dA	$\int \rho$ CpTn • (v - v_m) dA
TORQUE	$\int re_r \times$ ($\underline{T}$ • n) dA	
AVERAGE_CONC	$\int$ cjdA	
SURF_DISSIP	$\int \sigma\Delta$ v • (ζ - nn) dA	
AREA	$\int$ dA	
VOL_REVOLUTION	$\int \frac{1}{2} \frac{r}{\sqrt{1+(dr/dz)^2}}$ dA	
POR_LIQ_FLUX	$\int$ n • ($\rho_l v_{darcy}$) dA	
CHARGED_SPECIES_FLUX	$\int$ (-Djn • Δ cj) dA	$\int \rho$ n • (v - v_m) cjdA
CURRENT_FICKIAN	$\int$ (-Djn • Δ cj) dA	$\int \rho$ n • (v - v_m) cjdA
PVELOCITY[1-3]	$\int$ n • pvjdA	
ELEC_FORCE_NORMAL	$\int$ n $\underline{T}_e$ • ndA	
ELEC_FORCE_TANGENT1	$\int t_1$ • $\underline{T}_e$ • ndA	
ELEC_FORCE_TANGENT2	$\int t_2$ • $\underline{T}_e$ • ndA	
ELEC_FORCE_X	$\int$ i • $\underline{T}_e$ • ndA	

continues on next page

Table 1 – continued from previous page

ELEC_FORCE_Y	$\int$ j • $\underline{T}_e$ • ndA	
ELEC_FORCE_Y	$\int$ k • $\underline{T}_e$ • ndA	
NET_SURF_CHARGE	$\int$ (-ε $\underline{n}$ • $\underline{E}$) dA	
ACOUSTIC_FLUX_NORMAL	$\int$ (-$\frac{1}{kR}$ n • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ n • ΔP_{real}) dA
ACOUSTIC_FLUX_TANGENT1	$\int$ (-$\frac{1}{kR}$ t_1 • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ t_1 • ΔP_{real}) dA
ACOUSTIC_FLUX_TANGENT2	$\int$ (-$\frac{1}{kR}$ t_2 • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ t_2 • ΔP_{real}) dA
ACOUSTIC_FLUX_X	$\int$ (-$\frac{1}{kR}$ i • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ i • ΔP_{real}) dA
ACOUSTIC_FLUX_Y	$\int$ (-$\frac{1}{kR}$ j • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ j • ΔP_{real}) dA
ACOUSTIC_FLUX_Z	$\int$ (-$\frac{1}{kR}$ k • ΔP_{imag}) dA	$\int$ (-$\frac{1}{kR}$ k • ΔP_{real}) dA

The SURF_DISSIP card is used to compute the energy dissipated at a surface by surface tension (Batchelor, 1970). The VOL_REVOLUTION card is used in axi- symmetric problems to compute the volume swept by revolving a surface around the axis of symmetry (z-axis). Even though every flux card results in the area computation of the side set, the AREA card is used when the area of a surface is part of an augmenting condition. The POR_LIQ_FLUX term is valid only for saturated media and the Darcy velocity is defined by ν_{darcy} = (κ / μ) Δ p_{liq} . For the more general case, refer to the *POROUS_LIQ_FLUX_CONST* boundary condition.

References

Batchelor, JFM, 1970. ….. need to fill-in reference; get from RBS

For information on using flux calculations as part of augmenting conditions, see:

SAND2000-2465: Advanced Capabilities in Goma 3.0 - Augmenting Conditions, Automatic Continuation, and Linear Stability Analysis, I. D. Gates, I. D., Labreche, D. A. and Hopkins, M. M. (January 2001).

END OF FLUX

```
END OF FLUX
```

Description / Usage

This card is used to denote the end of a set of *FLUX* cards and is only used when the *Post Processing Fluxes* card is present and one or more *FLUX* cards are specified.

Examples

A simple example of using this card in context is shown below.

```
Post Processing Fluxes =
FLUX = FORCE_X 5 1 0 side5.out
END OF FLUX
```

Technical Discussion

No Discussion.

References

No References.

Post Processing Data

```
Post Processing Data =
```

Description / Usage

This card indicates that the cards between this and an *END OF DATA* card are to be read and processed. If this card (*Post Processing Data*) is not present, *DATA* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Data =
```

Technical Discussion

No Discussion.

References

No References.

DATA

```
DATA = {data_type} <bc_id> <blk_id> <species_id> <file_name>
```

Description / Usage

A DATA card directs *Goma* to output the indicated primitive variable on a specified node set. As many of these DATA cards as desired can be input to *Goma*. For example, multiple cards may be used to output a particular variable, e.g. VELOCITY1, on different node sets or different variables on the same node set. Cards with identical variables and identical node sets could be used to output the variables to different files. Definitions of the input parameters are as follows:

{data_type}	A keyword that can have any one of the following primitive values: • **VELOCITY[1-3]** • **TEMPERATURE** • **MASS_FRACTION** • **MESH_DISPLACEMENT[1-3]** • **SURFACE** • **PRESSURE** • **POLYMER_STRESS[1-3][1-3]** • **SOLID_DISPLACEMENT[1-3]** • **VELOCITY_GRADIENT[1-3][1-3]** • **VOLTAGE** • **FILL** • **SHEAR_RATE** • **PVELOCITY[1-3]** • **POLYMER_STRESS[1-3][1-3]_[1-7}** • **SPECIES_UNK[0-29]** • **VolFracPh_[0-4]** • **POR_LIQ_PRES** • **POR_GAS_PRES** • **POR_PORSITY** • **POR_SATURATION** • **VORT_DIR[1-3]** • **VORT_LAMBDA** Each request will result in the point coordinates and the quantity value being printed to the specified file.
<bc_id>	The boundary flag identifier, an integer associated with the boundary location (node set in EXODUS II) in the problem domain on which the quantity is desired.
<blk_id>	An integer that designates the mesh block (material) from which the variable value should be taken. This has implications for discontinuous variables on internal boundaries.
<species_id>	An integer that identifies the species number if a species variable is requested.
<file_name>	A character string corresponding to a file name into which the data should be printed.

Examples

A simple example of using this card in context is shown below.

```
Post Processing Data =
DATA = VELOCITY2 1 1 0 data.out
END OF DATA
```

Technical Discussion

If a fixed mesh or a subparametric mesh problem is being solved, the point coordinates printed to the file will be the undeformed coordinates.

References

No References.

END OF DATA

```
END OF DATA
```

Description / Usage

This card is used to denote the end of a set of *DATA* cards and is only used when the *Post Processing Data* card is present and one or more *DATA* cards are specified.

Examples

A simple example of using this card in context is shown below.

```
Post Processing Data =
DATA = VELOCITY2 1 1 0 data.out
END OF DATA
```

Technical Discussion

No Discussion.

References

No References.

Post Processing Flux Sensitivities

```
Post Processing Flux Sensitivities =
```

Description / Usage

This card indicates that the cards between this and an *END OF FLUX_SENS* card are to be read and processed. If this card (*Post Processing Flux Sensitivities*) is not present, *FLUX_SENS* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Flux Sensitivities =
```

Technical Discussion

No Discussion.

References

No References.

FLUX_SENS

```
FLUX_SENS = {flux_type} <bc_id> <blk_id> <species_id> {sens_type}
<sens_id> <sens_flt> <file_name>
```

Description / Usage

FLUX_SENS cards request the calculation of the sensitivity of an integrated flux with respect to a boundary condition or material parameter. As many FLUX_SENS cards as desired can be input to *Goma*. Definitions of the input parameters are as follows:

{flux_type}	A keyword that can have any one of the following values: • **FORCE_NORMAL** • **FORCE_TANGENT1** • **FORCE_TANGENT2** • **FORCE_X** • **FORCE_Y** • **FORCE_Z** • **VOLUME_FLUX** • **SPECIES_FLUX** • **HEAT_FLUX** • **TORQUE** • **AVERAGE_CONC** • **SURF_DISSIP** • **AREA** • **VOL_REVOLUTION** • **PORE_LIQ_FLUX** • **CHARGED_SPECIES_FLUX** • **CURRENT_FICKIAN** • **CURRENT** For every request, the specified sensitivity of the integrated diffusive flux followed by that of the convective portion over the requested boundary will be appended to the specified file. If the convective flux is not applicable (cf. FLUX card), the second quantity will be zero. In all cases the area of the face (covered by the entire side set) and the time value are also output.
<bc_id>	The boundary flag identifier, an integer associated with the boundary location (node set in EXODUS II) in the problem domain on which the integrated flux sensitivity is desired.
<blk_id>	An integer that designates the mesh block (material) from which the flux sensitivity integral should be performed. This has implications on internal boundaries.
<species_id>	An integer that identifies the species number if an integrated species flux sensitivity is requested.
{sens_type}	A two-character entry that identifies the sensitivity type, where: • **BC** denotes a sensitivity w.r.t. to a boundary condition parameter. • **MT** denotes a sensitivity w.r.t. to a material property parameter.
<sens_id>	An integer that identifies the sensitivity. If **BC** is specified for {sens_type}, then this value is the BC card number. If **MT** is specified for {sens_type}, this value is the material number.
<sens_flt>	A floating point number whose meaning is also dependent on the selection of {sens_type}. If **BC** is specified, this value is the BC data float number. If **MT** is specified, this value is the material property tag.
<file_name>	A character string corresponding to a file name into which these fluxes should be printed.

Examples

Here is an example input deck:

```
Post Processing Flux Sensitivities =
FLUX_SENS = VOLUME_FLUX 1 1 0 BC 5 3 flux_sens.out
END OF FLUX_SENS
```

Technical Discussion

Currently, the flux sensitivities do not account for the implicit sensitivity of material properties. That is, $dFORCE_X$ / dBC_{float} does not include a contribution from $d\mu$ / dBC_{float} , but $dFORCE_X$ / $dMT_{property}$ should be done correctly. In addition, sensitivities of integrated fluxes in solid materials have not been implemented yet.

NOTE: In order to compute flux sensitivities with respect to Dirichlet boundary condition floats, the boundary conditions need to use the *residual method* in the input file as described in the *Boundary Condition Specification* introduction, i.e. the optional parameter should be set to 1.0.

References

No References.

END OF FLUX_SENS

```
END OF FLUX_SENS
```

Description / Usage

This card is used to denote the end of a set of *FLUX_SENS* cards and is only used when the *Post Processing Flux Sensitivities* card is present and one or more *FLUX_SENS* cards are specified.

Examples

A simple example of using this card in context is shown below.

```
Post Processing Flux Sensitivities =
FLUX_SENS = VOLUME_FLUX 1 1 0 BC 5 3 flux_sens.out
END OF FLUX_SENS
```

Technical Discussion

No Discussion.

References

No References.

Post Processing Data Sensitivities

```
Post Processing Data Sensitivities =
```

Description / Usage

This card indicates that the cards between this and an *END OF DATA_SENS* card are to be read and processed. If this card (*Post Processing Data Sensitivities*) is not present, *DATA_SENS* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Data Sensitivities =
```

Technical Discussion

No Discussion.

References

No References.

DATA_SENS

```
DATA = {data_type} <bc_id> <blk_id> <species_id> {sens_type} <sens_id>
<sens_flt> <file_name>
```

Description / Usage

As many of these *DATA_SENS* cards as desired can be input to direct *Goma* to print the sensitivity of a specified variable with respect to a boundary condition or material parameter on a specified node set. Definitions of the input parameters are as follows:

{data_type}	A keyword that can have any one of the following values: • **VELOCITY[1-3]** • **TEMPERATURE** • **MASS_FRACTION** • **MESH_DISPLACEMENT[1-3]** • **SURFACE** • **PRESSURE** • **POLYMER_STRESS[1-3][1-3]** • **SOLID_DISPLACEMENT[1-3]** • **VELOCITY_GRADIENT[1-3][1-3]** • **VOLTAGE** • **FILL** • **SHEAR_RATE** • **PVELOCITY[1-3]** • **VolFracPh_[0-4]** • **POR_LIQ_PRES** • **POR_GAS_PRES** • **POR_PORSITY** • **POR_SATURATION** • **VORT_DIR[1-3]** • **VORT_LAMBDA** Each request will result in the point coordinates and the specified sensitivity value being printed to the specified file.
<bc_id>	The boundary flag identifier, an integer associated with the boundary location (node set in EXODUS II) in the problem domain on which the variable sensitivity is desired.
<blk_id>	An integer that designates the mesh block (material) from which the variable sensitivity should be taken. This has implications for discontinuous variables on internal boundaries.
<species_id>	An integer that identifies the species number if a species sensitivity is requested.
{sens_type}	A two-character entry that identifies the sensitivity type, where: • **BC** denotes a sensitivity w.r.t. to a boundary condition parameter. • **MT** denotes a sensitivity w.r.t. to a material property parameter.
<sens_id>	An integer that identifies the sensitivity. If **BC** is specified for {sens_type}, then this value is the BC card number. If **MT** is specified for {sens_type}, this value is the material number.
<sens_flt>	A floating point number whose meaning is also dependent on the selection of {sens_type}. If **BC** is specified, this value is the BC data float number. If **MT** is specified, this value is the material property tag.
<file_name>	A character string corresponding to a file name into which the data should be printed.

Examples

The following example shows a sample input deck section with three data requests:

```
Post Processing Data Sensitivities =
DATA_SENS = VELOCITY2 1 1 0 BC 5 3 data_sens.out
DATA_SENS = VELOCITY1 6 1 0 BC 5 3 data_sens.out
DATA_SENS = VELOCITY1 6 1 0 BC 4 0 data_sens.out
END OF DATA_SENS
```

Technical Discussion

NOTE: In order to compute data sensitivities with respect to dirichlet boundary condition floats, the boundary conditions need to be "soft" set in the input file, i.e. the optional parameter should be set to 1.0.

References

No References.

END OF DATA_SENS

```
END OF DATA_SENS
```

Description / Usage

This card is used to denote the end of a set of *DATA_SENS* cards and is only used when the *Post Processing Data Sensitivities* card is present and one or more *DATA_SENS* cards are specified.

Examples

A simple example of using this card in context is shown below.

```
Post Processing Data Sensitivities =
DATA_SENS = VELOCITY2 1 1 0 BC 5 3 data_sens.out
END OF DATA_SENS
```

Technical Discussion

No Discussion.

References

No References.

1.4.16 Post Processing Particle Traces

This option enables the calculation of particle trajectories and computed quantities along the trajectories. The coordinates of the trajectory starting point and time-stepping parameters are input using the cards in this section. The computation of quantities along the trajectories and their subsequent output is controlled through the *usr_ptracking* routine in user_post.c.

Post Processing Particle Traces

```
Post Processing Particle Traces =
```

Description / Usage

This card indicates that the cards between this and an *END OF PARTICLES* card are to be read and processed. If this card (*Post Processing Particle Traces*) is not present, *PARTICLE* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Particle Traces =
```

Technical Discussion

No Discussion.

References

No References.

PARTICLE

```
PARTICLE = <float_list> <file_name>
```

Description / Usage

Each *PARTICLE* card represents a separate particle trajectory. As many of these cards as desired can be input to direct *Goma* to compute a particle trajectory.

There are eleven values to specify in the <float_list>; definitions of the input parameters are as follows:

<float1>	**xpt**, the X-coordinate of the origin of the trajectory.
<float2>	**ypt**, the Y-coordinate of the origin of the trajectory.
<float3>	**zpt**, the Z-coordinate of the origin of the trajectory.
<float4>	**initial_time**, the start time for computing the trajectory (a value in units consistent with rest of the problem, i.e. length scale/velocity scale).
<float5>	**end_time**, the end time for computing the trajectory. The trajectory will be computed until the end time is reached or until the particle trajectory leaves the computational domain.
<float6>	**point_spacing**, the desired distance between successive points on the trajectory. The point spacing may be decreased below this value if required by the trajectory calculation but it will not exceed it.
<float7>	**mobility**, the mobility of the particle when particles with inertia are desired. Enter zero for inertia-less trajectories.
<float8>	**mass**, the mass of the particle. If the trajectory of an inertialess particle is desired, a value of 0.0 should be entered.
<float9>	**force_x**, the X-component of an external force (such as gravity) which is to be applied to the particle (with mass).
<float10>	**force_y**, the Y-component of an external force (such as gravity) which is to be applied to the particle (with mass).
<float11>	**force_z**, the Z-component of an external force (such as gravity) which is to be applied to the particle (with mass).
<file_name>	A character string corresponding to a file name into which the output should be printed.

Thus, the particle trajectory starts at the coordinates defined by **xpt, ypt,** and **zpt**. The trajectory of the particle is computed starting at a time value of **initial_time** and continuing until **end_time** is reached or until the particle exits the computational domain. The time step is adjusted so that the distance between successive points on the trajectory is, at most, equal to the **point_spacing** (it may be less if the time-stepping algorithm requires it). At each point along the trajectory, the *usr_ptracking* routine is called which the user may modify to control the output to **file_name**.

Examples

Here is an example of an input deck with 6 trajectory cards.

```
Post Processing Particle Traces =
PARTICLE = -1.8 -0.1 3.0 0 10000 0.02 0 0 0 0 0 part1.out
PARTICLE = -1.8 -0.1 3.0 0 1000 0.02 {mob1} {mass1} 0 {-f1} 0 part1.out
PARTICLE = -1.8 -0.1 3.0 0 1000 0.02 {mob2} {mass2} 0 {-f2} 0 part1.out
PARTICLE = -1.8 -0.15 3.0 0 10000 0.02 0 0 0 0 0 part2.out
PARTICLE = -1.8 -0.15 3.0 0 1000 0.02 {mob1} {mass1} 0 {-f1} 0 part2.out
PARTICLE = -1.8 -0.15 3.0 0 1000 0.02 {mob2} {mass2} 0 {-f2} 0 part2.out
END OF PARTICLES
```

Technical Discussion

The vorticity vector function, $\underline{\omega}$, is defined in terms of the velocity $\underline{v}$ as:

$$\frac{d\boldsymbol{x}}{dt} = v(\boldsymbol{x})$$

$$\boldsymbol{x}(t = t_0) = \boldsymbol{x}_0$$

Trapezoidal rule time integration is utilized with Euler prediction.

For trajectories with particle inertia (i.e. when the product of mass and mobility is greater than zero), the following evolution equation is used:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} + \omega\left(-m\frac{d^2\boldsymbol{x}}{dt^2} + \boldsymbol{f}\right)$$

where ω is the particle mobility (e.g. $1/(6\pi\mu r)$ for a sphere of radius r in a liquid of viscosity μ), m is the particle mass, and **f** is the external force vector acting on the particle.

The trajectory is computed using a coupled set of ordinary differential equations:

$$\boldsymbol{u}_1 = \boldsymbol{x}(t)$$

$$\boldsymbol{u}_2 = \frac{d\boldsymbol{x}}{dt}$$

$$\frac{d\boldsymbol{u}_1}{dt} = \boldsymbol{u}_2$$

$$m\omega\frac{d\boldsymbol{u}_2}{dt} = \boldsymbol{v} - \boldsymbol{u}_2 + \omega\boldsymbol{f}$$

$$\boldsymbol{u}_1(t = 0) = \boldsymbol{x}_0$$

$$\boldsymbol{u}_2(t = 0) = v(\boldsymbol{x}_0)$$

References

Russel, Saville, and Schowalter, Colloidal Dispersions, pp. 374-377.

END OF PARTICLES

```
END OF PARTICLES
```

Description / Usage

This card is used to denote the end of a set of *PARTICLE* cards and is only used when the *Post Processing Particle Traces* card is present and one or more *PARTICLE* cards are specified.

Examples

The *PARTICLE* card shows an example of using this card in context. Because the card has no input parameters, it always appears as

```
END OF PARTICLES
```

Technical Discussion

No Discussion.

References

No References.

1.4.17 Volumetric Integration

This option enables computation of overall integrated quantities for a specific volume. Several standard options are available and the user is permitted to define his/her own integrand to be evaluated in this section. However, the current implementation of these volumetric integrals is limited to generalized Newtonian fluids. Output is to a specified file.

Post Processing Volumetric Integration

```
Post Processing Volumetric Integration =
```

Description / Usage

This card indicates that the cards between this and an *END OF VOLUME_INT* card are to be read and processed. If this card (*Post Processing Volumetric Integration*) is not present, *VOLUME_INT* cards will be ignored.

Examples

There are no input parameters for this card, which always appears as follows:

```
Post Processing Volumetric Integration =
```

Technical Discussion

No Discussion.

References

No References.

VOLUME_INT

```
VOLUME_INT = {volume_type} <blk_id> <species_no> <file_string> [float_list]
```

Description / Usage

The VOLUME_INT card activates computation of specified volumetric integrals during post processing. As many of these VOLUME_INT cards as desired can be input to *Goma*. Definitions of the input parameters are as follows:

<volume_type>	Several choices of volumetric integral are allowed and are referenced through this parameter. The permissible values and corresponding volume integral follow: • **VOLUME**-Volume of the element block specified by <blk_id>. • **DISSIPATION**-Total viscous dissipation, τ: Δv , in the element block specified by <blk_id>. • **JOULE**-Total Joule or Ohmic heating, $\frac{1}{\sigma}$ ($\underline{J}$ • $\underline{J}$) , in the element block specified by <blk_id>. • **SPECIES_MASS**-Integral of concentration of the component specified by <species_no> in the element block specified by < blk_id>. • **MOMENTUMX, MOMENTUMY,** or **MOMENTUMZ**-Integral of appropriate component of the momentum flux $\rho\vec{v}$ over the element block <blk_id>. • **STRESS_TRACE**-Integral of the trace of the complete stress tensor (-$p\zeta + \tau$) over the element block *blk_id*. • **HEAT_ENERGY**-Integral of the sensible heat over <blk_id> (not currently implemented). • **POSITIVE_FILL, NEGATIVE_FILL**-Volume integral of region occupied by positive (negative) values of the FILL variable in element block < blk_id>. Note, for either of these cards, [float_list] is required. **NOTE for Level-Set users**: There are numerous other quantities (too-lengthy and esoteric to list here) that can be integraded vis-a-vis level set fields. Please see code. • **NEGATIVE_VX, NEGATIVE_VY, NEGATIVE_VZ**-Velocity integral in one of the three directions over just the region occupied by negative values of the FILL variable in level set problems. Note, for any of these cards, the [float_list] is required. • **POROUS_LIQ_INVENTORY**-Volume integral of bulk liquid component density (gas and liquid phase) in a porous medium. Result is a total inventory of liquid in the porous phase. • **SPEED_SQUARED**-Volume integral of the square of the speed, viz. $\underline{v}$ • $\underline{v}$. Used to measure norm of fluid kinetic energy level. Should tend to zero for a fluid at rest. • **USER**-Volume integral is supplied by the user (not currently implemented). • **SURFACE_SPECIES**-Generate locus of points which correspond to a surface of constant species concentration according to Ac_1 + Bc_2 + Cc_3 + D=0. Currently only implemented for 3D linear elements. • **LUB_LOAD**-"Volume integral" of lubrication pressure over entire mesh shell block, which is useful for computing the overall lubrication load. This is actually an area integral over the shell, thereby yielding a force.
	• **ELOADX; ELOADY; ELOADZ**-Volume integral of electric field or the gradient of the electric potential for electrostatic problems. • **RATE_OF_DEF_II**-Volume integral of the second invariant of the rate-of-deformation tensor.

Examples

Here is an example of an input deck with 3 VOLUME_INT cards.

```
Post Processing Volumetric Integration =
VOLUME_INT = VOLUME 1 0 volume.out
VOLUME_INT = SPECIES_MASS 2 3 species3.out
VOLUME_INT = NEGATIVE_FILL 1 0 fill.out 0.1
END OF VOLUME_INT
```

Technical Discussion

The volume integrations are carried out as follows:

volume_type	volume integral
VOLUME	$\int$ dV
DISSIPATION	$\int$ (-p $\zeta + \tau$) • Δ vdV
JOULE	$\int \frac{1}{\sigma}$ J • JdV
SPECIES_MASS	$\int c_j dV$
MOMENTUM_{X|Y|Z}	$\int \rho$ (i|j|k) • vdV
STRESS_TRACE	$\int$ tr(-p $\zeta + \tau$) dV
{POSITIVE|NEGATIVE}_FILL	$\int$ H(ϕ) dV
NEGATIVE_V{X|Y|Z}	$\int$ H(ϕ) {i|j|k} • vdV
POROUS_LIQ_INVENTORY	$\int$ [ρ_{gas} ϕ (1-S) + ρ_{liq} ϕ S] dV

References

No References.

END OF VOLUME_INT

```
END OF VOLUME_INT
```

Description / Usage

This card is used to denote the end of a set of *VOLUME_INT* cards and is only used when the *Post Processing Volumetric Integration* card is present and one or more *VOLUME_INT* cards are specified.

Examples

The *VOLUME_INT* card shows an example of using this card in context. Because the card has no input parameters, it always appears as

```
END OF VOLUME_INT
```

Technical Discussion

No Discussion.

References

No References.

1.5 Material Files

The material ("mat") file for *Goma* contains a description or specification of all the properties required for the multi-physics capabilities of *Goma*. A separate *.mat* file must be developed for each material present in each simulation. The *mat* file (see Figure 5) is split into seven sections: (1) Physical Properties (Section 5.1), (2) Mechanical Properties and Constitutive Equations (Section 5.2), (3) Thermal Properties (Section 5.3), (4) Electrical Properties (Section 5.4), (5) Microstructure Properties (Section 5.5), (6) Species Properties (Section 5.6), and (7) Source Terms (Section 5.7).

Each section in this chapter discusses a separate part of the material file specification and it indicates the data cards or input records that may be used, followed by the options available for each individual record (or line in the file) and the necessary input data/parameters. All input data are specified in a free field format with successive data items separated by blanks or tabs. In this version of the user's manual, a new format has been instituted in which each record is presented in a template structure. This template has eight parts: 1) a title, which is also the card name, 2) a syntax, which is enclosed in a framed box and shows the proper contents of the card, 3) a Description/Usage section, which presents the user options and descriptions of proper input records, 4) an Example, 5) a Technical Discussion to provide relevant information to help the user understand how to select from among various options or how to properly determine the desired parameters, 6) a Theory to provide an understanding of the physics and mechanics that have been implemented or are being exercised, 7) a FAQs section to present important user experience, and 8) a Reference section to identify citations and/or provide background information to the user. This is a more lengthly but a more complete form for documenting and instructing users of *Goma*.

The syntax entry denotes a unique string for each input record which *Goma* parses in the input file. All words in these unique strings are separated by a single white space and because the code parses for these exact strings, the parser becomes case sensitive. The identifying string for a particular specification is followed by an '=' character. Following this character will be all additional data for that record, if any. In the syntax box, this additional data is symbolically represented by one or more variables with some appropriate delimiters. Typically, the user will find a variable called *model_name* enclosed in **curly braces** '{}'; this would then be followed by a description of specific options for *model_name* in the Description/Usage section. The curly braces indicate a required input and that the user must select one of the offered options for *model_name*. **Required parameters**, if any, for the model option are enclosed in **angle brackets** '< >', while **optional parameters** for model_name are enclosed in **square brackets** '[ ]'. Following the '=' character, the user may use white space freely between and among the remaining parameters on the command line.

Figure 4 illustrates a typical material file. The section *headers*, e.g., "— Physical Properties", are user comments that are not processed by the input parser. In all sections of this chapter, *model_name* is a character string and *floating_point_const_list* is a list of floating point numbers of arbitrary length separated by a comma or one or more white spaces. The remainder of this chapter covers each card (line) of the material-description file in detail. For each parameter

that is not dimensionless, base units are indicated in square brackets ([ ]) at the end of the syntax line; the base units are those indicated in the Nomenclature section of this document. Empty brackets ([ ]) denote dimensionless parameters, while those without units or brackets are simply model names, other strings, or integers. Several model parameters, e.g., Diffusivity, where the model options include other than the **CONSTANT** type with a single input value, identify the units as [**varied**]. In these cases, the parameter units will be listed for the **CONSTANT** model option and the units for individual input parameters will be identified in the parameter description.

All property models will eventually have a **USER** and a **USER_GEN** option. When the former is selected, the user must add the user model to the appropriate routine in the file *user_mp.c.* This file contains a template to simplify the implementation of a model in a full-Newton context, but has the restriction that none of the models can contain a dependence on gradients of variables. For more complex models, which contain such dependencies, the user must resort to the more sophisticated mechanism that comprise the routines in *user_mp_gen.c*

A relatively new capability/model available on many of the properties is a table-lookup feature. That is, if the model is of type **TABLE**, then a linear or bilinear interpolation is used to extract the material property value from a table of numbers representing the dependence. The best way to explain this is with an example. Often times a property is dependent on temperature, or related dependent variable. If discrete data is available of the property value at various temperatures, as from a spreadsheet, then such a table can be read and with appropriate interpolation operations the property value is determined. Throughout the material property options, the reader might see aat **TABLE** option. The syntax for the input of that option is as follows:

<Property name> = TABLE *<integer1> <character_string1> [character_string2] {LINEAR | BILINEAR} [integer2] [FILE = filenm]*

Here, the integers, character strings and floats are defined as follows:

<integer1> - the number N of columns in the table. The first N-1 columns are the values of the independent variables (e.g. temperature, concentration, etc.) and the final Nth column is the property value. This number is usually 2.

<character_string1> - Required variable name for first column. Valid variable names are **TEMPERATURE, MASS_FRACTION, SPECIES, CAP_PRES, FAUX_PLASTIC,** and **LOWER_DISTANCE**. The last three are specific to the Saturation model of porous flow, the LAME Mu model, and the Lubrication Height function model, respectively. Temperature and mass fraction dependence are available in all properties with a **TABLE** option which make sense.

[character_string2] - Optional second variable name for bi-linear lookup dependence. This is exploratory.

{*LINEAR* | *BILINEAR*} - type of interpolation

[*integer2*] - species number required only for MASS_FRACTION, SPECIES, and FAUX_PLASTICITY variables.

[*FILE* = <*filenm*>] - The optional keyword 'FILE=' indicates that the table data is to be read from a separate file identified by <filenm>. Each row of the table corresponds to one variable value, and is input in free form CSV or space separated values. Note that if this 'FILE=' option is not present then the data will be read from the input material file itself following the TABLE model card. The end of the table is signaled by the keyword "END TABLE" (see example below).

Some examples are in order:

```
Lame MU = TABLE 2 FAUX_PLASTIC 0 LINEAR FILE=stress_strain_comp.txt
...
Lame MU = TABLE 2 TEMPERATURE LINEAR
1. 293
2. 300
3. 425.
END TABLE
```

Finally, before we get started, the following is an option added to allow existing Chemkin material property databases to be read in, basically obviating the need to even read the material (mat) file. The detailed description of input records provided in this chapter thus applies to the case when the Default Database is set to GOMA_MAT.

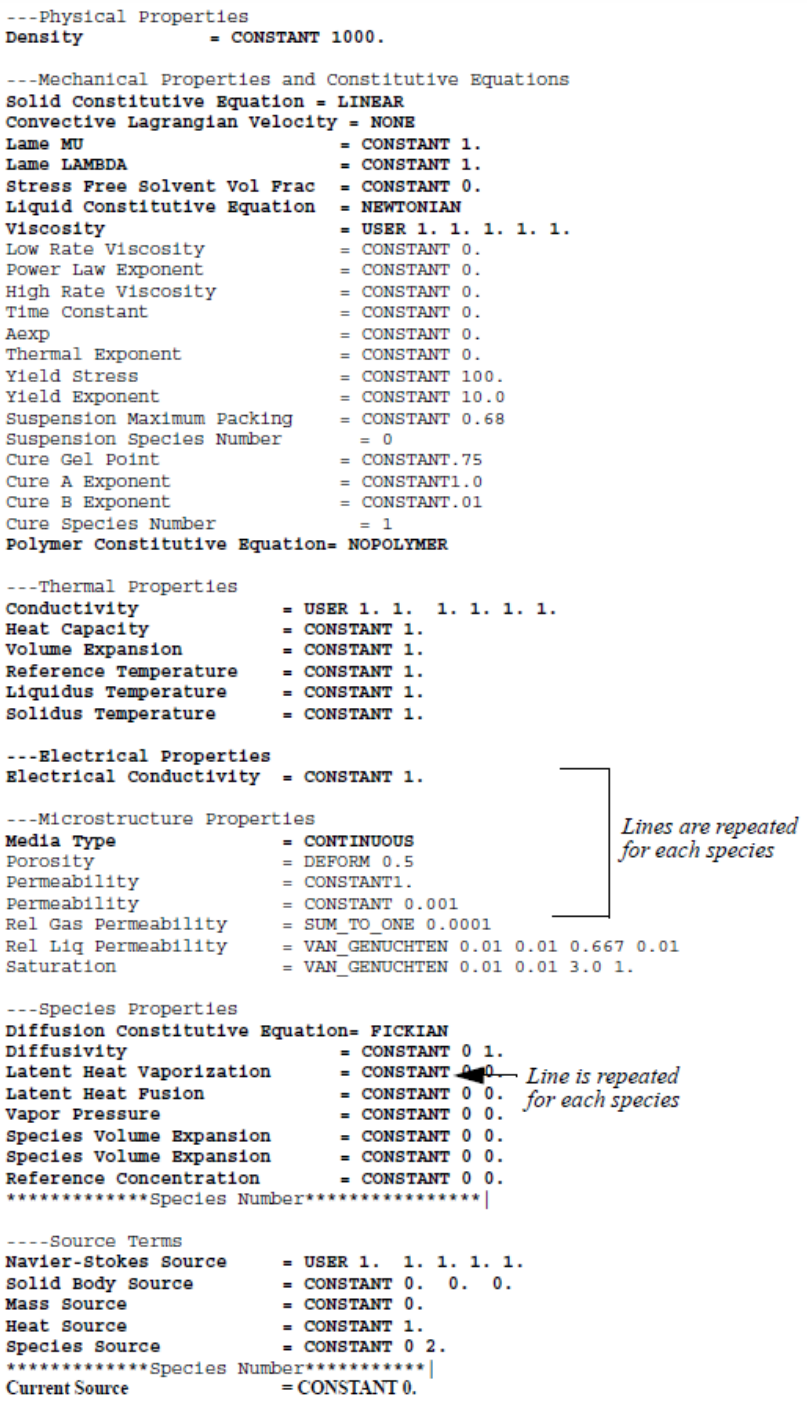


Fig. 5: Sample material-description file format. Lines highlighted in bold-face type are required.

1.5.1 Physical Properties

The intrinsic property of materials essential to *Goma* is the density. As *Goma* presumes that all materials are incompressible, density is a constant in the governing differential equations. However, several options for models of density are present in the code because numerous processes lead to density changes, though during any analysis cycle, the density is constant.

Default Database

```
Default Database = {GOMA_MAT | CHEMKIN_MAT}
```

Description / Usage

This card sets the default material database type. The default for this card is GOMA_MAT. In that case, all material properties for the current material are obtained from the current material file being read. If the default database is Chemkin, then the Chemkin 3 linking files are read in, and initialization of most of the methods and data for thermodynamic function evaluation, the stoichiometry and names of species and elements, the homogeneous and heterogeneous source terms for chemical reactions and their coupling into the energy equation, and transport property evaluations occurs. Many fields in the materials database file that were required now are optional. After Chemkin initialization, the rest of the materials database file is then read in. At that time, some fields containing methods and data that were initialized with Chemkin methods and data may be overwritten with methods and data specified by the material file. Other fields not initialized or even handled by Chemkin (such as surface tension) must be initialized for the first time by the materials file. Thus, the use of Chemkin materials database doesn't mitigate the need for a Goma materials file.

Examples

Following is a sample card:

```
Default Database = CHEMKIN_MAT
```

Technical Discussion

Chemkin includes its own rigorous treatment of ideal gas thermodynamics and transport property evaluations, providing it with a solid foundation on which to build kinetics mechanisms and a rigorous treatment of gas phase transport property evaluation. In order to maintain internal consistency, the new treatment must be used in its entirety.

References

No References.

Density

```
Density = {model_name} {float_list}  [M/L3]
```

Description / Usage

This required card is used to specify the model, and all associated parameters, for density. Definitions of the input parameters are as follows:

{model_name}	Name of the density model. This parameter can have one of the following values: **CONSTANT, USER, FILL, SUSPENSION, IDEAL_GAS, THERMAL_BATTERY, LEVEL_SET, CONST_PHASE_FUNCTION, FOAM, REACTIVE_FOAM, or SOLVENT_POLYMER.** Boussinesq models can be selected through the *Navier-Stokes Source* card.
{float_list}	One or more floating point numbers (<float1> through <floatn> whose interpretation is determined by the selection for {model_name}.

Thus, choices for {model_name} and the accompanying parameter list are given below; additional guidance to the user can be found in the Technical Discussion section following the Examples.

CONSTANT <float1>	For the **CONSTANT** density model, {float_list} is a single value: • <float1> - Density [M/L3]
USER <float1> … <floatn>	For a user-defined model, the set of parameters specified as <float1> through <floatn> are defined in the function usr_density.
FILL <float1> <float2>	The model is used with the fill equation when the location of the free surface between two fluids is tracked with a volume-of-fluid method. The {float_list} contains two values for this model, where: • <float1> - Density of the fluid in phase 1, denoted by F=1 • <float2> - Density of the fluid in phase 2, denoted by F=0 This card is required when using the FILL momentum source model (Navier-Stokes Source in Source Terms section of manual) since it makes use of this model to compute the value of the density.
SUSPENSION <float1> <float2> <float3>	The option is used to model a suspension where the solid particle phase and the carrier fluid have different densities. The {float_list} contains three values for this model, where: • <float1> - Species number associated with the solid particulate phase (the parser reads this as a float but it is cast as an integer when assigned). • <float2> - Density of the fluid in the carrier fluid, ρ_f . • <float3> - Density of the solid particulate phase, rho_s .
THERMAL_BATTERY <float1> <float2>	This model is used to relate electrolyte density to field variables such as mole fraction. A simple empirical form is used, with two constants in the {float_list}: • <float1> - Base Electrolyte Density, p_0. • <float2> - Constant, α . (See Technical Discussion.)
SOLVENT_POLYMER <float1>	This density model is used primarily in problems involving drying of polymeric solutions. The single float parameter on this card is specific volume of the solvent material. Note that the numerical value for this parameter must be chosen to be consistent with the specific volumes for each species in the solution set with the Specific Volumes card in the material file (discussed below).
LEVEL_SET <float1> <float2> <float3>	This model is used to vary the density in the flow regime when following an interface between two fluids using level set interface tracking. This choice assures a smooth transition in density across the zero level set contour. The {float_list} contains three values for this model, where: • <float1> - Fluid density in the negative regions of the level set function, ρ_- • <float2> - Fluid density in the positive regions of the level set function, ρ_+ • <float3> - Length scale over which the transition occurs, α . If this parameter is set to zero, it will default to one-half the Level Set Length Scale value already specified. This card is required when using the LEVEL_SET momentum source model (Navier-Stokes Source in Source

Examples

Following are some sample input cards:

```
Density = CONSTANT 1000.
```

```
Density = LEVEL_SET 0.05 0.0001 0.25
```

```
DENSITY = CONST_PHASE_FUNCTION 0.9 0.001 12.0 0.0 0.00001
```

Technical Discussion

- The **CONSTANT** density model prescribes an unchanging value for an incompressible fluid; only a single value need be specified by the user.
- The **USER** model provides a means for the user to create a custom density model for his/her problem. This user-defined model must be incorporated by modifying the usr_density function in the file user_mp.c. The parameters needed by this model are entered in the {float_list} and are passed to the usr_density routine as an array.
- The **FILL** model is used when the location of the interface between two fluids is tracked with an explicit volume-of-fluid method. The value of density is defined from the following:

$$\rho(F) = \rho_1 F + \rho_0(1-F)$$

where ρ_1 and ρ_0 are the phase densities obtained from the FILL density card, F is the value of the fill color function. As can be seen, ρ_1 is the density value when F = 1 while ρ_0 is the density value when F = 0. In the transition zone between these to extremes of F, the density will simply be a weighted average of the two values.

- The **SUSPENSION** model is used to model a suspension where the solid particle phase and the carrier fluid have different densities. The concentration of the continuum mixture is defined by the following relationship:

$$\rho = \rho_f + (\rho_s - \rho_f)\phi$$

where ϕ is the volume fraction of the solid particulate phase, ρ_f , is the density of the fluid in the carrier fluid and ρ_s, is the density of the solid particulate phase. The solid particulate phase has an associated species number as this is designates the species equation being solved for this component.

- The **THERMAL_BATTERY** model is used to relate electrolyte density to field variables such as mole fraction. A simple empirical form is used with the density of the system being specified by the following equation,:

$$\rho_i = \rho_0 + a x_i$$

where x_i is the mole fraction of ionic species *i*, ρ_0 is the base electrolyte density and is *a* constant.

- The **LEVEL_SET** density model is used to vary the density in the flow regime from one phase to the other when the interface between two fluids is being followed by level set interface tracking. The model assures a smooth

transition in density across the zero level set contour. The density as a function of the level set function value, ϕ, is as follows:

$$\rho(\phi) = \rho_-, \qquad \phi < -\alpha$$

$$\rho(\phi) = [\rho_- + (\rho_+ - \rho_-)H_\alpha(\phi)], \qquad -\alpha < \phi < \alpha$$

$$\rho(\phi) = \rho_+, \qquad \phi > \alpha$$

where

$$H_\alpha(\phi) = (1 + \phi/\alpha + \sin(\pi\phi/\alpha)/\pi)/2$$

is a smooth Heaviside function, ϕ is the value of the level set function, ρ_+ and ρ_- are density values of the fluids assigned positive or negative values of the level set function, respectively, and α is the density transition length scale, that is, half the width of the transition zone between density values. Note that this value may differ from the level set length scale parameter set elsewhere.

- The **CONST_PHASE_FUNCTION** model computes the density at a given point with the following relation:

$$\rho = \sum_{N_f} \rho_i H_\alpha(\phi_i) + \rho_\varnothing$$

where ρ_i are the individual phase function (ϕ_i) densities, $H_a(\phi_i)$, is the smoothed Heaviside function using the length scale specified on the card. The parameter $\rho_t heta$ is the null density and will only come into play at points were all phase function values are less than zero. In theory, this shouldn't happen for well posed problems, but in practice it is not uncommon.

- The **SPECIES_SOURCE** and **REACTIVE_FOAM** models both employ the following density formula:

where w_j is the mass fraction of component j and V_j is the specific volume of species j; these two parameters are set by the Specific Volume cards in the material file. The variable N is the total number of bulk species. The variable $V_n + 1$ is the specific volume specified in the density card.

1.5.2 Mechanical Properties and Constitutive Equations

This section of the material property input specifies the type of model, for both solids and fluids, that relates stress and strain (or strain-rate) as well as the various parameters for these models. Models for solids are relatively simple compared to solid mechanics codes but cover the primary needs in fluid-solid problems. The models for fluids are quite extensive, covering Newtonian, generalized-Newtonian, rate-dependent models, thermally-dependent models, curing and particleladen models and combinations of these. These properties are used in the solid and fluid momentum conservation equations.

$$\rho_{mix} = \left[V_{N+1} + \sum_{j=1}^{N} W_j (V_j - V_{N+1}) \right]^{-1}$$

Solid Constitutive Equation

```
Solid Constitutive Equation = {model_name}
```

Description / Usage

This required card specifies the constitutive equation used to control mesh motion and/ or the constitutive model describing solid material stress response to deformation. The single input parameter is defined as

{model_name}	The name of the constitutive equation. The permissible values for {model_name} are dependent on the selection for the Mesh Motion type, that being one of ARBITRARY, LAGRANGIAN/DYNAMIC_LAGRANGIAN, or TOTAL_ALE. For an ARBITRARY *Mesh Motion*, {model_name} can be one of the following: • **LINEAR** - a linear elastic model for which the deformations are assumed to be small, thus simplifying the analysis of strain and stress. • **NONLINEAR** - a nonlinear neo-Hookean elastic model for which the deformations can be large without loss of frame invariance. This is the recommended model (and all materials currently default to NONLINEAR if the mesh is arbitrary). For a LAGRANGIAN, DYNAMIC_LAGRANGIAN, or TOTAL_ALE *Mesh Motion*, {model_name} can be one of the following: • **LINEAR** - a linear elastic model for which the deformations are assumed to be small, thus simplifying the analysis of strain and stress. • **NONLINEAR** - a nonlinear neo-Hookean model applicable for 2D, 3D, and CYLINDRICAL coordinates. • **HOOKEAN_PSTRAIN** - a nonlinear neo-Hookean model with plane strain for which the deformations can be large without loss of frame invariance (2D only). • **HOOKEAN_PSTRESS** - a nonlinear neo-Hookean model with plane stress (not activated in *Goma* 4.0). • **INCOMP_PSTRAIN** - An incompressible nonlinear neo-Hookean model with plane strain and a Lagrangian pressure constraint on the volume. • **INCOMP_PSTRESS** - An incompressible nonlinear neo-Hookean model with plane stress and a Lagrangian pressure constraint on the volume (not activated in *Goma* 4.0). • **INCOMP_3D** - Incompressible version of the neo-Hookean solid in a special Segalman formulation that removes the volume-change from the strain tensor (like the *INCOMP_PSTRAIN* model above), and is specifically designed for 3D applications (not widely used).

The functional form of each of these equations is provided in the Technical Discussion with some important details.

Examples

The following is a sample input card:

```
Solid Constitutive Equation = LINEAR
```

This equation type requires two elastic constants be specified, *Lame Lambda* and *Lame Mu*. This constitutive equation can be used for all *Mesh Motion* types. It is not recommended for large mesh deformations, even for ALE problems, because of spurious stresses generated by solid body translation or rotation.

Technical Discussion

The general compressible form of Hooke's Law, which applies to the *LINEAR, NONLINEAR* and *HOOKEAN_PSTRAIN* options, can be written as

$$\underset{\sim}{\sigma} = \lambda\varepsilon\underset{\sim}{\boldsymbol{I}} + 2\mu\underset{\sim}{\boldsymbol{E}}$$

Here λ is the Lame coefficient for volume expansion, ε is the volume strain measure whose definition depends on the model chosen, μ is another elastic Lame coefficient for shear deformation, $\underline{E}$ and is the chosen strain tensor, the form of which also depends on the constitutive model chosen.

The general incompressible form of Hooke's Law, which applies to all *INCOMP** options, can be written as:

$$\underset{\sim}{\sigma} = p\underset{\sim}{\boldsymbol{I}} + 2\mu\underset{\sim}{\boldsymbol{E}}$$

with *p* being the solid phase pressure. An additional continuity equation is required in this case to account for the pressure (see *Continuity* equation card). Note, for these model options one must set the *Lame LAMBDA* coefficient to zero, or the pressure term and the expansion term are added together.

The volume change and strain tensors depend on the chosen solid constitutive equation and are as follows:

For the *LINEAR* option:

$$\underset{\sim}{\varepsilon} = tr\left[(\nabla\underset{\sim}{d} + (\nabla\underset{\sim}{d})^T) \right]$$

and

where $\underline{d}$ is the displacement field vector, tr is the tensor trace operator, and the gradient operator (Δ) is with respect to the deformed coordinates.

For all *NONLINEAR* models, we use the deformation gradient tensor as a building block:

The "material coordinates" are $\underline{X}$ and describe the original locations of all parcels of material in the domain; and the "current configuration/spatial coordinates" $\underline{x}$ are the deformed mesh coordinates. Of course we have

$$\underset{\sim}{E} = (\nabla \underset{\sim}{d} + (\nabla \underset{\sim}{d})^T)$$

$$\underset{\sim}{F}_m \equiv \frac{\partial \underset{\sim}{x}_m}{\partial \underset{\sim}{X}} = (\underset{\sim}{I} - \nabla \underset{\sim}{d})^{-1}$$

$$\underset{\sim}{d} \equiv \underset{\sim}{x} - \underset{\sim}{X}$$

$$\underset{\sim}{C} \equiv \frac{1}{2}[\underset{\sim}{F}\underset{\sim}{F}^T - \underset{\sim}{I}]$$

for all *LAGRANGIAN* mesh motion cases. We define a Cauchy-Green tensor as:

and invoke the linearized small strain theory (viz. that $\Delta_x \underline{d} \equiv \Delta_x \underline{d}$), and write

$$\underset{\sim}{C} = (\nabla \underline{d} + (\nabla \underline{d})^T + \nabla \underline{d}^T \nabla \underline{d})$$

With these quantities, we form the volume strain and strain tensor for the various models:

For *NONLINEAR, INCOMP_PSTRAIN, INCOMP_3D,* and *HOOKEAN_PSTRAIN*:

$$\varepsilon = 3\left(det(\underset{\sim}{F})^{\frac{1}{3}} - 1 \right)$$

For *INCOMP_PSTRAIN* and *INCOMP_3D* we use:

$$\underset{\sim}{E} = \frac{1}{2}\left(1 - det(\underset{\sim}{F})^{\frac{2}{3}} \right) I + det(\underset{\sim}{F})^{\frac{2}{3}} \underset{\sim}{C}$$

For all other models we use $\underline{E} = \underline{C}$. It is noteworthy that we use the linearized small strain theory for parts of the strain tensor, but the real Lagrangian deformation gradient for the volumetric strain. For elastoviscoplastic models and *TOTAL_ALE* solid mechanics, we do not invoke the linearized small strain theory.

