Elmer Models Manual

Peter Råback, Mika Malinen, Juha Ruokolainen,
Antti Pursula, Thomas Zwinger, Eds.

CSC – IT Center for Science

February 9, 2023

Elmer Models Manual

About this document

Elmer Models Manual is a part of the documentation of Elmer finite element software. It consists of independent chapters describing different modules a.k.a. solvers which the main program (ElmerSolver) uses to create a specific physical model, although some solvers take care of computational tasks which are not tied up with the concept of physical model.

The structure of the manual reflects the modular architecture of the software which enables the addition of new modules without making any changes to the main program. Each chapter typically has a separate section for theory and keywords, and often some additional information is also given, for example about the limitations of the model. The Elmer Models Manual is best used as a reference manual rather than a concise introduction to the matter.

The present manual corresponds to Elmer software version 9.0. The latest documentation and program versions of Elmer are available (or links are provided) at `http://www.csc.fi/elmer`.

Copyright information

Contents

Part I

Models of Fluid Mechanics and Transport Phenomena

Model 1

Heat Equation

Module name: HeatSolve,HeatSolveVec
Module subroutines: HeatSolver
Module authors: Juha Ruokolainen, Peter Råback, Matthias Zenker
Document authors: Juha Ruokolainen, Ville Savolainen, Peter Råback

1.1 Introduction

Heat equation results from the requirement of energy conservation. In addition the Fourier's law is used to model the heat conduction. The linearity of the equation may be ruined by temperature dependent thermal conductivity, or by heat radiation.

Note that there are two versions of the heat solver. The older one in module `HeatSolve` includes some more physics while the newer one in module `HeatSolveVec` has been optimized for speed and also has some new features.

1.2 Theory

1.2.1 Governing Equations

The incompressible heat equation is expressed as

$$\rho c_p \left(\frac{\partial T}{\partial t} + (\vec{u} \cdot \nabla) T \right) - \nabla \cdot (k \nabla T) = \overline{\overline{\tau}} : \overline{\overline{\varepsilon}} + \rho h, \tag{1.1}$$

where ρ is the density, c_p the heat capacity at constant pressure, T the temperature, $\vec{u}$ the convection velocity, k the heat conductivity and h is source of heat. The term $\overline{\overline{\tau}} : \overline{\overline{\varepsilon}}$ is the frictional viscous heating, which is negligible in most cases. For Newtonian fluids, the viscous part of the stress tensor is

$$\overline{\overline{\tau}} = 2\mu\overline{\overline{\varepsilon}}, \tag{1.2}$$

where $\overline{\overline{\varepsilon}}$ the linearized strain rate tensor.

Eq.1.1 applies also for solids, setting $\vec{u} = 0$. For solids, conduction may be anisotropic and the conductivity a tensor.

For compressible fluids, the heat equation is written as

$$\rho c_v \left(\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T \right) - \nabla \cdot (k \nabla T) = -p \nabla \cdot \vec{u} + \overline{\overline{\tau}} : \overline{\overline{\varepsilon}} + \rho h, \tag{1.3}$$

where c_v is the heat capacity at constant volume. The density needs to be calculated from the equation of state, e.g., perfect gas law. More information is given in the chapter describing the Navier-Stokes equation.

The Elmer heat equation module is capable of simulation heat transfer by conduction, convection, and diffuse gray radiation. Also a phase change model is included. Couplings to other modules include, convection by fluid flow, frictional heating (modules providing flow fields), and resistive heating (modules providing magnetic and/or electric fields).

1.2.2 Arbitrary Lagrangian-Eulerian (ALE) coordinates

For problems involving a deforming mesh the transient heat equation must be solved using Arbitrary Lagrangian-Eulerian (ALE) frame of reference. Assume that the mesh velocity is $\vec{c}$. Then the convective term yields

$$\rho c_p \left((\vec{u} - \vec{c})\right) \cdot \nabla) T \tag{1.4}$$

1.2.3 Phase Change Model

Elmer has an internal fixed grid phase change model. Modelling phase change is done by modifying the definition of heat capacity according to whether a point in space is in solid or liquid phase or in a 'mushy' region. The choice of heat capacity within the intervals is explained in detail below.

This type of algorithm is only applicable, when the phase change occurs within finite temperature interval. If the modelled material is such that the phase change occurs within very sharp temperature interval, this method might not be appropriate.

For the solidification phase change model Elmer uses, we need enthalpy. The enthalpy is defined to be

$$H(T) = \int_0^T \left(\rho c_p + \rho L \frac{\partial f}{\partial \lambda} \right) d\lambda, \tag{1.5}$$

where $f(T)$ is the fraction of liquid material as a function of temperature, and L is the latent heat. The enthalpy-temperature curve is used to compute an effective heat capacity, whereupon the equations become identical to the heat equation. There are two ways of computing the effective heat capacity in Elmer:

$$c_{p,\mathrm{eff}} = \frac{\partial H}{\partial T}, \tag{1.6}$$

and

$$c_{p,\mathrm{eff}} = \left(\frac{\nabla H \cdot \nabla H}{\nabla T \cdot \nabla T} \right)^{1/2}. \tag{1.7}$$

The former method is used only if the local temperature gradient is very small, while the latter is the preferred method. In transient simulations a third method is used, given by

$$c_{p,\mathrm{eff}} = \frac{\partial H / \partial t}{\partial T / \partial t}. \tag{1.8}$$

Note that for the current implementation of the heat equation heat capacity and enthalpy are additive. So if heat capacity is present in the command file it should not be incorporated to enthalpy as an integral.