Also noteworthy is that the elastic constants λ and μ are related to the more well known bulk and Young's moduli and the Poisson's ratio by simple expressions (see *Lame Mu* and *Lame Lambda* cards).

Theory

The incompressible options (i.e., INCOMP_PSTRAIN and INCOMP_PSTRESS and INCOMP_3D) use the theory of Segalman, et. al. (1992) to control mesh motion and couple the volume dilation to changes in solvent content. Plane strain implies that there is no deformation in the z-direction; plane stress implies there is no stress change in the z-direction.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Segalman, D., W. Witkowski, D. Adolf, M. Shahinpoor, "Theory and Application of Electrically Controlled Polymeric Gels", Smart Mater. Struct. 1:95-100 (1992).

Plasticity Equation

```
Plasticity Equation = {model_name} [ ]
```

Description / Usage

This optional card specifies the formulation for the potential yielding/plastic flow regime during solid deformation. This card is not to be used in place of the *Solid Constitutive Equation* card, but rather supplements that card to describe the constitutive behavior during plastic deformation. Elastic deformation still proceeds according to the model specified on the *Solid Constitutive Equation* card (i.e., for regimes that have not yielded). The single input parameter is defined as

{model_name}	Name of the plasticity model. This parameter can have one of the following values:
EVP_HYPER	A constitutive equation that uses the elasticity portion specified on the *Solid Constitutive Equation* card for unyielding material and a complex hyperelastic plasticity equation for the yielding/flowing material as determined by the Von Mises yield criterion.
NO_MODEL	Or any value other than EVP_HYPER, will result in no plastic deformation.

Requirements for the use of this model are

- Transient problems only
- *LAGRANGIAN* mesh motion only; no *TALE*
- Continuous media only; no porous media (as specified on the *Media Type* card)
- Elastic Plane Strain models only (i.e., INCOMP_PSTRAIN in the *Solid Constitutive Equation* card)
- a *Plastic Viscosity* card and an *EVP Yield Stress* card must also be supplied.

Examples

Following is a sample card:

```
Plasticity Equation = EVP_HYPER
```

which specifies hyperelastic elastoviscoplastic model is to be used for a solid phase constitutive equation. In addition to the Lame coefficients that are still required as the mechanical properties of the unyielded material, this model also requires a plastic viscosity and a yield stress, viz.

```
Plastic Viscosity = LINEAR 1.0 2.0
```

```
EVP Yield Stress = CONSTANT 50.0
```

Technical Discussion

Detailed theoretical discussion, usage tutorials and troubleshooting tips for this model are covered in the EVP tutorial (GT-019.1). Usage examples for four different strain scenarios are given, including a solid yielding from an applied mechanical load and a solid yielding from high shrinkage stress during drying.

FAQs

Problem – Trouble in continuing the first few time steps.

Solution – You may have a fast drying case with slow diffusion in the coating. Instead of decreasing the time step size according to the normal procedure and intuition, increase the time step size. With fast drying and slow diffusion, the initial concentration gradient is very steep at the drying surface. This is a very difficult numerical problem to solve. So when you increase the time step size, in effect, you are relaxing the concentration gradient the program is solving, that will get you past the initial numerical difficulty. However, even if the code can handle such a condition, the concentration and stress profile may appear very wavy. This waviness only reflects the degree of difficulty the code encountered and is not part of the real solution. In this case, refining the mesh towards the drying surface will only increase the waviness of the solution. Drawing from this observation, coarsening the mesh will also get you past this initial numerical difficulty. Although this condition may pose numerical stability problems initially, it does not affect subsequent solution. And most of the time, one is not interested in the solution from the initial time steps.

Problem – Trouble in converging in the plastic region.

Solution – Reduce the time step size because viscoplasticity is in itself a time dependent problem and elasticity in itself is not. Before the material yields, time dependency is induced only through the drying process. The reduction in time step size depends on the value of the plastic viscosity. The lower the viscosity, the small time step should be used. Also, it takes more iterations to converge a time step in the viscoplastic region than the elastic region, so increasing the maximum allowable iterations per time step will help.

Other Cautions: Always set the *MASS_FRACTION* in the input file to be the same as the *Stress Free Solvent Vol Frac* in the material file.

The code has been tested for a wide range of initial solvent volume fractions (up to 0.85). When using very high initial solvent volume fractions (approaching 0.85 or beyond), use with caution.

References

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

S.Y. Tam's thesis: "Stress Effects in Drying Coatings," Ph.D Dissertation, University of Minnesota, 1997

Convective Lagrangian Velocity

```
Convective Lagrangian Velocity = {model_name} {float_list} [L/t]
```

Description / Usage

In solid mechanics, when the deformation of the mesh is Lagrangian, i.e., motion of the solid can be described by a mapping from the stress-free state (undeformed state) to the deformed state, it is often desirable to prescribe a convective velocity of the stress-free state that can lead to inertial forces through deformation (see Technical Discussion below). This required card allows for the specification of solid-body translation or rotation of the stress-free state, and results in an inertial term on the otherwise quasi static solid momentum equation.

Definitions of the input parameters are as follows:

{model_name}	Name of the prescribed velocity model. This parameter can have one of the following values: **NONE**, **CONSTANT**, or **ROTATIONAL**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below. Note that not all models employ a {float_list}.

Thus,

NONE	the stress-free state is assumed to be unmoving. No floating point input values are required with this model.
CONSTANT <float1> <float2> <float3>	the stress-free state is one of solid-body translation, viz. it moves uniformly with a velocity specified by three orthogonal components: • <float1> - X-component of velocity • <float2> - Y-component of velocity • <float3> - Z-component of velocity (for 3-D)
ROTATIONAL <float1> <float2> <float3> <float4>	the stress-free state is one of solid-body rotation at a specified rotation rate. • <float1> - Rotation rate, in radians/sec. • <float2> - X-position of axis of rotation (must be constant in 3D). • <float3> - Y-position of axis of rotation (must be constant in 3D, viz. the axis must be perpendicular to both the X and Y axes, viz. the axis must be the Z axis. • <float4> - Set to zero. Parameter is not used for now. Note that this model is applicable in 2-D and certain 3-D problems in which the rotation axis is the Z-axis. To generalize this model to three-dimensions, the proper input will require a point and a direction of the rotation axis. In two-dimensions, the axis of rotation is the Z-direction.

Examples

The following is a sample input card:

```
Convective Lagrangian Velocity = ROTATIONAL 25.0 1. 1. 0.
```

This card is associated with a material file, and hence a material that is of *LAGRANGIAN* or *TOTAL_ALE* type (see *Mesh Motion* card). That material's stressfree state, as specified by this model, will rotate about an axis that is located at [1.0, 1.0, 0] at 25 radians/sec (assuming seconds are the time scale of the problem).

Technical Discussion

This capability is often used when problems require a force or a boundary condition to be applied to a solid material that is moving relative to the source, or the desired frame of reference. Such constraints arise mainly in fluid-structure interaction problems where one solid material is moving relative to another, with a fluid material in between, e.g. deformable blade or knife metering/pushing liquid over a flat or round substrate. These models have also been used in porous-material translation relative to a drying source (see references below).

Specification of any model but **NONE** on this card produces the left-hand-side term in the equation for quasi static equilibrium:

$$\nabla \bullet \rho \underset{\sim}{v}_m^0 \underset{\sim}{v}_m^0 = \nabla \bullet \underset{\sim}{\sigma}$$

σ is the Cauchy stress tensor of the solid material, and f is the body force per unit volume. The first term is a result of the specified advection of the stress-free state. v_m^0, which depends solely on the user-prescribed velocity and the current state of deformation, is by definition

$$\underset{\sim}{v}_m^0 = \frac{\partial x_m}{\partial X} \cdot \frac{\partial X}{\partial t} = \underset{\sim}{F}_m \cdot \underset{\sim}{v}_{sfs} \quad .$$

where F_m is the material deformation gradient tensor (computed somewhat differently depending on the formulation, as described in the references below), and $v_s fs$ is the stress-free state velocity field specified by this card.

References

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

Lame MU

```
Lame MU = {model_name} {float_list} [M/ :math:`Lt^2`]
```

Description / Usage

This required card is used to specify the model for the Lame coefficient μ for the solid constitutive equation (see Sackinger, et. al. 1995, and *Solid Constitutive Equation* card); this coefficient is equivalent to the shear modulus *G* in most cases, as described below.

Definitions of the input parameters are as follows:

{model_name}	Name of the Lame Mu coefficient model. This parameter can have one of the following values: **CONSTANT, POWER_LAW, CONTACT_LINE, SHEAR_HARDEN, EXPONENTIAL, DENSE_POWER_LAW, or USER.**
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model.

The details of each model option are given below:

CONSTANT <float1>	For the **CONSTANT** model, {float_list} is a single value: <float1> - Standard value of the coefficient μ. (See Technical Discussion.)
POWER_LAW <float1> <float2> <float3>	The **POWER_LAW** model is only to be used for deformable porous media where the shear modulus is allowed to vary as a power of the porosity, ϕ (see Scherer, 1992): The {float_list} contains three values for this model, where: .. figure:: /figures/360_goma_physics.png :align: center :width: 90% • <float1> - G_0 is the base shear modulus at the initial porosity (or $\mu 0$) • <float2> - ϕ_0 is the porosity in the stress free state • <float3> - *m* is the powerlaw exponent
CONTACT_LINE <float1> <float2> <float3> <float4>	The **CONTACT_LINE** model is a convenient way to control mesh deformation near a fixed point and is normally used ONLY for *ARBITRARY Mesh Motion* types. This model enables the user to make the shear modulus much larger near the contact line (fixed point) than far away from the contact line, so that elements near the contact line are forced to retain their shape. The shear modulus in this model varies inversely with distance from the contact line: $$G = G_0 + \frac{0.1}{((r/r_0)^3 + 0.1/G_1)}$$ *r* is the distance from the fixed point, r_0 is a decay length, G_0 is the modulus at the contact line. The {float_list} contains four values for this model, where: • <float1> - Node set number of the fixed point (converted to an integer by *Goma*) • <float2> - G_0 (or μ_0) • <float3> - G_1 • <float3> - r_0
SHEAR_HARDEN <float1> <float2>	The **SHEAR_HARDEN** model is: $$G = G_0 + \chi(II_E)^2$$ where χ is the coefficient of variation, II_E is the second invariant of the strain tensor (see *Solid Constitutive Equation* card), G_0 is the modulus at zero shear. The {float_list} contains two values for this model, where: • <float1> - G_0 (or μ_0) • <float2> - χ
EXPONENTIAL <float1> <float2> <float3>	The **EXPONENTIAL** model is used exclusively for poroelastic problems, and allows for an exponential dependence of the shear modulus μ (or G) on porosity: $$G = G_0 e^{\lambda(\phi_0 - \phi)}$$ where λ is the rate of decay, ϕ_0 is the porosity in the

All modulus values in these equations have the same units as Lame Mu, i.e., M/Lt2.

Examples

The following is a sample card:

```
Lame MU = CONSTANT 1.
```

Technical Discussion

Note that μ and λ, (see the *Lame LAMBDA* card) are related to the more often used Young's Modulus and Poisson's Ratio by the following standard expressions:

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu} \qquad \upsilon = \frac{\lambda}{2(\lambda + \mu)}$$

where E is the Young's modulus and υ is Poisson's ratio. A significant limiting case is approached as ν approaches 0.5, in which case the solid becomes incompressible.

The **POWER_LAW** option could easily be adapted to a concentration measure, viz. made dependent on the concentration of some species (see EQ = *species_bulk* card). This can be done through the user option, and in fact in usr_lame_mu function of file user_mp.c in the *Goma* distribution has an example that is appropriate. Also note that all of these models are available for the elastoviscoplastic option on the *Plasticity* card, and for the real-solid in *TOTAL_ALE* mesh motion.

FAQs

Important note that when one desires an incompressible solid through the use of INCOMP_PSTRAIN type models, by using an incompressible continuity equation in a LAGRANGIAN mesh region (see *EQ = continuity*), then the bulk modulus, or Lame Lambda expansion term is also added on. So to get a truly incompressible response, one must set the Lame LAMBDA coefficient to zero.

References

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Scherer, G.W., 1992, "Recent Progress in Drying of Gels", J. of Non-Crystalline Solids, 147&148, 363-374

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

GTM-027: Probing Plastic Deformation in Gelatin Films during Drying, M. Lu, S. Y. Tam, A. Sun, P. R. Schunk and C. J. Brinker, 2000

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Lame LAMBDA

```
Lame LAMBDA = {model_name} {float_list} [M/Lt2]
```

Description / Usage

This required card is used to specify the model for the Lame coefficient λ for the solid constitutive equation (see Sackinger, et. al., 1995). When using a nonlinear constitutive equation for ALE mesh motion, this coefficient is related to the bulk modulus:

$$\kappa = \lambda + \frac{2}{3}\mu$$

Definitions of the input parameters are as follows:

{model_name}	Name of the *Lame LAMBDA* model. This parameter can have one of the following values: **CONSTANT, POWER_LAW, EXPONENTIAL or USER**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below.

The models are described here.

CONSTANT <float1>	For the **CONSTANT** model, {float_list} is a single value (see *Lame MU* card for relationship to other more common elastic constants): • <float1> - Standard value of the elastic constant λ
POISSON_RATIO <float1>	For any Lame MU model (see Lame MU card) this option uses the following formula to compute Lame Lame LAMBDA: • <float1> - Poisson's ratio nu. $$\lambda = \frac{2\mu\nu}{(1-2\nu)}$$
POWER_LAW <float1> <float2> <float3>	The **POWER_LAW** model can be used in deformable porous media where the Lame coefficient varies as a power of the porosity, ϕ (Scherer, 1992): $$\lambda = \lambda_0\left(\frac{1-\phi}{1-\phi_0}\right)^m$$ The {float_list} contains three values for this model, where: • <float1> - λ_0 is the base *Lame LAMBDA* modulus at the initial porosity. • <float2> - ϕ_0 is the porosity in the stress-free state • <float3> - *m* is the powerlaw exponent, as shown
USER <float1>,…, <floatn>	For the **USER** model, {float_list} is of arbitrary length, and the values are used through the param[] array in usr_lame_lambda function to parameterize a userdefined model. See examples in user_mp.c.

Examples

Following is a sample card:

```
Lame LAMBDA = CONSTANT 1.
```

Technical Discussion

Please see the *Solid Constitutive Equation* card for details on the use of this parameter. Special consideration is required for *INCOMP** type constitutive equations. The isotropic stress term, or pressure, in that case is added onto the constitutive equation, and so this parameter must be set to zero so as to prevent any compressibility.

Important note that when one desires an incompressible solid through the use of *INCOMP_PSTRAIN* type models, by using an incompressible continuity equation in a *LAGRANGIAN* mesh region (see *EQ = continuity*), then the bulk modulus, or Lame Lambda expansion term is also added on. So to get a truly incompressible response, one must set the *Lame LAMBDA* coefficient to zero.

References

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Scherer, G.W., 1992, "Recent Progress in Drying of Gels", J. of Non-Crystalline Solids, 147&148, 363-374

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Stress Free Solvent Vol Frac

```
Stress Free Solvent Vol Frac = CONSTANT <float> []
```

Description / Usage

This required card is used to specify the model for the stress-free solvent volume fraction, which is the volume fraction of solvents in the solid material in its stress-free state. This card is used exclusively in materials of *LAGRANGIAN* or *TOTAL_ALE* Mesh Motion types (see *Mesh Motion* card) which are being modeled as gelled solids laden with solvent. At the gel-point, the solid is considered to be stress free, after which a reduction of solvent leads to volume shrinkage and hence a rising stress state. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the stress-free solvent volume fraction.
<float>	The value of the stress-free solvent volume fraction; this value is unitless.

Examples

The following is a sample card:

```
Stress Free Solvent Vol Frac = CONSTANT 0.5
```

This specification sets the volume fraction of solvent in the material to 50 per cent. That volume fraction is tantamount to the gel point of the material.

Technical Discussion

The stress free state volume fraction of solvent is basically the solvent fraction at which a material gels, viz., the state at which the material solidifies from a liquid state. This quantity is used in the continuity equation for incompressible solid materials, through which is transported by a variety of diffusion models (see *Diffusivity* card). The continuity equation, viz., *EQ* = *continuity*, is applied as follows:

$$det\underset{\sim}{F} = \frac{1 - y^0}{1 - \sum_i y^i}$$

where the dependent variable is the solid phase pressure (see *Solid Constitutive Equation* card). Here *det* **F** is the determinant of the deformation gradient tensor, yi is the volume fraction of component i (specified by the *EQ* = *species_bulk* card), and y0 is the volume fraction of total solvents at the stress free state. Clearly, as the solvent concentration decreases the local volume of solid decreases, creating a rising stress.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Solid Thermal Expansion

```
Solid Thermal Expansion = {CONSTANT | SHRINKAGE} <float> [1/T]
```

Description / Usage

This card is used to specify the model for thermal expansion of solid materials. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the thermal expansion coefficient.
SHRINKAGE	Model for adding solidification shrinkage stress effects for enthalpy models. Experimental only (1/25/2013).
<float>	The value of the thermal expansion coefficient. For the SRINKAGE model this float is not used.

Examples

The following is a sample card:

```
Solid Thermal Expansion = CONSTANT 0.001
```

Technical Discussion

When solid materials expand due to temperature changes, the strain field is composed of two components, the strain due to the stress field and the strain due to thermal expansion:

$$\underset{\sim}{\varepsilon} = \underset{\sim}{\varepsilon}^{(S)} + \underset{\sim}{\varepsilon}^{(T)}$$

The strain due to thermal expansion is given by

$$\underset{\sim}{\varepsilon}^{(T)} = \alpha(T - T_0)\underset{\sim}{\delta}$$

where α is the linear thermal expansion coefficient T_0 and is the reference temperature (see Solid Reference Temperature card). As a result, the solid constitutive relation contains an extra term:

$$\underset{\sim}{\sigma} = 2\mu\underset{\sim}{\varepsilon} + \lambda tr(\underset{\sim}{\varepsilon})\underset{\sim}{\delta} - 2\mu\alpha(T - T_0)\underset{\sim}{\delta} \quad .$$

Note, the linear thermal expansion coefficient is presumed to be independent of strain and the Lame constants are presumed to be independent of temperature. (Model is hardwired right now in GOMA source, PRS 1/23/2013).

In the case of the SHRINKAGE model, an additional term is added on to the deviatoric stress:

References

For a discussion of linear thermoelasticity, see (Section 6.2)

Malvern, L. E., 1969, Introduction to the Mechanics of a Continuous Medium, Prentice-Hall

Solid Reference Temperature

```
Solid Reference Temperature = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify the model for the solid reference temperature used in the thermal expansion of solid materials. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the reference temperature.
<float>	A floating point number that is the value of the solid reference temperature, T_{ref} .

Examples

The following is a sample card:

```
Solid Reference Temperature = CONSTANT 90.0
```

Technical Discussion

See the *Solid Thermal Expansion* card for a discussion of the use of this property in the linear thermoelasticity of solids.

References

No References.

Plastic Viscosity

```
Plastic Viscosity = {CONSTANT | LINEAR} <float1> [float2] [M/L-t]
```

Description / Usage

This card is used to specify the characteristic viscosity of plastic deformation and is required when the *Plasticity Equation* card is present. Definitions of the input model options are as follows:

CONSTANT	Name of the model for a constant plastic viscosity. • <float1> - the value of the viscosity.
LINEAR	LINEAR Name of the model for a linear variation in plastic viscosity; this model requires two floating point values as parameters. • <float1> - y_1, the lower limit of plastic viscosity • <float2> - y_2, the upper limit of plastic viscosity

Examples

Following is a sample card:

```
Plastic Viscosity = LINEAR 1.0 100.
```

This specification results in a linear variation of plastic viscosity of the elastoviscoplasticity constitutive equation with concentration of solvent species according to the equation above.

Technical Discussion

Using the concentration of solvent species as the independent variable in the *LINEAR* model, the viscosity y at a certain concentration c is:

$$y = y_1 + \left(\frac{V_{sf} - c}{V_{sf}} \right) (y_2 - y_1)$$

where V_{sf} is the stress-free solvent volume fraction and the solvent volume fraction at solidification, which is set by the *Stress Free Solvent Vol Fraction* card in the material file. The input parameters for the LINEAR model are the plastic viscosity limits y_1 and y_2. *NOTE: this model activates a linear dependence on concentration and hence can only be used for cases in which there is solvent transport.*

So for a typical drying/solidification problem, the material file input deck requirements are shown as follows:

```
Stress Free Solvent Vol Frac = CONSTANT 0.6
```

```
Plasticity Equation = EVP_HYPER
```

```
Plastic Viscosity = LINEAR 1.0 2.0
```

```
EVP Yield Stress = CONSTANT 50.0
```

Together with these properties one must specify the elastic constants *Lame Mu* and *Lame Lambda*.

Theory

See Schunk, et. al., 2001 (GT-019.1).

References

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

GTM-020.0: In-Situ Characterization of Stress Development in Gelatin Film During Controlled Drying, M. Lu, S-Y Tam, P. R. Schunk and C. J. Brinker, March 2000.

GTM-027.0: Probing Plastic Deformation in Gelatin Films during Drying, M. Lu, S. Y. Tam, A. Sun, P. R. Schunk and C. J. Brinker, 2000.

S.Y. Tam's thesis: "Stress Effects in Drying Coatings," Ph.D Dissertation, University of Minnesota, 1997

EVP Yield Stress

```
EVP Yield Stress = {CONSTANT | LINEAR} <float1> [<float2>] [M/L-t2]
```

Description / Usage

This card is used to specify the characteristic yield stress for Von Mises yield criterion of plastic deformation and is required when the *Plasticity Equation* card is present. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for a constant yield stress. • <float1> - the value of the yield stress.
LINEAR	LINEAR Name of the model for a linear variation in plastic viscosity; this model requires two floating point values as parameters. • <float1> - y_1, the lower limit of yield stress. • <float2> - y_2, the upper limit of yield stress.

Examples

Following is a sample card:

```
EVP Yield Stress = LINEAR 1.0 100.
```

This specification results in a linear variation of yield stress of the elastoviscoplasticity constitutive equation with concentration of solvent species according to the equation above.

Technical Discussion

Using the concentration of solvent species as the independent variable, the yield stress y at a certain concentration c is:

$$y = y_1 + \left(\frac{V_{sf} - c}{V_{sf}}\right)(y_2 - y_1)$$

where V_{sf} is the stress-free solvent volume fraction and the solvent volume fraction at solidification, which is set by the Stress Free Solvent Vol Fraction card in the material file. The input parameters for the *LINEAR* model are the plastic viscosity limits y_1 and y_2. *NOTE: this model activates a linear dependence on concentration and hence can only be used for cases in which there is solvent transport.*

So for a typical drying/solidification problem, the material file input deck requirements are shown as follows:

```
Stress Free Solvent Vol Frac = CONSTANT 0.6
```

```
Plasticity Equation = EVP_HYPER
```

```
Plastic Viscosity = LINEAR 1.0 2.0
```

```
EVP Yield Stress = CONSTANT 50.0
```

Together with these properties one must specify the elastic constants *Lame Mu* and *Lame Lambda*.

Theory

See Schunk, et. al., 2001 reference.

References

GT-019.1: Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial, P. R. Schunk, A. Sun, S. Y. Tam (Imation Corp.) and K. S. Chen, January 11, 2001

GTM-020.0: In-Situ Characterization of Stress Development in Gelatin Film During Controlled Drying, M. Lu, S-Y Tam, P. R. Schunk and C. J. Brinker, March 2000.

GTM-027.0: Probing Plastic Deformation in Gelatin Films during Drying, M. Lu, S. Y. Tam, A. Sun, P. R. Schunk and C. J. Brinker, 2000.

S.Y. Tam's thesis: "Stress Effects in Drying Coatings," Ph.D Dissertation, University of Minnesota, 1997

Polymer Viscosity

```
Pseudo-Solid Constitutive Equation = {model_name}
```

Description / Usage

This card specifies the constitutive equation used to control mesh motion for arbitrary Lagrangian Eulerian solid mechanics and is required for use with the *TOTAL_ALE* mesh motion type (see *Mesh Motion* card). Details are discussed in references provided below.

The single input parameter is the type of model for the constitutive equation:

{model_name}	The name of the constitutive equation; {model_name} can be one of the following: • **LINEAR** - the mesh deformations are assumed to be small and thus simplifies the analysis of strain and stress. • **NONLINEAR** - a nonlinear neo-Hookean elastic model for which the deformations can be large without loss of frame invariance. This is the recommended model (and all materials currently default to NONLINEAR if the mesh is arbitrary). The following models are allowed but not recommended. • **HOOKEAN_PSTRAIN** - a nonlinear neo-Hookean model with plane strain assumption 2D only). • **INCOMP_PSTRAIN** - an incompressible nonlinear neo-Hookean model with plane strain and a Lagrangian pressure constraint on the volume. • **INCOMP_3D** - Incompressible version of the neo-Hookean solid in a special Segalman formulation that removes the volume-change from the strain tensor (like the *INCOMP_PSTRAIN* model above), and is specifically designed for 3D applications (not a widely used option).

Note again the requirement that the *Mesh Motion* type for the material in which this constitutive equation applies must be *TOTAL_ALE*.

Examples

```
Pseudo-Solid Constitutive Equation = NONLINEAR
```

This card specifies the mesh motion in the ALE solid region is to conform to the nonlinear elastic model, as described on the *Solid Constitutive Equation* card. This card is required together with *Pseudo-Solid Lame Mu and Pseudo-Solid Lame Lambda* cards.

Technical Discussion

The Pseudo-Solid mesh motion, like the *ARBITRARY* mesh motion, is governed by the equations of elasticity. These cards, together with the other cards required by the real solid constitutive behavior, are required for ALE solid mechanics. The theory is explained in detail in the provided references. Throughout the boundary condition options, the user will notice an appended *_RS*. This signifies that the boundary conditions apply to the real-solid elasticity in *TOTAL_ALE* problems. All other boundary conditions on force and displacement, viz. those without the *_RS*, are applied to the mesh motion.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

SAND2000-0807: TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid- Structure Interaction Problems, P. R. Schunk (May 2000)

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Pseudo-Solid Lame MU

```
Pseudo-Solid Lame MU = {model_name} {float_list} [M/Lt2]
```

Description / Usage

This card is required only for *TOTAL_ALE* mesh motion types (see *Mesh Motion* card) and is used to specify the model for the Lame coefficient μ for the mesh motion solid constitutive equation (see Sackinger et al. 1995, and *Solid Constitutive Equation* card); this coefficient is equivalent to the shear modulus G. The model list here is abbreviated as compared to the *Lame MU* card as these properties are just used to aid in the elastic mesh motion, independent of the material.

Definitions of the input parameters are as follows:

{model_name}	Name of the Lame' Mu coefficient model. This parameter can have one of the following values: **CONSTANT** or **CONTACT_LINE**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below.

The details of each model option are:

CONSTANT <float1>	**For the CONSTANT model, {float_list} is a single value:** • <float1> - Standard value of the μ (or the shear modulus G for the mesh). See *Pseudo Solid Constitutive Equation* card.
CONTACT_LINE <float1> <float2> <float3> <float4>	The CONTACT_LINE model is a convenient way to control mesh deformation near a fixed point and is normally used ONLY for *TOTAL_ALE* or *ARBITRARY Mesh Motion* types. This model enables the user to make the shear modulus much larger near the contact line (fixed point) than far away from the contact line, so that elements near the contact line are forced to retain their shape. The shear modulus in this model varies inversely with distance from the contact line: This card specifies the mesh motion in the ALE solid region is to conform to the nonlinear elastic model, as described on the Solid Constitutive Equation card. This card is required together with Pseudo-Solid Lame Mu and Pseudo-Solid Lame Lambda cards. $$G = G_0 + \frac{0.1}{((r/r_0)^3 + 0.1/G_1)}$$ *r* is the distance from the fixed point, r_0 is a decay length, G_0 is the modulus far from the contact line, and G_0 is the modulus at the contact line. The {float_list} contains four values for this model, where: • <float1> - Node set number of the fixed point (converted to an integer by *Goma*) • <float2> - G_0 (or μ_0) • <float3> - G_1 • <float4> - r_0

Examples

```
Pseudo-Solid Lame MU = CONSTANT 0.5
```

This card specifies that the current material have a constant shear modulus of 0.5 for the mesh elasticity. Note that the real-solid mesh Lame MU is set with the *Lame MU* card.

Technical Discussion

It is best to consult the TALE tutorial (Schunk, 1999) for details of this card.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Pseudo-Solid Lame LAMBDA

```
Pseudo-Solid Lame LAMBDA = CONSTANT <float> [M/Lt2]
```

Description / Usage

This card is required only for *TOTAL_ALE* mesh motion types (see *Mesh Motion* card) and is used to specify the model for the Lame coefficient λ for the mesh motion elasticity (see Sackinger et al., 1995).

This material parameter currently has only one possible model type (CONSTANT) with only a single required input value, as follows:

CONSTANT	**Name of the *Lame LAMBDA* coefficient model.** • <float1> - Standard value of μ (or the shear modulus G for the mesh). See *Pseudo-Solid Constitutive Equation* card.

Examples

The following is a sample input card:

```
Pseudo-Solid Lame LAMBDA = CONSTANT 1.
```

This card specifies that the current material have a constant shear modulus of 0.5 for the mesh elasticity. Note that the real-solid mesh Lame MU is set with the *Lame MU* card.

Technical Discussion

See discussion on *Lame LAMBDA* card and *Solid Constitutive Equation* card for more details. The main difference here is that this modulus is applied only to the moving mesh, and not the real solid as in an ALE solid mechanics simulation.

References

GT-005.3: THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media, August 6, 1999, P. R. Schunk

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Liquid Constitutive Equation

```
Liquid Constitutive Equation = {model_name}
```

Description / Usage

This required card is used to specify the stress, strain-rate/strain constitutive equation associated with the momentum equations (e.g. Navier-Stokes equations) and contains Newtonian and generalized Newtonian models. The single input parameter is the {model_name} with the options listed below:

{model_name} Name of the constitutive equation, being one of the following values: **NEWTONIAN, POWER_LAW, CARREAU, BINGHAM, CARREAU_WLF, CURE, THERMAL, EPOXY, SUSPENSION, FILLED_EPOXY, POWERLAW_SUSPENSION, CARREAU_SUSPENSION,** or **HERSCHEL_BULKLEY**. Each of these constitutive models require additional parameters that are entered via additional cards, as described below.

Thus,

NEWTONIAN For a simple constant viscosity Newtonian fluid. This model requires one floating point value,

μ, where μ is the viscosity in the chosen units for the problem and is entered with the *Viscosity* card.

POWER_LAW For a power law model. This model requires two parameters. The first, μ0, is the zero strain-rate limit of the viscosity and is entered with the *Low Rate Viscosity* card. The second, n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning). n is entered with the *Power Law Exponent* card. The form of the equation is

$$\mu = \mu_0 \dot{\gamma}^{n-1}$$

where is the second invariant of the shear-rate tensor. To obtain solutions with the power law model, it is best to start with a Newtonian initial guess since the viscosity becomes infinite at zero shear-rate.

CARREAU For a Carreau-Yasuda strain-rate thinning or thickening relation. This option requires five floating point values. The first, μ0, is the zero strain-rate limit of the viscosity and is entered with the Low Rate Viscosity card. The second, n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning). n is entered with the Power Law Exponent card. The third, μinf, is the high-strainrate limit to

the viscosity and is entered with the High Rate Viscosity card. The fourth, λ, is the time constant reflecting the strain-rate at which the transition between μ0 and μinf takes place. λ is entered with the Time Constant card. The fifth, a, is a dimensionless parameter that describes the transition between the low-rate and the power-law region and is entered with the Aexp card. The form of the equation is

$$\mu = \mu_{inf} + (\mu_0 - \mu_{inf})(1 + (\lambda\dot{\gamma})^a)^{\frac{(n-1)}{a}}$$

where is the second invariant of the shear-rate tensor.

BINGHAM For a Bingham-Carreau-Yasuda fluid. This option requires eight floating point values. It uses the same parameters as the CARREAU model with the addition of coefficients to describe the yield and temperature dependent behavior. The first, μ0, is the zero strain-rate limit of the viscosity and is entered with the Low Rate Viscosity card. The second, n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning). n is entered with the Power Law Exponent card. The third, μinf, is the high-strain-rate limit to the viscosity and is entered with the High Rate Viscosity card. The fourth, λ, is the time constant reflecting the strain-rate at which the transition between μ0 and μinf takes place. λ is entered with the Time Constant card. The fifth, a, is a dimensionless parameter that describes the transition between the low-rate and the power-law region and is entered with the Aexp card. The form of the equation is

$$\mu = a_T\left(\mu_{inf} + \left(\mu_0 - \mu_{inf} + \tau_y \frac{\left(1 - e^{-a_T\dot{\gamma}F}\right)}{a_T\dot{\gamma}}\right)\left(1 + (a_T\lambda\dot{\gamma})^a\right)^{\frac{(n-1)}{a}}\right)$$

where is a simplified temperature dependent shift factor that is expressed as an Arrhenius type temperature dependence of the following form:

$$a_T = e^{\frac{E_\mu}{R}\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)}$$

The exponent for the temperature dependence, Eμ/R, is input using the Thermal Exponent card. Tref is input using the Reference Temperature card in the thermal properties section of the material file. The stress at which the material yields is input with the Yield Stress card. The sharpness of the transition from the solid to fluid state, F, is indicated with the Yield Exponent card.

CARREAU_WLF An extension of the Carreau-Yasuda model to incorporate a temperature-dependent shift in shear-rate according to the Williams-Landel-Ferry equation (Hudson and Jones, 1993). The form of the equation is

where a_T is another form of the temperature-dependent shift factor:

Here is a thermal exponential factor (can be Arrhenius) and is input by the *Thermal Exponent* card; c_2 is the WLF constant 2 and is input by the *Thermal WLF Constant2* card. μ0, is the zero strain-rate limit of the viscosity and is entered with the *Low Rate Viscosity* card. n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning) and is entered with the *Power Law Exponent* card. $_{inf}$, is the high-strain-rate limit to the viscosity and is entered with the *High Rate Viscosity* card. λ, is the time constant reflecting the strain-rate at which the transition between μ0 and μinf takes place and is entered with the *Time Constant* card. a, is a dimensionless parameter that describes the transition between the low-rate and the power-law region and is entered with the *Aexp* card.

CURE For a model to increase the viscosity with the extent of reaction. The Cure model can be used to represent polymerizing systems whose viscosity depends on the extent of reaction. The form of the equation is

$$\mu = a_T\left[\mu_{inf} + (\mu_0 - \mu_{inf})\left(1 + (a_T\lambda\dot{\gamma})^a\right)^{\frac{(n-1)}{a}}\right]$$

$$\alpha_T = \exp\left[\frac{c_1(T_{ref} - T)}{c_2 + T - T_{ref}}\right]$$

This option requires four floating point values. The first, μ0, is the reference state viscosity and is entered with the *Low Rate Viscosity* card. The constant, $_g$, is entered with the *Cure Gel Point* card and marks the extent of reaction at the transition from the liquid to the solid state. The exponents *A* and *B* are entered with the *Cure A Exponent* and *Cure B Exponent* cards.

THERMAL For a temperature-dependent viscosity. This option, which requires two floating point values, can be used to represent fluids that change viscosity with temperature. The form of the equation is

where the reference state viscosity, μ0, is entered with the *Low Rate Viscosity* card. The exponent, Eμ/R, is specified using the *Thermal Exponent* card.

EPOXY For a thermal and curing component. The Epoxy model combines the temperature dependence of the **THERMAL** option with the extent of reaction dependence of the **CURE** option. The functional form of the equation is:

Five cards must be used to specify all the parameters for this model. The first, μ0, is the reference state viscosity and is entered with the *Low Rate Viscosity* card. The thermal exponent, Eμ/R, is specified using the *Thermal Exponent* card. The constant, $_g$, is entered with the *Cure Gel Point* card and marks the extent of reaction at the transition from the liquid to the solid state. The exponents *A* and *B* are entered with the *Cure A Exponent* and *Cure B Exponent* cards.

SUSPENSION For simulating a carrier fluid with high-volume fraction particles. This option invokes a concentrationdependent viscosity model useful in modeling solid suspensions. The functional form associated with this option is,

where μ0 is effectively the viscosity of the suspending fluid specified with the *Low Rate Viscosity* card, n is an exponent specified by the *Power Law Exponent* card and is typically less than zero. C_{max} is the "binding" solid concentration and is specified with the Suspension Maximum Packing card. Ci is the solid concentration and is tied to a convective-diffusion equation specified in the equation section of the Problem Description. The correct species number "i" is specified with the Suspension Species Number card. Note that for $C_i > C_{max}$ and n < 0, the model as written above is physically undefined. For concentrations in this range, a very large value for viscosity will be used, effectively solidifying the material.

FILLED_EPOXY This option combines the cure and thermal dependence of the **EPOXY** model with the solid volume fraction dependence of the **SUSPENSION** model. The functional form of this equation is

with the temperature T_g being calculated from

$$\mu = \mu_0\left(\frac{\alpha_g}{\alpha_g - \alpha}\right)^{A + B\alpha}$$

$$\mu = \mu_0 e^{\frac{E_\mu}{RT}}$$

$$\mu = \mu_0 e^{\frac{E_\mu}{RT}\left(\frac{\alpha_g}{\alpha_g - \alpha}\right)^{A+B\alpha}}$$

Here the viscosity now depends on extent of reaction, temperature and solid volume fraction. Nine cards must be specified to define the parameters for this option and are entered in the following manner. The first, μ0, is the reference state viscosity and is entered with the *Low Rate Viscosity* card. n is the exponent for suspension behavior and is specified by the *Power Law Exponent* card; it is typically less than zero. C_{max} is the "binding" solid concentration and is specified with the *Suspension Maximum Packing* card. C_i is the solid concentration and is tied to a convective-diffusion equation specified in the equation section of the previous chapter. The correct species number "i" is identified with the *Suspension Species Number* card. Here c_1 is a thermal exponential factor and is input by the Thermal Exponent card; c_2 is a second thermal exponent and is entered via the *Cure B Exponent* card. The constant for the curing model, $_g$, is entered with the *Cure Gel Point* card and marks the extent of reaction at the transition from the liquid to the solid state. The cure exponent used in the **EPOXY** model is here assumed to be constant (-4/3) and is fixed in the model. The constant A in the gel temperature equation is entered with the *Cure A Exponent* card and the temperature is entered with the *Unreacted Gel Temperature* card. Although it does not appear directly in the model equations, the *Cure Species Number* must also be specified.

POWERLAW_SUSPENSION This is a specialized research model that incorporates the power law model with the suspension model to try and simulate particles suspending in shear-thinning fluid. This option requires five input values. The first, μ0, is the zero strain-rate limit of the viscosity of the solvent and is entered with the *Low Rate Viscosity* card. The second, n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning). n is entered with the *Power Law Exponent* card. The third value is the exponent for the suspension Krieger model, which is input through the *Thermal Exponent*, m. The fourth term is the suspension maximum packing, C_{max}, which is entered through the *Suspension Maximum Packing* card. C_i is the solid concentration and is tied to a convectivediffusion equation specified in the equation section of the previous chapter. The correct species number "i" is identified with the *Suspension Species Number* card. The form of the equation is

where y is the second invariant of the shear-rate tensor. It is best to start with a Newtonian initial guess for the power law suspension model, since the viscosity for the power law model will become infinite at zero shear-rate.

CARREAU_SUSPENSION This model is a hybrid for the flow of particle-laden suspensions in shear-thinning fluids. It uses a Carreau-Yasuda strain-rate thinning or thickening relation for the suspending fluid and a Krieger model for the suspension. This option requires eight input values. The first, μ0, is the zero strain- rate limit of the viscosity and is entered with the Low Rate Viscosity card. The second, n, is the exponent on the strain rate which can take on any value between 1 (Newtonian) and 0 (infinitely shear thinning). n is entered with the *Power Law Exponent* card. The third, μinf, is the high-strain-rate limit to the viscosity and is entered with the *High Rate Viscosity* card. The fourth, λ, is the time constant reflecting the strain-rate at which the transition between μ0 and μinf takes place. λ is entered with the *Time Constant* card. The fifth, a, is a dimensionless parameter that describes the transition between the low-rate and the power-law region and is entered with the Aexp card. The sixth value is the exponent for the suspension Krieger model, which is input through the Thermal Exponent, m. The seventh term is the suspension maximum packing, Cmax, which is entered through the Suspension Maximum Packing card. Ci is the solid concentration and is tied to a convective-diffusion equation specified in the equation section of the previous chapter. The correct species number "i" is identified with the *Suspension Species Number* card.The form of the equation is

$$\mu = \mu_0 \left(1 - \frac{C_i}{C_{max}}\right)^n$$

$$\mu = \mu_0 \left(1 - \frac{C_i}{C_{max}}\right)^n \left(10^{\frac{-C_1(T-T_g)}{C_2+T-T_g}}\right) \left(1 - \left(\frac{\alpha}{\alpha_g}\right)^2\right)^{-1.333}$$

$$T_g = \frac{T_g^0}{1 - A\alpha}$$

where y is the second invariant of the shear-rate tensor.

HERSCHEL_BULKLEY This is a variant on the power law model that includes a yield stress. It requires three input values to operate: a reference viscosity value, μ0, a power-law exponent, n. and a yield shear stress value, $_y$. The model for this constitutive relations is as follows:

The nature of this relation is best seen by multiplying the entire relation by the shear rate to produce a relation between shear stress and shear rate. In this manner it can be seen that the shear stress does not go to zero for zero shear rate. Instead it approaches the yield shear stress value. Put another way, only for imposed shear stresses greater than the yield stress will the fluid exhibit a nonzero shear rate. This is effective yielding behavior.

A caveat needs stating at this point. This model is essentially a superposition of two power-law models. One with the supplied exponent and the other with an implicit exponent of n = 0. It has long been observed that power-law models with exponents approaching zero exhibit very poor convergence properties. The Herschel_Bulkley model is no exception. To alleviate these convergence problems somewhat, the sensitivities of the yield stress term with respect to shear rate has not been included in the Jacobian entries for this viscosity model. This helps in that it allows for convergence at most yield stress values, but also means that the iteration scheme no longer uses an exact Jacobian. The difference is seen in that this model will take relatively more iterations to converge to an answer. The user should expect this and not be too troubled (it's alright to be troubled a little).

Examples

The following is a sample card setting the liquid constitutive equation type to **NEWTONIAN** and demonstrates the required cards:

```
Liquid Constitutive Equation = NEWTONIAN
```

```
Viscosity = CONSTANT 1.00
```

The following is a sample card setting the liquid constitutive equation type to **POWER_LAW** and demonstrates the required cards:

```
Liquid Constitutive Equation = POWER_LAW
```

$$\mu = \mu_0 \dot{\gamma}^{n-1} \left(1 - \frac{C_i}{C_{max}}\right)^m$$

$$\mu = \left(\mu_{inf} + (\mu_0 - \mu_{inf})(1 + (\lambda\dot{\gamma})^a)^{\frac{(n-1)}{a}} \right) \left(1 - \frac{C_i}{C_{max}} \right)^m$$

$$\mu = \frac{\tau_y}{\dot{\gamma}} + \mu_0(\dot{\gamma})^{n-1}$$

```
Low Rate Viscosity= CONSTANT 1.
```

```
Power Law Exponent= CONSTANT 1.
```

The following is a sample card setting the liquid constitutive equation type to **CARREAU** and demonstrates the required cards:

```
Liquid Constitutive Equation = CARREAU
```

```
Low Rate Viscosity= CONSTANT 1.
```

```
Power Law Exponent= CONSTANT 1.
```

```
High Rate Viscosity= CONSTANT 0.001
```

```
Time Constant = CONSTANT 1.
```

```
Aexp = CONSTANT 1.
```

The following is a sample card setting the liquid constitutive equation type to **BINGHAM** and demonstrates the required cards:

```
Liquid Constitutive Equation = BINGHAM
```

```
Low Rate Viscosity= CONSTANT 10.00
```

```
Power Law Exponent= CONSTANT .70
```

```
High Rate Viscosity= CONSTANT 0.01
```

```
Time Constant = CONSTANT 100.
```

```
Aexp = CONSTANT 2.5
```

```
Thermal Exponent = CONSTANT 1.
```

```
Yield Stress = CONSTANT 5.
```

```
Yield Exponent = CONSTANT 1.0
```

```
Reference Temperature= CONSTANT 273.
```

The following is a sample card setting the liquid constitutive equation type to **CARREAU_WLF** and demonstrates the required cards:

```
Liquid Constitutive Equation = CARREAU_WLF
```

```
Low Rate Viscosity= CONSTANT 10.00
```

```
Power Law Exponent= CONSTANT .70
```

```
High Rate Viscosity= CONSTANT 0.01
```

```
Time Constant = CONSTANT 100.
```

```
Aexp = CONSTANT 2.5
```

```
Thermal Exponent = CONSTANT 1.
```

```
Thermal WLF Constant2 = CONSTANT 0.5
```

```
Reference Temperature= CONSTANT 273.
```

The following is a sample card setting the liquid constitutive equation type to **CURE** and demonstrates the required cards:

```
Liquid Constitutive Equation = CURE
```

```
Low Rate Viscosity= CONSTANT 1.
```

```
Power Law Exponent= CONSTANT 1.
```

The following is a sample card setting the liquid constitutive equation type to **THERMAL** and demonstrates the required cards:

```
Liquid Constitutive Equation = THERMAL
```

```
Low Rate Viscosity= CONSTANT 1.
```

```
Thermal Exponent= CONSTANT 9.
```

The following is a sample card setting the liquid constitutive equation type to **EPOXY** and demonstrates the required cards:

```
Liquid Constitutive Equation = EPOXY
```

```
Liquid Constitutive Equation = FILLED_EPOXY
```

```
Low Rate Viscosity= CONSTANT 1.e5
```

```
Thermal Exponent= CONSTANT 9.
```

```
Cure Gel Point = CONSTANT 0.8
```

```
Cure A Exponent= CONSTANT 0.3
```

```
Cure B Exponent= CONSTANT 43.8
```

The following is a sample card setting the liquid constitutive equation type to **SUSPENSION** and demonstrates the required cards:

```
Liquid Constitutive Equation = SUSPENSION
```

```
Low Rate Viscosity= CONSTANT 1.e5
```

```
Power Law Exponent = CONSTANT -3.0
```

```
Suspension Maximum Packing= CONSTANT 0.49
```

```
Suspension Species Number = 0
```

The following is a sample card setting the liquid constitutive equation type to **FILLED_EPOXY** and demonstrates the required cards:

```
Liquid Constitutive Equation = FILLED_EPOXY
```

```
Low Rate Viscosity = CONSTANT 1.e5
```

```
Power Law Exponent = CONSTANT -3.0
```

```
Thermal Exponent = CONSTANT 9.
```

```
Suspension Maximum Packing = CONSTANT 0.49
```

```
Suspension Species Number = 0
```

```
Cure Gel Point = CONSTANT 0.8
```

```
Cure A Exponent = CONSTANT 0.3
```

```
Cure B Exponent = CONSTANT 43.8
```

```
Cure Species Number = 2
```

```
Unreacted Gel Temperature = CONSTANT 243
```

The following is a sample card setting the liquid constitutive equation type to **POWERLAW_SUSPENSION** and demonstrates the required cards:

```
Liquid Constitutive Equation = POWERLAW_SUSPENSION
```

```
Low Rate Viscosity= CONSTANT 1.
```

```
Power Law Exponent= CONSTANT 1.
```

```
Thermal Exponent = CONSTANT -1.82
```

```
Suspension Maximum Packing= CONSTANT 0.68
```

```
Suspension Species Number= 0
```

The following is a sample card setting the liquid constitutive equation type to **CARREAU_SUSPENSION** and demonstrates the required cards:

```
Liquid Constitutive Equation = CARREAU_SUSPENSION
```

```
Low Rate Viscosity= CONSTANT 1.
```

```
Power Law Exponent= CONSTANT 1.
```

```
High Rate Viscosity= CONSTANT 0.001
```

```
High Rate Viscosity= CONSTANT 0.001
```

```
Time Constant = CONSTANT 1.
```

```
Aexp = CONSTANT 1.
```

```
Thermal Exponent = CONSTANT -1.82
```

```
Suspension Maximum Packing= CONSTANT 0.68
```

```
Suspension Species Number= 0
```

The following card gives an example of the **HERSCHEL_BULKLEY** model

```
Liquid Constitutive Equation = HERSCHEL_BULKLEY
```

```
Low Rate Viscosity = CONSTANT 0.337
```

```
Power Law Exponent = CONSTANT 0.817
```

```
Yield Stress = CONSTANT 1.39
```

Technical Discussion

See Description/Usage section for this card.

Theory

The **NEWTONIAN, POWER_LAW,** and **CARREAU** models are described in detail in Bird, et al. (1987). Details of the continuous yield stress model used in the Bingham- Carreau-Yasuda (**BINGHAM**) model, which is a Carreau model combined with a continuous yield stress model, can be found in Papanastasiou (1987).

References

Bird, R. B., Armstrong, R. C., and Hassager, O. 1987. Dynamics of Polymeric Liquids, 2nd ed., Wiley, New York, Vol. 1.

Hudson, N. E. and Jones, T. E. R., 1993. "The A1 project - an overview", Journal of Non-Newtonian Fluid Mechanics, 46, 69-88.

Papanastasiou, T. C., 1987. "Flows of Materials with Yield," Journal of Rheology, 31 (5), 385-404.

Papananstasiou, T. C., and Boudouvis, A. G., 1997. "Flows of Viscoplastic Materials: Models and Computation," Computers & Structures, Vol 64, No 1-4, pp 677-694.

Viscosity

```
Viscosity = {model_name} {float_list} [M/Lt]
```

Description / Usage

This card is used to specify the viscosity model for the liquid constitutive equation (see Sackinger et al., 1995). Definitions of the input parameters are as follows:

{model_name}	The name of the viscosity model, which can be one of the following: **CONSTANT, USER, USER_GEN**, or **FILL, LEVEL_SET, CONST_PHASE_FUNCTION**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below. Note that not all models employ a {float_list}.

Thus,

CONSTANT <float1>	This option specifies a constant viscosity for a Newtonian fluid. The {float_list} has a single value: • <float1> - value of viscosity
USER <float1>… <floatn	This option specifies that the viscosity will be given by a user-defined model; the model must be incorporated into *Goma* by modifying function "usr_viscosity" in file user_mp.c. The model parameters are entered in the {float_list} as <float1> through <floatn> and passed to the routine as an array.
USER_GEN <float1>… <floatn>	This option specifies that the viscosity will be given by a generalized user-defined model. This user-defined model must be incorporated by modifying the routine "usr_viscosity_gen" in the file user_mp_gen.c. Any number of parameters can be passed (via <float1> through <floatn>) in here.
FILL <float1> <float2>	The {float_list} for this option requires two values. It invokes a FILL dependent viscosity that is set to the value of *float1* if the FILL variable is 1 and *float2* if the FILL variable is 0.
LEVEL_SET <float1> <float2> <float3>	This model is used to vary the viscosity in the flow region when a level set function is used to track the boundary between two fluids using level set interface tracking. This choice assures a smooth transition in density across the zero level set contour. The {float_list} contains three values for this model, where: • <float1> Fluid viscosity in the negative regions of the level set function, μ- • <float2> Fluid viscosity in the positive regions of the level set function, μ+ • <float3> Length scale over which the transition occurs, α . If this parameter is set to zero, it will default to one-half the Level Set Length Scale value specified elsewhere in the input deck. **Note**: a better way to specify the identical viscosity model is to make use of the 2nd Level Set Viscosity card documented also in this manual.
CONST_PHASE_FUNCTION <floatlist> <float1> <float2>	This model is used to vary the viscosity in the flow regime when phase functions are used to track the motion of muliple phases. This choice assures a smooth transition in viscosity across the phase boundaries. The {float_list} contains a variable number of values that depend on the number phase functions being tracked, where: • <floatlist> list of float variables equal to the number of phase functions. These are the constant viscosities associated with each phase in order from 1 to number of phase functions. • <float1> Length scale over which the transition between one phases viscosity value to theother occurs, α . If this parameter is set to zero, it will default to one-half the Level Set Length Scale value already specified. • <float3> The "null" value for viscosity. This is the value for viscosity which will be assigned to those regions of the flow where all the phase functions are less than or equal to zero. The user should examine the **CONST_PHASE_FUNCTION** density model for a detailed description of the relations used to compute

Examples

The following is a sample card that sets the viscosity to USER:

```
Viscosity = USER 1. 1. 1. 1. 1.
```

```
Viscosity = LEVEL_SET 0.083 0.0001 0.1
```

Technical Discussion

The viscosity specified by this input card is used with the **NEWTONIAN** *Liquid Constitutive Equation*.

References

Sackinger, P. A., Schunk, P. R. and Rao, R. R. 1995. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125 (1996) 83-103.

Low Rate Viscosity

```
Low Rate Viscosity = CONSTANT <float> [M/Lt]
```

Description / Usage

This card is used to specify the model for the low-rate viscosity parameter for the **POWER_LAW, CARREAU, CARREAU_WLF, BINGHAM, SUSPENSION, THERMAL, CURE, EPOXY, FILLED_EPOXY, POWER-LAW_SUSPENSION** and **CARREAU_SUSPENSION** model options of the *Liquid Constitutive Equation* card. This is also the reference viscosity value in the **HERSCHEL_BULKLEY** constitutive equation.

Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the low-rate viscosity. • <float> - the value of the low-rate viscosity. This value is also called the zero strain-rate limit of the viscosity and in models is normally called μ0.
LEVEL_SET	Name of the model for level-set dependent low-rate viscosity. Allows for this viscosity level to be a function of the level-set field. Specifically used for changing the low-rate viscosity from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of viscosity in the negative regions of the level set function. • <float2> - the value of viscosity in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the low rate viscosity to 10:

```
Low Rate Viscosity = CONSTANT 10.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

References

No References.

Power Law Exponent

```
Power Law Exponent = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the power-law exponent parameter of the **POWER_LAW, CARREAU, BINGHAM, CARREAU_WLF, CURE, SUSPENSION, FILLED_EPOXY, POWERLAW_SUSPENSION, CARREAU_SUSPENSION,** and **HERSCHEL_BULKLEY** fluid options of the *Liquid Constitutive Equation* card.

Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the power-law exponent. • <float> - the value of the power-law exponent. This variable is normally *n* in the constitutive laws.
LEVEL_SET	Name of the model for level-set dependent power law exponent. Specifically used for changing the exponent from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of power-law exponent in the negative regions of the level set function. • <float2> - the value of power-law exponent in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the power law exponent to 0.2:

```
Power Law Exponent = CONSTANT 0.2
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

References

No References.

High Rate Viscosity

```
High Rate Viscosity = CONSTANT <float> [M/Lt]
```

Description / Usage

This card is used to specify the model for the high-rate viscosity parameter of the **CARREAU, BINGHAM, CARREAU_WLF** and **CARREAU_SUSPENSION** fluid options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the high-rate viscosity. • <float> - the value of the high-rate viscosity. This value is normally called μinf in models.
LEVEL_SET	Name of the model for level-set dependent high-rate viscosity. Allows for this viscosity level to be a function of the level-set field. Specifically used for changing the high-rate viscosity from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of viscosity in the negative regions of the level set function. • <float2> - the value of viscosity in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the high rate viscosity to 10.:

```
High Rate Viscosity = CONSTANT 10.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

References

No References.

Time Constant

```
Time Constant = CONSTANT <float> [t]
```

Description / Usage

This card is used to specify the model for the time constant parameter of the **CARREAU, BINGHAM, CARREAU_WLF** and **CARREAU_SUSPENSION** fluid options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the time constant. • <float> - the value of the time constant, λ.
LEVEL_SET	Name of the model for level-set dependent time constant. Allows for this time constant level to be a function of the level-set field. Specifically used for changing the time constant from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of time constant in the negative regions of the level set function. • <float2> - the value of time constant in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the time constant to 0.2.

```
Time Constant = CONSTANT 0.2
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Aexp

```
Aexp = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the Aexp parameter of the **CARREAU,BINGHAM, CARREAU_WLF** and **CARREAU_SUSPENSION** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for Aexp. • <float> - the value of the a exponent in the liquid constitutive models; also, a dimensionless parameter that describes the transition between the low-rate and the power-law region for the Carreau model (see Bird, et. al., 1987).
LEVEL_SET	Name of the model for level-set dependent Aexp parameter. Allows for this parameter level to be a function of the level-set field. Specifically used for changing the Aexp parameter from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of Aexp parameter in the negative regions of the level set function. • <float2> - the value of Aexp parramete in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets Aexp to 3.0:

```
Aexp = CONSTANT 3.0
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

References

Bird, R. B., Armstrong, R. C., and Hassager, O. 1987. Dynamics of Polymeric Liquids, 2nd ed., Wiley, New York, Vol. 1.

Thermal Exponent

```
Thermal Exponent = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify a thermal exponential factor for **CARREAU_WLF, BINGHAM, THERMAL, EPOXY, FILLED_EPOXY, POWERLAW_SUSPENSION** and **CARREAU_SUSPENSION** viscosity models, as selected in the *Liquid Constitutive Equation* card. The value represented by the thermal exponent varies between these liquid constitutive models; the appropriate values for each model is indicated below.

Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the thermal exponent. • <float> - the value of the thermal exponent for the viscosity model specified in the *Liquid Constitutive Equation* card. • for the **BINGHAM, THERMAL, EPOXY,** or **FILLED_EPOXY** model, – <float> - the Eμ/R parameter. This has the dimensions of temperature in whatever units are consistent with the problem and describes the thinning of viscosity with temperature. • for the **CARREAU_WLF** model model, – <float> - the c1 constant of the equation for the temperature-dependent shift factor. • for the **POWERLAW_SUSPENSION** or **CARREAU_SUSPENSION** model, – <float> - the exponent for the Krieger viscosity model, *m*.
LEVEL_SET	Name of the model for level-set dependent thermal exponent factor. Allows for this exponent level to be a function of the level-set field. Specifically used for changing the thermal exponent from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of thermal exponent in the negative regions of the level set function. • <float2> - the value of thermal exponent in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the thermal exponent to 0.5.

```
Thermal Exponent = CONSTANT 0.5
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Thermal WLF Constant2

```
Thermal WLF Constant2 = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify the thermal constant 2 of the **CARREAU_WLF** viscosity model in the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for Thermal Constant2. • <float> - the value of c_2, in the equation representing the temperature-dependent shift factor for the **CARREAU_WL** constitutive model.
LEVEL_SET	Name of the model for level-set dependent WLF thermal constant 2. Allows for this thermal constant 2 level to be a function of the level-set field. Specifically used for changing the thermal constant 2 from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of thermal constant 2 in the negative regions of the level set function. • <float2> - the value of thermal constant 2 in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.

Examples

The following is a sample card that sets the *Thermal WLF Constant2* to 0.1.

```
Thermal WLF Constant2 = CONSTANT 0.1
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Yield Stress

```
Yield Stress = CONSTANT <float> [M/Lt2]
```

Description / Usage

This card is used to specify the model for the yield stress parameter, $_y$, of the **BINGHAM** and **HERSCHEL_BULKLEY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the yield stress. • <float> - the value of the yield stress, τy, which has the dimensions of stress, in whatever units are consistent with the problem and marks the transition from solid-like to fluid-like behavior for the Bingham-Carreau-Yasuda model.

Examples

The following is a sample card that sets the yield stress to 100:

```
Yield Stress = CONSTANT 100.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Yield Exponent

```
Yield Exponent = CONSTANT <float> [t]
```

Description / Usage

This card is used to specify the model for the yield exponent parameter, *F*, for the **BINGHAM** model option of the *Liquid Constitutive Equation* card, or when the *Polymer Constitutive Equation* card is **SARAMITO_OLDROYDB**, **SARAMITO_GIESEKUS**, or **SARAMITO_PTT**. Definitions of the input parameters are as follows:

CONSTANT <float> Name of the model for the yield exponent. <float> the value of the yield exponent, *F*, which has the dimensions of inverse shear-rate in whatever units are consistent with the problem of interest and which connotes the steepness of the transition from solid to fluid behavior for the Bingham-Carreau-Yasuda model or the Saramito yield stress model.

For the **BINGHAM** model, if *F* is large, the material has an abrupt transition from solid-like to fluid-like behavior, whereas for a small *F*, the transition is more gradual.

For the **SARAMITO_OLDROYDB**, **SARAMITO_GIESEKUS**, and **SARAMITO_PTT** models, the material has and abrupt transition when *F* equals zero. This Transistion becomes smooth for nonzero when *F* is greater than zero, with the transition becoming more gradual as *F* increases.

Examples

The following is a sample card that sets the yield exponent to 10.0

```
Yield Exponent = CONSTANT 10.0.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* and *Polymer Constitutive Equation* cards.

Suspension Maximum Packing

```
Suspension Maximum Packing = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the $Cmax$ parameter of the **SUSPENSION** and **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for suspension maximum packing. • <float> - the value of $Cmax$, which is the mass fraction at which the suspension begins to act as a solid.

Examples

The following is a sample card that sets the suspension maximum packing:

```
Suspension Maximum Packing = CONSTANT 0.68.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Suspension Species Number

```
Suspension Species Number = <integer>
```

Description / Usage

This card is used to specify the value of the species number "*i*" of the **SUSPENSION** and **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

<integer> - the species number "*i*".

Examples

The following is a sample card that sets the suspension species number to 1:

```
Suspension Species Number = 1
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Cure Gel Point

```
Cure Gel Point = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the $_g$ parameter for the **CURE**, **EPOXY**, and **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the $_g$ parameter. • <float> - the value of $_g$, which is the extent of reaction at the gel point of a polymerizing system.

Examples

The following is a sample card that sets the cure gel point to 0.75:

```
Cure Gel Point = CONSTANT 0.75
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Cure A Exponent

```
Cure A Exponent = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the A exponent of the **CURE**, **EPOXY**, and **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the *A* exponent. • <float> - the value of *A*.

Examples

The following is a sample card that sets the cure *A* exponent to 1.0:

```
Cure A Exponent = CONSTANT 1.0
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Cure B Exponent

```
Cure B Exponent = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the B exponent of the **CURE**, **EPOXY**, and **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the *B* exponent. • <float> - the value of *B*.

Examples

The following is a sample card that sets the cure *B* exponent to 0.1:

```
Cure B Exponent = CONSTANT 0.1.
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Cure Species Number

```
Cure Species Number = <integer>
```

Description / Usage

This card is used to specify the species number, e.g., the *i* in C_i, for the **FILLED_EPOXY** model options of the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

<integer> - the value of the species equation, *i*, associated with tracking the curing species.

Examples

The following is a sample card that sets the cure species number to 0.

```
Cure Species Number = 0
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Unreacted Gel Temperature

```
Unreacted Gel Temperature = CONSTANT <float>
```

Description / Usage

This card is used to specify the model for the unreacted gel temperature parameter for the **FILLED_EPOXY** fluid option of the *Liquid Constitutive Equation* card.

Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the unreacted gel temperature. • <float> - the value of the unreacted gel temperature,:math:*T_{g0}*.

Examples

The following is a sample card that sets the unreacted gel temperature to 273.0:

```
Power Law Exponent = CONSTANT 273.0
```

Technical Discussion

See Description/Usage for *Liquid Constitutive Equation* card.

Polymer Constitutive Equation

```
Polymer Constitutive Equation = {model_name}
```

Description / Usage

This required card is used to specify the polymer constitutive equation. A single input parameter must be defined, that being the {model_name}.

{model_name} Name of the constitutive equation model, being one of the following values: **NOPOLYMER, OLDROYDB, GIESEKUS, PTT, SARAMITO_OLDROYDB, SARAMITO_GIESEKUS, SARAMITO_PTT, WHITE-METZNER**. Several of these polymer constitutive models require additional parameters for the polymer properties that are entered via additional cards, as described below. Please see the Example sectionand the tutorial referenced below.

Thus,

NOPOLYMER For Newtonian and generalized Newtonian models. No floating point values are required.

OLDROYDB For the Oldroyd-B constitutive model. This option requires four floating point values, which are described below.

GIESEKUS For the Giesekus model. This option requires five floating point values, which are described below.

PTT For the Phan-Thien Tanner model. This option requires six floating point values, which are described below.

SARAMITO_OLDROYDB For the Oldroyd-B model used with the Saramito yield model. This option requires six floating point values, which are described below.

SARAMITO_GIESEKUS For the Giesekus model used with the Saramito yield model. This option requires seven floating point values, which are described below.

SARAMITO_PTT For the Giesekus model used with the Saramito yield model. This option requires eight floating point values, which are described below.

WHITE_METZNER For the White-Metzner model. This option is not currently working.

Examples

The following is a sample card that sets the polymer constitutive equation to **NOPOLYMER**. This option does not require any additional cards since it indicates that there is no polymer constitutive equation present.

```
Polymer Constitutive Equation = NOPOLYMER
```

The following is a sample card that sets the polymer constitutive equation to **OLDROYDB**. This option requires four cards describing the polymer stress formulation, weight function, viscosity and time constant.

```
Polymer Constitutive Equation = OLDROYDB
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 1.
```

```
Polymer Time Constant = CONSTANT 0.02
```

The following is a sample card that sets the polymer constitutive equation to **GIESEKUS**. This option requires five cards describing the polymer stress formulation, weight function, viscosity, time constant and mobility parameter.

```
Polymer Constitutive Equation = GIESEKUS
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 1.
```

```
Polymer Time Constant = CONSTANT 0.2
```

```
Mobility Parameter = CONSTANT 0.1
```

The following is a sample card that sets the polymer constitutive equation to **PHANTHIEN TANNER** (or **PTT**). This option requires six additional cards that set the polymer stress formulation, weight function for the stress equation, viscosity, time constant and nonlinear PTT parameters.:

```
Polymer Consitutive Equation = PTT
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 8000.
```

```
Polymer Time Constant = CONSTANT 0.01
```

```
PTT Xi parameter = CONSTANT 0.10
```

```
PTT Epsilon parameter = CONSTANT 0.05
```

The following is a sample card that sets the polymer constitutive equation to **SARAMITO_OLDROYDB**. This option requires six cards describing the polymer stress formulation, weight function, viscosity, time constant, yield stress, and yield exponent.

```
Polymer Constitutive Equation = SARAMITO_OLDROYDB
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 1.
```

```
Polymer Time Constant = CONSTANT 0.02
```

```
Polymer Yield Stress = CONSTANT 15.
```

```
Yield Exponent = CONSTANT 0.
```

The following is a sample card that sets the polymer constitutive equation to **SARAMITO_GIESEKUS**. This option requires seven cards describing the polymer stress formulation, weight function, viscosity, time constant, mobility parameter, yield stress, and yield exponent.

```
Polymer Constitutive Equation = SARAMITO_GIESEKUS
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 1.
```

```
Polymer Time Constant = CONSTANT 0.2
```

```
Polymer Yield Stress = CONSTANT 12.
```

```
Yield Exponent = CONSTANT 1.0
```

```
Mobility Parameter = CONSTANT 0.1
```

The following is a sample card that sets the polymer constitutive equation to **SARAMITO_PTT**. This option requires eight additional cards that set the polymer stress formulation, weight function for the stress equation, viscosity, time constant, nonlinear PTT parameters, yield stress, and yield exponent.

```
Polymer Consitutive Equation = SARAMITO_PTT
```

```
Polymer Stress Formulation = EVSS_F
```

```
Polymer Weight Function = GALERKIN
```

```
Polymer Viscosity = CONSTANT 8000.
```

```
Polymer Time Constant = CONSTANT 0.01
```

```
Polymer Yield Stress = CONSTANT 200.
```

```
Yield Exponent = CONSTANT 0.5
```

```
PTT Xi parameter = CONSTANT 0.10
```

```
PTT Epsilon parameter = CONSTANT 0.05
```

The following is a sample card that sets the polymer constitutive equation to **WHITE_METZNER**. This option is not currently functional for multimode viscoelasticity. If needed it could be resurrected with only minimal changes to the input parser.

```
Polymer Consitutive Equation = WHITE_METZNER
```

Technical Discussion

The viscoelastic tutorial is helpful for usage issues such as extensions from single mode to multimodes.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

Polymer Stress Formulation

```
Polymer Constitutive Equation = {model_name}
```

Description / Usage

This card specifies which formulation of the polymer constitutive equation should be used. Valid options are

EVSS_C	Uses the classic elastic-viscous stress splitting of Rajagopalan (1990) where the stress is the elastic stress only without a Newtonian component. This option is the default if this *Polymer Stress Formulation* card is not supplied. This formulation is almost never used.
EVSS_F	Uses the EVSS formulation of Guenette and Fortin (1995) that solves the standard stress equation with the addition of a new term to the momentum equation. This formulation is used most often.
EVSS_L	Uses a research formulation for viscoelasticity that includes a level set discretization that switches the equations from solid to fluid. This option is not currently in production usage.

Examples

The following is a sample card that sets the polymer stress formulation to EVSS_F:

```
Polymer Stress Formulation = EVSS_F
```

Technical Discussion

No Discussion.

References

Guenette, R. and M. Fortin, "A New Mixed Finite Element Method for Computing Viscoelastic Flow," J. Non-Newtonian Fluid Mech., 60 (1995) 27-52.

Rajagopalan, D., R. C. Armstrong and R. A. Brown, "Finite Element Methods for Calculation of Viscoelastic Fluids with a Newtonian Viscosity", J. Non-Newtonian Fluid Mech., 36 (1990) 159-192.

Polymer Weight Function

```
Polymer Weight Function = {GALERKIN | SUPG}
```

Description / Usage

This optional card is used to specify the weight function for the polymer stress equation. Valid options are

GALERKIN	Uses a Galerkin weight-function for the stress equation. This option is the default if this card is not present.
SUPG	Uses a streamline upwind Petrov-Galerkin weight-function for the stress equation. If this option is chosen, a weight must be specified via the Polymer Weighting card.

Examples

The following is a sample card that set the polymer weight function to **SUPG** and demonstrates the required cards.

```
Polymer Weight Function = SUPG
```

```
Polymer Weighting = CONSTANT 0.1
```

The following is a sample card that set the polymer weight function to **GALERKIN**.

```
Polymer Weight Function = GALERKIN
```

Technical Discussion

No Discussion.

Polymer Shift Function

```
Polymer Shift Function = {CONSTANT | MODIFIED_WLF} <float1> [float2]
```

Description / Usage

This optional card is used to specify the temperature shift function for the polymer relaxation times and viscosities in the polymer stress equation(s);

$$\lambda_k(T) = a(T)\lambda_k'$$

$$\eta_k(T) = a(T)\eta_k'$$

Valid options are

CONSTANT	Applies a constant temperature shift factor to the polymer relaxation time(s) and the polymer viscosities. • <float1> - the temperature shift factor. If this card is not present, this option is the default and a shift factor of 1.0 is applied.

This option may be useful for continuation in elasticity level since continuation in this parameter will uniformly increase or decrease the relaxation time(s) and viscosities of all viscoelastic modes.

MODIFIED_WLF	Applies a temperature shift factor which is a modified version of the Williams-Landel-Ferry shift model (cf. Bird, Armstrong, and Hassager 1987, pp.139-143); $a(T) = \exp\left[\frac{C_1(T_{ref}-T)}{C_2+T-T_{ref}}\right]$ The reference temperature, Tref, is taken from the Reference Temperature card. Note that if C2 is chosen equal to Tref, this model reduces to an Arrhenius form where C1 = Eμ/RTref. Also note that this form is based on the exponential function whereas the WLF model is based on 10x. • <float1> - constant C1 • <float2> - constant C2

Examples

The following is a sample card that sets a constant temperature shift.

```
Polymer Shift Function = CONSTANT 1.0
```

The following is a sample card that utilizes the modified WLF shift function.

```
Polymer Shift Function = MODIFIED_WLF 2.5 95.0
```

Technical Discussion

No Discussion.

References

Bird, R. B., Armstrong, R. C., and Hassager, O. Dynamics of Polymeric Liquids, Volume 1. John Wiley & Sons, Inc. 1987.

Polymer Shock Function

```
Polymer Shock Function = {NONE | DCDD <float1>}
```

Description / Usage

This optional card is used to specify the shock function for the polymer stress equation. Valid options are

NONE No polymer shock function is used, default

DCDD DCDD shock capturing term is used * <float1> the scaling value of the shock capturing term

Currently only available for log-conformation formulations.

Examples

The following is a sample card that set the polymer shock function to **SUPG** and demonstrates the required cards.

```
Polymer Shock Function = DCDD 1
```

The following is a sample card that set the polymer shock function to **NONE**.

```
Polymer Shock Function = NONE
```

Technical Discussion

No Discussion.

Polymer Weighting

```
Polymer Weighting = <float> [t/L]
```

Description / Usage

This card is only used if the value of the *Polymer Weight Function* card is **SUPG**. The single input parameter is defined as

<float> - scale factor for the upwind term in the Petrov-Galerkin formulation. If this is set to zero, a Galerkin weight function is used. The correct scaling for this term is the inverse of the average inflow velocity.

Examples

The following is a companion pair of sample input cards that includes setting the polymer weighting to 0.1:

```
Polymer Weight Function = SUPG
```

```
Polymer Weighting = 0.1
```

Technical Discussion

No Discussion.

References

No References.

Discontinuous Jacobian Formulation

```
Discontinuous Jacobian Formulation = {model_name} <float>
```

Description / Usage

This optional card is used to specify the off element Jacobian contributions for the discontinuous Galerkin (DG) discretization of the polymer stress equation. These terms are important because the DG method uses stress information from upstream elements to determine the flux in the current element. If the off element Jacobians are not included, convergence is poor, but including these terms greatly increases the complexity of the code, the matrix bandwidth and the matrix solution time.

The default sets this option to false, implying that no off element Jacobians are included. Valid options for {model_name} are:

FULL	adds in the full complement of off-element Jacobians; no floating point data required. This option does not always work in parallel computations.
EXPLICIT	approximates the off-element Jacobians by adding terms to the residual equation based on the previous iteration. • <float> - scales the lagged term.
SEGREGATED	approximates the off-element Jacobians by adding terms to the residual equation based on a mass lumping at the current iteration. • <float> - scales the lumped term.

Examples

The following is a sample card that set the discontinuous Jacobian formulation to full.

```
Discontinuous Jacobian Formulation = FULL
```

The following is a sample card that set the discontinuous Jacobian formulation to explicit. Note this is more of a research option than a production one and the choice of scaling requires tuning for each problem.

```
Discontinuous Jacobian Formulation = EXPLICIT 0.1
```

The following is a sample card that set the discontinuous Jacobian formulation to segregated. Note this is more of a research option than a production one and the choice of scaling requires tuning for each problem.

```
Discontinuous Jacobian Formulation = SEGREGATED 0.2
```

Technical Discussion

For a discussion of the discontinuous Galerkin method see Fortin and Fortin (1989), Baaijens (1994) or Baaijens (1998). Internal (Sandia) users may find T. A. Baer's Gordon Conference presentation (1997) helpful.

References

Baaijens, F. P. T. , "Application of Low-Order Discontinuous Galerkin Method to the Analysis of Viscoelastic Flows," J. Non-Newtonian Fluid Mech., 52, 37-57 (1994).

Baaijens, F. P. T., "An Iterative Solver for the DEVSS/DG Method with Application to Smooth and Non-smooth Flows of the Upper Convected Maxwell Fluid," J. Non- Newtonian Fluid Mech., 75, 119-138 (1998).

Fortin, M. and A. Fortin, "A New Approach for the FEM Simulations of Viscoelastic Flow, J. Non-Newtonian Fluid Mech., 32, 295-310 (1989).

Adaptive Viscosity Scaling

```
Adaptive Viscosity Scaling = CONSTANT <float>
```

Description / Usage

This optional card is used to specify the adaptive viscosity scaling and the ε parameter associated with its usage (see theory section below). It requires one floating point number that scales the term. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the adaptive viscosity scaling. • <float> - value of ε scaling parameter.

Examples

The following is a sample card that sets the adaptive viscosity scaling to 0.5:

```
Adaptive Viscosity Scaling = CONSTANT 0.5
```

Technical Discussion

The momentum equation is modified with the addition of a numerical adaptive viscosity to help maintain the elliptic character of the equation set as stress and velocity gradient increase

$$\nabla\bullet(\eta_s+\eta_p)\eta_a\gamma+\nabla\bullet\tau-\nabla p-\nabla\bullet(\eta_s+\eta_p)\eta_a(G+G^t)\ =\ 0$$

where ηs is the solvent viscosity and ηp is the polymer viscosity. If we set the adaptive viscosity to zero (ηa= 0), we obtain the Standard EVSS Formulation of Guenette and Fortin (1995). For adaptive viscosity, we use the following definition

$$\eta_a = \frac{\sqrt{1 + \frac{\varepsilon}{2}\tau \bullet \tau}}{\sqrt{1 + \frac{\varepsilon}{2}(G + G^t) \bullet (G + G^t)}}$$

with 0<ε<1.

The equations are unchanged in the limit of h, the element size, going to zero.

Please see the viscoelastic tutorial for a discussion of usage for the adaptive viscosity scaling. The papers by Sun, et. al. (1996) and Sun, et. al (1999) provide a good discussion of the theory behind its usage. CRMPC presentations by R.R. Rao demonstrates its usefulness for *Goma* calculations.

References

GT-014.1: Tutorial for Running Viscoelastic Flow Problems with GOMA, June 21, 2000, R. R. Rao

Guenette, R. and M. Fortin, "A New Mixed Finite Element Method for Computing Viscoelastic Flows," J. Non-Newtonian Fluid Mech., 60, 27-52 (1995).

Sun, J., N. Phan-Thien, R. I. Tanner, "An Adaptive Viscoelastic Stress Splitting Scheme and Its Applications: AVSS/SI and AVSS/SUPG," J. Non-Newtonian Fluid Mech., 65, 75-91 (1996).

Sun, J., M. D. Smith, R. C. Armstrong, R. A. Brown, "Finite Element Method for Viscoelastic Flows Bases on the Discrete Adaptive Viscoelastic Stress Splitting and the Discontinuous Galerkin Method: DAVSS-G/DG," J. Non-Newtonian Fluid Mech., 86, 281-307 (1999).

Polymer Viscosity

```
Polymer Viscosity = {model_name} <float> [M/Lt]
```

Description / Usage

This card is used to specify the polymer viscosity associated with the model set in the *Polymer Constitutive Equation* card. This is a required card for the **OLDROYDB**, **GIESEKUS** and **PTT** models.

Definitions of the input parameters are as follows:

{model_name}	Permissible names for the viscosity model are **CONSTANT, POWER_LAW** and **CARREAU**.
CONSTANT	a simple constant viscosity, Newtonian fluid.
POWER_LAW	a power-law model
CARREAU	a Carreau strain-rate thinning or thickening relation

Input parameters are not identified for the latter two models as they have not worked since the multimode port. They could be made to work again if the proper tweaking is done to the input parser, but are not currently functional.

Examples

The following is a sample card that sets the polymer viscosity to 8000.0:

```
Polymer Viscosity = CONSTANT 8000.0
```

Technical Discussion

No Discussion.

References

Polymer Time Constant

```
Polymer Time Constant = {model_name} <float> [t]
```

Description / Usage

This card is used to specify the polymer time constant associated with the *Polymer Constitutive Equation* card. It is a required card for the **OLDROYDB, GIESEKUS** and **PTT** options. Definitions of the input parameters are as follows:

Definitions of the input parameters are as follows:

{model_name}	Permissible names for the viscosity model are **CONSTANT, POWER_LAW** and **CARREAU**.
CONSTANT	a simple constant viscosity, Newtonian fluid.
POWER_LAW	a power-law model
CARREAU	a Carreau strain-rate thinning or thickening relation

Input parameters are not identified for the latter two models as they have not worked since the multimode port. They could be made to work again if the proper tweaking is done to the input parser, but are not currently functional.

If the polymer time constant varies with properties, it must do so in the same way as the polymer viscosity; thus, the model on this card must be the same as the model selected on the *Polymer Viscosity* card.

All three models are described in detail in Bird, et. al. (1987).

Examples

The following is a sample card that sets the polymer time constant to 1.0:

```
Polymer Time Constant = CONSTANT 1.0
```

Technical Discussion

No Discussion.

References

Bird, R. B., Armstrong, R. C., and Hassager, O. 1987. Dynamics of Polymeric Liquids, 2nd ed., Wiley, New York, Vol. 1.

Polymer Yield Stress

```
Polymer Yield Stress = CONSTANT <float> []
```

Description / Usage

This card is required when using the Saramito yield model. The card should be included in the input when the option selected for the *Polymer Constitutive Equation* card is **SARAMITO_OLDROYDB**, **SARAMITO_GIESEKUS**, or **SARAMITO_PTT**. Definitions of the input parameters are as follows:

CONSTANT <float> Name of the model for the yield stress. * <float> - the value of the yield stress.

This card does not have to be present for constitutive equations other than **SARAMITO_OLDROYDB**, **SARAMITO_GIESEKUS**, and **SARAMITO_PTT**

Examples

The following is a sample card that sets the polymer yield stress to 12:

```
Polymer Yield Stress = CONSTANT 12
```

Mobility Parameter

```
Mobility Parameter = CONSTANT <float> []
```

Description / Usage

This card is used in the Giesekus model in the nonlinear stress terms. The card should be included in the input when the option selected for the *Polymer Constitutive Equation* card is **GIESEKUS**. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the mobility parameter. • <float> - the value of the mobility parameter.

This card does not have to be present for constitutive equations other than **GIESEKUS**.

Examples

The following is a sample card that sets the mobility parameter to 0.2:

```
Mobility Parameter = CONSTANT 0.2
```

Technical Discussion

No Discussion.

PTT Xi parameter

```
PTT Xi parameter = {model_name} <float>
```

Description / Usage

This card is used in the Phan-Thien Tanner model in the nonlinear stress terms. The card should be included in the input when the option selected for the *Polymer Constitutive Equation* card is **PTT**. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for PTT Xi parameter. • <float> - the value of the PTT Xi parameter.

This card does not have to be present for constitutive equations other than **PTT**.

Examples

The following is a sample card that sets the PTT Xi parameter to 0.1:

```
PTT Xi parameter = CONSTANT 0.10
```

Technical Discussion

No Discussion.

PTT Epsilon parameter

```
PTT Epsilon parameter = {model_name} <float>
```

Description / Usage

This card is used in the Phan-Thien Tanner model in the nonlinear stress terms. The card should be included in the input when the option selected for the *Polymer Constitutive Equation* card is **PTT**. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for PTT Epsilon parameter. • <float> - the value of the PTT Epsilon parameter.

This card does not have to be present for constitutive equations other than **PTT**.

Examples

The following is a sample card that sets the PTT Epsilon parameter to 0.1:

```
PTT Xi parameter = CONSTANT 0.10
```

Technical Discussion

No Discussion.

Surface Tension

```
Surface Tension = {CONSTANT | DILATION | USER} <float_list> [M/t2]
```

Description / Usage

This card is used to specify the interfacial surface tension of the fluid which enters into the *CAPILLARY* boundary condition and *CAP_ENDFORCE* boundary condition cards. The surface tension, albeit a property of an interface and not of a bulk material, is sometimes influenced by thermophysical phenomena associated with a material, hence the inclusion of this card in the material file. It should be mentioned that this card is optional, and if it does not appear the surface tension is taken off the aforementioned boundary condition cards. *PLEASE see the important technical discussion below if you plan on using this card.* Definitions of the input parameters are as follows:

CONSTANT	Name of the model for a constant value of interfacial surface tension of the fluid: • <float1> - constant value of surface tension
DILATION	Name of a surface tension model that depends on mesh dilation (only useful if the free surface is constrained to be a material surface both normally and tangentially, see Schwartz ,et. al. 1996). The model mathematically is
USER	A user-defined surface tension model that is defined in the user-supplied routine usr_surface_tension in the file user_mp.c. This model will have an arbitrary number of user-defined parameters (<float1> to floatn>).

WARNING: When specifying surface tension on this card, be sure the surface tension (multiplier) on the boundary condition CAPILLARY card is set to 1. In other words, the value of surface tension on the boundary condition cards is multiplied with the value on this card before the calculation is carried out.

Examples

Following is a sample card:

```
Surface Tension = DILATION 70.0 1.
```

Technical Discussion

Please read and understand the warning issued above regarding the proper place to specify surface tension. Basically, for constant surface tension models, it is a good idea to leave this card out and simply enter the proper surface tension value for the current surface on the boundary condition cards **CAPILLARY** and **CAP_ENDFORCE**. For variable models, please set the surface tension values on these BC cards to 1.0, and then handle your model through this card. The surface tension is a thermodynamic property of the interface and actually depends on the chemical composition of the fluids (or fluid/solids) of the bounding phases. The property controls the importance of the capillary stress jump on a curved interface on the hydrodynamics of the flow and the meniscus position and motion.

References

GT-001.4: GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

Second Level Set Conductivity

```
Second Level Set Conductivity = {model_name} {float_list} {char_string} [M/Lt]
```

Description / Usage

This card allows to the user to specify a second thermal conductivity model that will be applied to one side of a level set interfacial curve:

{model_name}	The name of the conductivity model can only be **CONSTANT** at the current time.
{float1}	This is a single float parameter which is the value of Fourier thermal conducticity applied to the second level set phase fluid.
{char_string}	This string may take the values POSITIVE or NEGATIVE. It identifies which side of the interface the preceding conductivity model is applied to.

This card allows the user to apply a CONSTANT or USER model to one side of the interface while the other side recieves the constant conductivity value listed on this card. The side of the interface that corresponds to char_string appearing on this card receives the constant conductivity value. The opposite side's conductibity is determined from the other, (possibly) more complex model. Transition between them is accomplished using smooth Heaviside functions whose width is given on the Level Set Length Scale card. Note that it is the prescence of the this card in the material file that actually activates this selection process.

Examples

The following is a usage example for this card:

```
Conductivity = USER 1.e4 0.1 3.0
```

```
Second Level Set Conductivity = CONSTANT. 1.0e-4 POSITIVE
```

This setup will cause the negattive side of the interface to receive conductivity values obtained from the USER model with the parameters listed above . The positive side of the interface will show a constant conductivity of 1.0e-4.

Technical Discussion

No Discussion.

References

No References.

Second Level Set Density

```
Second Level Set Density = {model_name} {float_list} {char_string} [M/Lt]
```

Description / Usage

This card allows to the user to specify a second density model that will be applied to one side of a level set interfacial curve:

{model_name}	The name of the density model can only be **CONSTANT** at the current time.
{float1}	This is a single float parameter which is the value of density applied to the second level set phase fluid.
{char_string}	This string may take the values POSITIVE or NEGATIVE. It identifies which side of the interface the preceding density model is applied to.

This card allows the user to apply one of the several complex density models currently available in Goma to one side of the interface while the other side recieves the constant density value listed on this card. The side of the interface that corresponds to char_string appearing on this card recieves the constant density value. The opposite sides density is determined from the other, more complex model. Transition between them is accomplished using smooth Heaviside functions whose width is given on the Level Set Length Scale card. Note that it is the prescence of the this card in the material file that actually activates this selection process.

Examples

The following is a usage example for this card:

```
Density = SUSPENSION 1.0 1.0 1.0
```

```
Second Level Set Density = CONSTANT. 1.0 POSITIVE
```

This setup will cause the negattive side of the interface to receive density values obtained from the SUSPENSION model with the parameters listed above . The positive side of the interface will show a constant density of 1.0.

Technical Discussion

No Discussion.

Second Level Set Heat Capacity

```
Second Level Set Heat Capacity = {model_name} {float_list} {char_string} [M/Lt]
```

Description / Usage

This card allows to the user to specify a second thermal heat capacity model that will be applied to one side of a level set interfacial curve:

{model_name}	The name of the heat capacity model can only be **CONSTANT** at the current time.
{float1}	This is a single float parameter which is the value of heat capacity applied to the second level set phase fluid.
{char_string}	This string may take the values POSITIVE or NEGATIVE. It identifies which side of the interface the preceding heat capacity model is applied to.

This card allows the user to apply a CONSTANT or USER model to one side of the interface while the other side recieves the constant heat capacity value listed on this card. The side of the interface that corresponds to char_string appearing on this card receives the constant heat capacity value. The opposite side's heat capacity is determined from the other, (possibly) more complex model. Transition between them is accomplished using smooth Heaviside functions whose width is given on the Level Set Length Scale card. Note that it is the prescence of the this card in the material file that actually activates this selection process.

Examples

The following is a usage example for this card:

```
Heat Capacity = ENTHALPY 1.e4 0.1
```

```
Second Level Set Heat Capacity = CONSTANT. 1.0 POSITIVE
```

This setup will cause the negattive side of the interface to receive heat capacity values obtained from the USER model with the parameters listed above . The positive side of the interface will show a constant heat capacity of 1.0e-4.

Technical Discussion

No Discussion.

Second Level Set Momentum Source

```
Second Level Set Momentum Source = {model_name} {float_list} {char_string} [M/Lt]
```

Description / Usage

This card allows to the user to specify a second thermal Navier-Stokes volumetric momentum source model that will be applied to one side of a level set interfacial curve:

{model_name}	The name of the momentum source model can only be **CONSTANT** at the current time.
{float1}	This is a single float parameter which is the value of volumetric momentum source term applied to the second level set phase fluid.[F/L3]
{char_string}	This string may take the values POSITIVE or NEGATIVE. It identifies which side of the interface the preceding momentum source model is applied to.

This card allows the user to apply one of the several momentum source models implement in Goma to one side of the interface while the other side recieves the constant momentum source value listed on this card. The side of the interface that corresponds to char_string appearing on this card receives the constant momentum source value. The opposite side's momentum source is determined from the other, (possibly) more complex model. Transition between them is accomplished using smooth Heaviside functions whose width is given on the Level Set Length Scale card. Note that it is the prescence of the this card in the material file that actually activates this selection process.

Examples

The following is a usage example for this card:

```
Navier-Stokes Source = SUSPEND 0. 0. -980.0 1.34e3
```

```
Second Level Set Momentum Source = CONSTANT. 1.0e-4 POSITIVE
```

This setup will cause the negattive side of the interface to receive momentum source values obtained from the USER model with the parameters listed above . The positive side of the interface will show a constant momentum source of 1.0e-4.

Technical Discussion

An important thing to note is that the units of the quantity specified on this card are units of force per volume in exact correspondence to the units used with the preceding momentum source model. Note also that this card should not be used when using the LEVEL_SET momentum source model. For one thing, it makes no sense and for another thing the values specified on the latter model are simply the gravitational acceleration and therefore are inconsistent with this card.

Second Level Set Viscosity

```
Second Level Set Viscosity = {model_name} {float_list} {char_string} [M/Lt]
```

Description / Usage

This card allows to the user to specify a second viscosity model that will be applied to one side of a level set interfacial curve:

{model_name}	The name of the viscosity model can only be **CONSTANT** at the current time.
{float1}	This is a single float parameter which is the value of Newtonian viscosity applied to the second level set. phase fluid.
{char_string}	This string may take the values POSITIVE or NEGATIVE. It identifies which side of the interface the preceding viscosity model is applied to.

This card allows the user to apply one of the several complex viscosity models currently available in Goma to one side of the interface while the other side recieves the constant viscosity value listed on this card. The side of the interface that corresponds to char_string appearing on this card recieves the constant viscosity value. The opposite sides viscosity is determined from the other, more complex model. Transition between them is accomplished using smooth Heaviside functions whose width is given on the Level Set Length Scale card. Note that it is the prescence of the this card in the material file that actually activates this selection process.

Examples

The following is a usage example for this card:

```
Liquid Constitutive Equation = HERSCHEL_BULKLEY
```

```
Low Rate Viscosity = CONSTANT 10000
```

```
Power Law Exponent = CONSTANT 0.6
```

```
Yield Stress = CONSTANT 1.e6
```

```
Second Level Set Viscosity = CONSTANT. 1.0 POSITIVE
```

This setup will cause the negattive side of the interface to receive viscosity values obtained from the HERSCHEL_BULKLEY model with the parameters listed above . The positive side of the interface will show a constant viscosity of 1.0.

Technical Discussion

No Discussion.

References

No References.

Shell Bending Stiffness

```
Shell bending stiffness = {model_name} {float_list} [M/Lt2]
```

Description / Usage

This required card is used to specify the model for the Shell bending stiffness property D which is defined as D=Et3/12(1-ν2), where E is the elastic modulus, ν Poisson's ratio, and t the shell thickness. The units are M-L2/t2 (or F-L). The elastic modulus is set through the *Lame MU* and *Lame Lambda* cards. This property is needed for the inextensible cylindrical shell equations (see *EQ = Shell Tension*).

Definitions of the input parameters are as follows:

{model_name}	Name of the Shell bending stiffness coefficient model. This parameter can have one of the following values: **CONSTANT**. • {float1} - The value of the shell bending stiffness.

The details of each model option are given below:

CONSTANT <float1>	For the **CONSTANT** model, {float_list} is a single value: • <float1> - Standard value of the coefficient D.

References

GT-027.1: GOMA's Shell Structure Capability: User Tutorial (GT-027.0). P. R. Schunk and E. D. Wilkes.

GT-033.0: Structural shell application example: tensioned-web slot coater (GT-033.0). P. R. Schunk and E. D. Wilkes.

1.5.3 Thermal Properties

In this section of material properties, the user specifies the parameters of models for Fourier heat conduction and thermally-induced density changes (by volume expansion) and parameters controlling the onset of phase changes. Properties governing energy transport by convection, radiation and diffusion are specified elsewhere.

Heat Flux Model

```
Heat Flux Model = USER
```

Description / Usage

NOT TESTED. Use this optional card to specify a user-defined model for the conductive heat flux. The routine "usr_heat_flux" in file user_mp.c must appropriately define the heat flux/temperature gradient model. The single input parameter has only one possible value:

USER	the user-defined model for the conductive heat flux.

If this card is missing or has a different keyword, the Fourier conductive heat flux model will be used.

Examples

Following is the only permissible specification for the card:

```
Heat Flux Model = USER
```

Technical Discussion

No Discussion.

References

No References.

Conductivity

```
Conductivity = {model_name} {float_list} [E/LtT]
```

Description / Usage

This card is used to specify the model for thermal conductivity. Definitions of the input parameters are as follows:

{model_name}	Name of the model for thermal conductivity; this parameter can have the value **CONSTANT** or **USER**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below.

Thus,

CONSTANT <float>	a constant thermal conductivity model, {float_list} is a single value: • <float1> - Standard value of k
USER <float1>… <floatn>	a user-defined model. With the USER option the appropriate modifications to the routine "usr_thermal_conductivity" in the user_mp.c file must be undertaken. The {float_list} can be of arbitrary length and is used to parameterize the model. These parameters are made available in the subroutine via <float1> through <floatn>.
TABLE <integer1> <character_string1> {LINEAR | BILINEAR} [integer2] [FILE = filenm]	Please see discussion at the beginning of the material properties chapter 5 for input description and options. Most often character_string1 will be **TEMPERATURE** or maybe **MASS_FRACTION**.

Examples

Following is a sample card:

```
Conductivity = USER 1. 1. 1. 1. 1.
```

Technical Discussion

No Discussion.

References

No References.

Heat Capacity

```
Heat Capacity = {model_name} {float_list} [E/MT]
```

Description / Usage

This required card is used to specify the model for the heat capacity. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the heat capacity. This parameter can have one of the following values: **CONSTANT**, **USER**, or **ENTHALPY**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. These are identified in the discussion of each model below.

Thus,

CONSTANT <float>	This option specifies a constant heat capacity. The {float_list} has a single value: • <float1> - Heat capacity
USER <float1>…	the heat capacity will be a user-defined model. This user-defined model must be incorporated by modifying the routine "usr_heat_capacity" in the file user_mp.c. The model parameters are entered in the {float_list} as <float1> through <floatn> and passed to the routine as an array.
ENTHALPY <float1>	a model of heat capacity that uses the latent heat of fusion parameter. The model goes as follows: Here the {float_list} requires two values, where: • <float1> - Base heat capacity in the solid state, cp • <float2> - Latent heat of fusion Hf The liquidus temperature Tl and the solidus temperature Ts are taken from the material file. This model is currently available for single species only, and is used for rapid melting problems in alloys.
TABLE <integer1> <character_string1> {LINEAR | BILINEAR} [integer2] [FILE = filenm]	Please see discussion at the beginning of the material properties chapter 5 for input description and options. Most often character_string1 will be **TEMPERATURE** or maybe **MASS_FRACTION**.

Examples

Following is a sample card:

```
Heat Capacity = CONSTANT 1.
```

Technical Discussion

When the **ENTHALPY** option is used, the liquidus (Tl) and solidus (Ts) temperatures must be added through the *Liquidus Temperature and Solidus Temperature* cards.

References

No References.

Volume Expansion

```
Volume Expansion = CONSTANT <float> [1/T]
```

Description / Usage

This card is used to specify the model for the coefficient of volume expansion in the energy equation. This property is required for the **BOUSS** option on the *Navier-Stokes Source* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for a constant volume-expansion coefficient. • <float> - the value of the volume expansion coefficient.

Examples

The following is a sample input card:

```
Volume Expansion = CONSTANT 1.
```

Technical Discussion

Warning: Please be careful that the Species Volume Expansion card is set appropriately. If the **BOUSS** or **BOUSSINESQ** model is turned on on the Navier- Stokes Source card, then both thermal and species volume expansion effects are included if the coefficients are nonzero. .

References

No References.

Reference Temperature

```
Reference Temperature = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify the model for the reference temperature, which is required by the **BOUSS** option on the *Navier-Stokes Source* card and by the **BINGHAM** option on the *Liquid Constitutive Equation* card. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for a constant reference temperature. • <float> - the value of the reference temperature.

Examples

The following is a sample input card:

```
Reference Temperature = CONSTANT 1.
```

Technical Discussion

No Discussion.

References

No References.

Liquidus Temperature

```
Liquidus Temperature = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify the model for the liquidus temperature. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the liquidus temperature. • <float> - the value of the liquidus, Tl .

Examples

Following is a sample card:

```
Liquidus Temperature = CONSTANT 1.
```

Technical Discussion

This card is required when using the **ENTHALPY** option on the *Heat Capacity* card.

References

No References.

Solidus Temperature

```
Solidus Temperature = CONSTANT <float> [T]
```

Description / Usage

This card is used to specify the model for the solidus temperature. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the solidus temperature. • <float> - the value of the solidus, Ts .

Examples

The following is a sample card:

```
Solidus Temperature = CONSTANT 1.
```

Technical Discussion

This card is required when using the **ENTHALPY** option on the *Heat Capacity* card.

References

No References.

Energy Weight Function

```
Energy Weight Function = {GALERKIN | SUPG} <float>
```

Description / Usage

This card specifies the weight function to be used on the weighted residual of the energy equations. For high Peclet number cases, you may want to use a Petrov- Galerkin formulation rather than a Galerkin formulation. Definitions of the input parameters are as follows:

GALERKIN	Name of the model for the weight functions for a full Galerkin formulation. This is the default when this card is absent. • <float> - the value of the weight function, a number between 0. and 1.; a value of 0. corresponds to **GALERKIN**.
SUPG	Name of the model for the weight functions for a stream-wise upwinded Petrov-Galerkin formulation. • <float> - the value of the weight function, a number between 0. and 1.; a value of 1. corresponds to a full **SUPG**.

Examples

The following is a sample input card:

```
Energy Weight Function = GALERKIN 0.0
```

Technical Discussion

The **SUPG** weighting is applied only to the advective term in the Energy conservation equation and Jacobian assembly.

References

No References.

1.5.4 Electrical Properties

Models for material electrical properties are simple or specialized, being very applicationoriented. The primary need for modeling electrical potential effects are to activate mass transport mechanisms that are charge-dependent.

Electrical Conductivity

```
Electrical Conductivity = {model_name} {float} []
```

Description / Usage

This required card is used to specify the model for electrical conductivity. There are currently three options, so {model_name} can be either **CONSTANT, ELECTRONEUTRALITY_FICKIAN or ELECTRONEUTRALITY_SM.** Definitions of the input parameters are as follows:

CONSTANT	Name of the model for constant electrical conductivity. <float> - the value of electrical conductivity.
LEVEL_SET	Name of the model for constant electrical conductivity.Allows for the conductivity as a function of the level-set field. Specifically used for changing the conductivity from one constant value on the negative side of the interface to another constant value on the positive side. The model requires three floats: • <float1> - the value of electrical conductivity in the negative regions of the level set function. • <float2> - the value of electrical conductivity in the positive regioons of the level-set function. • <float3> Length scale over which the transition occurs. If this parameter is set to zero, it will default to one-half the Level-Set Length Scale value specified.
ELECTRONEUTRALITY_FICKIAN	Name of the model for the electrical conductivity. This model requires no parameter specification, i.e. no floats.
ELECTRONEUTRALITY_SM	Name of the model for the electrical conductivity. This model requires no parameter specification, i.e. no floats. In earlier versions of Goma, this model was referred to by the name ELECTRODE_KINETICS and it remains to be active so that Goma can be backward compatible. In other words, **ELECTRONEUTRALITY_SM** and **ELECTRODE_KINETICS** are interchangeable.

See Technical Discussion for information on the electrical conductivity for the two models of **ELECTRONEUTRALITY**.

Examples

Following are sample cards:

```
Electrical Conductivity = CONSTANT 1.
```

```
Electrical Conductivity = ELECTRONEUTRALITY_FICKIAN
```

```
Electrical Conductivity = ELECTRONEUTRALITY_SM
```

```
Electrical Conductivity = ELECTRODE_KINETICS
```

Technical Discussion

For concentrated electrolyte solutions in which Stefan-Maxwell flux equations are employed to relate species fluxes to concentrations and their gradients, the electrical conductivity is given by (Chen et al. 2000, Schunk et al. 2000):

$$\kappa = \frac{F^2}{RT} \sum_{i'=1}^{n} \sum_{k'=2}^{n} z_{i'} z_{k'} \left(-b^{-1}_{ik} \right) x_{k'}$$

where i = m(i' – 1) + 1 and k = m(k' – 1) + 1, m is dimension of the problem (m = 2 for a 2-D problem), and is species mole fraction. The tedious definition of can be found in Chapter 2 of Chen et al. (Chen et al. 2000) and in Chapter 7 of the Goma Developer's Guide (Schunk, et. al., 2000).

For dilute electrolyte solutions in which Fick's first law is used to relate the flux of a species to its concentration gradient, the electrical conductivity is given by (Chen, 2000; Schunk, et. al., 2000):

$$\kappa = \sum_{i=1}^{n-1} \frac{F^2}{RT} z_i^2 D_i c_i$$

where ci is the molar concentration and zi is the charge number of species i, respectively; and n is the total number of species present in the electrolyte solution. Note that the nth species is taken to be the neutral solvent species, which has no contribution to the electrical conductivity since its charge number is zero.

Lastly, *Goma* calculates the conductivity in function assemble_potential as material properties are being loaded.

References

GTM-025.0: Chen, K. S., "Modeling diffusion and migration transport of charged species in dilute electrolyte solutions: GOMA implementation and sample computed predictions from a case study of electroplating", Sandia technical memorandum, September 21, 2000.

SAND2000-0207: Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R., and Houf, W. G., "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", Sandia Technical Report, 2000.

GDM-1.3: Schunk, P. R., Sackinger, P. A., Rao, R. R., Subia, S. R., Baer, T. A., Labreche, D. A., Moffat, H. K., Chen, K. S., Hopkins, M. M., and Roach, R. A., "GOMA 3.0 - A Full-Newton Finite Element Program for Free and Moving Boundary Problems with Coupled Fluid/Solid Momentum, Energy, Mass, and Chemical Species Transport: Developer's Guide, 2000.

Electrical Permittivity

```
Electrical Permittivity = {model_name} {float} []
```

Description / Usage

This required card is used to specify the model for electrical permittivity. There is currently one option, so {model_name} must be either **CONSTANT**. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for constant electrical permittivity. • <float> -the value of electrical permittivity.

Examples

Following are sample cards:

```
Electrical Permittivity = CONSTANT 1.
```

Technical Discussion

This card is utilized to set the electrical permittivity for electrostatic problems.

References

No References.

1.5.5 Microstructure Properties

Microstructure property models address material parameters and constitutive equations required for multiphase continuum approaches to flow in porous media, viz. fluid flow in partially or fully saturated porous media. Actually, only a few of these model/property cards pertain directly to media structure or microstructure, but all are affected by intrinsic material properties of all involved phases. Cards or records typically appearing in this section fall into one of three categories: microstructural or flow-property specification, numerical treatment specification, and species transport property specifications. These "sections" appear in this order in most of the sample input files.

Media Type

```
Media Type = {model_name}
```

Description / Usage

This card is used to designate the characteristic medium type for solid materials so that the proper microstructural features/models may be imposed. Basically, the choices are dictated by whether the medium is to be modeled as *porous* (viz. a medium in which flow will be determined relative to the motion of a porous solid skeleton) or as *continuous* (viz., in which the mechanics equations apply to all parts of the medium and not weighted by a solid fraction). If porous flow through Darcy or Brinkman formulations are desired in the material, then the phase is designated as *continuous*.

The input parameter is a {model_name} and has the following possible values:

{model_name}	{model_name} Name of the media model; the choices are • **CONTINUOUS** • **POROUS_SATURATED** • **POROUS_UNSATURATED** • **POROUS_TWO_PHASE** • **POROUS_BRINKMAN** • **POROUS_SHELL_UNSATURATED**

Specific characteristics of these types are identified below, including other cards that must be present.

- If the type chosen is **CONTINUOUS**, then the material is assumed to be amorphous and no further microstructure properties need to be specified (next required card is the *Diffusion Constitutive Equation*).
- In a porous medium with one phase in the pores (i.e. a saturated medium), use **POROUS_SATURATED** then only the Porosity and Permeability cards are required. A **POROUS_SATURATED** medium model enables the user to solve the simplest porous flow equation for the liquid phase pressure only for rigid porous media (see the porous_sat or porous_liq equation cards). For deformable porous saturated media, one can employ a stress balance and porosity equation for deformable porous media (see mesh* equation cards and porous_deform equation card).
- In a porous medium with two phases in the pores (such as air-water, i.e., an unsaturated medium), two options exist - **POROUS_UNSATURATED**, a formulation of the porous flow problem using the capillary pressure as the field variable (gas pressure assumed to be uniform), and **POROUS_TWO_PHASE**, a formulation of the porous flow problem using the liquid pressure and gas pressure as field variables. All the cards in this Microstructure porous flow section, except the Brinkman cards (*FlowingLiquid Viscosity and Inertia Coefficient*), are needed for the unsaturated or two-phase models. As in the saturated case above, these options can also be chosen for deformable porous media, for which the Lagrangian mesh stress equations and the porosity equation are used to complete the effective stress principle formulation.

- The **POROUS_BRINKMAN** model is an extension of the Navier-Stokes equation for porous media. In addition, it has an inertia term intended to account for boundary and interface deficiencies at Reynold's numbers greater than one (Re > 1), a deficiency in all Darcy flow models (see, e.g., Gartling, et. al., 1996). It is a vector formulation (the momentum equations) of saturated flow in a porous medium which reduces to the Navier-Stokes equations as the porosity increases to one ($\varphi \rightarrow 1$). For Brinkman flow, the input parameters (i.e., cards) that must be specified from this section are *Porosity, Permeability, FlowingLiquid Viscosity*, and *Inertia Coefficient*. Please note the use of two viscosities; for the Brinkman media type, the viscosity entered via the (*Mechanical Properties and Constitutive Equations*) Viscosity card is interpreted to be the Brinkman viscosity (μB) and is used to calculate the viscous stresses (see Gartling, et. al., 1996) while the *FlowingLiquid Viscosity* (μ) is used in the correction term for nonlinear drag forces in porous media. Brinkman viscosity is an effective value and can be taken as the porosity weighted average of the matrix and fluid. It is generally not correct to set it equal to the liquid viscosity (Martys, et. al., 1994; Givler and Altobelli, 1994).
- The **POROUS_SHELL_UNSATURATED** model is used for thin shell, open pore, porous media, viz. the shell_sat_open equation. This media type instructs GOMA to obtain most of the media properties from the bulk continuum specifications just like **POROUS_UNSATURATED**. Exceptions are the Porous Shell Cross Permeability model and the Porous Shell Height material models. Please see the porous shell tutorial

Examples

Following is a sample card:

```
Media Type = POROUS_TWO_PHASE
```

This card will require a plethora of material models for Darcy flow of liquid and gas in a porous medium. It also will require the use of two Darcy flow mass balances in the *Problem Description EQ* specification section, specifically *porous_liq* and *porous_gas* equations. See references below for details.

Technical Discussion

In solving porous medium problems, it is important to understand that each conservation equation represents a component, or species balance. The *porous_liq* equation is actually a species balance for the liquid phase primary component (e.g. water) for all phases in the medium, viz. liquid, gas, and solid. This is the case even though the dependent variable is the liquid phase pressure. This is the only required equation for rigid **POROUS_SATURATED** media. The same holds true for rigid **POROUS_UNSATURATED** media, as the liquid solvent is present in liquid and gas vapor form (it is actually taken as insoluble in the solid). For deformable media, one must add a stress balance through the *mesh** equations (in *LAGRANGIAN* form, as described on the *Mesh Motion* card) and a solid phase "solvent" balance which is used to solve for the porosity, viz. the *porous_deform* equation. In these cases, the gas is taken to be at constant pressure. If pressure driven Darcy flow is important in the gas, an additional species balance for the primary gas component is required through the porous_gas equation. This last case is the so-called **POROUS_TWO_PHASE** media type.

Options for representing the solid medium as rigid or deformable are discussed under the *Saturation, Permeability* and *Porosity* cards. When rigid porous media are modeled, both porosity and permeability are constant. In *Goma* 4.0, these concepts were being researched and improved, with much of the usage documentation residing in technical memos.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

Gartling D. K., C. E. Hickox and R. C. Givler 1996. "Simulations of Coupled Viscous and Porous Flow Problems", Comp. Fluid Dynamics, 7, 23-48.

Givler, R. C. and S. A. Altobelli 1994. "A Determination of the Effective Viscosity for the Brinkman-Forchheimer Flow Model." J. Fluid Mechanics, 258, 355-370.

Martys, N., D. P. Bantz and E. J. Barboczi 1994. "Computer Simulation Study of the Effective Viscosity in Brinkman's Equation." Phys. Fluids, 6, 1434-1439

Porosity

```
Porosity = {model_name} <float> []
```

Description / Usage

This card is used to specify the model for the porosity, which is required for the Brinkman or Darcy formulations for flow through porous media, viz. for **POROUS_BRINKMAN, POROUS_TWO_PHASE, POROUS_SATURATED,** and **POROUS_UNSATURATED** media types (see *Media Type* card).

Definitions of the {model_name} and <float> parameters are as follows:

CONSTANT	Name {model_name} of the constant porosity model. • <float> - Value of porosity.
DEFORM	Name {model_name} of the model for a porosity that varies with deformation of the porous medium. A conservation balance is required for the solid material skeleton and is invoked in the equation specification section (see *EQ* section). • <float> - Value of porosity (in the stress-free-state, i.e., undeformed state).

Examples

The following is a sample input card:

```
Porosity = DEFORM 0.5
```

This model will result in a porosity of 0.5 (volume fraction of the interstitial space of a porous skeleton) in the undeformed or stress-free state, but will allow the porosity to vary affinely with the volume change invariant of the deformation gradient tensor (see technical discussion). As mentioned above, the **DEFORM** model requires a field equation for the mass-conservation of the solid matrix through the porous_deform equation.

Technical Discussion

Porosity is a microstructural attribute of a porous medium which describes the fraction of volume not occupied by the solid skeleton. For rigid porous media, it is a parameter that weights the capacitance term (time-derivative term) of the Darcy flow equations for liquid solvent and gas "solvent" concentrations. It often affects the Saturation function (see *Saturation* card) and the permeability function (see *Permeability* card). The references cited below elucidate the role of the porosity parameter in these equations.

For deformable porous media, *Goma* uses the porosity as a measure of fraction solid concentration, as a part of a mass balance for the solid skeleton. The reason this equation is required is a result of the lack of an overall conservation law for the mixture. Instead, we close the system by individual conservation equations for all species components in the medium, including the solid; the liquid and gas phase components are accounted for with individual Darcy flow equations. The conservation law which governs the porosity assumes there is an affine deformation of the pores with the overall deformation of the solid, and hence can be written as:

$$det(\underline{F}) = \frac{V}{V_0} = \frac{1-\phi_0}{1-\phi}$$

where F˜ is the deformation gradient tensor, φ0 is the initial porosity, and φ is the porosity. This equation is invoked with the *porous_deform* option on the *EQ* specifications.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Permeability

```
Permeability = {model_name} {float_list} [L2]
```

Description / Usage

This card is used to specify the model for permeability, which is required for the Brinkman and Darcy formulations for flow through porous media. Definitions of the input parameters are as follows:

{model_name}	Name of the permissible models for permeability: **CONSTANT, TENSOR, KOZENY_CARMEN, SOLIDIFICATION** and **PSD_VOL, PSD_WEXP,** or **PSD_SEXP**. (No USER model as of 6/13/2002; contact Developers for this addition).
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

Permeability model choices and their parameters are discussed below.

CONSTANT <float1>	Model for constant permeability with a single parameter. This model is allowed for all Media Types (cf. *Media Type* card). • <float1> - k, Permeability [L2]
TENSOR <float1> <float2> <float3> <float4>	Model for a two dimensional, constant anisotropic permeability; it has not been implemented in three dimensions. All media types (cf. Media Type card) except **POROUS_BRINKMAN** may use this model. • <float1> - k_{xx} permeability [L2] • <float2> - k_{yy} permeability [L2] • <float3> - k_{xy} permeability [L2] • <float4> - k_{yx} permeability [L2]
PSD_VOL <float1> <float2> <float3> <float4>	This is a model of a deformable medium with a probabilistic distribution of pore sizes; see Technical Discussion section. Four parameters are required for the **PSD_VOL** model: • <float1> - φ0, porosity in undeformed state • <float2> - rmax(φ0), maximum pore radius in undeformed state • <float3> - α, ratio of smallest pore size to largest pore size • <float4> - $1/\tau 2$, a geometric tortuosity factor All media types (cf. *Media Type* card) except **POROUS_BRINKMAN** may use this model
PSD_WEXP <float1> <float2> <float3> <float4>	Same <float> specifications as **PSD_VOL** model. This model is allowed for all media types except **POROUS_BRINKMAN** (cf. *Media Type* card).
PSD_SEXP <float1> <float2> <float3> <float4>	Same <float> specifications as **PSD_VOL** model. This model is allowed for all media types except **POROUS_BRINKMAN** (cf. *Media Type* card).
SOLIDIFICATION <float1>	Used to phase in a porous flow term in the liquid momentum equations for low volume fraction packing of particles in the Brinkman porous flow formulation (see discussion below). Used for Phillip's model of suspensions for the *Liquid Constitutive Equation*, viz. **CARREAU_SUSPENSION, SUSPENSION, FILLED_EPOXY** or **POWER_LAW_SUSPENSION**. • <float1> - the species number of the suspension flow model; it is used to indicate that maximum packing, or solidification has occurred. (The float is converted to an integer). This model is **ONLY** allowed for media type **POROUS_BRINKMAN** (cf. *Media Type* card). The functional form is: $$\frac{k}{\mu} = \frac{1}{2}h_{avg}^2 \frac{\left(\frac{(1-\phi_{part})^2}{\phi_{part}(1.43-\phi_{part})}\right)}{\mu_0}$$ where μ0 is the clear fluid viscosity, φpart is the volume fraction of particles, or concentration divided by the maximum packing (0.68 for monodisperse spheres), and havg is the average element size.
KOZENY_CARMAN <float1> <float2>	The Kozeny-Carman equation relates the permeability to the porosity for a porous medium and has been shown to fit well the experimental results in many

Examples

Following is a sample card:

```
Permeability = CONSTANT 0.001
```

This specification leads to a constant permeability of 0.001.

Technical Discussion

For all models, this card provides the permeability, in units of [L2]. For saturated porous materials (viz. **POROUS_BRINKMAN** or **POROUS_SATURATED** media types), the viscosity from the *Viscosity* card is used to compute the porous conductivity, viz., permeability divided by viscosity. For unsaturated media types, the viscosity factor comes through the relative permeability cards (see *Rel Gas Permeability* and *Rel Liq Permeability* cards). Please consult the references below for the proper form of the equations.

The **PSD_VOL** (Probability Size Distribution, PSD) model treats the medium as a bundle of capillary tubes with a distribution of pores such that over a range of porek sizes the volume of pores is evenly distributed. For such a model, the maximum poresize varies with the porosity:

$$r_{max}(\phi) = r_{max}(\phi_0)\left(\frac{\phi}{\phi_0}\right)^{1/2}\left(\frac{1-\phi_0}{1-\phi}\right)^{1/3}$$

Then, the permeability is a function of the maximum pore-size and the pore-size distribution:

$$k = \phi\frac{1}{60\tau^2}\frac{(1-\alpha^3)}{(1-\alpha)}r_{max}^2$$

The input parameters for the PSD models are $\varphi 0$, $r_{max}(\varphi 0)$, α, and $1/\tau 2$. More detail on the deformable porous medium models is given in Cairncross, et. al., 1996. The **PSD_WEXP** and **PSD_SEXP** are similar pore-size distribution models to **PSD_VOL**. The references below should be consulted for details on how to use these models.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Liquid phase compressibility

```
Permeability = {model_name} {float_list} [L2]
```

Description / Usage

This card specifies the model and model parameters for liquid-phase compressibility, and was specifically designed for use in porous-media flow problems that are partially saturated (viz. *Media Type* card values of **POROUS_UNSATURATED** or **POROUS_TWO_PHASE**). This feature was added partially for numerical convenience in rigid porous media to accommodate regimes where the saturation level is at or near unity; at these saturation levels the capacitance term (see Technical Discussion below) all but vanishes, viz. there is no sensitivity of the saturation level to liquid phase pressure, and the mathematical behavior can change type. This occurs in situations of low permeability, narrow pore-size distribution, and sudden pressure spikes during simulation startup.

CONSTANT	Name of the model for the compressibility coefficient, currently the only option. It requires a single parameter: • <float> - Compressibility coefficient, in units of inverse pressure.

This card requires a companion card *Liquid phase reference pressure*.

Examples

The cards (using APREPRO variables)

```
Liquid phase compressibility = CONSTANT {beta_liquid}
```

```
Liquid phase reference pressure = CONSTANT {p_not}
```

leads to the application of a linearized compressibility model for the density of liquid in the time-derivative capacitance term. This is useful for rigid porous media when the conditions are such that the saturation front is sharp.

Technical Discussion

For Cls the most part, we have needed the *Liquid Phase Compressibility* capability to ease the startup of impregnation problems, in which an external pressure load is impulsively applied to a liquid layer being forced into a rigid porous matrix. The capacitance term as the saturation level approaches 1.0 (S->1) in the porous Darcy flow equation appears in *Goma* as

$$\frac{dC_{ls}}{dt} \Rightarrow \frac{d(\phi\rho_{ls})}{dt} = \phi\left(\frac{d\rho_{ls}}{dt}\right) \cong \phi\frac{d\rho_{ls}}{dP_{liq}}\frac{dP_{liq}}{dt}$$

Here is the liquid solvent concentration (in both gas and liquid phases), φ is the porosity, ρls and is the liquid phase density. Here we employ the linearized density model:

$$\rho_{ls} = \rho^{0}_{ls}(1 + (\beta_{ls})(p_{liq} - p^{0}_{liq}))$$

where βls is the coefficient of compressibility entered on this card, viz. dρls/dPliq defined above, p0liq is the reference liquid pressure (see Liquid phase reference pressure* card)

FAQs

The following troubleshooting tips regarding startup of partially saturated porous media problems are part of the authors experience presented in Schunk, 2002 (GT- 009.3):

-Linear elements, viz. Q1 elements, are better for saturation front startup at an external boundary if the difference between the boundary specified liquid-phase pressure and the medium-initialized liquid phase pressure are drastically different. Quadratic elements in this case can lead to zero or low Saturation values at all computational Gaussian integration points and the front may never penetrate.

-Time stepping is all important. There are three relevant parameters: time-step scheme, initial time step size, and time-step error factor. The rules of thumb that can be established are as follows:

If you are using Porous Mass Lumping, you must set the Time Step Parameter to 0.0, or your performance will suffer. In fact, it is always a good idea in steep penetration front problems to use backward Euler techniques.

With mass lumping and first order time integration, you must control your step size with the tolerance setting. Too big of time step early on can propagate to large errors at later times when time stepping. You may need to experiment with the error tolerance on the Time step error card. Constantly scrutinize your results for correctness and suspect an error growth here.

You must have a significant capacitance term on the first time step. If your capacitance term is small, then the problem is elliptic and will try to satisfy all boundary conditions, and this can mess up your penetration front.You can use *Liquid phase compressibility property* to help this for steep front startup.

Are you getting stagnant calculations with time-step decreases but not change in iteration history? Problem is that you have lost your capacitance term. Compressibility of the liquid is sometimes a remedy, but also a more accurate predictor. Mass lumping can help too and accomplishes the same thing. Sometimes your initial time step can be too small for a good start. Try increasing it …

-Another startup issue: Steep discontinuities at boundaries and internally for initial conditions are bad, obviously. If your time step is such that the front cannot penetrate beyond one element in one time step, then with linear elements the capacitance term is ineffective (small) upon reduced time steps. Somehow you have got to get the front beyond one or two elements before things work properly. I find that ramping up the initial boundary conditions helps. Sometimes a large first time step to kick it is good too.

-On startup of a pressurized column of liquid penetrating into a porous substrate, I noticed that at zero-based p_liq, there was no problem elevating the applied pressure on the penetration, but at Atm-based p_liq we couldn't start the problem without severe compressibility. However, compressibility affects the solution, and in fact allows you to push all of your column of liquid into a compressed layer in the substrate, with no Sat from propagation. So beware of poorly defined compressibility of liquid. Also, refinement in the porous layer helped the startup. But the most significant thing for the problem I was solving, don't be surprised if just a little perturbation on externally applied pressure greatly affects the penetration rate. In fact, in one problem simply changing from p_ext of 1.01325e+6 to 1.11325+6 increases the penetration rate 2-fold initially. The steeper curves are harder to handle.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

Liquid phase reference pressure

```
Liquid phase reference pressure = CONSTANT <float> [M/L-t2] or [N/L2]
```

Description / Usage

This card is used to specify the model and model parameters for the liquid-phase compressibility reference pressure. See *Liquid phase compressibility* card for discussion and theory.

CONSTANT	model for the reference pressure, currently the only available option. It requires a single floating point value: • <float> - The reference pressure, in units of pressure.

Examples

The cards

```
Liquid phase compressibility = CONSTANT {beta_liquid}
```

```
Liquid phase reference pressure = CONSTANT {p_not}
```

leads to the application of a linearized compressibility model for the density of liquid in the time-derivative capacitance term. This is useful for rigid porous media when the conditions are such that the saturation front is sharp.

Technical Discussion

See discussion on *Liquid phase compressibility* card.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

Flowing Liquid Viscosity

```
FlowingLiquid Viscosity = CONSTANT <float> [M/Lt]
```

Description / Usage

This card is used to specify the model for the viscosity of liquid flowing through pores with the Brinkman model of flow through porous media, viz. see *Media Type* card with **POROUS_BRINKMAN** option. In the Brinkman model, the viscosity input through the *Viscosity* card is used as the Brinkman viscosity, and the viscosity input through this card is used in determining the hydraulic resistance. Detailed discussion of these two viscosities can be found by consulting the references below.

Definitions of the input parameters are as follows:

CONSTANT	Name for the constant viscosity model. • <float> - The value of the viscosity.

Examples

Following is a sample card:

```
Flowing Liquid Viscosity = CONSTANT 101.0
```

This card is only applicable to the **POROUS_BRINKMAN** media type and results in a hydraulic resistance viscosity of 101.0.

Technical Discussion

See references below for discussion on use of this card.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

Gartling, D. K., C. E. Hickox and R. C. Givler 1996. "Simulations of Coupled Viscous and Porous Flow Problems", Comp. Fluid Dynamics, 7, 23-48.

Inertia Coefficient

```
Inertia Coefficient = CONSTANT <float> []
```

Description / Usage

This card is used to specify the model for the inertia coefficient cˆ in the Brinkman formulation for flow through porous media, viz. see **POROUS_BRINKMAN** option on *Media Type* card. Detailed discussion of this coefficient can be found by consulting the references below. Definitions of the input parameters are as follows:

CONSTANT	Name of the model for the inertia coefficient. • <float> - The value of the inertia coefficient.

Examples

Following is a sample input card that produces a weighting coefficient of 1.0 on the inertial term in the **POROUS_BRINKMAN** equations.

```
Inertia Coefficient= CONSTANT 1.0
```

Technical Discussion

See references below for discussion on use of this card.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

Gartling, D. K., C. E. Hickox and R. C. Givler 1996. "Simulations of Coupled Viscous and Porous Flow Problems", Comp. Fluid Dynamics, 7, 23-48.

Capillary Network Stress

```
Capillary Network Stress = {model_name}
```

Description / Usage

This card specifies the mechanism by which capillary stress and capillary pressure in the liquid phase of a partially saturated porous medium is transferred to the solid network. This model is active only when the *porous_deform* equation (see *EQ* card) is active, and the drained network is deformable under liquid phase pressure. The principles of this card rest in the theory of the effective stress principle. In effect, the model specified here can be used to change the affinity of the pore liquid to the solid network (more discussion below). The input parameter is the model for capillary network stress.

The options for {model_name} are the names of transfer mechanisms:

WETTING	specifies that the porous skeleton has the same hydrostatic pressure as the liquid. This model has not been tested recently. See discussion below.
PARTIALLY_WETTING	specifies that the porous skeleton has a hydrostatic pressure that is the average of the liquid and gas phase pressures, weighted by their saturations (see related report on drying of deformable porous media by Cairncross, et. al., 1996).
COMPRESSIBLE	functions the same as the **PARTIALLY_WETTING** option but includes a factor that accounts for the compressibility of the solid material, viz. the actual struts of the solid material, not the network (see Cairncross, et. al., 1996).

Examples

The following is a sample input card:

```
Capillary Network Stress = PARTIALLY_WETTING
```

Technical Discussion

Basically, this card sets the functional form of the capillary stress contribution to the composite effective stress in a porous medium. The constitutive equation is as follows:

$$\underset{\sim}{\sigma} = -\underset{\sim}{I}F(p_{liq}, p_{gas}, S, \phi, K_{solid}) + \underset{\sim}{\sigma}_{network}$$

where ˜network is the drained network stress that would result in the absence of any pore fluid (gas or liquid). The function F depends on the model type specified on this card. For **POROUS_SATURATED** media types, this card is not used and F = pliq. For **POROUS_UNSATURATED** and **POROUS_TWO_PHASE** media types, F is as follows for different transfer mechanisms:

- **WETTING**: The assumption here is that a thin liquid layer covers all surfaces.

$$F = (1-(1-S)\phi)p_{liq} + (1-S)\phi p_{gas}$$

- **PARTIALLY_WETTING**: The most commonly used model.

$$F = Sp_{liq} + (1-S)p_{gas}$$

- **COMPRESSIBLE**: If the solid struts are also significantly compressible, viz. the solid bulk modulus Ks is of the same order of magnitude as the network skeleton bulk modulus, Kn, this model should be used. Not recently tested; please consult with Developers before using this option. PRS (6/13/2002)

References

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Rel Gas Permeability

```
Rel Gas Permeability = {model_name} <float> []
```

Description / Usage

This card specifies the model for the relative gas phase permeability for flow in a partially saturated porous media, such that the gas flow is the pressure gradient in the gas times the permeability times the relative gas phase permeability divided by the gas viscosity. This card rests on a consistency in the specification of the relative liquid permeability (see models on the *Rel Liq Permeability* card) and this, the relative gas permeability. Definitions of the input parameters are as follows:

CONSTANT	{model_name} for constant relative gas phase permeability with a single input value: • <float> - the gas phase viscosity. For this model, one must account for the gas viscosity in the specification of this value.

The **CONSTANT** model is rarely used, as it is dependent on the saturation level and the relative liquid permeability value. Please see the *Rel Liq Permeability* card.

SUM_TO_ONE	{model_name} for the relative gas phase permeability. This model assumes that the relative liquid permeability and relative gas permeability add to one. • <float> - the value of the gas phase viscosity.

Examples

Following is a sample card:

```
Rel Gas Permeability = SUM_TO_ONE 0.0001
```

This card specifies that the relative gas permeability in Darcy's law for the gas flux is to depend on the liquid phase relative permeability such that the two sum-to-one. The gas viscosity here is specified to be 0.0001, in the appropriate viscosity units of M/L/t.

Technical Discussion

This card is only required for *Media Type* **POROUS_TWO_PHASE**. Darcy's law for gas flow is, in its simplest form:

$$\underset{\sim}{v}_g = k\frac{k_{rel}^g}{\mu_{gas}}\nabla p_{gas}$$

where, the Darcy velocity is proportional to the gradient in gas pressure, with k being the permeability, krelg being the relative gas permeability and μgas the viscosity of the gas. For the **SUM_TO_ONE** option above, the floating point constant is the gas phase viscosity, and the gas-phase relative permeability is calculated using

$$\mu_{gas}k_{gas}^{rel} + \mu_{liq}k_{liq}^{rel} = 1$$

For the **CONSTANT** option the floating point constant must include the effect of viscosity, viz. the constant represents krelg/μgas

References

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Rel Liq Permeability

```
Rel Liq Permeability = {model_name} {float_list} []
```

Description / Usage

This card is required for *Media Type* **POROUS_TWO_PHASE**. This card specifies the model for the relative liquid phase permeability for flow in a partially saturated porous media, such that the liquid flow is the pressure gradient in the liquid times the permeability times the relative liquid phase permeability divided by the liquid viscosity. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the relative gas phase permeability; the permissible values are **CONSTANT, VAN_GENUCHTEN, PSD_VOL, PSD_WEXP, and PSD_SEX**.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

Permeability model choices and their parameters are discussed below.

CONSTANT <float1>	a constant relative liquid permeability; this is a rarelyused option. • <float1> - relative liquid permeability, obtained by dividing the relative permeability desired by the liquid-phase viscosity
VAN_GENUCHTEN <float1> <float2> <float3> <float4>	assumes that the relative liquid permeability is a function of the saturation (as specified in the *Saturation* card). The {float_list} contains four values for this model, where: • <float1> - Irreducible water saturation • <float2> - Irreducible air saturation • <float3> - Exponent ($\lambda = 1 - 1/\beta$) in krel model • <float4> - Liquid viscosity
PSD_VOL <float1>	This model can only be used in conjunction with the same model for permeability and saturation; a single input value isrequired: • <float1> - Liquid phase viscosity All other parameters are loaded up from the *Saturation* and *Permeability* cards.
PSD_WEXP <float1>	This model can only be used in conjunction with the same model for permeability and saturation; a single input value is required: • <float1> - Liquid phase viscosity
PSD_SEXP <float1>	This model can only be used in conjunction with the same model for permeability and saturation; a single input value is required: • <float1> -Liquid phase viscosity

Examples

Following is a sample card:

```
Rel Liq Permeability = VAN_GENUCHTEN 0.01 0.01 0.667 0.01
```

Technical Discussion

The most often used model is that of VAN_GENUCHTEN. The functional form of this model is as follows:

$$k_{rel} = \left[\frac{(S - S_{min})}{(S_{max} - S_{min})}\right]^{\frac{1}{2}\left(1 - \left(1 - S_{eff}^{\frac{1}{\lambda}}\right)^{\lambda}\right)^2} \mu$$

where

$$S_{eff} = \frac{(S - S_{min})}{(S_{max} - S_{min})}$$

$$\lambda = 1 - \frac{1}{\beta} \quad ,$$

and is the viscosity. This function is clipped to zero as and clipped to one as Seff → 1. **PSD_*** model theory details can be found in the references cited below. These models bring in more explicit dependence on pore size and size distribution, as well as other microstructural features. In the **VAN_GENUCHTEN** model, such parameter effects are embodied in the Saturation dependence, which is empirically fit through the saturation function.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's capabilities for partially saturated flow in porous media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Saturation

```
Saturation = {model_name} {float_list} []
```

Description / Usage

This card specifies the model for the liquid saturation in a partially saturated porous media, which is frequently observed experimentally to be a function of the capillary pressure (gas pressure minus liquid pressure). This card is required for *Media Type* specifications of **POROUS_PART_SAT, POROUS_UNSAT,** and **POROUS_TWO_PHASE**. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the liquid in a partially saturated porous media. The permissible values are **CONSTANT,VAN_GENUCHTEN, TANH, PSD_VOL, PSD_WEXP,** and **PSD_SEXP.**
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

Saturation model choices and their parameters are discussed below.

CONSTANT <float1>	For the constant value of saturation model. This model is rarely used, unless one wanted to study the flow of gas and liquid at some constant, pre-specified saturation as a function of gas and liquid phase pressure.
VAN_GENUCHTEN	The **VAN_GENUCHTEN** model assumes that saturation is a function of the capillary pressure. The {float_list} contains four values, where: • <float1> - Irreducible water saturation • <float2> - Irreducible air saturation • <float3> - An exponent β • <float4> - A scaling to convert from capillary pressure to suction ($\alpha / \rho l / g$)
TANH <float_list>	The first version of the **TANH** model assumes that saturation is only a function of capillary pressure. The {float_list} contains four values, where: • <float1> - Irreducible water saturation, θw • <float2> - Irreducible air saturation, θair • <float3> - A constant c • <float3> - A constant d
PSD_VOL <float1> <float2>	This model can only be used in conjunction with the same model for permeability and relative liquid permeability; two input values are required: • <float1> - Surface tension of the liquid • <float2> - Contact angle the liquid-vapor menisci makes with the solid surfaces
PSD_WEXP <float1> <float2>	This model can only be used in conjunction with the same model for permeability and relative liquid permeability; two input values are required: • <float1> - Surface tension of the liquid • <float2> - Contact angle the liquid-vapor menisci makes with the solid surfaces
PSD_SEXP <float1> <float2>	This model can only be used in conjunction with the same model for permeability and relative liquid permeability; two input values are required: • <float1> - Surface tension of the liquid • <float2> - Contact angle the liquid-vapor menisci makes with the solid surfaces

Examples

Following is a sample card:

```
Saturation = VAN_GENUCHTEN 0.01 0.01 3.9 1.
```

The parameters on this **VAN_GENUCHTEN** specification are basically curve fit parameters to experimental measured saturation values versus capillary pressure. They do have some physical meaning, as is described below, and in the references.

Technical Discussion

The saturation function specification is perhaps the most critical and most influential function for capturing accurate behavior of flow through partially saturated porous media. The basic cap of this function versus capillary pressure is depicted in the figure below: Notice the plateau of saturation at unity at low capillary pressures (high positive liquid pressures) and the dip to the irreducible water saturation at high capillary pressures. In most real operations, this dependence will be highly sensitive to many factors: viz. whether you are filling or vacating the pore space, whether network stress in poroelastic problems is leading to liquid tension, etc.

The Van Genuchten model has the following functional form:

$$S = \theta_w + (1 - \theta_w - \theta_{air})\left(\frac{1}{1 + (\alpha P_c)^{\beta}}\right)^m$$

Here the irreducible water saturation is θw, the irreducible air saturation 0air, the suction factor is α, and the exponents β and m, the latter of which is $1 - 1/\beta$.

The **TANH** model has the following functional form:

$$S = a - b\tanh\left(c - \frac{d}{P_c}\right)$$

where *a* and *b* are automatically calculated from

$$a = 0.5 + \frac{\theta_w}{2} - \frac{\theta_{air}}{2} \quad \text{and} \quad b = 0.5 - \frac{\theta_w}{2} - \frac{\theta_{air}}{2}$$

and *c* and *d* are two fitted coefficients provided as input parameters. Here the irreducible water saturation is 0w, the irreducible air saturation 0air, and are also provided by the user as input parameters. Pc is the capillary pressure which has a lower limit of 1.E-5.

References

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk.

GTM-029.0: SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996).

Porous Weight Function

```
Porous Weight Function = {GALERKIN | SUPG} <float>
```

Description / Usage

This required card is used to specify the weight function form on the capacitance term of the Darcy flow equations for partially saturated flow (viz. for Media Type specifications of **POROUS_PART_SAT** and **POROUS_UNSAT**, and **POROUS_TWO_PHASE**.) The standard approach is to use a Galerkin formulation, but often times the SUPG option allows for a more stable time integration algorithm using the classic Streamwise Upwinding Petrov Galerkin weight function (see references below). The model options for this card are as follows:

GALERKIN	Name of the weight function formulation. This option requests a standard Galerkin finite element weighted residual treatment. A parameter is required, viz. <float>, but it is not used by *Goma*; it should be set to zero. • <float> - 0.0
SUGP	Name of the weight function formulation. This option requests a streamwise upwinding Petrov-Galerkin formulation. A floating point parameter is required as a SUPG weighting parameter and it should be set between 0.0 (for no upwinding) and 1.0 (for full upwinding). • <float> - a SUPG weighting parameter

The default model if this card is missing is **GALERKIN**.

Examples

An example card

```
Porous Weight Function = SUPG 1.0
```

Technical Discussion

As mentioned above, this card is used to invoke a streamwise upwinding scheme for purposes of stabilizing the solution around steep saturation fronts. Galerkin finite element treatment is often an extremely inaccurate discretization for propagating a discontinuity, such as is the case around these fronts, and often has to be supplemented with streamwise diffusion and/or mass lumping so that the saturation variable remains monotonic and well behaved, viz. to keep it from going below zero. Another expedient to aid in keeping the front smooth and monotonic is to use mass lumping (cf. *Mass Lumping* card).

References

GTM-029.0: SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

Bradford, S. F. and N. D. Katopodes, "The anti-dissipative, non-monotone behavior of Petrov-Galerkin Upwinding," International J. for Numerical Methods in Fluids, v. 33, 583-608 (2000).

Brooks, A. N. and T. J. R. Hughes, "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations," Comp. Math. In Appl. Mechanics and Eng., 32, 199 - 259 (1992).

Gundersen, E. and H. P. Langtangen, "Finite Element Methods for Two-Phase Flow in Heterogeneous Porous Media," in Numerical Methods and Software Tools in Industrial Mathematics, Morten Daehlen, Aslak Tveito, Eds., Birkhauser, Boston, 1997.

Helmig, R. and R. Huber, "Comparison of Galerkin-type discretization techniques for two-phase flow in heterogeneous porous media," Advances in Water Resources, 21, 697-711 (1998).

Unger, A. J. A., P. A. Forsyth and E. A. Sudicky, "Variable spatial and temporal weighting schemes for use in multi-phase compositional problems," Advances in Water Resources, 19, 1 - 27 (1996).

Porous Mass Lumping

```
Porous Mass Lumping = {yes | true | no | false}
```

Description / Usage

Mass lumping is a technique for handling stiff problems with propagation of discontinuities. By "Mass" we mean the so-called mass matrix, or the submatrix generated by the time-derivative term in the physical equations. Discretization of this term with the standard Galerkin finite element method produces a symmetric, but nondiagonal matrix, also known as the consistent mass matrix as it adheres to the proper weak form. This required card specifies the mode in which the mass matrix is computed. If mass lumping is turned on, then the matrix is formed on a nodal, collocated basis and the mass matrix becomes diagonal. This technique expedites timeintegration during the propagation of steep fronts.

The mass lumping here applies ONLY to the time-derivative term in the *EQ=porous_liq* or *EQ=porous_gas* equations in *Goma*, and only when the *Media Type* is either **POROUS_UNSATURATED** or **POROUS_TWO_PHASE**. Mass

lumping is not enabled for saturated porous flow. Please see technical discussion below for other usage tips. The card options are as follows:

yes | true Compute mass matrix with the lumped approach.

no | false Compute mass matrix with the standard Galerkin approach. This is the default.

Examples

```
Porous Mass Lumping = true
```

Technical Discussion

Mass lumping is almost essential for unsaturated porous flow problems, especially at low permeabilities and in conditions for which the saturation front is sharp. It is recommended that mass lumping always be used for all unsaturated porous flow problems. However, with such use it is also recommended to use ONLY 1st order time integration (see *Time step parameter* card and choose Backward-Euler, 0.0). For second order time integration on the porous flow equations, mass lumping does not provide any benefit as the increased accuracy in time tends to lead to insufficient accuracy in space, and wiggles form.

Mass lumping is not currently available for saturated deformable porous flow.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

GTM-029.0: SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

Bradford, S. F. and N. D. Katopodes, "The anti-dissipative, non-monotone behavior of Petrov-Galerkin Upwinding," International J. for Numerical Methods in Fluids, v. 33, 583-608 (2000).

Gundersen, E. and H. P. Langtangen, "Finite Element Methods for Two-Phase Flow in Heterogeneous Porous Media," in Numerical Methods and Software Tools in Industrial Mathematics, Morten Daehlen, Aslak Tveito, Eds., Birkhauser, Boston, 1997.

Helmig, R. and R. Huber, "Comparison of Galerkin-type discretization techniques for two-phase flow in heterogeneous porous media," Advances in Water Resources, 21, 697-711 (1998).

Unger, A. J. A., P. A. Forsyth and E. A. Sudicky, "Variable spatial and temporal weighting schemes for use in multi-phase compositional problems," Advances in Water Resources, 19, 1 - 27 (1996).

Porous Diffusion Constitutive Equation

```
Porous Diffusion Constitutive Equation = {model_name}
```

Description / Usage

This required card is used to specify the species diffusion model for the gas phase in a porous medium. Just now there is only one option, but plans are to expand the options to include multicomponent diffusion models (cf. *Diffusion Constitutive Equation* card). It is important to note that this model specification only applies to the gas phase of each component. Liquid phase species diffusive transport has not been implemented as of 12/19/01.

Definitions of the input parameters are as follows, with only a single permissible value:

DARCY_FICKIAN	Name of the model for the diffusion constitutive equation in the porous gas phase.

This model simply implies that gas species can be transported relative to the solid skeleton phase not only by a pressure gradient, as in Darcy's law, but also by Fickian diffusion.

Examples

The following sample input card uses the APREPRO variable model_name (which is set to **DARCY_FICKIAN**.

```
Porous Diffusion Constitutive Equation = {model_name}
```

Technical Discussion

Currently, the **DARCY_FICKIAN** model is the only option for the porous diffusion equation and it only applies to one phase. When this card is parsed, it is contained in a solvent species loop. When we allow more than one volatile species, we will eventually allow for other diffusion constitutive equation models, e.g. of the Stefan-Maxwell type. Also, we will have to build a phase dependence into this card, as the diffusion law may be different in the liquid and in the gas. Right now, we do not allow for diffusion transport (viz. by a chemical potential or concentration gradient) in the liquid phase of a porous medium. Please consult references below for theoretical discussion.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Porous Gas Diffusivity

```
Porous Gas Diffusivity = {model_name} <integer> <float_list> [L2/t]
```

Description / Usage

This card sets the model for the porous gas diffusivity, or the diffusion coefficient for diffusive species flux in the gas phase of a porous medium. It is applicable to media types **POROUS_UNSATURATED** and **POROUS_TWO_PHASE** (see *Media Type* card).

Definitions of the input options for {model_name} and the <integer> and <float> parameters fro each model are as follows:

CONSTANT <integer> <float1>	the name for the constant diffusivity model. • <integer> - phase/component; always set to zero until a multicomponent capability exists • <float1> - D, Diffusivity [L2/t]
POROUS <integer> <float1> <float2> <float3> <float4> <float5>	the name for a microstructure dependent porous medium model. • <integer> - phase/component; always set to zero until a multicomponent capability exists • <float1> - Dvo, binary diffusion coefficient in free space [L2/t] • <float2> - τ, tortuosity of the matrix skeleton • <float3> - P*gas, reference gas phase pressure • <float4> - T0, reference temperature. • <float5> - n, exponent on the temperature dependence (see below).

For two-phase or unsaturated flow in a porous medium, the diffusivity calculated by this model is the diffusivity of solvent vapor through the gas phase in the pore-space (see Martinez, 1995).

Examples

```
Porous Gas Diffusivity = POROUS 0 1.e-5 0.5 1.e+6 25.0 3
```

See the equation below for the diffusivity model that this card represents.

Technical Discussion

The generalized flux of liquid phase solvent, in both gas and liquid phases, contains a term that accounts for diffusion of the liquid solvent species as gas vapor (see references below). That flux is as follows:

$$D_{gv} = D_v^0\left(\frac{\phi(1-S)}{\tau}\right)$$

If the media type is **POROUS_TWO_PHASE**, this expression is divided by

$$p_{gas} / P^*_{gas}$$

and if in addition it is temperature dependent, this expression is multiplied by

$$\left(\frac{T}{T^0}\right)^n$$

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND94-0379: "Formulation and Numerical Analysis of Nonisothermal Multiphase Flow in Porous Media", Sandia Technical Report, Martinez, M. J., 1995

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Porous Latent Heat Vaporization

```
Porous Latent Heat Vaporization = CONSTANT <integer> <float> [E/M]
```

Description / Usage

This required card is used to specify the model for the latent heat of vaporization for each liquid solvent species in a partially saturated porous media flow problem, viz. *Media Type* card set to **POROUS_UNSATURATED** or **POROUS_TWO_PHASE**. As of 6/13/2002, we only allow single liquid phase solvent, and the porous enthalpy equation is being tested. Definitions of the input parameters are as follows:

CONSTANT	Name of the constant latent heat of vaporization model. • <integer> - the species equation of liquid phase solvent; MUST BE SET TO ZERO for now. • <float> - the value of the latent heat of vaporization.

Examples

The following is a sample input card:

```
Porous Latent Heat Vaporization = CONSTANT 0 1000.2
```

See the equation below for the diffusivity model that this card represents.

Technical Discussion

First order phase change involves the adsorption or expulsion of heat. This thermal effect is modeled through the porous energy equation (see EQ cards; this equation was under development and testing as this manual was being assembled) with a source term that depends on the evaporation/condensation rate.

References

No References.

Porous Latent Heat Fusion

```
Porous Latent Heat Fusion = CONSTANT <integer> <float> [E/M]
```

Description / Usage

This required card is used to specify the model for the latent heat of fusion (or freezing) for each liquid solvent species in a partially saturated porous media flow problem, viz. *Media Type* card set to **POROUS_UNSATURATED** or **POROUS_TWO_PHASE**. As of 6/13/2002, we only allow single liquid phase solvent and the porous enthalpy equation is being tested. Definitions of the input parameters are as follows:

CONSTANT	Name of the constant latent heat of fusion model. • <integer> - the species equation of liquid phase solvent; MUST BE SET TO ZERO for now. • <float> - the value of the latent heat of fusion.

Examples

The following is a sample input card:

```
Porous Latent Heat Fusion = CONSTANT 0 1000.2
```

See the equation below for the diffusivity model that this card represents.

Technical Discussion

First order phase change involves the adsorption or expulsion of heat. This thermal effect is modeled through the porous energy equation (see *EQ* cards; this equation was under development and testing as this manual was being assembled) with a source term that depends on the evaporation/condensation rate. *Fusion* implies a liquid to solid transition. It is envisioned that this card will someday be used for porous flow in mushy zones of solidifying metals, or the freezing of water in a porous solid.

References

No References.

Porous Vapor Pressure

```
Porous Vapor Pressure = {model_name} {integer} {float_list} [M/L-t2]
```

Description / Usage

Used to specify the model for the vapor pressure for each multiphase flow component in the porous medium that is activated for *Media Type* **POROUS_UNSATURATED** or **POROUS_TWO_PHASE**.

Definitions of the input parameters are as follows:

{model_name}	The permissible values for the model in this class are **KELVIN** and **FLAT** for a volatile liquid, and **NON_VOLATILE** for a non-volatile liquid.
{integer}	All models require an integer field after the model name which is the species_number; always set to zero until a multicomponent capability exists.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

Porous vapor pressure model choices and their parameters are presented below; consult the Technical Discussion for relevant details.

KELVIN <integer> <float1> <float2>… <float5>	For the **KELVIN** porous vapor pressure model, the {float_list} has a five values: • <float1> - p*v, vapor pressure on a flat interface • <float2> - ρl, the liquid density • <float3> - Mw , molecular weight of liquid • <float4> - R, the gas law constant • <float5> - T, the operating temperature
FLAT <integer> <float1> <float2>… <float5>	For the **FLAT** porous vapor pressure model, the {float_list} has a five values (same as **KELVIN** above): • <float1> - p*v, vapor pressure on a flat interface • <float2> - ρl, the liquid density • <float3> - Mw , molecular weight of liquid • <float4> - R, the gas law constant • <float5> - T, the operating temperature The **FLAT** option requires the same parameters as the **KELVIN** model but leaves out the exponential function.
NON_VOLATILE <integer>	The **NON_VOLATILE** model requires no additional input.

Examples

The sample input card:

```
Porous Vapor Pressure = FLAT 0 {Vap_Pres} {density} {30.} {Rgas} {T}
```

applies the **FLAT** model as described above to vapor-liquid equilibrium (assumed to be single component for now) using all APREPRO-defined parameters.

Technical Discussion

The **KELVIN** option is used to include the effect of vapor-pressure lowering that results in equilibrium over high curvature menisci, i.e., small pores. The equation form of this is:

The **FLAT** option requires the same parameters but leaves out the exponential function. The constants are still needed so that the gas-phase concentration can be calculated with the ideal gas law. The functional form is

where S is the local saturation, and ρgv is the gas phase density of vapor. This model is ad-hoc but nonetheless leads to some interesting results. It basically says that as saturation increases, the gas-liquid menisci, and correspondingly the interfacial area available for evaporation, become more concentrated and hence the gas-phase vapor concentration increases.

$$p_v = p^*_v \exp\left[-\frac{p_c M_w}{\rho_l RT}\right]$$

$$\rho_{gv} = \frac{M_w p_v}{RT} S$$

The **NON_VOLATILE** option should be set if no gas-phase transport of vapor of the liquid phase component is desired, as if the liquid phase were non-volatile. Goma, with this choice, sets the gas phase concentration of liquid vapor to zero.

For nonvolatile pore liquids, the vapor pressure on a flat interface, viz. the first required floating point on this card, should be set to zero. As of 6/13/02 this card has only been implemented for pure liquid solvents, so that no equilibrium solvent partitioning across the interface is present.

FAQs

Sometimes system aborts can happen with the Kelvin model because of real large, negative capillary pressures. In this case, the exponential term can exceed the machine limit. This can happen well into a transient run. The user should be aware of this; consult GT-009.3 for tips related to dealing with this problem.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

Porous Liquid Volume Expansion

```
Not currently enabled (12/21/2001)
```

Description / Usage

This card is not currently activated.

Examples

No examples.

Technical Discussion

No Discussion.

References

No References.

Porous Gas Constants

```
Porous Gas Constants = IDEAL_GAS <float_list> [varies]
```

Description / Usage

This required card is used for *Media Types* of **POROUS_UNSATURATED** and **POROUS_TWO_PHASE**, and is used to input some standard thermodynamic gas constants needed for vapor-liquid equilibrium calculations (see *Media Type* card). Eventually more than one model may be allowed for nonideal gas situations.

The **IDEAL_GAS** model is the only model currently requiring standard constants; they are defined as follows:

IDEAL_GAS	the model name requiring constants for the thermodynamic ideal gas law. • <float1> - MWair, the molecular weight of the insoluble gas in the gas phase [g/mole]. • <float2> - R, the universal gas law constant [ML2/t2/K] • <float3> - T, the temperature [deg K] • <float4> - pamb, the ambient gas pressure.

Examples

The sample input card follows:

```
Porous Gas Constants = IDEAL_GAS 28.0 8. 315 275 1.06e+5
```

Technical Discussion

For **POROUS_UNSATURATED** media types the ambient pressure dictates the equilibrium pressure for the calculation of the gas-phase density of solvent (viz. the total ambient pressure minus the vapor pressure will be the gas partial pressure, from which the concentration of gas can be computed based on the other gas constants). In **POROUS_TWO_PHASE** media types, the gas partial pressure is a dependent variable and computed as a part of the Darcy law mass balance. In this case the dynamic pressure is used instead of <float4> here for the calculation of the gas-phase concentrations.

It is important to realize that setting the ambient pressure on this card for Media Types of **POROUS_UNSATURATED** will potentially affect your saturation curve and the appropriate values of your liquid phase pressure boundary conditions. If possible, you should set this value to zero, and base your Saturation versus vapor pressure curve accordingly. Also, in that case your liquid pressure boundary conditions can all be referenced to zero. However, if you choose a gauge pressure, or thermodynamic pressure, you Saturation/capillary pressure curve must be shifted accordingly, as do your boundary conditions. Also, remember these pressures will affect your solid pressure state in poroelastic problems.

References

GT-008.2: Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems, August 11, 1999, P. R. Schunk

GT-009.3: GOMA's Capabilities for Partially Saturated Flow in Porous Media, September 1, 2002, P. R. Schunk

SAND96-2149: Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, Cairncross, R. A., P. R. Schunk, K. S. Chen, S. S. Prakash, J. Samuel, A. J. Hurd and C. Brinker (September 1996)

1.5.6 Species Properties

The section of material properties defines the models and parameters governing diffusive mass transport, whether it be ordinary, forced or thermal diffusion of species. Included in those generalizations are electrical potential-driven species movements. Models include those for single species, especially particle-laden suspensions, binary species and multi-component systems. Models for various equations of mass transport are included, various models of diffusion properties, different representations of species by means of molar, mass or volume concentrations, various models of vapor pressure for multiphase flow and on material boundaries for lumped parameter analyses, and properties for charged species.

Number of Species

```
Number of Species =
```

Description / Usage

This card is no longer used. It may be removed from the Material file.

Examples

No Example.

Technical Discussion

The *Number of Species* is now determined by *Goma* from the *Problem Specification* for each material in the *Goma* input file.

References

No References.

Diffusion Constitutive Equation

```
Diffusion Constitutive Equation = {model_name}
```

Description / Usage

This card is used to specify the constitutive equation governing mass transport. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the diffusion constitutive equation. The currently supported options are: • **NONE** • **FICKIAN** • **DARCY** • **DARCY_FICKIAN** • **HYDRODYNAMIC** • **GENERALIZED_FICKIAN** • **FICKIAN_CHARGED** • **STEFAN_MAXWELL** • **STEFAN_MAXWELL_CHARGED**

This card requires only the specification of a {model_name}. The Technical Discussion subsection below presents each of these models.

Examples

The following is a sample input card:

```
Diffusion Constitutive Equation = DARCY
```

Technical Discussion

NONE indicates that the material block to which this material file applies is a nondiffusing material. **FICKIAN** implies that the rate of diffusion is proportional to the gradient in volume fraction and the diffusion coefficient of each species. **DARCY** implies that mass transport occurs by pressure-driven flow through a porous medium. **DARCY_FICKIAN** implies that mass transport occurs by both diffusion and pressuredriven flow in a porous medium.

HYDRODYNAMIC implies that mass transport of at least one species is driven by gradients in the second invariant of the rate of deformation tensor (shear rate) and gradients in viscosity (Phillips, et.al. 1992). This model also includes a sedimentation flux term to account for the motion of non-neutrally buoyant particles resulting from gravitation (Zhang and Acrivos, 1994) and a curvature-driven flux term from the normal component of the acceleration vector (Krishnan et al., 1996). This model is used in predicting the particle distributions of particulate suspensions undergoing flow. For this model, the mass flux vector J is given by the following:

$$J = J_c + J_\mu + J_r + J_g$$

where

$$J_c = -C_i D_c \nabla(\dot{\gamma} C_i) \quad ,$$

$$J_\mu = C_i^2 \dot{\gamma} D_\mu \nabla(\ln \mu) \quad ,$$

$$J_r = C_i \dot{\gamma}^2 D_r \frac{\hat{n}}{r} \quad , \text{and}$$

where Ci is the particulate phase volume fraction, i is the species number designation of the particulate phase, the shear rate, μ the viscosity, the normal unit acceleration vector, r the curvature of streamlines, Dc, Dμ, Dr and Dg the “diffusivity” parameters, ρs and ρf the particle and fluid phase densities, respectively, and , the gravitational acceleration vector.

GENERALIZED_FICKIAN is based on the generalized-Fick’s law (Taylor and Krishna, 1993). The mass transport of each species is influenced by all of the species in the mixture.

$$J_g = D_g \frac{(\rho_s - \rho_f)}{\mu} C_i(1 - C_i)\hat{\vec{g}}$$

$$J = \rho D(\nabla w)$$

$$J = D(\nabla \rho)$$

ρ is the mass-concentration of species. The elements along the diagonal, Dii, are selfdiffusivities, while Dij are mutual-diffusivities between species i and j. Note that mutual diffusivities in generalized formulation can be both positive and negative.

FICKIAN_CHARGED indicates a model for multicomponent transport (diffusion and migration) of charged species in dilute electrolyte solutions will be used. The Fickian diffusivity of species i, Di, as defined in the following Fickian flux model (cf. Newman 1991; Chen 2000)

$$J_i = -D_i \nabla c_i - \frac{F}{RT} z_i D_i c_i \nabla \Phi$$

is taken to be constant. Here, ci is molar concentration of species i, Φ is electrical potential in electrolyte solution, zi is charge number of species i, F is the Faraday constant (96487 C/mole), R is the universal gas constant (8.314 J/mole-K), and T the electrolyte solution temperature.

STEFAN-MAXWELL activates a model for multicomponent diffusion of neutral species in concentrated solutions. The Stefan-Maxwell diffusivities, Dij, as defined in the following Stefan-Maxwell flux model (cf. Chen et al. 2000, Chen et al. 1998):

are taken to be constant. Here, xi is mole fraction of species i, Ji the molar flux of species i, and c the total molar concentration. Since Dij = Dji and Dii are not defined, only n(n-1)/2 Stefan-Maxwell diffusivities are required (here, n is the total number of diffusing species). For example, for n = 3 (i.e., a solution having three species), three Stefan-Maxwell diffusivities are needed: D12, D13, and D23.

STEFAN-MAXWELL_CHARGED For multicomponent transport (diffusion and migration) of charged species in concentrated electrolyte solutions. The Stefan- Maxwell diffusivities, Dij, as defined in the following Stefan-Maxwell flux model (cf. Chen et al. 2000, Chen et al. 1998)

$$\nabla x_i = \sum_{j \neq i} \frac{x_i J_j - x_j J_i}{c D_{ij}}$$

$$\nabla x_i + \frac{F}{RT} z_i x_i \nabla \Phi = \sum_{j \neq i} \frac{x_i J_j - x_j J_i}{c D_{ij}}$$

are taken to be constant, as in the case of multicomponent diffusion of neutral species in concentrated solutions. Here, the charged species definitions are the same as for the **FICKIAN_CHARGED** model.

References

GTM-025.0: Chen, K. S., "Modeling diffusion and migration transport of charged species in dilute electrolyte solutions: GOMA implementation and sample computed predictions from a case study of electroplating", Sandia memorandum, September 21, 2000.

Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R., and Houf, W. G. "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", SAND2000- 0207, Sandia National Laboratories Technical Report (2000).

Chen, K. S., Evans, G. H., Larson, R. S., Coltrin, M. E., and Newman, J. "Multidimensional modeling of thermal batteries using the Stefan-Maxwell formulation and the finite-element method", in Electrochemical Society Proceedings, Volume 98-15, p. 138-149 (1998).

Krishnan, G. P., S. Beimfohr, and D. Leighton, 1996. "Shear-induced radial segregation in bidisperse suspensions," J. Fluid Mech. 321, 371

Newman, J. S., Electrochemical Systems, Prentice Hall, Inc., Englewood Cliffs, New Jersey (1991).

Phillips, R.J., R.C. Armstrong, and R.A. Brown, 1992, "A constitutive equation for concentrated suspensions that accounts for shear-induced particle migration," Physics of Fluids A, 4(1), 30-40.

Taylor, R. and R. Krishna. 1993. Multicomponent Mass Transfer. John Wiley & Sons, New York.

Zhang K., and A. Acrivos, 1994, "Viscous resuspension in fully-developed laminar pipe flows," Int. J. Multiphase Flow, (20)3, 579-591.

Species Weight Function

```
Species Weight Function = {model_name} <float>
```

Description / Usage

This optional card is used to specify the weight functions to be used on the weighted residual of the species convective diffusion equations. For high Peclet number cases, you may want to use a Petrov-Galerkin formulation rather than a Galerkin formulation.

{model_name}	Name of the formulation model. Valid entries are **GALERKIN**, for a full Galerkin formulation, **SUPG**, for a streamwise upwinded Petrov-Galerkin formulation. • <float> - the weight function parameter, chosen between 0. and 1.. The value 0. corresponds to **GALERKIN** weighting and 1. corresponds to a full **SUPG**.

When this card is absent, the default {model_name} is **GALERKIN**.

Examples

The following is a sample input card:

```
Species Weight Function = SUPG 0.5
```

Technical Discussion

No Discussion.

References

No References.

Number of Chemical Reactions

```
Number of chemical reactions = <integer>
```

Description / Usage

This card is used to specify the number of electrochemical reactions being modeled in an electrode (anode or cathode), as in a thermal-battery cell.

Examples

Following is a sample card:

```
Number of chemical reactions = 1
```

Technical Discussion

No Discussion.

References

No References.

Reaction Rate

```
Reaction Rate = <model_name> <float1> <float2>
```

Description / Usage

This card is used to specify rates of species electrochemical reactions in the anode and cathode regions in a LiSi/LiCl-KCl/FeS2 thermal battery cell using Butler-Volmer kinetics.

This property currently allows for a single {model_name} which has two parameters:

ELECTRODE_KINETICS	the name of reaction rate model • <float1> - Anodic direction transfer coefficient • <float2> - Cathodic direction transfer coefficient

Two companion cards, *THERMODYNAMIC POTENTIAL* and *INTERFACIAL AREA* are required to complete the specification of parameters present in the Butler-Volmer kinetic model of current density.

Examples

The following are two sample cards:

```
Reaction Rate = ELECTRODE_KINETICS 0.5 0.5
```

```
Reaction Rate = ELECTRODE_KINETICS 1.0 1.0
```

Technical Discussion

No Discussion.

References

No References.

Thermodynamic Potential

```
Thermodynamic Potential = {model_name} {float_list}
```

Description / Usage

This card is used to specify the anodic or cathodic thermodynamic potential in a thermal battery cell.

{model_name}	Name of thermodynamic potential model. Currently, two thermodynamic potential models are available, namely **LiSi** and **FeS2**. Each of these and their accompanying input parameters (the <float_list>) is given below:
LiSi	This model requires seven floating-point parameters: • <float1> - Limit of electrode utilization for the first anode reaction. • <float2> - Limit of electrode utilization for the second anode reaction. • <float3> - Anode thickness. • <float4> - Anode porosity. • <float5> - Molar volume of active anode material. • <float6> - Current density output by the thermal battery cell. • <float7> - Number of electrons involved in anode reactions.
FeS2	This model requires eight floating-point parameters: • <float1> - Limit of electrode utilization for the first cathode reaction. • <float2> - Limit of electrode utilization for the second cathode reaction. • <float3> - Limit of electrode utilization for the third cathode reaction. • <float4> - Cathode thickness. • <float5> - Cathode porosity. • <float6> - Molar volume of active cathode material. • <float7> - Current density output by the thermal battery cell. • <float8> - Number of electrons involved in cathode reactions.

Examples

The following are two sample input cards:

```
Thermodynamic Potential = LiSi 0.283 0.474 0.088 0.275 54.61 0.0246 3.25
```

```
Thermodynamic Potential = FeS2 0.375 0.434 0.5 0.046 0.244 23.93 0.0246 4.0
```

Technical Discussion

No Discussion.

Interfacial Area

```
Interfacial Area = {model_name} {float_list}
```

Description / Usage

This card is used to specify the product of interfacial area per unit volume by exchange current density (i.e., ai0) in the Butler-Volmer kinetic model of current density.

{model_name}	Name of the model for interfacial area, of which there are currently two available, namely **CONSTANT** and **THERMAL_BATTERY**.
CONSTANT	constant value of Interfacial Area • <float1> - the value of the product of interfacial area per unit volume and exchange current density.
THERMAL_BATTERY	this option requires the following nine parameters: • <float1> - Initial value of the product of interfacial area per unit volume by exchange current density. • <float2> - Limit of electrode utilization beyond which ai0 = 0. • <float3> - Activation energy for the Arrhenius dependency of ai0 on temperature. • <float4> - Initial electrode/electrolyte temperature. • <float5> - Cathode thickness. • <float6> - Cathode porosity. • <float7> - Molar volume of active cathode material. • <float8> - Current density output by the thermal battery cell. • <float9> - Number of electrons involved in cathode reactions.

Examples

The following are two sample input cards:

```
Interfacial Area = CONSTANT 1.0
```

```
Interfacial Area = THERMAL_BATTERY 20.0 0.375 20000.0 846.0 0.046 0.244 23.93 0.0246␣
↪4.0
```

Technical Discussion

No Discussion.

References

No References.

Butler_Volmer_j

```
None
```

Unused; has been removed from *Goma* as of 12/20/2001

Butler_Volmer_ij

```
None
```

Unused; has been removed from *Goma* as of 12/20/2001

Solution Temperature

```
Solution Temperature = {model_name} <float_list>
```

Description / Usage

This card is used to specify the temperature of an electrolyte solution (i.e., when diffusion and migration transport of charged species is involved).

{model_name}	Name of the electrolyte-solution model, for which there are currently two options: **CONSTANT** and **THERMAL_BATTERY**; the former model has a single parameter in the <float_list> while the latter has six.
CONSTANT	A constant model of the solution temperature. • <float1> - the value of electrolyte-solution temperature.
THERMAL_BATTERY	A specialized model of electrolyte solutions for Thermal Batteries (Chen, et. al., 2000). • <float1> - Initial electrolyte solution temperature (K) • <float2> - Ambient temperature (K) • <float3> - Cross-sectional area from which heat is lost to ambient (m2 • <float4> - Heat transfer coefficient (W/m2/K) • <float5> - Mass of battery cell (kg) • <float6> - Heat capacity of electrolyte solution (J/kg/K)

Examples

The following are two sample input cards:

```
Solution Temperature = CONSTANT 313.0
```

```
Solution Temperature = THERMAL_BATTERY 846. 298. 0.0316 7.7 0.6 1030.
```

Technical Discussion

No Discussion.

References

SAND2000-0207: Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions, Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R., and Houf, W. G., January 2000.

Porosity

```
Porosity = {model_name} <float1> [float2]
```

Description / Usage

This card is used to specify the porosity model for the anode or separator or cathode region in a thermal battery cell.

Definitions of the {model_name} and the associated input parameters (<float>) are as follows:

CONSTANT	the name of the porosity model. • {float1} - the porosity value.
THERMAL_BATTERY	the name of the porosity model. • <float1> - the initial value of porosity • <float2> - specifies the change of molar volume in the anode or cathode electrode material per electron transferred, as stated in $\sum_{solid\ phases\ i} \frac{s_i \tilde{V}_i}{n}$ where si is stoichiometric coefficient of species or phase i, V is molar volume of species or phase i, n is the number of electrons transfer in the anodic or cathodic electrochemical reaction, and the summation is over the number of solid phases.

Examples

A sample input card for this material property might look like this:

```
Porosity = THERMAL_BATTERY 0.244 8.1185
```

Technical Discussion

- This is a porosity model for a special application in which the model for the diffusion constitutive equation is *STEFAN_MAXWELL_CHARGED*, which enables modeling the transport of multiple charged species with simultaneous electrochemical reaction(s) in a concentrated solution, as in a thermal-battery cell.
- See the reference below for a discussion of Thermal Battery modeling with *Goma*.

References

SAND2000-0207: Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions, K. S. Chen, G. H. Evans, R. S. Larson, D. R. Noble and W. G. Houf, January 2000.

Diffusivity

```
Diffusivity = {model_name} <species> <float_list> [varies]
```

Description / Usage

This required card is used to specify the model for the diffusivity for each species. Definitions of the input parameters are as follows:

{model_name}	Name of the diffusivity model. This parameter can have one of the following values: • **CONSTANT** • **USER** • **POROUS** • **GENERALIZED** • **FREE_VOL** • **GENERALIZED_FREE_VOL** • **HYDRO** • **ARRHENIUS** • **TABLE**
<species>	An integer designating the species equation.
{float_list}	the name of the porosity model. One or more floating point numbers (<float1> through <floatn> whose value is determined by the selection for {model_name}. Note that not all the models employ a {float_list}.

Thus, choices for {model_name} and the accompanying input parameter list are dependent on the {model_name} selected for the *Diffusion Constitutive Equation*. In some cases, the above model choices have special definitions, while for others some of the above choices do not exist. Thus, the presentation below is keyed to the value chosen for the *Diffusion Constitutive Equation* model.

When the *Diffusion Constitutive Equation* model is set to **NONE**, meaning the material block to which this material file applies is a non-diffusing material, this *Diffusivity* card should be present in the Material file specification but the model and its parameters will not be used.

For the **FICKIAN, GENERALIZED_FICKIAN, DARCY** and **DARCY_FICKIAN** flux models, the following options are valid choices for the Diffusivity {model_name} and accompanying parameter lists.

CONSTANT <species> <float1>	a constant diffusivity model • <species> - an integer designating species i • <float1> - Diffusivity of species i, in units [L2/t]
USER <species> <float_list>	a user-defined model, the <species> is specified and the set of parameters <float1> through <floatn> is defined by the function usr_diffusivity in the file user_mp.c.
POROUS <species> <float_list>	a diffusivity that depends on the saturation and porosity in a porous medium. For two-phase or unsaturated flow in a porous medium, the diffusivity calculated by this model is the diffusivity of solvent vapor through the gas phase in the pore-space This model has been deprecated as the porous equation rewrite has proceeded; it is not recommended for use!
GENERALIZED <species> <float1> <float2>	For constant diffusivities used by generalized Fick's law. The {float_list} consists of two values for each species i or i-j species pair: • <species> - an integer designating species i • <float1> - Dii, the self-diffusivity • <float2> - Dij; the mutual diffusivities
FREE_VOL <species> <floatlist>	For a diffusivity determined by free volume theory. The {float_list} for this model contains twelve values: • <species> - an integer designating species i • <float1> - V*1, solvent specific critical-hole volume • <float2> - V*2, polymer specific critical-hole volume • <float3> - K11/γ, solvent free volume parameter • <float4> - K12/γ, solvent free volume parameter • <float5> - K21 - Tg1, free volume/transition parameter • <float6> - K22 - Tg2, free volume/transition parameter • <float7> - χ, Flory-Huggins polymer/solvent interaction parameter • <float8> - ξ, ratio of solvent and polymer jumping units • <float9> - D01, binary diffusivity for 0-1 system [L2/t] • <float10> - E/R, ratio of activation energy to gas constant • <float11> - V01, solvent specific volume • <float12> - V02, polymer specific volume Note, this model can be run only with a single species equation, i.e., two components
GENERALIZED_FREE_VOL <species> <floatlist>	For constant diffusivities used by generalized Fick's law. The {float_list} consists of two values for each species i or i-j species pair: • <species> - an integer designating species i • <float1> - Dii, the self-diffusivity • <float2> - Dij; the mutual diffusivities
FREE_VOL <species> <floatlist>	For a diffusivity determined by free volume theory. The {float_list} for this model contains twelve values: • <species> - an integer designating species i • <float1> - V* 1, solvent specific critical-hole volume • <float2> - V* 2, polymer specific critical-hole vol-

For the **HYDRODYNAMIC** flux model (*Diffusion Constitutive Equation*), there is only one valid choice for the *Diffusivity* {model_name}, i.e., **HYDRO**. There are no accompanying parameters but several additional cards are required to define different portions of the model; these cards are identified below. The user is referred to each individual card (identified by italic typeset) definition for the associated model choices and parameter lists.

HYDRO	For mass transport driven by the hydrodynamic field. No <species> or {float_list} is required, although five additional input cards are required with this diffusivity model. The first specifies Dc in the Shear Rate Diffusivity card. The second specifies Dμ in the Viscosity Diffusivity card. The third specifies Dr in the Curvature Diffusivity card. The fourth specifies the diffusivity of a purely Fickian diffusion mode in the Fickian Diffusivity card; it is usually set to zero. The last card specifies Dg, in the Gravity-based Diffusivity card for the flotation term in variable density transport problems.
ARRHENIUS <integer 1 <integer 2> <float1> <float2> <float3>	This is a model for describing effect of temperature on Stefan-Maxwell diffusivities for application in modeling thermal batteries and thus it is used in conjunction with the **STEFAN_MAXWELL_CHARGED** or **STEFAN_MAXWELL** flux model (*Diffusion Constitutive Equation*). Two integers and three floats are required for this diffusivity model: • <integer 1> index for species i. • <integer 2> index for species j. • <float 1> Stefan-Maxwell diffusivity, dij in units [L2/t]. • <float 2> activation energy, ED . • <float 3> reference temperature, T0 . • Note: the units of ED and T0 are such that is dimensionless with R being the universal gas constant.

For the **FICKIAN_CHARGED** flux model (*Diffusion Constitutive Equation*), only constant diffusivities are allowed. So the *Diffusivity* model option is:

CONSTANT <species> <float1>	a constant diffusivity model • <species> - an integer designating species i • <float1> - Diffusivity of species i, in units [L2/t]

In addition, the *Charge Number* and *Solution Temperature* cards must also be specified in the material file so that the migration flux may be calculated.

The **STEFAN_MAXWELL** and **STEFAN_MAXWELL_CHARGED** flux models (*Diffusion Constitutive Equation*) should be used to model the transport of two or more species only. The diffusivity model for species in these transport problems is currently limited to being **CONSTANT** and **ARRHENIUS**. In the **CONSTANT&** Stefan- Maxwell diffusivity model, a set (only n(n-1)/2 values since Dij = Dji and Dii are not defined) of diffusivities, Dij, is required:

CONSTANT <species> <float1>	a constant diffusivity model • <species> - an integer designating species i • <species> - an integer designating species j • <float1> - Dij, mutual diffusivity of species i and j, in units [L2/t]

In addition, the *Charge Number, Molecular Weight* and *Solution Temperature* cards must also be specified in the material file so that the migration flux may be calculated.

Examples

Sections of material input files are shown below for several of the Diffusivity model options presented above.

Following is a sample input card for the **CONSTANT** *Diffusivity* model:

```
Diffusivity = CONSTANT 0 1.
```

Following is a sample section of the material file for the **HYDRO** *Diffusivity* model:

```
Diffusion Constitutive Equation = HYDRODYNAMIC
```

```
Diffusivity = HYDRO 0
```

```
Shear Rate Diffusivity = LINEAR 0 6.0313e-5
```

```
Viscosity Diffusivity = LINEAR 0 6.0313e-5
```

```
Curvature Diffusivity = CONSTANT 0 -48.02e-6
```

```
Fickian Diffusivity = ANISOTROPIC 0 0. 0.1e-5 0.
```

```
Gravity-based Diffusivity = RZBISECTION 0 2.14e-5 5.1 0.5 0.5
```

Following is a sample section of the material file for the **GENERALIZED_FREE_VOL** *Diffusivity* model:

```
Diffusion Constitutive Equation = GENERALIZED_FICKIAN
```

```
Diffusivity = GENERALIZED_FREE_VOL 1 0.943 1.004 0.000983 0.000239 -12.12 -96.4 0.395␣
↪0.266 0.00143 0 1.265983036 0.9233610342
```

Sample section of the material file for the **STEFAN_MAXWELL_CHARGED** *Diffusion Constitutive Equation* with the CONSTANT *Diffusivity* model:

```
Diffusion Constitutive Equation = STEFAN_MAXWELL_CHARGED
```

```
Diffusivity = CONSTANT
```

```
0 1 2.0e-05
```

```
0 2 2.0e-05
```

```
1 2 2.0e-05
```

```
Solution Temperature = THERMAL_BATTERY 846. 298. .03 7.7 0.6 1030.
```

```
Molecular Weight = CONSTANT 0 6.939
```

```
Charge Number = CONSTANT 0 1.0
```

```
Molecular Weight = CONSTANT 1 39.098
```

```
Charge Number = CONSTANT 1 1.0
```

```
Molecular Weight = CONSTANT 2 35.4
```

```
Charge Number = CONSTANT 2 -1.0
```

Sample section of the material file for the **STEFAN_MAXWELL_CHARGED** *Diffusion Constitutive Equation* with the **ARRHENIUS** *Diffusivity* model:

```
0 1 1.5e-05 80000.0 846.0
```

```
0 2 1.5e-05 80000.0 846.0
```

```
1 2 1.5e-05 80000.0 846.0
```

(the *Charge Number, Molecular Weight* and *Solution Temperature* cards are similarly specified as above in the **CONSANT** *Diffusivity* case)

Technical Discussion

Following are brief comments on the various *Diffusivity* models.

POROUS For this model, diffusivity depends on the saturation and porosity in a porous medium. For two-phase or unsaturated flow in a porous medium, the diffusivity calculated by this model is the diffusivity of solvent vapor through the gas phase in the pore-space (see Martinez, 1995). However as indicated above, this model is not recommended for use at his time.

GENERALIZED This model generalizes Fick's Law for multicomponent diffusion. The elements along the diagonal, Dii, are self-diffusivities, while Dij are mutual diffusivities between species i and j. Note that mutual diffusivities in generalized formulation can be both positive and negative, and are constant values.

FREE_VOL For a diffusivity determined by free volume theory (cf. Duda et al. 1982). In mathematical form, the binary mutual diffusion coefficient (solvent diffusion in a polymeric solution), using the free volume theory, is given by:

where

Here, ω1 is the solvent weight fraction, ω2 polymer weight fraction; V0 1 and V02 are, respectively, solvent and polymer specific volumes; φ1 solvent volume fraction, φ2 polymer volume fraction; γ overlap factor to account for shared free volume; Tg1 and Tg2 respectively solvent and polymer glass transition temperature, T absolute temperature; K11, K12, K21 and K22 solvent free-volume parameters; V* 1 and V*2 respectively, solvent and polymer specific critical-hole volumes; D01 constant preexponential factor when E is presumed to be zero (E is energy required to overcome attractive

$$D = D_{01}(1-\phi_1)^2(1-2\chi\phi_1)\exp\left[-\frac{(\omega_1 V_1^* + \omega_2 \xi V_2^*)}{V_{FH}/\gamma}\right]$$

$$\frac{V_{FH}}{\gamma} = \frac{K_{11}}{\gamma}\omega_1(K_{21}+T-T_{g1}) + \frac{K_{12}}{\gamma}\omega_2(K_{22}+T-T_{g2})$$

$$\phi_1 = \frac{\omega_1 V_1^0}{\omega_1 V_1^0 + \omega_2 V_2^0}$$

forces from neighboring molecules); ξ ratio of solvent and polymer jumping units; and χ Flory-Huggins polymer/solvent interaction parameter. In general, D01 should be expressed as D01 e- E/RT with R being the universal gas constant. Dependence of diffusivity, D, on temperature and mass fraction can be determined once the above twelve parameters are specified.

Note: This model (FREE_VOL) can be run ONLY with 1 species equation, i.e., with two components.

GENERALIZED_FREE_VOL This is a diffusivity model based on free volume theory and the generalized Fick's law. For a ternary mixture of solvent (1), solvent (2), and polymer (3), the concentration-dependent self-diffusivity is given by (Vrentas, et. al., 1984):

$$D_{11} = D_{01}\exp\left[-\frac{\left(\omega_1 V_1^* + \omega_2 V_2^* \frac{\xi_{13}}{\xi_{23}} + \omega_3 V_3^* \xi_{13}\right)}{V_{FH}/\gamma}\right]$$

where

The parameters for this model are the same twelve parameters as for the binary **FREE_VOL** model and so can be specified in the exact same order. The mutual diffusivities required to fill the cross-terms are also concentration-dependent. In addition, the gradient in chemical potential is also accounted for (Alsoy and Duda, 1999; Zielinski and Hanley, 1999).

$$\frac{V_{FH}}{\gamma} = \frac{K_{11}}{\gamma}\omega_1(K_{21} + T - T_{g1}) + \frac{K_{12}}{\gamma}\omega_2(K_{22} + T - T_{g2}) + \frac{K_{13}}{\gamma}\omega_2(K_{23} + T - T_{g3})$$

$$D_{ij} = D_{ii}\rho_i\frac{\partial \ln a_i}{\partial \rho_j}$$

$$\ln a_i = \ln \gamma_i \phi_i$$

ai is the activity of species i, which can be written in terms of the activity coefficient, γi, and volume fraction, φi. The current implementation of species activity is based on the Flory-Huggins model for multicomponent polymer-solvent mixtures (Flory, 1953).

HYDRO implies that mass transport of at least one species is driven by gradients in the second invariant of the rate of deformation tensor (shear rate) and gradients in viscosity (Phillips, et.al. 1992). This model also includes a sedimentation flux term to account for the motion of non-neutrally buoyant particles resulting from gravitation (Zhang and Acrivos, 1994) and a curvature-driven flux term from normal component of the acceleration vector (Krishnan, et. al., 1996). This model is used in predicting the particle distributions of particulate suspensions undergoing flow. For this model, the mass flux vector J is given by the following:

$$J = J_c + J_\mu + J_r + J_g$$

where

$$J_c = -C_i D_c \nabla(\dot{\gamma} C_i) \quad ,$$

$$J_\mu = C_i^2 \dot{\gamma} D_\mu \nabla(\ln \mu) \quad ,$$

$$J_r = C_i \dot{\gamma}^2 D_r \frac{\vec{n}}{r} \quad , \text{and}$$

$$J_g = D_g \frac{(\rho_s - \rho_f)}{\mu} C_i (1 - C_i) \vec{g}$$

where Ci is the particulate phase volume fraction, i is the species number designation of the particulate phase, the shear rate, μ the viscosity, the normal unit acceleration vector, r the curvature of streamlines, Dc, Dμ, Dr and Dg the “diffusivity”

parameters, ρs and ρf the particle and fluid phase densities, respectively, and , the gravitational acceleration vector.

ARRHENIUS Diffusivities can be strongly dependent on temperature as in processes such as thermal batteries. Such temperature dependency can be described using the following constitutive model that makes use of Arrhenius temperature dependency:

$$D_{ij} = D_{ij}^{0} e^{-\frac{E_D}{R}\left(\frac{1}{T}-\frac{1}{T_0}\right)}$$

where Dij are the Stefan-Maxwell diffusivities as defined in Equations 13 and 14. are the reference Stefan-Maxwell diffusivities at reference temperature T0; ED is the activation energy that controls the temperature dependency and R is the universal gas constant; and T is temperature. The units of ED, R and T are such that is dimensionless.

STEFAN-MAXWELL For multicomponent diffusion of neutral species in concentrated solutions. The Stefan-Maxwell diffusivities, Dij, as defined in the following Stefan-Maxwell flux model (cf. Chen, et. al., 2000, Chen, et. al., 1998):

$$\nabla x_i = \sum_{j \neq i} \frac{x_i J_j - x_j J_i}{c D_{ij}}$$

are taken to be constant. Here, xi is mole fraction of species i, Ji the molar flux of species i, and c the total molar concentration. Since Dij = Dji and Dii are not defined, only n(n-1)/2 Stefan-Maxwell diffusivities are required (here, n is the total number of diffusing species). For example, for n = 3 (i.e., a solution having three species), three Stefan-Maxwell diffusivities are needed: D12, D13, and D23.

STEFAN-MAXWELL_CHARGED For multicomponent transport (diffusion and migration) of charged species in concentrated electrolyte solutions. The Stefan- Maxwell diffusivities, Dij, as defined in the following Stefan-Maxwell flux model (cf. Chen et al. 2002, Chen, et. al., 2000, Chen, et. al., 1998):

are taken to be constant, as in the case of multicomponent diffusion of neutral species in concentrated solutions. Here, Φ is electrical potential in electrolyte solution, zi charge number of species i, F Faraday constant (96487 C/mole), R universal gas constant (8.314 J/mole-K), and T electrolyte solution temperature.

FICKIAN_CHARGED For multicomponent transport (diffusion and migration) of charged species in dilute electrolyte solutions. The Fickian diffusivity of species i, Di, as defined in the following Fickian flux model (cf. Newman, 1991; Chen, et. al., 2000):

is taken to be constant. Here, ci is molar concentration of species i.

$$\nabla x_i + \frac{F}{RT} z_i x_i \nabla \Phi = \sum_{j \neq i} \frac{x_i J_j - x_j J_i}{c D_{ij}}$$

$$J_i = -D_i \nabla c_i - \frac{F}{RT} z_i D_i c_i \nabla \Phi$$

FAQs

The following is a discussion of Units in *Goma* but covers several important Diffusionrelated items. It comes from some emails exchanged at Sandia during January 1998; while the discussions are relevant for each user of the code, the deficiencies or lack ofclarity have been since been remedied prior to *Goma* 4.0.

Unit Consistency in Goma (Jan 98)

Question:… I know what you are calling volume flux is mass flux divided by density. The point I am trying to make is that the conservation equations in the books I am familiar with talk about mass, energy, momentum, and heat fluxes. Why do you not write your conservation equations in their naturally occurring form? If density just so happens to be common in all of the terms, then it will be obvious to the user that the problem does not depend on density. You get the same answer no matter whether you input rho=1.0 or rho=6.9834, provided of course this does not impact iterative convergence. This way, you write fluxes in terms of gradients with the transport properties (viscosity, thermal conductivity, diffusion coefficient, etc.) being in familiar units.

Answer: First let me state the only error in the manual that exists with regard to the convection-diffusion equation (CDE) is the following:

Ji in the nomenclature table should be described as a volume flux with units of L/t, i.e. D · ∇yi, where D is in L2/t units.

Now, this is actually stated correctly elsewhere, as it states the Ji is a diffusion flux (without being specific); to be more specific here, we should say it is a "volume flux of species i." So, in this case D is in L · L/t units, yi is dimensionless and it is immaterial that the CDE is multiplied by density or not, *as long as density is constant*.

Now, in *Goma* we actually code it with no densities anywhere for the FICKIAN diffusion model. For the HYDRO diffusion model, we actually compute a Ji/ρ in the code, and handle variable density changes through that . In that case Ji as computed in *Goma* is a mass flux vector, not a volume flux vector, but by dividing it by and sending it back up to the CDE it changes back into a volume flux. i. e., everything is the same.

Concerning the units of the mass transfer coefficient on the YFLUX boundary condition, the above discussion now sets those. *Goma* clearly needs the flux in the following form:

and dimensionally for the left hand side

where D is in units L2/t, the gradient operator has units of 1/L so K has to be in units of L/t (period!) because yi is a fraction.

$$\underset{\sim}{n} \cdot D\nabla Y = K \cdot (y_i - y_i^{\infty})$$

$$(L^2/t) \cdot (1/L) = L/t$$

$$\underset{\sim}{n} \cdot D\nabla Y = \hat{K}(p_i - p_i^{\infty})$$

then K's units will have to accommodate for the relationship between p1 and y1 in the liquid, hopefully a linear one as in Raoult's law, i.e. if pi = PVyi where pv the vapor pressure, then

$$\underline{n} \cdot D\nabla Y = KP_V(y_i - y_i^{\infty})$$

and so K on the YFLUX command has to be KPv ….and so on.

Finally, you will note, since we do not multiply through by density, you will have to take care of that, i. e., in the Price paper (viz., Price, et. al., 1997) he gives K in units of t/L. So, that must be converted as follows:

$$K_{price}(P_V/\rho) = K_{goma} \quad : \quad (t/L)(M/Lt^2)(L^3/M) = L/t$$

This checks out!

References

Alsoy, S. and Duda, J. L., 1999. "Modeling of Multicomponent Drying of Polymer Films." AIChE Journal, (45)4, 896-905.

Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R. and Houf, W. G. "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", SAND2000- 0207, Sandia National Laboratories Technical Report (2000).

Chen, K. S., Evans, G. H., Larson, R. S., Coltrin, M. E. and Newman, J. "Multidimensional modeling of thermal batteries using the Stefan-Maxwell formulation and the finite-element method", in Electrochemical Society Proceedings, Volume 98-15, p. 138-149 (1998).

Chen, K. S., "Modeling diffusion and migration transport of charged species in dilute electrolyte solutions: GOMA implementation and sample computed predictions from a case study of electroplating", Sandia memorandum, September 21, 2000.

Chen, K. S., Evans, G. H., and Larson, R. S., "First-principle-based finite-element modeling of a Li(Si)/LiCl-KCl/FeS2 thermal battery cell", in Electrochem. Soc. Proc. Vol. 2002-30, p. 100 (2002).

Duda, J. L., Vrentas, J. S., Ju, S. T. and Liu, H. T. 1982. "Prediction of Diffusion Coefficients for Polymer-Solvent Systems", AIChE Journal, 28(2), 279-284.

P.J. Flory, Principles of Polymer Chemistry, Cornell University Press, 1953. Krishnan, G. P., S. Beimfohr and D. Leighton, 1996. "Shear-induced radial segregation in bidisperse suspensions," J. Fluid Mech. 321, 371.

Martinez, M. M., Mathematical and Numerical Formulation of Nonisothermal Multicomponent Three-Phase Flow in Porous Media, SAND95-1247, Sandia National Laboratories Technical Report, 1995.

Newman, J. S., Electrochemical Systems, Prentice Hall, Inc., Englewood Cliffs, New Jersey (1991).

Phillips, R.J., R.C. Armstrong and R.A. Brown, 1992, "A constitutive equation for concentrated suspensions that accounts for shear-induced particle migration," Physics of Fluids A, 4(1), 30-40.

Price, P. E., Jr., S. Wang, I. H. Romdhane, "Extracting Effective Diffusion Parameters from Drying Experiments," AIChE Journal, 43, 8, 1925-1934 (1997)

Vrentas, J.S., J.L. Duda and H.-C. Ling, 1984. "Self-Diffusion in Polymer-Solvent- Solvent Systems" Journal of Polymer Sciences: Polymer Physics edition, (22), 459- 469.

Zhang K. and A. Acrivos, 1994, "Viscous resuspension in fully-developed laminar pipe flows," Int. J. Multiphase Flow, (20)3, 579-591.

Zielinski, J.M. and B.F. Hanley, 1999. "Practical Friction-Based Approach to Modeling Multicomponent Diffusion." AIChE Journal, (45)1, 1-12.

Shear Rate Diffusivity

```
Shear Rate Diffusivity = {model_name} <species> <float>
```

Description / Usage

This card is used to specify the coefficient for the shear-rate gradient term when **HYDRO** is specified in the *Diffusivity* card. Definitions of the input parameters follow for the {model_name} options **CONSTANT** and **LINEAR** based on the model:

$$D_c = 1.4 k_c C_i$$

CONSTANT	Name of the model for constant shear rate diffusivity. • <species> - an integer designating the species equation. • <float> - Dc when there is no concentration dependency.
LINEAR	Name of the model in which shear rate diffusivity is a linear function of concentration. • <species> - an integer designating the species equation. • <float> - kc when the diffusivity is a linear function of concentration.

Examples

The following is a sample input card:

```
Shear Rate Diffusivity = CONSTANT 0 0.
```

Technical Discussion

Please refer to the technical discussion given under **HYDRO** section of the *Diffusivity* card.

References

No References.

Viscosity Diffusivity

```
Viscosity Diffusivity = {model_name} <species> <float>
```

Description / Usage

This card is used to specify Dr when the model in the *Diffusivity* card is **HYDRO**. Definitions of the input parameters follow for the {model_name} options **CONSTANT** and **LINEAR** based on the model:

$$D_\mu = 1.4 k_\mu C_i$$

CONSTANT	Name of the model for a constant curvature diffusivity. • <species> - An integer designating the species equation. • <float> - Dμ when there is no concentration dependency.
LINEAR	Name of the model in which the diffusivity is a linear function of concentration. • <species> - an integer designating the species equation. • <float> - kμ when the diffusivity is a linear function of concentration.

Examples

The following is a sample input card:

```
Viscosity Diffusivity = CONSTANT 0 0.
```

Technical Discussion

Please refer to the technical discussion given under **HYDRO** section of the Diffusivity card.

References

No References.

Curvature Diffusivity

```
Curvature Diffusivity = {model_name} <species> <float>
```

Description / Usage

This card is used to specify Dr when the model in the *Diffusivity* card is **HYDRO**. Definitions of the input parameters follow for the {model_name} options **CONSTANT** and **LINEAR** based on the model:

$$D_r = 1.4 k_r C_i$$

CONSTANT	Name of the model for a constant curvature diffusivity. • <species> - An integer designating the species equation. • <float> - Dr when there is no concentration dependency.
LINEAR	Name of the model in which the diffusivity is a linear function of concentration. • <species> - an integer designating the species equation. • <float> - kr when the diffusivity is a linear function of concentration.

Examples

The following is a sample input card:

```
Curvature Diffusivity = CONSTANT 0 0.
```

Technical Discussion

It was proposed that adding a curvature contribution of the diffusive flux for suspension particles would correct suspension migration behavior in parallel-plate and cone-and-plate. However, this correction term is not frame-invariant; hence, it cannot be used in generalized flow geometry. It is therefore not recommended.

References

No References.

Fickian Diffusivity

```
Fickian Diffusivity = {model_name} <species> {float_list}
```

Description / Usage

This card allows the user to select a Fickian diffusion mode when the model in the *Diffusivity* card is **HYDRO**. There are two {model_name} options for this mode; definitions of the input parameters are as follows:

ANISOTROPIC	an anisotropic Fickian diffusion. • <species> - an integer designating the species equation. • <float1> - the value of the diffusivity for the X direction, Dx. • <float2> - the value of the diffusivity for the Y direction, Dy. • <float3> - the value of the diffusivity for the Z direction, Dz.
EXP_DECAY	an exponential decay of flux. • <species> - an integer designating the species equation. • <float1> - the coefficient to the exponential decay, Do • <float2> - the exponent value for exponential decay, D1

Examples

Following are two sample cards:

```
Fickian Diffusivity = ANISOTROPIC 0 2.e-6 2.e-6 0.
```

```
Fickian Diffusivity = EXP_DECAY 0 0.01 1.e-3
```

Technical Discussion

In modeling suspension flow, often a sharp concentration gradient is encountered, and the numerical convergence becomes very poor. This card should be used for numerical stability (smooth out the wiggles) and should only be introduced as a last resort. The magnitudes should remain small relative to shear rate and viscosity diffusivities.

As the name implied, anisotropic Fickian diffusivity defines an additional flux contribution much like a classic Fickian diffusion term; i.e.,

$$J_x = D_x \frac{\partial C}{\partial x}, \qquad J_y = D_y \frac{\partial C}{\partial y}, \qquad J_z = D_z \frac{\partial C}{\partial z}$$

If the exponential decay option is used, the flux vector has the form,

$$J_i = D_o(\exp(-D_1 C) + \exp(-D_1(C_{max} - C)))$$

where C and Cmax are volume fractions of suspension locally and at maximum packing.

References

No References.

Gravity-based Diffusivity

```
Gravity-based Diffusivity = {model_name} <species> {float_list}
```

Description / Usage

This card is used to specify Dg when the model in the *Diffusivity* card is **HYDRO**. There are two {model_name} options for this mode; definitions of the input parameters are as follows:

CONSTANT	constant gravity-based diffusivity. • <species> - an integer designating the species equation. • <float>- the value of Dg.
RICHARDSON_ZAKI	constant gravity-based diffusivity. • <species> - an integer designating the species equation. • <float1> - the value of Dg. • <float2> - the exponent in the Richardson-Zaki hindered settling function.

Examples

The following is a sample input card:

```
Gravity-based Diffusivity = CONSTANT 0 8.88e-7
```

```
Gravity-based Diffusivity = RICHARDSON_ZAKI 0 8.88e-7 5.1
```

Technical Discussion

When a suspension of particles settles or floats in a fluid, particle-particle interactions serve to slow the terminal velocity of all the particles relative to the Stokes velocity. The terminal velocity is then corrected by what is known as the hindered settling function. If a **CONSTANT** model is chosen, the form of this function is

$$f(\phi) = \frac{(1-\phi)}{\eta(\phi)} \quad \eta(\phi) = \frac{\mu(\phi)}{\mu_0}$$

where φ is the volume fraction of suspension, η(φ) is the relative viscosity of the mixture, μ0 is the viscosity of the pure fluid.

On the other hand if **RICHARDSON_ZAKI** is chosen for the function,

where n is the exponent specified by the user. n=5.1 has been found to fit well for suspensions of monodisperse spherical particles at low Reynolds number by Garside and Al-Dibouni (1977). Richardson-Zaki approach will not yield a zero f(φ) if φ approaches maximum packing, so it is recommended that **CONSTANT** is used.

$$f(\phi) = (1 - \phi)^n$$

References

GTM-010.0: The Hindered Settling Function for a Glass Microballoon Suspension, March 3, 1999, C. A. Romero.

Garside, J. and M.R. Al-Dibouni, "Velocity-voidage relationship for fluidization and sedimentation in solid-liquid systems," Ind. Eng. Chem. Process Des. Dev., 16, 206 (1977).

Q Tensor Diffusivity

```
Q Tensor Diffusivity = <integer> <float_list>
```

Description / Usage

This card specifies the coefficients for use in the Q-tensor suspension rheology model. The <float_list> has three values, one for each direction, so the input parameters are defined as follows:

<integer>	Species number for suspension volume fraction.
<float1>	Coefficient of eigenvectors in the flow direction.
<float2>	Coefficient of eigenvectors in the gradient direction.
<float3>	Coefficient of eigenvector in the vorticity direction.

Examples

The current best selection of coefficients is given by:

```
Q Tensor Diffusivity = 0 1.0 1.0 0.5
```

Technical Discussion

The three directions (1, 2, 3) are often called the (flow, gradient, vorticity) directions. Here, vorticity is *not* curl(u), but defined (along with the other three) for a particular set of circumstances: steady simple shear flow. Their analogous definitions in other regimes, as well as the selection of the coefficients, is an active area of research. The interested reader should review the references listed below.

References

Brady, J. F. and Morris J. F., "Microstructure of strongly sheared suspensions and its impact on rheology and diffusion," J. of Fluid Mechanics, v. 348 pp.103-139, Oct 10, 1997.

Fang, Z. W., Mammoli, A. A., Brady, J.F., Ingber, M.S., Mondy, L.A. and Graham, A.L., "Flow-aligned tensor models for suspension flows," Int. J. of Multiphase Flow, v. 28(#1) pp. 137-166, January 2002.

Hopkins, M. M., Mondy, L. A., Rao, R. R., Altobelli, S. A., Fang, Z., Mammoli, A. A. and Ingber, M. S., 2001. "Three-Dimensional Modeling of Suspension Flows with a Flow-Aligned Tensor Model", The 3rd Pacific Rim Conference on Rheology, July 8- 13, 2001, Vancouver, B.C., Canada.

Morris, J. F. and Boulay, F., "Curvilinear flows of noncolloidal suspensions: The role of normal stresses," J. of Rheology, v. 43(#5) pp. 1213-1237 Sep-Oct 1999.

Species Time Integration

```
Species Time Integration= {model_name} <species>
```

Description / Usage

Sharp gradients are often a feature of convective-diffusive computations involving species. Traditional Galerkin time integration is not optimal under these circumstances. This optional card is used to change the species time integration scheme to be different from the global time integration. Each species equation can use a different time integration. The new time integration schemes are based upon a Taylor-Galerkin formulation which has better behavior when sharp fronts are present.

Following are the {model_name} options for species time integration, each of which requires only a species designation to which the model should be applied:

STANDARD	the input deck formulation, i.e., the global time integration scheme; this is the default. • <species> - the index of the species equation.
TAYLOR_GALERKIN	an implicit or semi-implicit Taylor-Galerkin time integration scheme • <species> - the index of the species equation.
TAYLOR_GALERKIN_EXP	An explicit Taylor-Galerkin time integration scheme • <species> - the index of the species equation.

Examples

The following sample input card invokes the explicit Taylor-Galerkin time integration of the species equation.

```
Species Time Integration = TAYLOR_GALERKIN_EXP 0
```

Technical Discussion

The Taylor-Galerkin schemes are designed for advection dominated problems with sharp fronts where rigorous mass conservation is important.

- **TAYLOR_GALERKIN** uses an implicit or semi-implicit form of the Taylor- Galerkin time integrals depending on what is chosen in the input deck.
- **TAYLOR_GALERKIN_EXP** uses an explicit form of the equations and is favored for volume-of-fluid simulations where the diffusive character of the implicit solver creates mass balance errors. The drawback of explicit time integration methods is that the time step used is governed by the Courant limit and must be quite small for stability.

References

No References.

Advective Scaling

```
Advective Scaling = {model_name} <species> <float>
```

Description / Usage

This material property card permits the user to scale only the advective terms in one or more of the species transport equations by a fixed constant. This may be useful when solving problems with non-standard concentrations or for stability reasons.

A single {model_name} is available; it and its parameters are described below:

CONSTANT	Model used to specify the advective scaling. • <species> - the index of the species equation to which the advective scaling will occur. • <float> - scaling, the actual value for the multiplicative scaling factor.

Examples

Here is an example of the card:

```
Advective Scaling = CONSTANT 0 0.0
```

In this case, the card is being used to eliminate the advective terms in the conservation equation for species 0.

Technical Discussion

The advective terms in the species conservation equations take the form, $u \cdot \nabla c$ where c is the species concentration and u the fluid velocity.

References

No References.

Latent Heat Vaporization

```
Latent Heat Vaporization = CONSTANT <species> <float> [E/M]
```

Description / Usage

This required card is used to specify the model for the latent heat of vaporization for each species. Definitions of the input parameters are as follows:

CONSTANT	Name of the constant latent heat of vaporization model. • <species> - an integer designating the species equation. • <float> - the value of the latent heat of vaporization.

Examples

The following is a sample input card:

```
Latent Heat Vaporization = CONSTANT 0 0.0
```

Technical Discussion

See the discussion for the *Latent Heat Fusion* model.

References

No References.

Latent Heat Fusion

```
Latent Heat Fusion = CONSTANT <species> <float> [E/M]
```

Description / Usage

This card is used to specify the model for the latent heat of fusion for each species. Thus an input deck may include several of these cards. Definitions of the input parameters are as follows:

CONSTANT	Name of the latent heat of fusion model, the only one available. • <species> - an integer designating the species equation. • <float> - the value of the latent heat of fusion.

Examples

The following is a sample input card:

```
Latent Heat Fusion = CONSTANT 0 0.0
```

Technical Discussion

This card is used on a species-basis and is unrelated to the latent heat of fusion specification for the **ENTHALPY** model of heat capacity. It is used to calculate the standard state heat of formation for the species. A related important card is the *Latent Heat Vaporization*.

References

No References.

Vapor Pressure

```
Vapor Pressure = {model_name} <species> {float_list} [varies]
```

Description / Usage

This required card is used to specify the model for the vapor pressure for each species; it has two main classes of use. The first class regards multiphase flow in porous media, which is activated when the media type is set to **POROUS_UNSATURATED** or **TWO_PHASE** (cf. the *Media Type* card). The second class of use of this data card is for specification of vapor pressure at the external boundary of a liquid domain, for which the bounding gas phase is modeled with a lumped parameter approach, or at an internal interface between a liquid and a gas. No curvature effects are included here. Eventually the models in this class will be supported in the porous-media cases. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the vapor pressure, based on the class of use. For the first class of multiphase flows in porous media, {model_name} can be one of the following: • **KELVIN** - for a volatile liquid • **IDEAL_GAS** - for a non-condensable gas • **FLAT** - for a volatile liquid For the second class regarding specification of vapor pressure at the external boundary of a liquid domain or the interface between a gas and a liquid, {model_name} can be one of the following: • **CONSTANT** - for a constant vapor pressure model • **ANTOINE** - for temperature-dependent, nonideal gases • **RIEDEL** -for temperature-dependent, nonideal gases
<species>	An integer designating the species equation. Typically this value is zero if the problem is one of a single solvent in a partially saturated medium.
{float_list}	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

$$p_v = p^*_v \exp\left[\frac{p_c M_w}{\rho_l RT}\right]$$

Vapor pressure model choices and their parameters are discussed below.

Models in the first class…

KELVIN <species> <float_list>	The <float_list> for the **KELVIN** option specifies input values for seven parameters: • <float1> - Equilibrium vapor pressure across a flat interface • <float2> - Liquid density • <float3> - Molecular weight of the liquid • <float4> - Gas law constant • <float5> - Operating temperature • <float6> - Molecular weight of air or gas phase • <float7> - Ambient pressure of that gas phase The **KELVIN** option is used to include the effect of vapor-pressure lowering that results in equilibrium over high curvature menisci, i.e., small pores. The equation form of this is
FLAT <species> <float_list>	The **FLAT** option requires the same seven parameters as the **KELVIN** model but leaves off the exponential function, i.e., the vapor pressure is independent of the level of capillary pressure. The constants are still needed so that the gas-phase concentration can be calculated with the ideal gas law. See the **KELVIN** option above for definition of the <float_list> values.
IDEAL_GAS <species> <float_list>	The <float_list> for this model has three values, where: • <float1> - Molecular weight of the gas • <float2> - Gas law constant • <float3> - Operating temperature

Models in the second class..

CONSTANT <species> <float1>	This model is used for a constant species source such as a homogeneous reaction term. The <float_list> has a single value: • <float1> - Vapor pressure
ANTOINE <species> <float_list>	The **ANTOINE** model for vapor pressure is used in conjunction with the *VL_EQUIL* boundary condition. If specified, a temperature-dependent vapor pressure for species i is calculated. The model requires six values in the <float_list>, where: • <float1> - A, the unit conversion factor for pressure based on the units in the material file • <float2> - Bi, Antoine coefficient for species i • <float3> - Ci, Antoine coefficient for species i • <float4> - Di, Antoine coefficient for species i • <float5> - Tmin, Minimum temperature of the range over which the Antoine relation will hold • <float6> - Tmax, Maximum temperature of the range over which the Antoine relation will hold
RIEDEL <species> <float_list>	The **RIEDEL** model for vapor pressure is used in conjunction with the *VL_EQUIL* boundary condition card. If specified, a temperature-dependent vapor pressure for species i is calculated. The model requires eight values in the <float_list>, where: • <float1> - A, the unit conversion factor for pressure based on the units in the material file • <float2> - Bi, Riedel constant for species i • <float3> - Ci, Riedel constant for species i • <float4> - Di, Riedel constant for species i • <float5> - Ei, Riedel constant for species i • <float6> - Fi, Riedel constant for species i • <float7> - Tmin, Minimum temperature of the range over which the relation will hold • <float8> - Tmax, Maximum temperature of the range over which the relation will hold

$$P_i^{vap} = A \exp\left(B_i + \frac{C_i}{(T + D_i)}\right) \quad T_{min} < T < T_{max}$$

$$P_i^{vap} = A \exp\left(B_i + \frac{C_i}{T} + D_i \ln T + E_i T^{F_i}\right) \quad T_{min} < T < T_{max}$$

Examples

An example use of the Antoine model for vapor pressure follows:

```
Vapor Pressure = ANTOINE 0 1 9.380340229 3096.516433 -53.668 0.1 1000
```

Technical Discussion

No Discussion.

References

No References.

Species Volume Expansion

```
Species Volume Expansion = CONSTANT <species> <float> [1/T]
```

Description / Usage

This card is used to specify the model for the coefficient of volume expansion associated with the concentration of a particular species. This property is optional for the **BOUSS** and **BOUSSINESQ** option on the *Navier-Stokes Source* card and, if nonzero, will result in a buoyancy term to be added to the Navier-Stokes equation that is apportioned to the species volume expansion coefficient, defined as the logarithmic sensitivity of density to concentration, or (dlnr)/(dC).

CONSTANT	Name of the constant volume expansion coefficient model. • <species> - an integer designating the species equation. • <float> - the value of the constant expansion coefficient.

Examples

The following is a sample input card:

```
Species Volume Expansion = CONSTANT 0 0.
```

Technical Discussion

WARNING: Please be aware that if the thermal volume expansion coefficient is also nonzero, the buoyancy force will be augmented.

References

No References.

Standard State Chemical Potential

```
Standard State Chemical Potential = CONSTANT <integer> <float>
```

Description / Usage

This card sets the standard state chemical potential of a species, <integer>, in the current material to a specified value, <float>. Currently, only the generic CONSTANT model is implemented. However, extensions to polynomial expressions in the temperature are easily implemented and forthcoming.

CONSTANT	Model name for the standard chemical state chemical potential model. • <species> - an integer designating the species equation. • <float> - the value of the chemical potential

The standard state chemical potential, μk, o(T) , which is defined to be only a function of the temperature, is used in the evaluation of the definition of the pure species chemical potential of species k, μk (T, P) , which in turn is used in the evaluation of the mixture chemical potential of species k, μk(T, P, Xi) .

Examples

The following is a sample input card:

```
Standard State Chemical Potential = CONSTANT 0 1.0
```

Technical Discussion

The values in this card are currently only applicable to the *IS_EQUIL_PSEUDORXN* boundary condition.

References

No References.

Pure Species Chemical Potential

```
Pure Species Chemical Potential = {model_name} <integer>
```

Description / Usage

This card takes the specification of the standard state chemical potential, which is defined as a function of temperature only, and completes the definition of the pure species chemical potential by possibly adding in a pressure dependence. Two model values are permissible:

PRESSURE_INDEPENDENCE	No pressure dependence to the pure species state when this value of {model_name} is specified. The standard state chemical potential is equal to the pure species chemical potential. The <integer> argument specifies the species subindex, k.
PRESSURE_IDEALGAS	The following expression holds for the pressure dependence: μk*(T, P) μ= k, o(T) + RTln(P ⁄ 1 atm) The <integer> argument specifies the species subindex, k. The standard state chemical potential, μk, o(T), which is defined to be only a function of the temperature, is used in the evaluation of the definition of the pure species chemical potential of species k, μk*(T, P) , which in turn is used in the evaluation of the mixture chemical potential of species k, μk (T P Xi).

Examples

Following is a sample card:

```
Pure Species Chemical Potential = PRESSURE INDEPENDENT 0
```

Technical Discussion

The values in this card are only applicable to the *IS_EQUIL_PSEUDORXN* boundary condition currently.

References

No References.

Chemical Potential

```
Chemical Potential = {IDEAL_SOLUTION | STOICHIOMETRIC_PHASE}
```

Description / Usage

This card is used to specify the formulation of the chemical potential for the phase. It is currently unconnected to *Goma's* functionality. Two values are permissible:

IDEAL_SOLUTION	Ideal solution thermodynamics
STOICHIOMETRIC_PHASE	Phase consists of fixed set of molecular composition

Examples

Following is a sample card:

```
Chemical Potential = IDEAL_SOLUTION
```

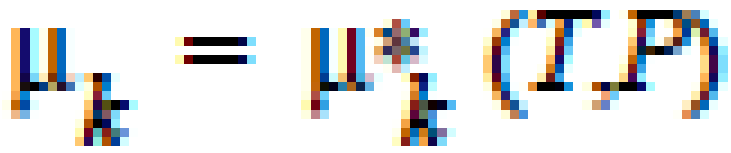
Technical Discussion

The chemical potential of species k in an ideal solution is given by the expression, [Denbigh, p. 249],

$$\mu_k = RT\ln\ (X_k) + \mu^*_k(T,P)$$

where μk*(T, P) is defined as the chemical potential of species k in its pure state (or a hypothetical pure state if a real pure state doesn't exist) at temperature T and pressure P. μk*(T, P) is related to the standard state of species k in the phase, μk, o(T), which is independent of pressure, through specification of the pressure dependence of the pure species k. Xk is the mole fraction of species k in the phase.

The chemical potential of species k (actually there is only one species!) in a stoichiometric phase is equal to



References

Denbigh, K., The Principles of Chemical Equilibrium, 4th Ed., Cambridge University Press, 1981

Reference Concentration

```
Reference Concentration = CONSTANT <species> <float> []
```

Description / Usage

This required card is used to specify the model for the reference concentration, which is required by the BOUSS option on the *Navier-Stokes Source* card. Definitions of the input parameters are as follows:

CONSTANT	Model for a constant reference concentration. • <species> - the species equation to which this specification applies. • <float> - the value of the reference concentration.

Examples

The following is a sample input card:

```
Reference Concentration = CONSTANT 0 0.
```

Technical Discussion

The Boussinesq model subtracts out the pressure head in its final equations. Thus, to zeroth order, hydrodynamic pressure field doesn't include a static variation in the gravity direction due to the pressure head. But, the source term in the momentum equations then becomes $-g(\rho - \rho o)$ instead of simply $-g\rho$. The reference concentration values entered via this card are used to evaluate ρo for use in calculating the natural convective force due to concentration differences.

The card is also used in various places where a value for a species concentration is needed. However, the species unknown variable is not included in the solution vector.

References

No References.

Molecular Weight

```
Molecular Weight = CONSTANT <integer> <float>
```

Description / Usage

This card specifies the molecular weight of a species. It is required when the Stefan- Maxwell flux model is used in modeling multicomponent transport of neutral or charged species. It is also required when vapor-liquid phase equilibrium is considered at the material boundaries. Molecular weight is used to convert units of mass fraction to mole fraction in a species material balance.

CONSTANT	Molecular weight model type. • <integer> - species number • <float> - molecular weight of the species

Examples

The following is a sample input card:

```
Molecular Weight = CONSTANT 0 6.939
```

Technical Discussion

This card originated from the development of a multicomponent diffusion model based on the Stefan-Maxwell equation. However, it has been generalized to include problems where mole fractions are necessary for the consideration of phase equilibria. For example, when *YFLUX_EQUIL* is invoked in the input deck, an equilibrium problem is solved rigorously which requires gas and liquid mole fractions. The conversion from mass fraction to mole fraction requires molecular weight information.

References

No References.

Specific Volume

```
Specific Volume = CONSTANT <integer> <float>
```

Description / Usage

This card specifies the specific volume of a species. It is required when polymersolvent vapor-liquid phase equilibrium is considered at the material boundaries. Specific volume is used to convert units of mass fraction to volume fraction in species material balance.

CONSTANT	Specific volume model type. • <integer> - species number • <float> - pure component specific volume

Examples

Following is a sample card:

```
Specific Volume = CONSTANT 0 1.154734411
```

Technical Discussion

This is the place where pure component density (inverse of specific volume) information is entered in the material property. When Flory-Huggins vapor-liquid equilibrium model was first developed in *Goma*, the equations were based on volume fractions, not mass fractions. In order to convert these units, the specific volume parameter is required for each component in the mixture.

This card is used only in conjunction with Flory-Huggins nonideal liquid activity model for polymer-solvent mixtures. This occurs when two types of BCs are specified: 1) when *VL_POLY* is specified at an discontinuous internal boundary and 2) when FLORY model under *YFLUX_EQUIL* boundary card is specified.

References

No References.

Molar Volume

```
Molar Volume = CONSTANT <integer> <float> [L3/mole]
```

Description / Usage

This card is referred when molar based equilibrium models are used on the boundaries, such as *VL_POLY*. The float value specified is necessary for converting mass fractions to mole fractions.

CONSTANT	Model for converting mass to mole fractions. • <integer> - species number • <float> - molar volume of the species. [L3/mole]

Examples

An example usage for this card:

```
Molar Volume = CONSTANT 0 1.
```

Technical Discussion

The same conversion from mass fraction to mole fraction can be obtained through specification of the *Molecular Weight* and *Specific Volume*. The redundancy, which will be allowed to remain, arose through simultaneous additions to the code by developers working on different projects.

References

No References.

Charge Number

```
Charge Number = CONSTANT <integer> <float>
```

Description / Usage

This card is required when charged species are involved, e.g. when using the **FICKIAN_CHARGED** or the **STEFAN_MAXWELL_CHARGED** *Diffusion Constitutive Equation* card. It specifies the charge number (e.g., the charge number for Ni2+ is 2, and that for SO2-is -2) of a species.

CONSTANT	Model for specifying constant charge on species. • <integer> - species number • <float> - charge number of the species

Examples

Sample usage for this card is shown belo

```
Charge Number = CONSTANT 0 1.0
```

Technical Discussion

No Discussion.

References

No References.

Non-condensable Molecular Weight

```
Non-condensable Molecular Weight = CONSTANT <integer> <float>
```

Description / Usage

This card specifies the molecular weight of a species when the species is implicit in the mixture. This means that in most problems involving n+1 species, only n species are independent; i.e.,

$$y_{n+1} = 1 - \sum_{i=1}^{n} y_i$$

It is required when Flory-Huggins vapor-liquid phase equilibrium is considered at the material boundaries, as used in *VL_POLY* and in **FLORY** under *YFLUX_EQUIL*. This is used to convert units of mass fraction to mole fraction in species material balance.

CONSTANT	Model for converting mass to mole fractions. • <integer> - species number • <float> - molecular weight of the non-condensable species, usually the n+1 component in *Goma* convention.

Examples

The following is an example card:

```
Non-condensable Molecular Weight = CONSTANT 2 36.
```

This example shows that two species are solved in the *Goma* problem explicitly: species 0 and species 1.

Technical Discussion

In the current set up, species balance in *Goma* considers the species to be independent of each other. However, the mass or volume fractions of all species must add up to unity in any mixtures. This means that some properties of the last species must be entered in the material file although that component is not solved explicitly in the problem. This is the case for molecular weight, molar volume, and specific volume specifications, which are required for calculating Flory-Huggins liquid activity.

References

No References.

Non-volatile Molar Volume

```
Non-volatile Molar Volume = CONSTANT <integer> <float>
```

Description / Usage

This card specifies the molar volume of a species when the species is implicit in the mixture. This means that in most problems involving n+1 species, only n species are independent; i.e.,

IIt is required when Flory-Huggins vapor-liquid phase equilibrium is considered at the material boundaries, as used in *VL_POLY* and in **FLORY** under *YFLUX_EQUIL*. This is used to convert units of mass fraction to mole fraction in species material balance.

CONSTANT	Model for converting mass to mole fractions. • <integer> - species number • <float> - molar volume of the non-volatile species, usually the n+1 component in *Goma* convention.

$$y_{n+1} = 1 - \sum_{i=1}^{n} y_i$$

Examples

The following is an example card:

```
Non-volatile Molar Volume = CONSTANT 2 1.5e-3
```

This example shows that two species are solved in the *Goma* problem explicitly: species 0 and species 1.

Technical Discussion

In the current set up, species balance in *Goma* considers the species to be independent of each other. However, the mass or volume fractions of all species must add up to unity in any mixtures. This means that some properties of the last species must be entered in the material file although that component is not solved explicitly in the problem. This is the case for molecular weight, molar volume, and specific volume specifications, which are required to calculate Flory-Huggins liquid activity.

References

No References.

Non-volatile Specific Volume

```
Non-volatile Specific Volume = CONSTANT <integer> <float>
```

Description / Usage

This card specifies the specific volume of a species when the species is implicit in the mixture. This means that in most problems involving n+1 species, only n species are independent; i.e.,

It is required when Flory-Huggins vapor-liquid phase equilibrium is considered at the material boundaries, as used in *VL_POLY* and in **FLORY** under *FLUX_EQUIL*. This is used to convert units of mass fraction to mole fraction in species material balance.

$$y_{n+1} = 1 - \sum_{i=1}^{n} y_i$$

CONSTANT	Model for converting mass to mole fractions. • <integer> - species number • <float> - specific volume of the non-volatile species, usually the n+1 component in *Goma* convention.

Examples

The following is an example card:

```
Non-volatile Specific Volume = CONSTANT 2 0.855e-3
```

This example shows that two species are solved in the *Goma* problem explicitly: species 0 and species 1.

Technical Discussion

In the current set up, species balance in *Goma* considers the species to be independent of each other. However, the mass or volume fractions of all species must add up to unity in any mixtures. This means that some properties of the last species must be entered in the material file although that component is not solved explicitly in the problem. This is the case for molecular weight, molar volume, and specific volume specifications, which are required for calculating Flory-Huggins liquid activity.

References

No References.

Flory-Huggins parameters

```
Flory-Huggins parameters = CONSTANT <integer1> <integer2> <float>
```

Description / Usage

This card specifies the Flory-Huggins binary interaction parameters. It is assumed that the binary parameters are symmetric; i.e.,

Therefore, one set of i-j coefficients is sufficient to describe the binary interaction coefficients.

CONSTANT	Model for constant Flory-Huggins parameters. • <integer1> - first species number. • <integer2> - second species number. • <float> - Flory-Huggins binary interaction coefficient.

Examples

Following is an example set of cards for a three-species mixture:

```
Flory-Huggins parameters = CONSTANT 0 1 0.3
Flory-Huggins parameters = CONSTANT 0 2 0.3
Flory-Huggins parameters = CONSTANT 1 2 0.3
```

This example shows that two species are solved in the *Goma* problem explicitly: species 0 and species 1.

Technical Discussion

No discussion; see Sun (1998).

References

GTM-007.1: New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

1.5.7 Source Terms

Source term models cover the internal generation of pressure (in fluids and solids), energy, species component mass and electrical potential for each of the main differential equations. Several representations are available for fluids, and the user should be aware of some consistencies required with density models (see details below). For all of the source models, the user must heed the following warning:

Make sure the equation term multipliers for the source terms being used are set to unity *(Section 4.12 - Problem Description and Equation specification in Volume 1).*

Navier-Stokes Source

```
Navier-Stokes Source = {model_name} {float_list} [varies]
```

Description / Usage

This required card is used to specify the model for the fluid momentum source term vector in the Navier-Stokes equations. Gravitational and buoyancy effects often enter through this card.

Definitions of the input parameters are as follows:

{model_name}	Name of the fluid momentum source term model for the Navier-Stokes equations. The model name will be one of the following strings: • **CONSTANT** • **USER** • **BOUSS** • **BOUSS_JXB** • **BOUSSINESQ** • **FILL** • **PHASE_FUNCTION** • **SUSPEND** • **SUSPENSION** • **VARIABLE_DENSITY** • **EHD_POLARIZATION** • **ACOUSTIC**
{float_list}	One or more floating point numbers (<float1> through <floatn>); the specific number is determined by the selection for {model _name}.

Choices for {model_name} and the accompanying parameter list are given below; additional user guidance can be found in the Technical Discussion section following the Examples.

CONSTANT <float1> <float2> <float3>	For a constant source model where the body force [M/L2t2] for this material does not vary. The {float_list} contains three values to specify the three components of the body force vector, where: • <float1> - a0, x-component of body force • <float2> - a1, y-component of body force • <float3> - a2, z-component of body force Note this source term has units of force/volume or, equivalently, density times acceleration. This is not true of all source term models.
USER <float1>… <floatn>	For a user-defined model; the set of {float_list} parameters are those required by specifications in the function usr_momentum_source.
BOUSS <float1> <float2> <float3>	This option specifies a generalized Boussinesq source where the density is linearly dependent upon temperature and concentration (species). The individual components of the constant acceleration vector a0 are read from the three entries in the {float_list}: • <float1> - a0, x-component of acceleration • <float2> - a1, y-component of acceleration • <float3> - a2, z-component of acceleration Unlike the **CONSTANT** model the units for these vector components are (L/t2), that is, they are true acceleration values. See the technical discussion below for the other parameters needed for this model.
BOUSSINESQ <float1> <float2> <float3>	This model prescribes a body force source term that is very similar to the BOUSS option except that the hydrostatic component is eliminated. The individual components of the constant acceleration vector a0 are read from the three entries in the {float_list}: • <float1> - a0, x-component of acceleration • <float2> - a1, y-component of acceleration • <float3> - a2, z-component of acceleration
BOUSS_JXB <float1> <float2> <float3> <float4>	This source model option specifies a generalized Boussinesq source term, as above, but also including Lorentz (electromagnetic) forces. The constant acceleration vector a0 is again specified using the first three constants that appear in the {float_list}. The fourth constant of the list is a Lorentz scaling factor (lsf). It may be used to scale the Lorentz term; see the Technical Discussion for more information. • <float1> - a0, x-component of acceleration • <float2> - a1, y-component of acceleration • <float3> - a2, z-component of acceleration • <float4> - lsf, Lorentz scaling factor.
EHD_POLARIZATION <float1>	This source model option can be used to add on a dielectrophoretic force to the Navier-Stokes equations of the form ρE • ∇E ., where E is the electric field vector and ρ is a user-supplied constant with dimensions [q2T2/ L3]. This term requires the vector *efield* equation and the *voltage* equation to be solved simultaneously with the fluid-phase momentum equation. cf. *EQ* card definitions. • <float1> is the constant ρ as described above

FILL <float1> <float2> <float3>	This model prescribes the body force momentum source term for problems making use of volume-of-fluid interface tracking. The card prescribes a constant accelera-

WARNING: Make sure the equation term multipliers for the source terms are set to unity.

Examples

Following are some sample input cards:

```
Navier-Stokes Source = BOUSS 0. -980. 0.
Navier-Stokes Source = LEVEL_SET 0. -980. 0.
```

Technical Discussion

This section contains user guidance, and theoretical background when appropriate, for each of the options for Navier-Stokes Source models.

CONSTANT	A constant source model has a body force [M/L2t2] for the material which does not vary. A common usage of this model is for an incompressible fluid in a uniform gravitational field. Note that the source term has units of force/volume or, equivalently, density times acceleration. Thus, the values in the {float_list} would need to be specified as the product of the fluid density and the acceleration of gravity.
USER	This model option provides a means for the user to create a custom Navier-Stokes Source model for his/her special problem. The parameters of the model will be used by the source term model defined in the usr_momentum_source function. The {float_list} parameters are passed to this function as a one dimensional array named param in the order in which they appear on the card. The model must return a body force (force/volume) vector. An example use of this specification might be to construct a Coriolis acceleration term for a fluid in a rotating reference frame.
BOUSS	A generalized Boussinesq source term has the form where the linear dependence of the density upon temperature and concentration is used for this source term only. Density is assumed constant wherever else it happens to appear in the governing conservation equations. The density has been expanded in a Taylor series to first order about a reference state that is chosen so that, at the reference temperature T0 and concentration C0 the density is ρ0. The reference density is taken from the CONSTANT density model specified earlier in the material file on the Density card. The coefficient of thermal expansion of the fluid, β, is taken from the Volume Expansion card specified under Thermal Properties for this material. βc, is taken from the Species Volume Expansion card specified under species Properties for this material. The individual components of the constant acceleration vector a0 are the three entries of the {float_list} after the BOUSS string. Note that this BOUSS form includes the body force of the reference state so that a motionless fluid at a uniform temperature of T0 must be sustained by a linearly varying pressure field. Below, an alternative means for solving Boussinesq problems is presented that eliminates the constant hydrostatic feature of the BOUSS formulation. T0 is set on the Reference Temperature card.

$$g(T) = \rho_0(1 - \beta[T - T_0] - \beta_c[C - C_0])a_0$$

BOUSSINESQ	This model prescribes a body force source term that is very similar to the BOUSS option except that the hydrostatic component is eliminated. Thus the form so that a no-flow solution with uniform temperature and concentration may be maintained by a constant pressure field. This form for the Boussinesq equations can sometimes provide a more well-conditioned equation system for weakly buoyant flows. Note again the implied convention that the coefficient of thermal expansion is positive when the density decreases with increasing temperature. That is, The same convention holds for the coefficient of solutal expansion. A source of confusion with buoyancy problems is that many sign conventions are applied. In addition to the convention for β, another possible source of confusion arises from a negative sign on the gravitational acceleration vector in many coordinate systems. That is, is a frequent choice for the constant acceleration for a twodimensional problem posed in Cartesian coordinates.T0 is set on the Reference Temperature card.

$$g(T) = -\rho_0\{\beta[T - T_0] + \beta_c[C - C_0]\}a_0$$

$$\beta = -\left(\frac{\partial \rho}{\partial T}\right)_{T = T_0}$$

$$a_0 = -g e_y$$

BOUSS_JXB	The JXB model is a generalized Boussinesq source term, as above, but also includes Lorentz forces. That is, the source term has the form where, in addition to the term defined for the BOUSS option, there is an added term due to electromagnetic forces acting upon a conducting fluid. The constant acceleration vector a0 is again specified using the first three constants that appear in the {float_list}. The fourth constant, lsf, may be used to scale the Lorentz term as desired (for example, lsf = 1 using a Gaussian system of units, or lsf = 1/c using a rationalized MKSA system of units). The two vector fields J, the current flux, and B, the magnetic induction, must be supplied to Goma in order to activate this option. At present, these fields must be supplied with the External Field cards, which provide the specific names of nodal variable fields in the EXODUS II files from which the fields are read. The three components of the J field must be called JX_REAL, JY_REAL, and JZ_REAL. Likewise the B field components must be called BX_REAL, BY_REAL, and BZ_REAL. These names are the default names coming from the electromagnetics code TORO II (Gartling, 1996). Because of the different coordinate convention when using cylindrical components, the fields have been made compatible with those arising from TORO II. It is the interface with TORO that also makes the Lorentz scaling (lsf) necessary so that the fixed set of units in TORO (MKS) can be adjusted to the userselected units in Goma. T0 is set on the Reference Temperature card.

$$g(T) = \rho_0(1 - \beta[T - T_0])a_0 + lsfJ \times B$$

FILL	The body force applied when using this momentum source model is as follows: where ρ1 and ρ0 are the phase densities obtained from the FILL density card, F is the value of the fill color function and the constant acceleration vector a0 is read from the three entries in the {float_list} of the FILL momentum source card.

$$g(F) = [\rho_1 F + \rho_0(1 - F)]a_0$$

LEVEL_SET	The body force applied when this model is used is given by the following function of the level set function value, φ: where is a smooth Heaviside function, φ is the value of the level set function, ρ+ and ρ- are the positive and negative phase densities, and α is the density transition length scale. The latter three parameters are obtained from the LEVEL_SET density card. The individual components of the constant acceleration vector a0 are three float parameters appearing in the {float_list} following the LEVEL_SET model name.

$$g(\phi) = \rho_- a_0, \qquad \phi < -\alpha$$

$$g(\phi) = [\rho_- + (\rho_+ - \rho_-)H_\alpha(\phi)]a_0, \qquad -\alpha < \phi < \alpha$$

$$g(\phi) = \rho_+ a_0, \qquad \phi > \alpha$$

where

$$H_\alpha(\phi) = (1 + \phi/\alpha + \sin(\pi\phi/\alpha)/\pi)/2$$

PHASE_FUNCTION	The body force applied when this model is specified is identical in concept to that applied with the above LEVEL_SET model. The parameters on this card are simply the components of a constant acceleration vector (gravity in most applications). This card must be used in conjunction with the CONST_PHASE_FUNCTION density model because the actual body force vector is obtained by multiplying the acceleration vector specified with this card by the density computed by that latter model. Again this is identical in concept to the LEVEL_SET body force source model.
SUSPEND	This model prescribes a body force source term that is for simulating suspensions when the suspending fluid and particle phase have different densities. The difference in density can lead to buoyancy driven flow. The form of the source term is given below: where Ci is the solid particle volume fraction tracked using a species equation with a HYDRO diffusion model. Four parameters must be set for this card using the {float_list}. The first three parameters are the three components of the gravity vector. The fourth parameter is a reference concentration, Cref. The density values are those entered by a SUSPENSION density model on the Density card. NOTE: If this momentum source term is used in conjunction with the HYDRODYNAMIC mass flux option, only one species can use the HYDRO diffusivity model.

$$g(C_i) = -(C_i - C_{ref})[\rho_f - \rho_s]a_0$$

SUSPENSION	This model is identical to the SUSPEND momentum source model in terms of the assembly of the momentum equation. However, this model creates a source term that gets applied during the assembly of the continuity equation due to transport of species with different densities. The suspension density models meet the definition of a locally variable density model, so the Lagrangian derivative of their densities can be represented as a divergence of mass flux. This term is integrated by parts and this particle phase flux is included separately as a source term for the continuity equation.
ACOUSTIC	This model contains the usual gravitational source terms in the CONSTANT model plus the gradient of the acoustic Reynolds stress as an additional momentum source. The acous_reyn_stress equation must be present to use this source model.

The user should take special note of the distinction between the different use of the {float_list} for **CONSTANT** body force problems and for the various buoyant options. For the **CONSTANT** model, the three components are the force per unit volume, and the user must remember to include density specifically if it is desired. For the buoyancy options, the three components are acceleration, and the density value specified on a previous card is automatically used by *Goma* to construct the overall body force source term. This is also true for the **FILL, LEVEL_SET, SUSPENSION** and **SUSPEND** momentum source models.

The user must also take special care that the source term multipliers for the momentum equation are set to unity.

References

Gartling, D. K., TORO II - A Finite Element Computer Program for Nonlinear Quasi- Static Problems in Electromagnetics, Part I - Theoretical Background, SAND95-2472, Sandia National Laboratories, Albuquerque, NM, May 1996.

Solid Body Source

```
Solid Body Source = CONSTANT <species_number> <float1> <float2> <float3>
```

Description / Usage

This card is used to specify the model for the body force source term on the solid mechanics momentum equations. This card is used most to impose gravitational forces on solid phase material elements in the problem. It can be also used to impose body forces on the pseudo-solid mesh material if that is desirable.

Definitions of the input parameters are as follows:

CONSTANT	A string identifying the constant force model. Currently, this is the only body force model for solid materials. • <float1> - the x-component of the body force vector in (F/L3). • <float2> - the y-component of the body force vector in (F/L3). • <float3> - the z-component of the body force vector in (F/L3).
JXB	A string identifying a body force model based on external current density fields J and external magnetic fields B. See Technical discussion. • <float1> - a scale factor, usually set to 1.

Examples

The following is a sample input card:

```
Solid Body Source = CONSTANT 0.0 0.0 -2000.0
```

Technical Discussion

Just as there is a body force vector that can be applied to fluid material regions, there is a capability to apply a similar body force term to solid material regions. Most often this is used to apply gravitational forces in which case the values of the components supplied on this card would be the solid density multiplied by the gravitational acceleration vector.

The **JXB** model requires external nodal fields loaded through External Field capability. These fields must be named JE_N_1, JE_N_2, and JE_N_3 for the three components of the current density and BE_N_1, BE_N_2, and BE_N_3 for the three components of the magnetic field.

References

No References.

Mass Source

```
Mass Source = CONSTANT 0.
```

Description / Usage

This source term is inactive in *Goma* but the card must be present in the input at this time. Definitions of the input parameters are as follows.

CONSTANT	Name of the model (to prevent heartburn for *Goma*). 0. A floating point number (the value zero).

Examples

Following is the only allowable card specification:

```
Mass Source = CONSTANT 0.
```

Technical Discussion

No Discussion.

References

No References.

Heat Source

```
Heat Source = {model_name} <float_list> [varies]
```

Description / Usage

This required card is used to specify the model for the source term on the energy equation. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the source term on the energy equation. The permissible values are • **CONSTANT** • **USER** • **USER_GEN** • **JOULE** • **EPOXY** • **VISC_DISS** • **BUTLER_VOLMER** • **ELECTRODE_KINETICS**
<float_list>	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. Note that not all models have a <float_list>.

Source-term model choices and their parameters are discussed below. WARNING: make sure the equation term multipliers for the source terms are set to unity (see the Equation Cards segment in the previous chapter).

CONSTANT <float1>	The constant source model adds a constant homogenous source term [E/L3t] to the heat equations. The <float_list> has a single value: <float1> - Heat source.
USER <float1>… <floatn>	The **USER** option indicates that a user-defined model has been introduced into the usr_heat_source routine in the user_mp.c file. The <float_list> is of arbitrary length subject to the user's requirements to parameterize the model.
USER_GEN <float1>… <floatn>	The **USER_GEN** option provides a user-defined model with low-level, general capabilities. For this option one must make the appropriate modifications to the routine usr_heat_source_gen in the user_mp_gen.c file. The difference between the **USER** and **USER_GEN** capabilities is described at the beginning of this chapter.
JOULE	The **JOULE** model is used to specify a Joule heating source term. No input is required for this model as the sole independent parameter of the model for the voltage equation is the "Electrical Conductivity", which is specified in the material file.
EPOXY <float1>	The **EPOXY** model is used to specify the heat generated by an epoxy curing reaction. The single input value is the: • <float1> - heat of reaction due to curing
VISC_DISS <float>	The **VISC_DISS** model is used to specify the heat generated by viscous dissipation. The <float_list> has a single value: • <float1> - A multiplier to facilitate different unit combinations. Selection of this option activates its use.
BUTLER_VOLMER <integer> <float1> <float2> <float3> <float4> <float5> <float6> <float7> <float8>	The **BUTLER_VOLMER** model is used to specify the current source or sink due to a homogeneuous electrochemical reaction involving a single species (e.g., the hydrogen oxidation and oxygen reduction reactions in a hydrogen-feuled polymerelectrolyte- membrane fuel cell), which is computed using the Butler-Volmer kinetic model (as described in the Theory section of the **BUTLER_VOLMER** current source card). This is due to that the voltage equation is used to solve for the electrical potential in the liquid electrolyte phase whereas the energy equation is utilized to solve for the electrical potential in the solid-electrode phase such that the electrode potential unknowns is represented by the temperature unknown. Parameters required for this **BUTLER_VOLMER** heat source model are the same as those for the **BUTLER_VOLMER** current source model; accordingly, detailed description of the model parameters can be found in the Current Source section of this manual.
ELECTRODE_KINETICS	The **ELECTRODE_KINETICS** model is used to specify the current generated or consumed in the solid electrode phase in electrochemical processes involving concentrated electrolyte solution and multiple species as in thermal batteries. As in the case of the **BUTLER_VOLMER** model, this is due to that the voltage equation is used to solve for the electrical potential in the liquid electrolyte phase whereas the energy equation is utilized to solve for the electrical potential in the solid- electrode phase such that the electrode potential unknowns is represented by the temperature unknown. The

Examples

The following is a sample input card:

```
Heat Source = CONSTANT 1.
```

Technical Discussion

The energy equation solved by *Goma*, which can be found elsewhere, is a convectionconduction equation given by

$$\frac{d(\rho C_p T)}{dt} = -(v - v_m) \bullet \nabla(\rho C_p T) - \nabla \bullet q + H$$

The heat source-term model represented by H is specified by this input record. The **CONSTANT, USER** and **USER_GEN** options provide the standard means of specifying model input and will not be discussed.

JOULE Model: The **JOULE** model is used to specify a Joule heating term. It is based on heat generation in a medium of specified electrical resistance subjected to an electrical voltage potential. It computes the heat source as:

$$h(\phi) = \frac{1}{\sigma} J \cdot J = \nabla\phi \cdot \sigma \nabla\varphi$$

where J is the current flux density which is represented as $-\sigma\nabla\varphi$ and φ is represented using the voltage equation. No input is required for this model since the *Electrical Conductivity* is specified in the material file for the voltage equation.

EPOXY Model: The **EPOXY** model is used to specify the heat generated by a condensation reaction, which is the heat of reaction, ΔHrxn, multiplied by the reaction rate as measured by the extent of reaction. The form of the equation is:

$$h(\alpha) = \Delta H_{rxn} \cdot R_\alpha$$

This card is used in conjunction with the EPOXY Species Source so that the reaction rate Rα can be determined.

VISC_DISS Model: In heat transfer problems that are accompanied by fluid flow, the energy balance equation contains a term which represents the (rate of) work done on the fluid by viscous forces. These forces have the potential to raise the fluid temperature and therefore it may be necessary to include these forces in your analysis. Typically, problems in which this term is significant may be characterized as highspeed flows with large velocity gradients, rapid extrusion and lubrication problems. The Brinkman number (Br = (μV2)/(kΔT)) is an indicator of the importance of viscous heating relative to the heat flow caused by temperature gradients.

$$R_\alpha = (k_1 + k_2\alpha^m)(1-\alpha)^n$$

$$H = (\tau : \nabla v)$$

Mathematically, the Heat Source term, *H*, in the energy equation is given by

where ((:) indicates a double dot product. This mechanism is an irreversible process whereby mechanical energy is degraded into thermal energy, as the right hand side expands to the sum of quadratic terms which will always be positive, at least for Newtonian fluids. In *Goma*, the **VISC_DISS** model computes the source term in function **visc_diss_heat_source** in *mm_std_models.c*; the expression looks like

$$multiplier \times \mu\dot{\gamma}^2$$

where μ is the viscosity and γ is the shear rate.

The *multiplier* allows the user the flexibility to choose appropriate units for the momentum and energy equations. For example, for many problems the momentum equations are scaled appropriately using viscosity units of psi-sec. But in the absence of this multiplier, this would force the energy equation to be in units of psi/sec; in other words, (*density*heat capacity*) would need to be in units of psi/(deg C) and thermal conductivity would need to be in units of psi-in2/(sec-deg C) - these aren't exactly common units! Instead, we can set the multiplier to 6891 (i.e., 6891 Pa = 1 psi) in order to have the energy equation in units of J/sec - i.e. leaving (*density*heat capacity*) in units of J/(m3 deg C) and thermal conductivity in terms of J/(m-sec deg C). Another use of the multiplier is to allow appropriate scales when the momentum equation is diffusion-dominated and the thermal equation is convection-dominated. Suppose we keep all quantities in MKS units. The convection terms in the thermal equation can then be scaled to roughly order unity by dividing through by (*density*heat capacity*) - i.e. set *heat capacity=1/density, thermal conductivity = thermal diffusivity*, and set the dissipation multiplier to *1/(density*heat capacity)*.

So, in essence, this multiplier allows flexibility in the choice of mechanical and thermal units in a convenient manner - i.e. it's on the term that couples the energy and momentum equations.

BUTLER_VOLMER and **ELECTRODE_KINETIC Models**: As mentioned above, these two models are used to specify the current generated or consumed in the solid electrode phase in electrochemical processes such as polymer-electrolyte-membrane fuel cells and thermal batteries. This is due to that the voltage equation is used to solve for the electrical potential in the liquid electrolyte phase whereas the energy equation is utilized to solve for the electrical potential in the solid-electrode phase such that the electrode potential unknown is actually represented by the temperature unknown.

Further details for the **BUTLER_VOLMER** model are presented in the current source model section of this manual and that for the **ELECTRODE_KINETIC** model can be found in the reference provided below (Chen et al. 2000).

VARIABLE_DENSITY Model: Work was begun on a **VARIABLE_DENSITY** model for drying problems but has not been completed. The roots for this may be found in the source code but the model is not yet functional; it was not listed above as a valid Heat Source option.

References

SAND2000-0207: Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions, K. S. Chen, G. H. Evans, R. S. Larson, D. R. Noble and W. G. Houf, January 2000.

Species Source

```
Species Source = {model_name} <species> <float_list> [varies]
```

Description / Usage

This required card is used to specify the model for the source term on the species convection diffusion equations. Definitions of the input parameters are as follows:

{model_name}	Name of the model for the source term on the species convection diffusion equations. The permissible values are • **CONSTANT** • **BUTLER_VOLMER** • **ELECTRODE_KINETICS** • **ELECTROOSMOTIC** • **EPOXY** • **EPOXY_DEA** • **FOAM** • **USER**
<species>	An integer designating the species equation.
<float_list>	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}.

Source-term model choices and their parameters are discussed below. Details are contained in the Technical Discussion section below. The <species> definition given above applies to all the following choices for which it is specified; its definition will not be repeated.

CONSTANT <species> <float1>	This model of a constant species source has a single input value: <float1>-Constant species source
BUTLER_VOLMER <species> <float1> <float2> <float3> <float4> <float5> <float6> <float7> <float8> <float9>	This is the homogeneous species source or sink term (in units of moles per unit volume, e.g. moles/cm3-s) as described by the Butler-Volmer kinetic model (see the Theory section below). One integer and 9 flotas are required: • <species> - Index of the species involved in the electrochemical reaction (here, we assume that only a single species is involved). • <float1> - Stoichiometric coefficient, s. • <float2> - Product of interfacial area per unit volume by exchange current density, ai0, in units of A/cm3. • <float3> - Reaction order, β. • <float4> - Reference species concentration, cref, in units of moles/cm3. • <float5> - Anodic transfer coefficient, αa. • <float6> - Cathodic transfer coefficient, αc. • <float7> - Temperature, T, in unit of K. • <float8> - Open-circuit potential, U0, in unit of V. • <float9> - Number of electrons involved in the reaction, n.
ELECTRODE_KINETICS	The **ELECTRODE_KINETICS** model is used to specify the species generation or consumption in electrochemical processes involving concentrated electrolyte solutions and multiple species such as thermal batteries. The {model_name} **ELECTRODE_KINETICS** toggles on the option in the equation assembly; no parameters are required.
ELECTROOSMOTIC <int1> <int2> <float1> <float2> <float3> <float4> <float5> <float6> <float7> <float8> <float9> <float10>	This is the source or sink term (in units of moles per unit volume, e.g. moles/cm3-s) for thw water species due to electro-osmotic drag by the protons (H+). Two integers and 10 flotas are required: • <int1> - Water species index. • <int2> - Index of the species involved in the electrochemical reaction that generates the electrical current (here, we assume that only a single species is involved). • <float1> - Stoichiometric coefficient, s. • <float2> - Product of interfacial area per unit volume by exchange current density, ai0, in units of A/cm3. • <float3> - Reaction order, β. • <float4> - Reference species concentration, cref, in units of moles/cm3. • <float5> - Anodic transfer coefficient, αa. • <float6> - Cathodic transfer coefficient, αc. • <float7> - Temperature, T, in unit of K. • <float8> - Open-circuit potential, U0, in unit of V. • <float9> - Number of electrons involved in the reaction, n. • <float10> - Electro-osmotic drag coefficient, nd.
EPOXY <species> <floatlist>	The **EPOXY** model adds a reaction source term for a condensation polymerization reaction based on an extent of reaction variable. Six model parameters make up the <float_list> for the **EPOXY** species source model, as follows:

Examples

Sample card for the **CONSTANT** model:

```
Species Source = CONSTANT 0 2.
```

Sample card for the **BUTLER_VOLMER** model:

```
Species Source = BUTLER_VOLMER 1 -1. .02 1. 4.e-5 1. 1. 353. 1.18 4.
```

Sample card for the **ELECTROOSMOTIC** model:

```
Species Source = ELECTROOSMOTIC 2 1 1. .02 1. 4.e-5 1. 1. 353. 1.18 4.0 1.4
```

Technical Discussion

A discussion of units for species flux terms can be found under **FAQs** on the *Diffusivity* card.

The **CONSTANT** option offers the simplest way for prescribing a constant homogeneous rate of species generation or consumption involving in a speciestransport process.

In the **BUTLER_VOLMER** model, the current source or sink due to a homogeneuous electrochemical reaction involving a single species (e.g., the hydrogen oxidation and oxygen reduction reactions in a hydrogen-feuled polymer-electrolyte-membrane fuel cell) is computed using the Butler-Volmer kinetic model as described below in the Theory section.

The **ELECTRODE_KINETIC** model computes the molar rate of electrolyte-species generation or consumption in electrochemical processes involving concentrated electrolyte solutions and multiple species as in thermal batteries. The molar rate of electrolyte-species consumption is evaluated using Butler-Volmer kinetics along with Faraday's law. Further details can be found in the reference listed below in the References sub-section (Chen et al. 2000).

The **ELECTROOSMOTIC** model computes the water-species flux due to the electroosmotic drag of protons (H+), which is proportional to the average current density with the proportionality constant being the electro-osmotic drag coefficient, nd.

The **EPOXY** model adds a reaction source term for a condensation polymerization reaction based on an extent of reaction variable. The extent of reaction is tracked as a convection equation with a reaction source term. The form of the EPOXY species source term is

$$R_\alpha = (k_1 + k_2\alpha^m)(1-\alpha)^n$$

where α is the extent of reaction, the rate constants, k1 and k2, can depend on temperature in the Arrhenius manner, and m and n are exponents.

where R is the gas constant in the appropriate units, Ai is the prefactor, and Ei is the activation energy for reaction. Six parameters are required to define the model: A1 and A2 (prefactors), E1 and E2 (activation energies), and m and n (exponents), with R being the universal gas constant.

The **EPOXY_DEA** model was created specifically for diethanolamine-epoxy curing reaction. While the expression for the source term is identical to the **EPOXY** model (with n=1.6),

the reaction kinetics differs, having three reaction regimes for exponent m and rate constant k2. For T< 65 C, m = 2 and

for 65 C < T< 90C, m = 74*k2 and

and for T > 90C, m = k2 = 0. Rate constant k1 is fixed for all these regimes and is determined from the prefactor A1 and activation energy E1.

The **FOAM** model computes the mixture volume change rate as:

$$k_1 = A_1 e^{\frac{E_1}{RT}}$$

$$R_\alpha = (k_1 + k_2 \alpha^m)(1-\alpha)^{1.6}$$

$$k_2 = A_2 e^{\frac{E_2}{RT}}$$

$$k_2 = A_2\left(\frac{90 - T}{T^6}\right) \quad \text{where T in C;}$$

$$\frac{\partial \rho}{\partial t} = \rho_{mix} \sum_{i=0}^{N} (V_i - V_{N+1}) r_i$$

where ρmix is the mixture density as defined in the REACTIVE_FOAM density model (which is required for this model) and Vi is the specific volume of component i.

The **USER** option indicates that a user-defined model has been introduced into the usr_species_source routine in the user_mp.c file. The <float_list> is of arbitrary length subject to the user's requirements to parameterize the model.

Theory

The rate of species generation or consumption in electrochemical processes involving a single species such as polymer-electrolyte-membrane fuel cells can be computed using the Butler-Volmer kinetic model and the Faraday's law (cf. Newman 1991, Chen et al. 2000, Chen and Hickner 2006):

$$r = \frac{sai_0}{nF}\left(\frac{c}{c_{ref}}\right)^{\beta}\left[e^{\frac{\alpha_a F(\Phi_1 - \Phi_2 - U_0)}{RT}} - e^{\frac{-\alpha_c F(\Phi_1 - \Phi_2 - U_0)}{RT}}\right]$$

where r is the homogeneous species source or sink in units of moles/cm3-s; s is the stoichiometric coefficient with a sign comvention such that r represents a source when s > 0 and sink when s < 0; n is the number of electrons involved in the electrochemical reaction; ai0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/cm3; c and cref are, respectively, species and reference molar concentrations in units of moles/cm3; β is reaction order; αa and αc are, respetively, the anodic and cathodic transfer coefficients; F is the Faraday's constant (≡ 96487 C/mole) and R is the universal gasl constant (≡ 8.314 J/mole-K); and are, respectively, the electrode and electrolyte potentials in unit of V; U0 is the open-circuit potential in unit of V; and T is temperature in unit of K.

References

for EPOXY_DEA Model GTM-011.0: Validation of 828/DEA/GMB Encapsulant using GOMA, August 20, 1999, A. C. Sun

for BUTLER_VOLMER and ELECTRODE_KINETIC Models:

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, Englewood Cliff, NJ (1991).

K. S. Chen, G. H. Evans, R. S. Larson, D. R. Noble, and W. G. Houf, "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", Sandia Report SAND2000-0207 (2000).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

Current Source

```
Current Source = {model_name} <optional integer> <float_list> [E/M]
```

Description / Usage

This card is used to specify the model for the source term on the voltage potential equation. Values for the permissible {model_names} and the associated <optional integer> and <floatlist> parameters are given below.

{model_name}	Name of the model for the source term on the voltage equation having permissible values • **CONSTANT** • **USER** • **BUTLER_VOLMER** • **ELECTRODE_KINETICS** • **FICKIAN_CHARGED** • **NET_CHARGE** • **STEFAN_MAXWELL_CHARGED**
<optional integer>	This is required for the **BUTLER_VOLMER** model only.
<float_list>	One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. Note that not all models have a <float_list>.

Source-term model choices and their parameters are discussed below. WARNING: make sure the equation term multipliers for the source terms are set to unity (see the Equation Cards segment in the previous chapter).

CONSTANT <float1>	For the **CONSTANT** current source term, there is a single input parameter corresponding to the current density. • <float1> - Current density [E/M]
USER <float_list>	For a user-defined model, the set of parameters specified in the <floatlist> are defined in file user_mp.c in the function usr_current_source.
BUTLER_VOLMER <integer> <float1> <float2> <float3> <float4> <float5> <float6> <float7> <float8>	This is the homogeneous current source or sink term (in units of amphere per unit volume, e.g. A/cm3) as described by the Butler-Volmer kinetic model (see the Theory section below).One integer and 8 flotas are required: • <integer> - Index of the species involved in the electrochemical reaction (here, we assume that only a single species is involved). • <float1> - Stoichiometric coefficient, s. • <float2> - Product of interfacial area per unit volume by exchange current density, ai0, in units of A/cm3. • <float3> - Reaction order, β. • <float4> - Reference species concentration, cref, in units of moles/cm3. • <float5> - Anodic transfer coefficient, αa. • <float6> - Cathodic transfer coefficient, αc. • <float7> - Temperature, T, in unit of K. • <float8> - Open-circuit potential, U0, in unit of V.
ELECTRODE_KINETICS	This is a toggle, turning the model on; no parameters are required.
FICKIAN_CHARGED	This is a toggle, turning the model on; no parameters are required.
NET_CHARGE	This is a toggle, turning the model on; no parameters are required.
STEFAN_MAXWELL_CHARGED	This is a toggle, turning the model on; no parameters are required.

Examples

Sample card for the **CONSTANT** model:

```
Current Source = CONSTANT 0.50
```

Sample card for the **BUTLER_VOLMER** model:

```
Current Source = BUTLER_VOLMER 0 1. 1000. 0.5 4.e-5 1. 1. 353. 0.
```

Technical Discussion

The **CONSTANT** and **USER** models are those standardly available in *Goma*.

In the **BUTLER_VOLMER** model the current source or sink due to a homogeneuous electrochemical reaction involving a single species (e.g., the hydrogen oxidation and oxygen reduction reactions in a hydrogen-feuled polymer-electrolyte-membrane fuel cell) is computed using the Butler-Volmer kinetic model as described below in the Theory section.

In the **FICKIAN_CHARGED** model, current source or sink for electrochemical processes involving dilute electrolyte solution and multiple species as in LIGA electrodeposition is computed.

The **NET_CHARGE** model is used to compute the net charge or current source in a region where the concentrations of positively and negatively charged species differ as in the space layer of a atmospheric copper sulfidation process, in which the copper hole and vacancy concentrations differ such that charge separation occur (see the reference listed below in the Reference sub-section, Chen 2004, for further details).

In the **STEFAN_MAXWELL_CHARGED** and **ELECTRODE_KINETICS** models, current sources or sinks for electrochemical processes involving concentrated electrolyte solutions and multiple species as in thermal batteries are computed.

Further details of these models can be found in the SAND Reports and proceeding paper referenced below in the Theory sub-section.

Theory

BUTLER_VOLMER model: for the Butler-Volmer kinetic model with the exchange current density being dependent on a single species is given by (cf. Newman 1991, Chen et al. 2000, Chen and Hickner 2006):

$$j = sai_0\left(\frac{c}{c_{ref}}\right)^{\beta}\left[e^{\frac{\alpha_a F(\Phi_1-\Phi_2-U_0)}{RT}} - e^{\frac{-\alpha_c F(\Phi_1-\Phi_2-U_0)}{RT}}\right]$$

where j is the homogeneous current source or sink in units of A/cm3; s is the stoichiometric coefficient with a sign convention such that j represents a source when s > 0 and sink when s < 0; ai0 denotes the product of interfacial area per unit volume by exchange current density, which has units of A/cm3; c and cref are, respectively, species and reference molar concentrations in units of moles/cm3; β is reaction order; αa and αc are, respetively, the anodic and cathodic transfer coefficients; F is the Faraday's constant (≡ 96487 C/mole) and R is the universal gasl constant (≡ 8.314 J/mole-K); Φ1 and Φ2 are, respectively, the electrode and electrolyte potentials in unit of V; U0 is the open-circuit potential in unit of V; and T is temperature in unit of K.

NET_CHARGE model: The net charge or current source in a region with charge separation (e.g., in a space charge layer in which hole and vacancy concentrations differ as in the atmospheric copper sulfidation corrosion process) is given by

where j is the net charge or current source in units of A/cm3; zi is the charge number and ci is the molar concentration in units of moles/cm3, respectively, of species i; F is the Faraday's constant (≡ 96487 C/mole); and n is the number of charge species present.

$$j = F \sum_{i=1}^{n} z_i c_i$$

References

J. Newman, Electrochemical Systems, 2nd Edition, Prentice-Hall, NJ (1991).

K. S. Chen, G. H. Evans, R. S. Larson, D. R. Noble, and W. G. Houf, "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", Sandia Report SAND2000-0207 (2000).

K. S. Chen and G. H. Evans, "Multi-dimensional Multi-species Modeling of Transient Electrodeposition in LIGA Microfabrication", Sandia Report SAND2004-2864 (2004).

K. S. Chen, "Multi-dimensional Modeling of Atmospheric Copper-Sulfidation Corrosion on non-Planar Substrates", Sandia Report SAND2004-5878 (2004).

K. S. Chen and M. A. Hickner, "Modeling PEM fuel cell performance using the finiteelement method and a fully-coupled implicit solution scheme via Newton's technique", in ASME Proceedings of FUELCELL2006-97032 (2006).

Moment Source

```
Moment Source = {model_name} <float_list> [varies]
```

Description / Usage

This required card is used to specify the model for the source term on the energy equation. Definitions of the input parameters are as follows:

{model_name} Name of the model for the source term on the energy equation. The permissible values are * FOAM_PMDI_10 * CONSTANT_GROWTH * FOAM_PBE

<float_list> One or more floating point numbers (<float1> through <floatn>) whose values are determined by the selection for {model_name}. Note that not all models have a <float_list>

Source-term model choices and their parameters are discussed below. WARNING: make sure the equation term multipliers for the source terms are set to unity (see the Equation Cards segment in the previous chapter).

FOAM_PMDI_10 Foam PMDI_10 model from Ortiz et al. * <float1> Growth rate coefficient, multiplicative with Moment Growth Rate Kernel * <float2> Coalescence kernel (Unused, set to 1.)

CONSTANT_GROWTH Constant growth rate * <float1> Growth rate coefficient

Examples

The following is a sample input card:

```
Moment Source = FOAM_PMDI_10 1. 1.
```

Technical Discussion

References

Ortiz, Weston, et al. "Population balance modeling of polyurethane foam formation with pressure-dependent growth kernel." AIChE Journal (2021): e17529.

Initialize

```
Initialize = {char_string} <integer> <float> [varies]
```

Description / Usage

This optional card provides a mechanism to set one of the field variables to a constant value within the current material block. Definitions of the input parameters are as follows:

<char_string>	Permissible values for this input string are any variable names identified in source file rf_fem_const.h beginning at the section labeled Variable Names of unknowns, though they should be active in this material block. Examples include, but are not limited to, the following: **VELOCITY1, VELOCITY2, VELOCITY3 (V123),MESH_DISPLACEMENT (MD123), SOLID_DISPLACEMENT (SD123), MASS_FRACTION, TEMPERATURE, PRESSURE,VOLTAGE, FILL, LS, POLYMER_STRESS (6 components, 8 modes), VELOCITY_GRADIENT (9 components), SHEAR_RATE, VOLF_PHASE (6 phases), POR_LIQ_PRES, POR_GAS_PRES, POR_POROSITY, POR_SATURATION, POR_LAST, LAGR_MULT (LM123), SURF_CHARGE, EXT_VELOCITY, EFIELD(123), SHELL (4 variables), SPECIES (7 variables).** *Note: for a comprehensive list of initializable variables, consult Volume 1 "Initialize" card.*
<integer>	Species number to be initialized if the value of {char_string} is one of the **SPECIES** variables (see Technical Discussion); otherwise, set <integer> to zero.
<float>	Value to which the variable should be initialized.

Multiple applications of this card are valid; Goma automatically counts the number of Initialize cards.

Examples

Following is a sample card:

```
INITIALIZE = POLYMER_STRESS11 0 1.25E4
```

Technical Discussion

This card provides the means to set initial values for any of the field variables in the element block for a particular material. Since the setting of variables initialized on this card takes place after reading the initial guess (see function init_vec in file rf_util.c), it can be used to override the value in the initial guess file.

In order to set a field to a specific value over the entire problem domain, a similar *Initialize* capability is provided as a global variable in the *General Specifications* section of the *Goma* input file. Please check in the Problem Description section of this manual.

Note, the **SPECIES_UNK** variables are **NOT** used to initialize any of the species variables. Rather, the special definition called **MASS_FRACTION** is the variables used in Goma input or mat files for this input record. Multiple species are initialized by combining **MASS_FRACTION** with the second parameter (<integer>) on this card. These cards are particularly handy for mass transfer problems, where the initial conditions need to specify different concentrations of the same species in different materials.

Note: for a comprehensive list of initializable variables, consult Volume 1 "Initialize" card.

1.5.8 Shell Equation Properties and Models

In this section we list all "material-region" specific models and properties associated with GOMA's extensive shell equation capability. Currently we have specialized shell equations for Reynolds lubrication flow (lubp), open Reynolds film flow (shell_film_H), energy (shell_energy, convection and diffusion, coupled with lubrication), thin porous media (closed cell and open cell), melting and phase change and more. While many of these cards are actual material properties, most are geometry and kinematic related. The most appropriate place for these cards are region/ material files because they are actually boundary conditions and related parameters which arise from the reduction of order (integration through the thin film). For more information, please see the shell-equation tutorial (GT-036).

Upper Height Function Constants

```
Upper Height Function Constants = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the upper-height function for the confined channel lubrication capability, or the lub_p equation. This function specifies the height of the channel versus distance and time. Currently three models for {model_name} are permissible:

CONSTANT_SPEED	This model invokes a squeeze/separation velocity uniformly across the entire material region, viz. the two walls are brought together/apart at a constant rate. This option requires two floating point values • <float1> the separation velocity (rate) in units of length/time • <float2> the initial wall separation in units of length • <float3> An OPTIONAL parameter which scales the addition of an external field called "HEIGHT" which is read in using the External Field or External Pixel Field capabilities. If this field is present, the value of it is added to the height calculated with this model.
ROLL_ON	This model invokes a squeeze/separation velocity in a hinging-motion along one boundary. The model is best explained with the figure in the technical discussion section. The equation for the gap h as a function of time and the input parameters (floats) is as follows: • <float1> is x0 in units of length • <float2> is hlow in units of length • <float3> is h Δ, in units of length • <float4> is the verticle separation velocity (if negative then squeeze velocity) in units of length/time • <float5> is the length of the plate, L.

$$h(t,x) = (v_{sq}t + h_\Delta)\left(\frac{x - x_0}{L}\right) + h_{low}$$

ROLL	This model is used for a roll coating geometry. This option requires 8 floats: • <float1> x-coordinate of origin, L. • <float2> y-coordinate of orgin, L. • <float3> z-coordinate of origin, L. • <float4> Direction angle 1 of rotation axis • <float5> Direction angle 2of rotation axis • <float6> Direction angle 3of rotation axis • <float7> rotation speed L/t.
FLAT_GRAD_FLAT	This model used two arctan functions to mimic a flat region, then a region of constant slope, then another flat region. The transitions between the two regions are curved by the arctan function. This currently on works for changes in the x direction. This option requires five floating point values • <float1> x location of the first transition (flat to grad) • <float2> height of the first flat region • <float3> x location of the second transition (grad to flat) • <float4> height of the second flat region • <float5> parameter controlling the curvature of the transitions
POLY_TIME	This time applies a time-dependent lubrication height in the form of a polynomial. It can take as many arguments as GOMA can handle, and the resulting height function is • <floati> value of Ci

$$h(t) = \sum_{i}^{N} C_i t^i$$

JOURNAL	This model simulates a journal bearing. It is intended to be run on a cylindrical shell mesh aligned along the z axis and centered at (0,0). It could be extended to be more flexible, but this is all it is currently capable of. The height is defined by h(θ) = C(1+ε cos(0)) Where C is the mean lubrication height and is the eccentricity of the two cylinders, with the smallest gap in the –y direction. • <float1> C • <float2> ε
EXTERNAL_FIELD	Not recognized. Oddly, this model is invoked with the extra optional float on the CONSTANT_SPEED option.

```
External Field = HEIGHT Q1 name.exoII (see this card)
```

Examples

Following is a sample card:

```
Upper Height Function Constants = CONSTANT_SPEED {v_sq = -0.001} {h_i=0.001}
```

This results in an upper wall speed of 0.001 in a direction which reduces the gap, which is initial 0.001.

Technical Discussion

The material function model ROLL_ON prescribes the squeezing/separation motion of two non-parallel flate plates about a hinge point, as shown in the figure below.

Lower Height Function Constants

```
Lower Height Function Constants = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the lower-height function for the confined channel lubrication capability, or the lub_p equation. This function specifies the height of the channel versus distance and time. Currently three models for {model_name} are permissible:

CONSTANT_SPEED	This model invokes a squeeze/separation velocity uniformly across the entire material region, viz. the two walls are brought together/apart at a constant rate. This option requires two floating point values • <float1> the separation velocity (rate) in units of length/time • <float2> the initial wall separation in units of length • <float3> An OPTIONAL parameter which scales the addition of an external field called "HEIGHT" which is read in using the External Field or External Pixel Field capabilities. If this field is present, the value of it is added to the height calculated with this model.
ROLL_ON	This model invokes a squeeze/separation velocity in a hinging-motion along one boundary. The model is best explained with the figure in the technical discussion section. The equation for the gap h as a function of time and the input parameters (floats) is as follows: • <float1> is x0 in units of length • <float2> is hlow in units of length • <float3> is h Δ, in units of length • <float4> is the verticle separation velocity (if negative then squeeze velocity) in units of length/time • <float5> is the length of the plate, L.

$$h(t,x) = (v_{sq}t + h_\Delta)\left(\frac{x - x_0}{L}\right) + h_{low}$$

ROLL	This model is used for a roll coating geometry. This option requires 8 floats: • <float1> x-coordinate of origin, L. • <float2> y-coordinate of orgin, L. • <float3> z-coordinate of origin, L. • <float4> Direction angle 1 of rotation axis • <float5> Direction angle 2 of rotation axis • <float6> Direction angle 3 of rotation axis • <float7> rotation speed L/t.
TABLE <integer1> <character_string1> {LINEAR | BILINEAR} [integer2] [FILE = filenm]	Please see discussion at the beginning of the material properties Chapter 5 for input description and options. Most likely *character_string1* will be **LOWER_DISTANCE** This option is good for inputing table geometry versus distance. Specifically, an arbitrary lower height function model is input as a function of the x-direction coordinate of the Lower Velocity Function model. This option in turn requires the use of SLIDER_POLY_TIME lower velocity function model. See example below.

Examples

Following is a sample card:

```
Lower Height Function Constants = CONSTANT_SPEED {v_sq = -0.001} {h_i=0.001}
```

This results in an lower wall speed of 0.001 in a direction which reduces the gap, which is initial 0.001.

In another example:

Lower Height Function Constants = TABLE 2 LOWER_DISTANCE 0 LINEAR FILE=shell.dat

where shell.dat is a table with 2 columns, the first the position, the second the height.

Technical Discussion

The material function model ROLL_ON prescribes the squeezing/separation motion of two non-parallel flate plates about a hinge point, as shown in the figure below.

Upper Velocity Function Constants

```
Upper Velocity Function Constants = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the upper-wall velocity function for the confined channel lubrication capability, or the lub_p equation. This function specifies the velocity of the upper channel wall as a function of time. Currently two models for {model_name} are permissible:

CONSTANT	This model invokes a squeeze/separation velocity uniformly across the entire material region, viz. the two walls are brought together/apart at a constant rate. This option requires two floating point values • <float1> is the velocity component in the x-direction. L/t. • <float2> is the velocity component in the y-direction. L/t • <float3> is the velocity component in the z-direction. L/t (NOTE: this is usually taken as zero as it is set in the Upper Wall Height Function model)
ROLL	This model invokes a wall velocity which corresponds to a rolling-motion. This model takes nine constants ???? : • <float1> Roll radius, L. • <float2> x-coordinate of axis origin, L. • <float3> y-coordinate of axis orgin, L. • <float4> z-coordinate of axis origin, L. • <float5> Direction angle 1 of rotation axis • <float6> Direction angle 2of rotation axis • <float7> Direction angle 3of rotation axis • <float8> Squeeze rate • <float9> rotation rate
TANGENTIAL_ROTATE	his model allows a velocity that is always tangential to a shell surface, not necessarily aligned along the coordinate directions. It requires three specifications. First, a vector (v) that is always non-colinear to the normal vector of the shell must be specified. This is used to make unique tangent vectors. The last two specifications are the two tangential components to the velocity. The first velocity is applied in the direction of t1 = v×n. The second velocity is then applied in the t = t ×n direction. • <float1> vx • <float2> vy • <float3> vz • <float4> velocity in the t1 direction • <float5> velocity in the t2 direction
CIRCLE_MELT	Model which allows a converging or diverging height that is like a circle. Also works for melting. • <float1> - x-location of the circle center (circle is in x-y plane) • <float2> - radius of circle • <float3>- minimum height of circle

Examples

Following is a sample card:

```
Upper Velocity Function Constants = CONSTANT {v_x= -0.001} {vy=0.00} {vz=0}
```

This card results in an upper wall speed of -0.001 in the x-direction which is tangential to the substrate, thus generating a Couette component to the flow field.

Technical Discussion

For non-curved shell meshes, most of the time they are oriented with the x-, y-, or zplane. This card is aimed at applying a tangential motion to that plane, and so one of the three components is usually zero.

Lower Velocity Function Constants

```
Lower Velocity Function Constants = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the Lower-wall velocity function for the confined channel lubrication capability, or the lub_p equation. This function specifies the velocity of the Lower channel wall as a function of time. Currently two models for {model_name} are permissible:

CONSTANT	This model invokes a squeeze/separation velocity uniformly across the entire material region, viz. the two walls are brought together/apart at a constant rate. This option requires two floating point values • <float1> is the velocity component in the x-direction. L/t • <float2> is the velocity component in the y-direction. L/t • <float3> is the velocity component in the z-direction. L/t (NOTE: this is usually taken as zero as it is set in the Lower Wall Height Function model)
SLIDER_POLY_TIME	This model implements a spatially-uniform velocity in the x-direction that is specified as a polynomial in time. The value of time may be scaled by a given scaling factor and the polynomial may have an unlimited number of terms. • <float1> is the time scaling factor • <float2-N> are the coefficients in front of the t^(i-2) term

$$v_x = \sum_{i=2}^{i=N_{float}} \text{float}_i t_s^{\,i-2}$$

$$t_s = \text{float}_1 \times t$$

ROLL	This model invokes a wall velocity which corresponds to a rolling-motion. This model takes nine constants ???? : • <float1> Roll radius, L. • <float2> x-coordinate of axis origin, L. • <float3> y-coordinate of axis orgin, L. • <float4> z-coordinate of axis origin, L. • <float5> Direction angle 1 of rotation axis • <float6> Direction angle 2of rotation axis • <float7> Direction angle 3of rotation axis • <float8> Squeeze rate. • <float9> rotation rate
TANGENTIAL_ROTATE	This model allows a unique specification of tangential motion in a lubrication shell element. Previous implementations allowed specification only in terms of coordinate direction, but this option can be used to rotate a cylinder. Five floats are required • <float1> x-comnponent of a vector tangential to the shell. This vector must never be normal to the shell. It is then projected onto the shell. • <float2> y-comnponent of a vector tangential to the shell. • <float3> z-comnponent of a vector tangential to the shell. • <float4> U1, or scalar speed of wall velocity in a direction determined by the cross product ot the tangent vector and the normal vector to the shell. (L/t) • <float5> U2 scalar speed component in direction normal to U1. (L/t)

Examples

Following is a sample card:

```
Lower Velocity Function Constants = CONSTANT {v_x= -0.001} {vy=0.00} {vz=0}
```

This card results in an Lower wall speed of -0.001 in the x-direction which is tangential to the substrate, thus generating a Couette component to the flow field.

Technical Discussion

For non-curved shell meshes, most of the time they are oriented with the x-, y-, or zplane. This card is aimed at applying a tangential motion to that plane, and so one of the three components is usually zero.

Upper Contact Angle

```
Upper Contact Angle = {model_name} <floatlist>
```

Description / Usage

This card sets contact angle of the liquid phase on the upper-wall for the two-phase capability in the lub_p equation (viz. when using the level-set equation to model the motion of a meniscus in a thin gap, where the in-plan curvature is neglected. Currently one model {model_name} is permissible:

CONSTANT	This model is used to set a constant contact able of the the free surface at the upper wall. Contact angle of less than 90 degrees is considered as nonwetting with respect to the heavier level-set phase. Only one floating point value is required. • <float1> is the contact angle in degrees.

Examples

Following is a sample card:

```
Upper Contact Angle = CONSTANT 180.
```

This card results in an upper wall contact able to 180 degrees, which is perfectly wetting. If the lower wall is given the same angle, then the capillary pressure jump will go as 2/h, where h is the gap.

Lower Contact Angle

```
Lower Contact Angle = {model_name} <floatlist>
```

Description / Usage

This card sets contact angle of the liquid phase on the lower-wall for the two-phase capability in the lub_p equation (viz. when using the level-set equation to model the motion of a meniscus in a thin gap, where the in-plan curvature is neglected. Currently one model {model_name} is permissible:

CONSTANT	This model is used to set a constant contact able of the the free surface at the lower wall. Contact angle of less than 90 degrees is considered as nonwetting with respect to the heavier level-set phase. Only one floating point value is required. • <float1> is the contact angle in degrees.

Examples

Following is a sample card:

```
Lower Contact Angle = CONSTANT 180.
```

This card results in an lower wall contact able to 180 degrees, which is perfectly wetting. If the lower wall is given the same angle, then the capillary pressure jump will go as 2/h, where h is the gap.

Lubrication Fluid Source

```
Lubrication Fluid Source = {model_name} <floatlist>
```

Description / Usage

This card sets a fluid mass source term in the lub_p equation. Can be used to specify inflow mass fluxes over the entire portion of the lubrication gap in which the lub_p equation is active (over the shell material). This flux might be the result of an injection of fluid, or even melting. Currently two models {model_name} are permissible:

CONSTANT	This model is used to set a constant fluid source in units of velocity. Only one floating point value is required. • <float1> is the velocity of the fluid source.
MELT	This model is used to set fluid source in units of Velocity which results from an analytical model of lubricated melt bearing flow due to Stiffler (1959). Three floating point values are required. • <float1> is load on the slider in units of pressure • <float2> is the Stiffler delta factor. Unitless but depends on the aspect ratio. • <float3> is the length of the slider in the direction of the motion.

Examples

Following is a sample card:

```
Lubrication Fluid Source = CONSTANT 180.
```

Lubrication Momentum Source

```
Lubrication Momentum Source = {model_name} <floatlist>
```

Description / Usage

This card sets a fluid "body force per unit volume" source term in the lub_p equation. This capability can be used to specify a force field over the entire shell area (over the shell material).. Currently two models {model_name} are permissible:

CONSTANT	THIS MODEL NOT IMPLEMENTED AS OF 11/11/2010. This model is used to set a constant fluid momentum source in units of force per unit volume. Only one floating point value is required. • <float1> is the fluid momentum source in F/L^3.
JXB	This model is used to set fluid momentum source in units of force per unit volume which comes from externally supplied current density J field and magnetic B fields. These fields are suppled with the external field capability in Goma in a component wise fashion. Please consult the technical discussion below. • <float1> is scale factor which may be used for nondimensionalization. Typically this is set to 1.0.

Examples

Following is a sample card:

```
Lubrication Momentum Source = JXB 1.
```

Technical Discussion

The two vector fields J, the current flux, and B, the magnetic induction, must be supplied to Goma in order to activate this option. At present, these fields must be supplied with the External Field cards, which provide the specific names of nodal variable fields in the EXODUS II files from which the fields are read. The three components of the J field must be called JX_REAL, JY_REAL, and JZ_REAL. Likewise the B field components must be called BX_REAL, BY_REAL, and BZ_REAL. These names are the default names coming from the electromagnetics code like Alegra. Because of the different coordinate convention when using cylindrical components, the fields have been made compatible with those arising from TORO II. It is the interface with TORO that also makes the Lorentz scaling (lsf) necessary so that the fixed set of units in TORO (MKS) can be adjusted to the user-selected units in Goma.

References

No References.

Turbulent Lubrication Mode

```
Turbulent Lubrication Model = {model_name}
```

Description / Usage

This card activates a turbulent model for viscosity in the lub_p equation. Currently one model {model_name} is permissible:

PRANDTL_MIXING	The model used to determine the pre-multiplier on the molecular viscosity in the Reynolds lubrication equation. For confined, laminar flow, this multiplier is 12. For turbulent flow it is taken as K(Re), where Re is the local Reynolds number. Specifically, invoking a analytical approximation for K from Hirs (1973), we set k0 according to the: Reynolds number Re= :ρh U μ For 0 < Re < 2000 K0=12 (Laminar case), Else 2000 < Re < 100000 K0 = 0.05Re 3/4 Here the wall velocity is used to compute The Reynolds number, as this turbulence model is specific to turbulent Couette flow.

Examples

Following is a sample card:

```
Turbulent Lubrication Model = PRANDTL_MIXING
```

Technical Discussion

Several other models can be implemented in this instance. We chose this simple model which derives from Prandtl mixing length theory.

References

G.G. Hirs, “Bulk flow theory for turbulence in lubricant films”, Trans. ASME, ser. F, 95, pp 137-146, 1973.

Shell Energy Source QCONV

```
Shell Energy Source QCONV = {model_name} <float_list>
```

Description / Usage

This card activates a heat source (or sink, as it were) in the shell_energy equation. The functional form of this source/sink is a lumped heat-transfer coefficient model, hence the QCONV in its name (see BC = QCONV card in main user manual). Currently two models {model_name} are permissible:

CONSTANT	This model invokes a simple constant heat-transfer coefficient and reference temperature, viz. q = H(T − Tref). Commensurately there are two floats required: <float1> - Heat transfer coefficient in units of Energy/time/L2/deg T E.g. W/m2-K in MKS units. <float2> - Reference temperature.
MELT_TURB	This model also invokes a lumped parameter model, but the heat-transfer coefficient depends on the flow strength (Reynolds number), viz. q = H(T − Tref). Three floats are required: <float1> - Thermal conductivity in units of Energy time/L/deg (e.g. W/m/k). <float2> - Reference temperature. <float3> - Latent heat of melting (Energy/M, e.g. J/Kg). This quantity is required due to the cross use of this in the shell_deltah equation (viz. EQ = shell_deltah).

Examples

Following is a sample card:

```
Shell Energy Source QCONV = MELT_TURB {thermal_conductivity} {Tref} {latent_heat}
```

Technical Discussion

The MELT_TURB model warrants further discussion. The functional form of the heat transfer coefficient H is

$$H = 0.0735 \rho C_p c_f^{1/2} u_{wall} \left(\frac{\mu C_p}{K}\right)$$

.

Here cf is the coefficient of friction, which for now is taken as 8./Re.

Shell Energy Source Sliding Contact

```
Shell Energy Source Sliding_Contact = {model_name} <float_list>
```

Description / Usage

This card activates a heat source (or sink, as it were) in the shell_energy equation. The functional form of this source/sink is a sliding contact model derived in the frame-of-reference of the slider on a stationary surface, so that the surface is moving in the simulation. In this case, the conditions for the flux vary from the leading edge to the trailing edge of the slider as a thermal boundary layer builds up. Think of this as a hot slider moving over a cold stationary wall, so that the flux at the leading edge of the slider into the cold wall will be larger, due to a steeper thermal boundary layer. Clearly the contact time will play a role. Currently two models {model_name} are permissible:

LOCAL_CONTACT	This model invokes the following functional form: Commensurately there are seven floats required: • <float1> - length l of slider. • <float2> - Sink temperature of substrate. • <float3> - Thermal conductivity of substrate. • <float4> - Density of substrate • <float5> - Heat capacity of substrate. • <float6> - Delta L, or L1 – L2. This parameter sets the segment size (less than the total slider length) over which the heat flux is resolved. • <float7> - Leading edge coordinate of slider.

$$K \left.\frac{dT}{dz}\right|_0 = \frac{2K(T - T_0)}{\sqrt{\pi K \left(\frac{1}{u_{slider}}\right)}} \left[\frac{\sqrt{L2} - \sqrt{L1}}{L2 - L1}\right]$$

AVERAGE_CONTACT	This model invokes the following functional form: Commensurately there are seven floats required: • <float1> - length l of slider. • <float2> - Sink temperature of substrate. • <float3> - Thermal conductivity of substrate. • <float4> - Density of substrate • <float5> - Heat capacity of substrate.

$$K \left. \frac{dT}{dz} \right|_0 = \frac{2K(T - T_0)}{\sqrt{\pi\kappa\left(\frac{L}{u_{slider}}\right)}}$$

l:

Examples

Following is a sample card:

```
Shell Energy Source Sliding_Contact = LOCAL_CONTACT {L= 2.5} {t_r=20} {t_cond_cu_cgs}
↪{density_cu_cgs} {heat_capacity_cu_cgs} {delta_L = 0.1} {leading_edge_coordx = 2.5}
```

Technical Discussion

Technical Discussion This boundary condition was derived using the analytical solution for heat conduction into an infinite slab, as derived by Carslaw and Jaeger. The modification here is that the temperature source accommodates a motion relative to the substrate, which is what leads to the need to segment the slider into bins over which a local heat flux solution is derived.

NOTE: If this card is used and there is no upper-wall or lower wall sliding motion, and error is thrown.

Shell Energy Source Viscous Dissipation

```
Shell Energy Source Viscous Dissipation = {model_name} <float_list>
```

Description / Usage

This card activates a heat source (or sink, as it were) in the shell_energy equation resulting from viscous dissipation due to shear combined Couette and pressure-driven flow in the Reynolds lubrication equation (lubp equation).

LUBRICATION	This model invokes a viscous dissipation model simplified for the lubrication approximation. • <float1> - Scale factor for the term, typically taken as 1.0
LUBRICATION_FRICTION	This model invokes the same viscous dissipation source term as in the LUBRICATION model, but adds on an additional linear friction model of the form μ*Pload vs-lider. Commensurately there are seven floats required: • <float1> - coefficient of friction, μ. • <float2> - Applied external load Pload

Examples

Following is a sample card:

```
Shell Energy Source Viscous Dissipation= LUBRICATION_FRICTION {load=5e8} {coeff=0.9}
```

References

No References.

Shell Energy Source External

```
Shell Energy Source Viscous External = {model_name} <float_list>
```

Description / Usage

This card activates a heat source (or sink, as it were) in the shell_energy equation which corresponds to a user-supplied or constant value. Two models are available.

CONSTANT	This model invokes a constant heat source term (heat sink if negative) in units of energy per area per time. • <float1> - Value of heat source.
JOULE	This model invokes a constant energy source which is determined by an external current density field of the form Qjoule = hξ J · J. Here J is the current density, h is the gap, and is the electrical resistivity, or the inverse conductivity. Both h and ξ are determined from other models in the material file. J is brought in as an external field variable from another exodusII file (see discussion below). • <float1> - Scale factor, usually set to 1.0.
JOULE_LS	This model differs from JOULE only in that the electrical conductivity is pulled out and must be specified with a LEVEL_SET model. This model is not well tested (PRS 12/14/2012) • <float1> - Scale factor, usually set to 1.0.

Examples

Following is a sample card:

```
Shell Energy Source External = JOULE {scale=1.0}
```

Technical Discussion

To bring in an external field of the appropriate form, see the main Goma user's manual and refer to the External Field card. As an example, you might consider solving a simple electrostatic problem using the EQ = V (voltage) equation and output the magnitude of the current density vector. In Goma, this is done with the post processing

```
Electric Field Magnitude = yes
```

Card. This card outputs this J-magnitude as the exodusII variable EE. You then bring it in as follows in the input script:

```
External Field = EE Q1 current_dens_out.exoII
```

With the JOULE model, this field is used to compute the Joule heating term.

References

No References.

FSI Deformation Model

```
FSI Deformation Model = {model_name}
```

Description / Usage

This card specifies the type of interaction the lubrication shell elements will have with any surrounding continuum element friends. When not coupling the lubrication equations to a continuum element, this card should be set to the default value, FSI_SHELL_ONLY. All models are described below:

FSI_MESH_BOTH This model should be used when both the shell and neighboring continuum elements use deformable meshes and the user wishes to full couple these behaviors. This model is not currently implemented and should not be used.

FSI_MESH_CONTINUUM In this model, the neighboring continuum elements use mesh equations, but the lubrication shell does not. This model features a two-way coupling, where the lubrication pressure can deform the neighboring solid (through the appropriate boundary condition) and deformations to the mesh in turn affect the height of the lubrication gap. This is equivalent to the old "toggle = 1".

FSI_MESH_SHELL This model accounts for mesh equations present in the lubrication shell, but not in the adjoining continuum elements. This model is not currently implemented and should not be used.

FSI_SHELL_ONLY This model can be thought of as the default behavior, where there is no coupling between the lubrication shell elements and any neighboring continuum elements. This should also be used if only shells are present.

FSI_MESH_UNDEF This model is similar to FSI_MESH_CONTINUUM, but the normal vectors in the shell are calculated using the original undeformed configuration, rather than the current deformed state. Implementation of this model is currently in progress and needs to be fully tested.

FSI_MESH_ONEWAY This model is similar to FSI_MESH_CONTINUUM, but only utilizes a one way coupling. Deformations in the neighboring continuum element do not affect the lubrication height, but do affect the calculated normal vectors. This is equivalent to the old "toggle = 0".

Examples

Technical Discussion

References

No References.

Film Evaporation Model

```
Film Evaporation Model = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the evaporation rate for the film-flow equation capability, specifically the shell_filmp equation. This function specifies the rate of evaporation in the unit of length per time. Currently two models for {model_name} are permissible:

CONSTANT	This model specifies a constant evaporation rate. This option requires one floating point values • <float1> is the evaporation rate in the unit of length/time
CONC_POWER	This model specifies evaporation rate function of particles volume fraction and the input parameters (floats). This model is proposed by Schwartz et al (2001). The functional form is: • <float1> is the pure liquid evaporation rate in units of length per time • <float2> is exponent v and it should satisfy 0 <v <1 • <float3> is the maximum packing volume fraction 0max

$$\dot{E} = \dot{E}_0 \left[1 - \frac{\varphi}{\varphi_{\max}} \right]^{\nu}$$

Examples

Following is a sample card:

```
Film Evaporation Model = CONC_POWER 1.0e-3 0.5 0.64
```

This results in film evaporation with the pure liquid evaporation rate of 1.0e-3, exponent of 0.5, and maximum packing volume fraction of 0.64.

References

Leonard W. Schwartz, R. Valery Roy, Richard R. Eley, and Stanislaw Petrash, “Dewetting Patterns in a Drying Liquid Film”, Journal of Colloid and Interface Science 234, 363–374 (2001)

Disjoining Pressure Model

```
Disjoining Pressure Model = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the disjoining pressure model for the film-flow equation capability, specifically the shell_filmh equation. This function specifies the disjoining pressure in the unit of force per area. Currently four models for {model_name} are permissible:

CONSTANT	This model specifies a constant disjoining pressure. This option requires one floating point values • <float1> is the evaporation rate in the unit of length/time

ONE_TERM This model specifies disjoining pressure and the input parameters (floats). This model only employs the repulsion part of the van der Waals force. The functional form is:

- <float1> is the equilibrium liquid-solid contact angle θe
- <float2> is exponent n and it should satisfy n >1
- <float3> is the precursor film thickness h*

TWO_TERM This model specifies disjoining pressure and the input parameters (floats). Here, the model only employs both repulsion and attraction part of the van der Waals force. The functional form is:

where

- <float1> is the equilibrium liquid-solid contact angle 0e
- <float2> is exponent n corresponding to the repulsive part of the van der Waals force. It should satisfy n >1
- <float3> is exponent m corresponding to the attractive part of the van der Waals force. It should satisfy m > n since the attractive part acts in longer range than the repulsive one.
- <float4> is the precursor film thickness h
- <float5> is the parameter a describing relative importance of the attractive part to the repulsive part. Typically, a is chosen to be 0 <α <1 in order to achieve more numerical stability.

TWO_TERM_EXT_CA This model is identical with **TWO_TERM** except that it uses contact angle from an external field identifies as THETA.

$$\Pi = B\left[\frac{h}{h_*}\right]^n$$

where

$$B = \frac{\sigma\left(1-\cos\theta_e\right)\left(n-1\right)}{h_*}$$

$$\Pi = B\left\{\left[\frac{h}{h_*}\right]^n - \alpha\left[\frac{h}{h_*}\right]^m\right\}$$

$$B = \frac{\sigma(n-1)(m-1)(1-\cos\theta_e)}{\alpha(n-1)-(m-1)}$$

Examples

Following is a sample card:

```
Disjoining Pressure Model = TWO_TERM 120.3 2 1.0e-4 0.1
```

This results in disjoining pressure with contact angle of 120, repulsive exponent of 3, attractive repulsion of 2, precursor film thickness of 1.0e-4, and relative importance of attractive part of 0.1.

Technical Discussion

A thorough discussion of disjoining pressure can be found in Teletzke et al (1987). The premultiplying constant is related to contact angle and surface tension by balancing capillary and disjoining force where the wetting line meets the precursor film. See Schwartz (1998) for further detail.

References

Leonard W. Schwartz, R. Valery Roy, Richard R. Eley, and Stanislaw Petrash, “Dewetting Patterns in a Drying Liquid Film”, Journal of Colloid and Interface Science 234, 363–374 (2001)

Teletzke, G. F., Davis, H. T., and Scriven, L. E., “How liquids spread on solids”, Chem. Eng. Comm., 55, pp 41-81 (1987).

Diffusion Coefficient Model

```
Diffusion Coefficient Model = {model_name} <floatlist>
```

Description / Usage

This card takes the specification of the diffusion coefficient model for the conservation of particles inside film-flow capability, i.e. equation describing shell_partc. Currently two models for {model_name} are permissible:

CONSTANT	This model specifies a constant diffusion coefficient. This option requires one floating point values • <float1> is the diffusion coefficient
STOKES_EINSTEIN	This model specifies diffusion coefficient that depends on particles radius and the film viscosity. The functional form is: • <float1> is the Boltzmann constant kb where the magnitude depends on the units chosen by the user. • <float2> is temperature T in unit of Kelvin. • <float3> is the particles radii R.

$$D = \frac{k_B T}{6\pi\mu R}$$

Examples

Following is a sample card:

```
Diffusion Coefficient Model = STOKES_EINSTEIN 1.3807e-16 298 1.0e-6
```

This results in diffusion coefficient calculated with Stokes Einstein model with Bolztmann constant of 1.3807e-16 in CGS units, 298 K temperature, and 1.0e-6 cm radius particles.

Technical Discussion

Viscosity dependence of diffusion coefficient can be exploited to relate particles concentration (or volume fraction in this case) to diffusion coefficient by employing SUSPENSION viscosity model in the material file. See SUSPENSION viscosity model for further detail

Porous Shell Radius

```
Porous Shell Closed Radius = {model_name} <floatlist>
```

Description / Usage

This card specifies the radius of the pores used in porous_shell_closed and porous_shell_open equations. Currently two models for {model_name} are permissible:

CONSTANT	This model applies a constant pore radius for the entire model. It requires a single floating point value. • <float1> is the pore radius. L.
EXTERNAL_FIELD	This model reads in an array of values for the radius from an initial exodus file. This allows for spatial variations in the parameter value.

Examples

Following is a sample card:

```
Porous Shell Radius = CONSTANT 0.00001
```

References

No References.

Porous Shell Height

```
Porous Shell Height = {model_name} <floatlist>
```

Description / Usage

This card specifies height of the pores used in porous_shell_closed and porous_shell_open equations. Currently two models for {model_name} are permissible:

CONSTANT	This model applies a constant pore height for the entire model. It requires a single floating point value. • <float1> is the pore height. L.
EXTERNAL_FIELD	This model reads in an array of values for the height from an initial exodus file. This allows for spatial variations in the parameter value.

Examples

Following is a sample card:

```
Porous Shell Height = CONSTANT 0.00001
```

References

No References.

Porous Shell Closed Porosity

```
Porous Shell Closed Porosity = {model_name} <floatlist>
```

Description / Usage

This card specifies the porosity used in porous_shell_closed equations. Currently two models for {model_name} are permissible:

CONSTANT	This model applies a constant porosity for the entire model. It requires a single floating point value. • <float1> is the porosity
EXTERNAL_FIELD	FIELDThis model reads in an array of values for the porosity from an initial exodus file. This allows for spatial variations in the parameter value. • <float1> scale factor for scaling field value The ExodusII field variable name should be "SH_SAT_CL_POROSITY", viz.

```
External Field = SH_SAT_CLOSED_POROSITY Q1 name.exoII (see this card)
```

Examples

Following is a sample card:

```
Porous Shell Closed Porosity= CONSTANT 0.5
```

References

No References.

Porous Shell Closed Gas Pressure

```
Porous Shell Closed Gas Pressure = {model_name} <floatlist>
```

Description / Usage

This card specifies the gas pressure used in porous_shell_closed equations. Currently one model for {model_name} are permissible:

CONSTANT	This model applies a constant gas pressure for the entire model. It requires a single floating point value. • <float1> is the gas pressure

Examples

Following is a sample card:

```
Porous Shell Closed Gas Pressure = CONSTANT 0.5
```

References

No References.

Porous Shell Atmospheric Pressure

```
Porous Shell Atmospheric Pressure = {model_name} <floatlist>
```

Description / Usage

This card is used to set the atmospheric pressure level in a open-cell shell porous equation (for partially saturated flow). As of 11/27/2012 this card is NOT used.

Examples

No Examples.

References

No References.

Porous Shell Reference Pressure

```
Porous Shell Atmospheric Pressure = {model_name} <floatlist>
```

Description / Usage

This card is used to set the reference pressure level in a open-cell shell porous equation (for partially saturated flow). This pressure is used to shift the saturation-capillary pressure curve appropriately. As of 11/27/2012 this card is NOT used as all saturation curve information is handled in the main porous flow property input framework, even for shell formulations.

Examples

No Examples.

References

No References.

Porous Shell Cross Permeability

```
Porous Shell Cross Permeability = {model_name} <floatlist>
```

Description / Usage

This card is used to set the permeability in the thin direction of a shell porous region. The property is used for the porous_sat_open equation. The in-shell (in-plane for a flat shell) permeabilities are set on the Permeability card. Please consult the references for the equation form. The property can take on one of two models:

CONSTANT	This model applies a constant cross-region permeability. It requires a single floating point input: • <float1> is the cross region permeability
EXTERNAL_FIELD	This model is used to read a finite element mesh field representing the cross-term permeability. Please consult tutorials listed below for proper usage. This model requires one float: • <float1> scale factor for incoming exodusII field and desired level. The ExodusII field variable name should be "CROSS_PERM", viz.

Examples

No Examples.

References

S. A. Roberts and P. R. Schunk 2012. in preparation.

Randy Schunk 2011. GT-038 "Pixel-to-Mesh-Map Tool Tutorial for GOMA". Memo to distribution.

Porous Shell Gas Diffusivity

```
Porous Shell Gas Diffusivity = {model_name} <floatlist>
```

Description / Usage

This card is used to set the gas diffusivity for the trapped gas in the porous_sat_closed equation. Basically, the gas trapped in closed pores during the imbibition process is allowed to diffuse into the liquid, and this property is a part of that model gas inventory equation R_SHELL_SAT_GASN. Only one model is available for this property:

CONSTANT	This model applies a constant cross-region permeability. It requires a single floating point input: • <float1> is the cross region permeability
EXTERNAL_FIELD	This model applies a constant gas diffusivity. It requires a single floating point input: • <float1> is the gas diffusivity (L2/t)

Examples

Porous Shell Gas Diffusivity = CONSTANT 1.e-5

References

S. A. Roberts and P. R. Schunk 2012. in preparation.

Porous Shell Gas Temperature Constant

```
Porous Shell Gas Temperature Constant = {model_name} <floatlist>
```

Description / Usage

This card is used to set the temperature constant in Henry's law for the trapped gas in the porous_sat_closed equation. Basically, the gas trapped in closed pores during the imbibition process is allowed to diffuse into the liquid, and this property is a part of that model for the dissolution constant of gas in pressurized liquid. It is only needed if the equation R_SHELL_SAT_GASN is used. Only one model is available for this property:

CONSTANT	This model sets the Henry's law temperature constant. It requires a single floating point input: • <float1> is the gas temperature constant

Examples

Porous Shell Gas Temperature Constant= CONSTANT 1.e10

References

S. A. Roberts and P. R. Schunk 2012. in preparation.

Porous Shell Henrys Law Constant

```
Porous Shell Henrys Law Constant = {model_name} <floatlist>
```

Description / Usage

This card is used to set the partitionconstant in Henry's law for the trapped gas in the porous_sat_closed equation. Basically, the gas trapped in closed pores during the imbibition process is allowed to diffuse into the liquid, and this property is a part of that model for the dissolution constant of gas in pressurized liquid. It is only needed if the equation R_SHELL_SAT_GASN is used. Please consult the references for a detailed explanation. Only one model is available for this property:

CONSTANT	This model sets the Henry's law constant. It requires a single floating point input: • <float1> is the Henry's law constant

Examples

Porous Shell Gas Temperature Constant= CONSTANT 1.e10

References

S. A. Roberts and P. R. Schunk 2012. in preparation.

1.5.9 Moment Properties

Properties related to Population Balance Methods using Quadrature Method of Moments (QMOM)

Moment Weight Function

```
Moment Weight Function = {GALERKIN | SUPG } <float>
```

Description / Usage

This card specifies the weight function to be used on the weighted residual of the Moment equations. Definitions of the input parameters are as follows:

GALERKIN Name of the model for the weight functions for a full Galerkin formulation. This is the default when this card is absent. * <float> - set but unused for Galerkin

SUPG Name of the model for the weight functions for a streamwise upwinded Petrov-Galerkin formulation.

- <float> - the value of the weight function, a number between 0. and 1.; a value of 1. corresponds to a full **SUPG**

Examples

The following is a sample input card:

```
Moment Weight Function = GALERKIN 0.0

Moment Weight Function = SUPG 1.0
```

Technical Discussion

References

No References.

Moment Shock Function

```
Moment Shock Function = {NONE | YZBETA } <float>
```

Description / Usage

This card specifies the shock capturing function to be used on the Moment equations. Definitions of the input parameters are as follows:

NONE No shock capturing is applied. This is the default when this card is absent

YZBETA A YZBETA shock capturing term is applied.

- <float> - the value of the weight function, a number between 0. and 1.; a value of 1. corresponds to a full YZBETA shock capturing

Examples

The following is a sample input card:

```
Moment Weight Function = NONE (default)

Moment Weight Function = YZBETA 1.0
```

Technical Discussion

References

No References.

Moment Growth Kernel

```
Moment Growth Kernel = {CONSTANT | VISCOSITY_SCALED | VISCOSITY_PRESSURE_SCALED}
↪<float1> [float2]
```

Description / Usage

This card specifies the growth kernel to be used for the Moment equations. Currently only used for *PMDI_10* QMOM models. Definitions of the input parameters are as follows:

CONSTANT Constant growth rate * <float1> the prefactor

VISCOSITY_SCALED Growth rate scaled by viscosity $G = G_0\eta_0/\mu$, for use with *PMDI_10* models * <float1> - G_0

VISCOSITY_PRESSURE_SCALED Growth rate scaled by viscosity and pressure $G = G_0(\eta_0/\mu)/(p - p_ref)^2$, for use with *PMDI_10* models

- <float1> G_0
- <float2> p_ref

Examples

The following is a sample input card:

```
Moment Growth Kernel = CONSTANT 1e-6

Moment Growth Kernel = VISCOSITY_SCALE 1e-3

Moment Growth Kernel = VISCOSITY_PRESSURE_SCALED 1e-8
```

Technical Discussion

Note this is multiplicative with the growth rate in the *FOAM_PMDI_10* moment source model

References

Ortiz, Weston, et al. “Population balance modeling of polyurethane foam formation with pressure-dependent growth kernel.” AIChE Journal (2021): e17529.

Moment Coalescence Kernel

```
Moment Coalescence Kernel = {ADDITION | VISCOSITY_SCALED_ADDITION | VISCOSITY_
↪ADDITION_BUBBLE_RATIO} <float1>
```

Description / Usage

This card specifies the coalescence kernel to be used for the Moment equations. Currently only used for *PMDI_10* QMOM models. Definitions of the input parameters are as follows:

ADDITION $\beta(v,v') = \beta_0(v+v')$ * <float1> β_0

VISCOSITY_SCALED_ADDITION $\beta(v,v') = \beta_0(\eta_0/\mu)(v+v')$ * <float1> β_0

VISCOSITY_ADDITION_BUBBLE_RATIO $\beta(v,v') = \beta_0(\eta_0/\mu)(v+v')(v/(v'+\epsilon))$ * <float1> β_0

Examples

The following is a sample input card:

```
Moment Coalescence Kernel = ADDITION 1e-12

Moment Coalescence Kernel = VISCOSITY_SCALED_ADDITION 1e-3

Moment Coalescence Kernel = VISCOSITY_ADDITION_BUBBLE_RATIO 1e-8
```

Technical Discussion

Currently only for *FOAM_PMDI_10* moment source model

References

Ortiz, Weston, et al. "Population balance modeling of polyurethane foam formation with pressure-dependent growth kernel." AIChE Journal (2021): e17529.

1.6 References

[1] Alsoy, S. and J. L. Duda, 1999. "Modeling of Multicomponent Drying of Polymer Films." AIChE Journal, (45) 4, 896-905.

[2] Baaijens, F. P. T., 1994. "Application of Low-Order Discontinuous Galerkin Method to the Analysis of Viscoelastic Flows," J. Non-Newtonian Fluid Mech., 52, 37-57.

[3] Baaijens, F. P. T., 1998. "An Iterative Solver for the DEVSS/DG Method with Application to Smooth and Non-smooth Flows of the Upper Convected Maxwell Fluid," J. Non-Newtonian Fluid Mech., 75, 119-138.

[4] Baer, T. A., Schunk P. R., Cairncross, R. A., Rao, R. R., and Sackinger, P. A., 2000. "A finite element method for free surface flows of incompressible fluids in three dimensions. Part II. Dynamic Wetting Lines", Int. J. Numer. Meth. Fluids, 33, 405-427.

[5] Bertram, L. A., Schunk P. R., Kempka, S. N., Spadafora, F., Minisandram, R., 1998. "The Macroscale Simulation of Remelting Processes", J. of Metals, Minerals, Metals, and Materials Society, 50, 18-21. [6] Bird, R. B., Armstrong, R. C., and Hassager, O., 1987. Dynamics of Polymeric Liquids, 2nd ed., Wiley, New York, Vol. 1.

[7] Blacker, T. D., 1988. "FASTQ Users Manual: Version 1.2", Sandia Technical Report SAND88-1326. [8] Bradford, S. F. and N. D. Katopodes, 2000. "The anti-dissipative, non-monotone behavior of Petrov- Galerkin Upwinding," Int. J. Numer. Meth. Fluids, v. 33, 583-608.

[9] Brady, J. F. and Morris J. F., "Microstructure of strongly sheared suspensions and its impact on rheology and diffusion,", J. of Fluid Mechanics, v. 348 pp.103-139, Oct 10, 1997.

[10] Brooks, A. N. and T. J. R. Hughes, 1992. "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations," Comp. Math. In Appl. Mechanics and Eng., 32, 199 - 259.

[11] Brown, R. A., M. J. Szady, P. J. Northey, and R. C. Armstrong, 1993. "On the Numerical Stability of Mixed Finite-Element methods for Viscoelastic Flows Governed by Differential Constitutive Equations", Theoretical and Computational Fluid Mechanics, 5, 77-106.

[12] Cairncross, R. A., Chen, K. S., Schunk, P. R., Brinker, C. J., and Hurd, A. J., 1995. "Recent advances in theoretical modeling of deposition, drying, and shrinkage in sol-gel coating processes," Proceedings on Computational Modeling of Materials and Processing Symposium at the American Ceramic Society National Meeting, Cincinnati, OH, 30 April - 3 May.

[13] Cairncross, R. A., P.R. Schunk, K.S. Chen, S. Prakash, J. Samuel, A.J. Hurd, and C. J. Brinker, 1996. "Drying in Deformable Partially Saturated Porous Media: Sol-Gel Coatings", Sandia Technical Report, SAND96-2149.

[14] Cairncross, R. A., Schunk, P. R., Baer, T. A., Rao, R. R., and Sackinger, P. A., 2000. "A finite element method for free surface flows of incompressible fluids in three dimensions. Part I. Boundary fitted mesh motion." Int. J. Numer. Meth. Fluids, 33, 375-403.

[15] Chen, K. S., Schunk, P. R., and Sackinger, P. A., 1995. "Finite element analyses of blade and slot coating flows using a Newton-Raphson pseudo-solid domain mapping technique and unstructured grids", Proceedings of the 1995 TAPPI conference.

[16] Chen, K. S., Evans, G. H., Larson, R. S., Coltrin, M. E., and Newman, J., 1998. "Multi-dimensional modeling of thermal batteries using the Stefan-Maxwell formulation and the finite-element method", in Electrochemical Society Proceedings, Volume 98-15, p. 138-149.

[17] Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R., and Houf, W. G., 2000. "Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions", Sandia Technical Report, SAND2000-0207.

[18] Davis, T.A. and I. S. Duff, 1997. "An unsymmetric-pattern multifrontal method for sparse LU factorization", SIAM J. Matrix Analysis and Applications, January.

[19] Denbigh, K., 1981. The Principles of Chemical Equilibrium, Cambridge University Press, Cambridge.

[20] Dohrmann, C. R. and Bochev, P. B. 2004. "A stabilized finite element method for the Stokes problem based on polynomial pressure projections", Int. J. Numer. Meth. Fluids, 46, pp183-201.

[21] Droux, J. J. and T. J. R. Hughes, 1994. "A Boundary Integral Modification of the Galerkin Least Squares Formulation for the Stokes Problem," Comput. Methods Appl. Mech. Engrg., 113 (1994) 173-182.

[22] Duda, J. L., Vrentas, J. S., Ju, S. T., and Liu, H. T., 1982. "Prediction of Diffusion Coefficients for Polymer- Solvent Systems", AIChE Journal, 28(2), 279-284.

[23] Fang, Z. W., Mammoli, A. A., Brady, J.F., Ingber, M.S., Mondy, L.A. and Graham, A.L., "Flow-aligned tensor models for suspension flows," Int. J. of Multiphase Flow, v. 28(#1) pp. 137-166, January 2002.

[24] Flory, P.,1953. Principles of Polymer Chemistry, Cornell University Press, New York.

[25] Fortin, M. and A. Fortin, 1989. "A New Approach for the FEM Simulations of Viscoelastic Flow", J. Non-Newtonian Fluid Mech., 32, 295-310.

[26] Garside, J. and M.R. Al-Dibouni, 1977. "Velocity-voidage relationship for fluidization and sedimentation in solid-liquid systems," Ind. Eng. Chem. Process Des. Dev., 16, 206.

[27] Gartling, D. K., 1987. "NACHOS 2: A Finite Element Computer Program for Incompressible Flow Problems - Part 2 - User's Manual", Sandia Technical Report, SAND86-1816.

[28] Gartling, D. K., 1996. "TORO II - A Finite Element Computer Program for Nonlinear Quasi-Static Problems in Electromagnetics, Part I - Theoretical Background", Sandia Technical Report, SAND95-2472.

[29] Gartling, D. K., C. E. Hickox and R. C. Givler, 1996. "Simulations of Coupled Viscous and Porous Flow Problems", Comp. Fluid Dynamics, 7, 23-48.

[30] Gates, I. D., Labreche, D. A., and Hopkins, M. M., 2000. "Advanced Capabilities in GOMA 3.0 - Augmenting Conditions, Automatic Continuation and Linear Stability Analysis", Sandia Technical Report, SAND2000-2465.

[31] Gilkey, A. P. and Glick, J. H., 1989. "BLOT - A Mesh and Curve Plot Program for the Output of a Finite Element Analysis", Sandia Technical Report, SAND88-1432.

[32] Givler, R. C. and S. A. Altobelli, 1994. "A Determination of the Effective Viscosity for the Brinkman- Forchheimer Flow Model", J. Fluid Mechanics, 258, 355-370.

[33] Glass, M. W., 1995. Personal communication.

[34] Golub, G. H. and C. F. V. Loan, 1996. Matrix Computations, Johns Hopkins University Press, Baltimore, MD 3rd ed.

[35] Gooray, A., Roller, G., Galambos, P., Zavadil, K., Givler, R., Peter, F. and Crowley, J., A MEMS Ejector for Printing Applications, Proceedings of the Society of Imaging Science & Technology, Ft. Lauderdale FL, September 2001.

[36] Griffiths, D.F., 1997. "The 'no boundary condition' outflow boundary condition," Int. J. Numer. Meth. Fluids, 24, 393-411.

[37] Guenette, R. and M. Fortin, 1995. "A New Mixed Finite Element Method for Computing Viscoelastic Flow," J. Non-Newtonian Fluid Mech., 60 27-52.

[38] Gundersen, E. and H. P. Langtangen, 1997. "Finite Element Methods for Two-Phase Flow in Heterogeneous Porous Media," in Numerical Methods and Software Tools in Industrial Mathematics, Morten Daehlen, Aslak Tveito, Eds., Birkhauser, Boston.

[39] Helmig, R. and R. Huber, 1988. "Comparison of Galerkin-type discretization techniques for two-phase flow in heterogeneous porous media," Advances in Water Resources, 21, 697-711.

[40] Heroux, M. A., 1992. "A proposal for a sparse BLAS toolkit." Technical Report, TR/PA/92/90, CERFACS, December.

[41] Hood, P., 1976. "Frontal Solution Program for Unsymmetric Matrices", Int. J. Numer. Meth. Engr., 10, 379-399.

[42] Hopkins, M. M., Mondy, L. A., Rao, R. R., Altobelli, S. A., Fang, Z., Mammoli, A. A. and Ingber, M. S., 2001. "Three-Dimensional Modeling of Suspension Flows with a Flow-Aligned Tensor Model", The 3rd Pacific Rim Conference on Rheology, July 8-13, 2001, Vancouver, B.C., Canada.

[43] Hudson, N. E. and Jones, T. E. R., 1993. "The A1 project - an overview", J. Non-Newtonian Fluid Mech, 46, 69-88.

[44] Hughes, T. J. R. and L. P. Franca, 1987. "A New Finite Element Formulation for Computational Fluid Dynamics: VII The Stokes Problem with Various Well-Posed Boundary Conditions: Symmetric Formulations that Converge For All Velocity/Pressure Spaces," Comput. Methods Appl. Mech. Engrg., 65, 85- 96.

[45] Hughes, T. J. R., L. P. Franca and M. Balestra, 1986. "A New Finite Element Formulation for Computational Fluid Dynamics: V. Circumventing the Babuska-Brezzi Condition: A Stable Petrov-Galerkin Formulation of the Stokes Problem Accommodating Equal-Order Interpolations," Comput. Methods Appl. Mech. Engrg., 59, 85-99.

[46] Hutchinson, S. A., Shadid, J. N. and Tuminaro, R. S., 1995. "Aztec User's Guide Version 1.0", Sandia Internal Report, SAND95-1559.

[47] Irons, B. M., 1970. "A frontal solution program for finite element analysis," Int. J. Numer. Meth. Eng., 2:5-12.

[48] Kernighan, B. W. and Ritchie, D. M., 1988. The C Programming Language, 2nd Ed., PTR Prentice Hall, New Jersey.

[49] Kistler, S. F. and Scriven, L. E., 1983. Coating Flows. In Computational Analysis of Polymer Processing. Eds. J. A. Pearson and S. M. Richardson, Applied Science Publishers, London.

[50] Kool, J. B. and Parker, J. B., 1987. "Development and Evaluation of Closed-Form Expressions for Hysteretic Soil Hydraulic Properties", Water Resources Research, Vol. 23, pp 105-114.

[51] Krishnan, G. P., S. Beimfohr, and D. Leighton, 1996. "Shear-induced radial segregation in bidisperse suspensions," J. Fluid Mech. 321, 371

[52] Kundert, K. S. and Sangiovanni-Vincentelli, A., 1988. "Sparse User's Guide: Version 1.3a" Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley.

[53] Labreche, D. A., Wilkes, E. D., Hopkins, M. M. and Sun, A. C., 2006. "Advanced Capabilities in GOMA 5.0 - Augmenting Conditions, Automatic Continuation and Linear Stability Analysis", Sandia Technical Report (in preparation).

[54] Malvern, L. E., 1969, Introduction to the Mechanics of a Continuous Medium, Prentice-Hall

[55] Martinez, M. J., 1995. "Formulation and Numerical Analysis of Nonisothermal Multiphase Flow in Porous Media", Sandia Technical Report, SAND94-0379.

[56] Martinez, M. J. 1995, "Mathematical and Numerical Formulation of Nonisothermal Multicomponent Three-Phase Flow in Porous Media", Sandia Technical Report, SAND95-1247.

[57] Martys, N., D. P. Bantz and E. J. Barboczi, 1994. "Computer Simulation Study of the Effective Viscosity in Brinkman's Equation." Phys. Fluids, 6, 1434-1439

[58] Morris, J. F. and Boulay, F., "Curvilinear flows of noncolloidal suspensions: The role of normal stresses," J. of Rheology, v. 43(#5) pp. 1213-1237 Sep-Oct 1999.

[59] Newman, J. S., Electrochemical Systems, Prentice Hall, Inc., Englewood Cliffs, New Jersey (1991).

[60] Papanastasiou, T. C., 1987. "Flows of Materials with Yield", Journal of Rheology, 31 (5), 385-404.

[61] Papanastasiou, T. C., N. Malamataris, and K. Ellwood, 1992. "A New Outflow Boundary Condition", Int. J. for Numerical Methods in Fluids, 14, 587-608.

[62] Papananstasiou, T. C., and Boudouvis, A. G., 1997. "Flows of Viscoplastic Materials: Models and Computation," Computers & Structures, Vol 64, No 1-4, pp 677-694.

[63] Patterson, D., Y.B. Tewari, H.P. Schreiber, and J.E. Guillet, 1971. "Application of Gas-Liquid Chromatography to the Thermodynamics of Polymer Solutions", Macromolecules, 4, 3, 356-358.

[64] PDA Engineering, 1990. "PATRAN Plus User Manual", Publication No. 2191024, Costa Mesa, California, January.

[65] Phillips, R.J., R.C. Armstrong, and R.A. Brown, 1992. "A constitutive equation for concentrated suspensions that accounts for shear-induced particle migration", Physics of Fluids A, 4(1), 30-40.

[66] Price, P. E., Jr., S. Wang, I. H. Romdhane, 1997. "Extracting Effective Diffusion Parameters from Drying Experiments", AIChE Journal, 43, 8, 1925-1934.

[67] Rajagopalan, D., R. C. Armstrong and R. A. Brown, 1990. "Finite Element Methods for Calculation of Viscoelastic Fluids with a Newtonian Viscosity", J. Non-Newtonian Fluid Mech., 36 159-192.

[68] Rao, R. R., Mondy, L. A., Schunk, P. R., Sackinger P. A., and Adolf, D. B., 2001. "Verification and Validation of Encapsulation Flow Models in GOMA, Version 1.1", Sandia Technical Report, SAND2001- 2947.

[69] Renardy, M., 1997. "Imposing 'NO' boundary conditions at outflow: Why does this work?" Int. J. for Numerical Methods in Fluids, 24, 413-417.

[70] Rew, R. K., Davis, G. P., and Emmerson, S., 1993. "NetCDF User's Guide: An Interface for Data Access", Version 2.3, University Corporation for Atmospheric Research, Boulder, Colorado, April.

[71] Saad, Y., 1994. "ILUT: a dual threshold incomplete ILU factorization", Numerical Linear Algebra with Applications, 1:387-402.

[72] Sackinger, P. A., Schunk, P. R. and Rao, R. R., 1996. "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", J. Comp. Phys., 125, 83-103.

[73] Salinger, A. G., N.M. Bou-Rabee, E.A. Burroughs, R.B. Lehoucq, R.P. Pawlowski, L.A. Romero, and E.D. Wilkes, 2002. "LOCA 1.0: Theory and Implementation Manual", Sandia Technical Report, SAND2002-0396.

[74] Sani, R. L., and P. M. Gresho, 1994. "Resume and remarks on the open boundary condition minisymposium," Int. J. for Numerical Methods in Fluids, 18, 983-1008.

[75] Scherer, G.W., 1992. "Recent Progress in Drying of Gels", J. of Non-Crystalline Solids, 147 & 148, 363- 374.

[76] Schoof, L. A. and Yarberry, V. R., 1994. "EXODUS II: A Finite Element Data Model", Sandia Technical Report, SAND92-2137.

[77] Schunk, P. R., 1999. "TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid-Structure Interaction Problems", Sandia Technical Report, SAND2000-0807.

[78] Schunk, P. R., Sackinger, P. A., Rao, R. R., Chen, K. S., Cairncross, R. A., Baer, T. A., and Labreche, D. A., 1997. "GOMA 2.0- A Full-Newton Finite Element Program for free and Moving boundary Problems with Coupled Fluid/Solid Momentum, Energy, Mass, and chemical Species Transport: User's Guide", SAND97-2404.

[79] Schunk, P. R. and Shadid, J. N., 1992. "Iterative Solvers in Implicit Finite Element Codes" Sandia Technical Report, SAND92-1158.

[80] Schunk, P. R. and Rao, R. R., 1994. "Finite element analysis of multicomponent two-phase flows with interphase mass and momentum transport", Int. J. Numer. Meth. Fluids, 18, 821-842.

[81] Schunk, P. R., M. A. Heroux, R. R. Rao, T. A. Baer, S. R. Subia and A. C. Sun., 2002. "Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems", Sandia Technical Report, SAND2001-3512J.

[82] Schwartz, L.W., R.A. Cairncross and D.E. Weidner, 1996. "Anomalous Behavior During Leveling of Thin Coating Layers with Surfactant", Phys. Fluids, 8, (7), 1693-1695.

[83] Segalman, D., Witkowski, W., Adolf, D. and Shahinpoor, M., 1992. "Theory and Application of Electrically Controlled Polymeric Gels", Smart Mater. Struct., 1, 95-100.

[84] Shadid, J. N., Moffat, H. K., Hutchinson, S. A., Hennigan, G. L., Devine, K. D. and Salinger, A. G., 1995. "MPSalsa: A finite element computer program for reacting flow problems, Part 1 - Theoretical development." Sandia Technical Report, SAND95-2752.

[85] Sjaardema, G. D., 1992. "APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses", Sandia Technical Report, SAND92-2291.

[86] Sjaardema, G. D., 1993. "Overview of the Sandia National Laboratories Engineering Analysis Code Ac cess System", Sandia Technical Report, SAND92-2292.

[87] Sun, J., N. Phan-Thien, R. I. Tanner, 1996. "An Adaptive Viscoelastic Stress Splitting Scheme and Its Applications: AVSS/SI and AVSS/SUPG," J. Non-Newtonian Fluid Mech., 65, 75-91.

[88] Sun, J., M. D. Smith, R. C. Armstrong, R. A. Brown, 1999. "Finite Element Method for Viscoelastic Flow Bases on the Discrete Adaptive Viscoelastic Stress Splitting and the Discontinuous Galerkin Method: DAVSS-G/DG," J. Non-Newtonian Fluid Mech., 86, 281-307.

[89] Tam, S. Y., 1997. "Stress Effects in Drying Coatings", Ph. D. Thesis, University of Minnesota,. Available on from University Microfilms, Ann Arbor, MI.

[90] Taylor, R. and R. Krishna, 1993. Multicomponent Mass Transfer. John Wiley & Sons, New York. [91] Tuminaro, R. S., Heroux, M. A., Hutchinson, S. A. and Shadid, J. N., 1999. "Official Aztec User's Guide Version 2.1", Sandia Technical Report, SAND99-8801J.

[92] Unger, A. J. A., P. A. Forsyth and E. A. Sudicky, 1996. "Variable spatial and temporal weighting schemes for use in multi-phase compositional problems," Advances in Water Resources, 19, 1 - 27.

[93] Vrentas, J.S., J.L. Duda and H.-C. Ling, 1984. "Self-Diffusion in Polymer-Solvent-Solvent Systems", Journal of Polymer Sciences: Polymer Physics edition, (22), 459-469.

[94] Zhang K. and A. Acrivos, 1994, "Viscous resuspension in fully-developed laminar pipe flows," Int. J. Multiphase Flow, (20)3, 579-591.

[95] Zielinski, J.M. and B.F. Hanley, 1999. "Practical Friction-Based Approach to Modeling Multicomponent Diffusion." AIChE Journal, (45)1, 1-12.

[96] Zlatev, Z., Wasniewski, J., and Schaumburg, K., 1981. "Y12M. Solution of large and sparse systems of linear algebraic equations." Lecture notes in computer science, 121, Springer-Verlag, New York.

1.7 Appendix 1: Goma Documentation Lists

The documents identified in this appendix constitute a current list of instructional, technical and reference material for Goma and the CRMPC Consortium of Companies.

1.7.1 Reference Manuals

GDM-1. GOMA 3.0 - A Full-Newton Finite Element Program for Free and Moving Boundary Problems with Coupled Fluid/Solid Momentum, Energy, Mass, and Chemical Species Transport: Developer's Guide, Schunk, P. R., Sackinger, P. A., Rao, R. R., Subia, S. R., Baer, T. A., Labreche, D. A., Moffat, H. K., Chen, K. S., Hopkins, M. M. and Roach, R. A., October 2000. (no link here; see printed document)

GSR-01.3 Advanced Capabilities in GOMA 4.0 - Augmenting Conditions, Automatic Continuation, and Linear Stability Analysis, SAND Report, Labreche, D. A., Wilkes, E. D., Hopkins, M. M. and Sun, A. C., (In Prep).

SAND95-1559 Aztec User's Guide Version 1.0, SAND95-1559, Hutchinson, S. A., Shadid, J. N. and Tuminaro, R. S., October 1995.

SAND96-2149 Drying in Deformable Partially-Saturated Porous Media: Sol-Gel Coatings, SAND96-2149, Cairncross, R. A., Schunk, P. R., Chen, K. S., Prakash, S. S., Samuel, J., Hurd, A. J. and Brinker, C. J., September 1996.

SAND97-2404 GOMA 2.0 - A Full-Newton Finite Element Program for Free and Moving Boundary Problems with Coupled Fluid/Solid Momentum, Energy, Mass, and Chemical Species Transport: User's Guide, SAND97-2404, Schunk, P. R., Sackinger, P. A., Rao, R. R., Chen, K. S., Cairncross, R. A., Baer, T. A. and Labreche, D. A.

SAND2000-0207 Final Report on LDRD Project: A Phenomenological Model for Multicomponent Transport with Simultaneous Electrochemical Reactions in Concentrated Solutions, SAND2000-0207, Chen, K. S., Evans, G. H., Larson, R. S., Noble, D. R., and Houf, W. G., January 2000.

SAND2000-0807 TALE: An Arbitrary Lagrangian-Eulerian Approach to Fluid-Structure Interaction Problems, SAND2000-0807, Schunk, P. R., May 2000.

SAND2000-2465 Advanced Capabilities in GOMA 3.0 - Augmenting Conditions, Automatic Continuation, and Linear Stability Analysis, SAND2000-2465, Gates, I. D., Labreche, D. A. and Hopkins, M. M., January, 2001.

SAND2001-2947 Verification and Validation of Encapsulation Flow Models in GOMA, Version 1.1, SAND2001-2947, Rao, R. R., Mondy, L. A., Schunk, P. R., Sackinger, P. A., Adolf, D. B., October 2001.

SAND2001-3512J Iterative Solvers and Preconditioners for Fully-coupled Finite Element Formulations of Incompressible Fluid Mechanics and Related Transport Problems, SAND2001-3512J, Schunk, P. R., Heroux, M. A., Rao, R. R., Baer, T. A., Subia, S. R. and Sun., A. C., March 2002.

SAND2002-0396 LOCA 1.0: Library of Continuation Algorithms: Theory and Implementation Manual, SAND2002-0396, Salinger, A. G., Bou-Rabee, N. M., Pawlowski, R. P., Wilkes, E. D., Burroughs, E. A., Lehoucq, R. B. and Romero, L. A., March 2002.

SAND2002-3204 GOMA 4.0 - A Full-Newton Finite Element Program for Free and Moving Boundary Problems with Coupled Fluid/Solid Momentum, Energy, Mass, and Chemical Species Transport: User's Guide, SAND2002-3204, Schunk, P. R., Sackinger, P. A., Rao, R. R., Chen, K. S., Baer, T. A., Labreche, D. A, Sun., A. C., Hopkins, M. M., Subia, S. R., Moffat, H. K., Secor, R. B., Roach, R. A.,Wilkes, E. D., Noble, D. R., Hopkins, P. L. and Notz, P. K., November 2002 (link elsewhere)

1.7.2 Technical Memoranda

GTM-001.0 Pressure Stabilization in Goma using Galerkin Least Squares, July 17, 1996, R. R. Rao

GTM-002.0 Suspension flow in a concentric Couette device, a benchmark of the GOMA code, November 24, 1997, S. R. Subia, T. A. Baer and R. R. Rao

GTM-003.0 Reactive LDRD Task 1 Report: Governing Equations for Liquid-Phase Multicomponent Transport of Ionic or Charged Species, March 16, 1998, K. S. Chen, G. H. Evans and R. S. Larson

GTM-004.1 Corners and Outflow Boundary Conditions in GOMA, April 24, 2001, P. R. Schunk

GTM-005.2 GOMA Simulation of Monodisperse Suspensions in Torsional Flow Viscometers, April 27, 2000, A. C. Sun

GTM-006.0 Demonstration of GOMA/DAKOTA Interface for Parameter Estimation September 28, 1998, B. F. Blackwell

GTM-007.1 New Multicomponent Vapor-Liquid Equilibrium Capabilities in GOMA, December 10, 1998, A. C. Sun

GTM-008.0 Finite Element Modeling of Suspensioned Particle Migration in Non-Newtonian Fluids, R. R. Rao, L. A. Mondy, T. A. Baer, S. Altobelli and T. Stephens

GTM-009.0 NMR Measurements and Finite Element Modeling of Non-Neutrally Buoyant Suspensions, L. A. Mondy, R. R. Rao, A. C. Sun, S. Altobelli and J. Seymour

GTM-010.0 The Hindered Settling Function for a Glass Microballoon Suspension, March 3, 1999, C. A. Romero

GTM-011.0 Validation of 828/DEA/GMB Encapsulant using GOMA, August 20, 1999, A. C. Sun

GTM-012.0 Parameter Estimation of Drying Models using GOMA and DAKOTA, September 28, 1999, A. C. Sun

GTM-013.0 On the Verification and Validation of the Stefan-Maxwell Flux Model in GOMA: Ternary Gaseous Diffusion in a Stefan Tube, December 23, 1999, K. S. Chen

GTM-014.0 Parallel Simulation of Three-Dimensional Free-Surface Fluid Flow Problems, January, 2000, T. A. Baer, S. R. Subia and P. A. Sackinger

GTM-015.1 Implementation Plan for Upgrading Boundary Conditions at Discontinuous- Variable Interfaces, January 8, 2001, H. K. Moffat

GTM-016.0 Laser Spot Weld Modeling using an ALE Finite Element Method, D. R. Noble, P. R. Schunk, A. Kassinos and M. P. Kanouff (Unpublished DRAFT)

GTM-017.0 Parallel Plate Viscometer Verification of GOMA, March 8, 2000, E. R. Lindgren

GTM-018.0 Cone-and-Plate Viscometer Verification of GOMA, March 8, 2000, E. R. Lindgren

GTM-019.1 Assessment and Plan to Implement a VGI in GOMA, February 15, 2001, P.R. Schunk

GTM-020.0 In-Situ Characterization of Stress Development in Gelatin Film During Controlled Drying, M. Lu, S.-Y. Tam, P. R. Schunk and C. J. Brinker, March 2000.

GTM-021.0 Multiparameter continuation and linear stability analysis on highly deformable meshes in Goma, M. M. Hopkins, June 22, 2000

GTM-022.0 On the Verification of GOMA's Capability for Modeling Transient Diffusion Processes Involving Dilute Solute Species and Slow Surface Chemical Reaction, K. S. Chen, March 31, 2000.

GTM-023.0 On the Verification of GOMA Baseline Model for Atmospheric Copper Sulfidation in the Gas-phase Diffusion Regime – Fixed Sulfidation-Front Approximation, K. S. Chen, May 5, 2000.

GTM-024.0 Simulations with the Pressure-Stabilized Petrov-Galerkin (PSPG) Finite-Element Method, J. R. Torczynski, September 13, 2000

GTM-025.0 Modeling diffusion and migration transport of charged species in dilute electrolyte solutions: GOMA implementation and sample computed predictions from a case study of electroplating, K. S. Chen, September 21, 2000

GTM-026.0 A Generic GOMA Model for Drying of Two-Layer, Ternary, Polymeric Film- Coatings Involving Two Solvents and a Non-Porous/Impermeable Substrate, K. S. Chen, November 9, 2000.

GTM-027.0 Probing Plastic Deformation in Gelatin Film during Drying, M. Lu, S. Y. Tam, A. Sun, P. R. Schunk and C. J. Brinker, 2000.

GTM-028.0 Modeling Drying of Dip-Coated Films with Strongly-Coupled Gas Phase Natural Convection, R. A. Cairncross, August 1999.

GTM-029.0 SUPG Formulation for the Porous Flow Equations in Goma, H. K. Moffat, August 2001 (DRAFT).

GTM-030.0 A Baseline Multi-Dimensional Mathematical Model of Copper Sulfidation for the Initial Implementation in GOMA, K. S. Chen, January 22, 1999.

GTM-031.0 On Implementing and Verifying in Goma the Poisson Equation Governing Electric Potential in Electrochemical Processes Involving Charge Separation such as in Copper Sulfidation, K. S. Chen, May 15, 2002

1.7.3 Tutorials

GT-001.4 GOMA and SEAMS tutorial for new users, February 18, 2002, P. R. Schunk and D. A. Labreche

GT-002.1 Slot coating templates and tutorial for GOMA and SEAMS, (GT-002.1), July 29, 1999, P. R. Schunk

GT-003.1 Roll coating templates and tutorial for GOMA and SEAMS (GT-003.1), February 29, 2000, P. R. Schunk and M. Stay

GT-004.1 REVISED: DAKOTA tutorial for new users, October 24, 1997, T. Simmermacher and M. Eldred

GT-005.3 THE NEW TOTAL-ARBITRARY-LAGRANGIAN-EULERIAN (TALE) CAPABILITY and its applicability to coating with/on deformable media (GT- 005.3), August 6, 1999, P. R. Schunk

GT-006.3 Slot and Roll coating with remeshing templates and tutorial for GOMA and CUBIT/MAPVAR (GT-006.3), August 3, 1999, R. R. Lober and P. R. Schunk

GT-007.2 Tutorial on droplet on incline problem (GT-007.2), July 30, 1999, T. A. Baer

GT-008.2 Porous Media Capabilities/Tutorial for GOMA. User Guidance for Saturated Porous Penetration Problems (GT-008.2), August 11, 1999, P. R. Schunk

GT-009.3 GOMA's Capabilities for Partially Saturated Flow in Porous Media (GT-009.3), September 1, 2002, P. R. Schunk

GT-010.1 Slot Coating Optimization, March 16, 1999, T. Simmermacher

GT-011.1 Slide Coating Templates and Tutorial for GOMA (GT-011.1), March 17, 1999, P. R. Schunk and D. A. Labreche

GT-012.0 3D Roll coating template and tutorial for GOMA (GT-012.0), February 21, 2000, P.R. Schunk

GT-013.2 Computations for slot coater edge section (GT-013.2), October 10, 2002, T.A. Baer

GT-014.1 Tutorial for Running Viscoelastic Flow Problems in GOMA (GT-014.1), June 21, 2000, R. R. Rao

GT-015.0 Template for parameter continuation and operability window estimation using Perl scripts and Goma for a slot coater (GT-015.0), June 22, 2000, M. M. Hopkins

GT-016.1 Software Developer's Tutorial for GOMA (GT-016.1), January 9, 2001, P. R. Schunk

GT-017.1 Parallel GOMA Tutorial (GT-017.1), S. R. Subia and P. A. Sackinger, January 22, 2001

GT-018.1 ROT card tutorial (GT-018.1), January 22, 2001, T. A. Baer

GT-019.2 Elastoviscoplastic (EVP) Constitutive Model in GOMA: Theory, Testing, and Tutorial (GT-019.1), P. R. Schunk, A. Sun, S.Y. Tam and K. S. Chen, March 13, 2003

GT-020.3 Tutorial on Level Set Interface Tracking in GOMA (GT-020.3), July 31, 2005, T.A. Baer

GT-021.2 Common Geometry Model (CGM) Usage for GOMA (GT-021.2), August 20, 2002, M. M. Hopkins

GT-022.0 Library of Continuation Algorithms (LOCA) Usage for GOMA (GT-022.0), August 15, 2002, E. D. Wilkes

GT-023.1 Usage of ARPACK eigensolver for linear stability analysis in GOMA (GT-023.1), April 2, 2004, E. D. Wilkes

GT-024.0 Solution Procedure for Three Dimensional Free Surface Flow and 3D Remeshing (GT-024.0), August 23, 2002, T.A. Baer

GT-025.0 Using Element Quality Metrics in GOMA (GT-025.0), September 15, 2003, E. D. Wilkes

GT-026.4 GOMA's Overset Mesh Method: User Tutorial (GT-026.4), January 11, 2006, P. R. Schunk and E. D. Wilkes

GT-027.1 GOMA's Shell Structure Capability: User Tutorial (GT-027.1), March 1, 2004, P. R. Schunk and E. D. Wilkes

GT-028.0 Liquid Drop Impact on a Porous Substrate: a level-set tutorial (GT-028.0), July 31, 2005, P. R. Schunk

GT-029.1 Modeling wetting contact and dewetting Phenomena with Goma level-set capability (GT-029.1), August 10, 2005, T. A. Baer and P. R. Schunk

GT-030.0 Tutorial memo on 2D overflow problem from Corning Inc. (GT-030.0), July 12, 2005, T. A. Baer

GT-031.0 Tutorial on solving microfilling problem using level set method implemented in GOMA (GT-031.0), June 13, 2005, T. A. Baer

GT-032.0 Tutorial on solution of 3D microfilling problem using level set method implemented in GOMA (GT-32.0), June 13, 2005, T. A. Baer

GT-033.0 Structural shell application example: tensioned-web slot coater (GT-033.0), April 1, 2006, E. D. Wilkes and P. R. Schunk

1.7.4 Goma Collections

GC-001 Goma Tutorials, Documents and Related Memos – March 1997 to October 1, 1998 (Distributed: October 1998 CRMPC Meeting) SAND96-2149 GDM-1.0 GT-001.2 GTM-001.0 GT-002.0 GTM-002.0 GT-003.0 GTM-003.0 GT-004.0 GTM-004.0 GT-005.1 GTM-005.0 GT-006.1 GT-007.1

GC-002 Goma Tutorials, Documents and Related Memos – October 1, 1998 to March 31, 1999 (Distributed: March 1999 CRMPC Meeting) GSR-01.0 GT-008.1 GTM-006.0 GT-009.0 GTM-007.1 GT-010.1 GTM-008.0 GT-011.0 GTM-009.0 GTM-010.0

GC-003 Goma Tutorials, Documents and Related Memos – April 1, 1999 to March 31, 2000 (Distributed: March 2000 CRMPC Meeting) GT-003.1 GTM-011.0 GT-005.3 GTM-012.0 GT-006.3 GTM-013.0 GT-012.0 GTM-014.0 GT-013.0 GTM-015.0 GTM-016.0 GTM-017.0 GTM-018.0 GTM-019.0 GTM-020.0

GC-004 Goma Tutorials, Documents and Related Memos – April 1, 2000 to January 31, 2001 (Distributed: January 2001 CRMPC Meeting) GT-015.0 GTM-015.1 GT-016.1 GTM-021.0 GT-017.1 GTM-022.0 GT-018.1 GTM-023.0 GT-019.1 GTM-024.0 GTM-025.0 GTM-026.0 GTM-027.0

GOMA Document List - 01/29/01]