1.2.4 Additional Heat Sources

Frictional heating is calculated currently, for both incompressible and compressible fluids, by the heat source

$$h_f = 2\mu \bar{\bar{\varepsilon}} : \bar{\bar{\varepsilon}}. \tag{1.9}$$

In case there are currents in the media the also the the resistive heating may need to be considered. The Joule heating is then given by

$$h_m = \frac{1}{\sigma} \vec{J} \cdot \vec{J}. \tag{1.10}$$

In the above equations, $\vec{B}$ and $\vec{E}$ are the magnetic and electric fields, respectively. The current density $\vec{J}$ is defined as

$$\vec{J} = \sigma(\vec{E} + \vec{u} \times \vec{B}). \tag{1.11}$$

In modeling biological tissue perfused with blood acting as heat sink an additional heat source term of the Pennes' Bioheat equation is needed. The term is

$$h_b = c_b \rho_b w (T_b - T) \tag{1.12}$$

where c_b is the specific heat capacity, ρ_b the density, and T_b the temperature of the blood. The perfusion rate w is the volume of blood flowing through a unit volume of tissue per second. This additional source term is modeled so that the part including T is treated implicitly for better convergence. Even though the model was written for the biological application in mind the additional heat source may find also other uses.

1.2.5 Boundary Conditions

For temperature one can apply boundary conditions and have either temperature or heat flux prescribed.

Dirichlet boundary condition (temperature is prescribed) reads as

$$T = T_b. \tag{1.13}$$

The value of T_b can be constant or a function of time, position or other variables.

Heat flux depending on heat transfer coefficient α and external temperature T_{ext} may be written as

$$-k\frac{\partial T}{\partial n} = \alpha(T - T_{ext}). \tag{1.14}$$

Both variables α and T_{ext} can be constant or functions of time, position or other variables. If the heat transfer coefficient α is equal to zero, it means that the heat flux on a boundary is identically zero. The Neumann boundary condition $-k\partial T/\partial n = 0$ is also used in a symmetry axis in 2D, axisymmetric or cylindrical problems.

Heat flux can consist of idealized radiation whereupon

$$-k\frac{\partial T}{\partial n} = \sigma\varepsilon(T^4 - T_{\text{ext}}^4). \tag{1.15}$$

Above, σ is the Stefan-Boltzmann constant and ε the surface emissivity. The emissivity and the external temperature can again be constant or functions of time, position, or other variables.

The heat equation is nonlinear when radiation is modelled. The nonlinear term in the boundary condition (1.15) can be linearized as

$$T^4 - T_{\text{ext}}^4 \approx (\mathcal{T}^3 + T_{\text{ext}}\mathcal{T}^2 + T_{\text{ext}}^2\mathcal{T} + T_{\text{ext}}^3)(T - T_{\text{ext}}), \tag{1.16}$$

where $\mathcal{T}$ is the temperature from the previous iteration. Alternatively Newton's linearization may be used. The latter is usually converging faster but may have a smaller radius of convergence.

In many problems involving thermal radiation the geometry is such that the objects sees itself at least partially. This involves the use of view factors

If the surface k is receiving radiation from other surfaces in the system, then the heat flux for the Gebhart factors model is

$$-k_k\frac{\partial T_k}{\partial n_k} = \sigma\varepsilon_k(T_k^4 - \frac{1}{A_k\varepsilon_k}\sum_{i=1}^{N} G_{ik}\varepsilon_i T_i^4 A_i), \tag{1.17}$$

where the subscripts i and k refer to surfaces i and k, and the parameters A_i and A_k to the specific surface areas. The factors G_{ik} are Gebhart factors, and N represents the total number of radiating surfaces present in the system. The radiosity model computes the radiosity vectors J which relate to the flux as

$$-k_k\frac{\partial T_k}{\partial n_k} = \frac{\varepsilon_k A_k}{(1-\varepsilon_k)}(\sigma T_k^4 - J_k) \tag{1.18}$$

A "radiator" is a pointlike source radiating energy, giving raise to flux

$$-k\frac{\partial T}{\partial n} = \sum_i P_i \alpha_i \tag{1.19}$$

where P_i is the power of the source and α_i is vector of geometrical factors giving the portion of energy incident on a given boundary element. The factors take into account the orientation of the boundary element with respect to the sources and shadowing. Note that this type of source is not available in cylindrically symmetric cases.

One may also give an additional heat flux term as

$$-k\frac{\partial T}{\partial n} = q. \tag{1.20}$$

1.3 Keywords

`Constants`

`Stefan Boltzmann` Real
: The value of the Stefan-Boltzmann constant needed for thermal radiation.

`Simulation`
The simulation section gives the case control data:

`Simulation Type` String
: Heat equation may be either `Transient` or `Steady State`.

`Coordinate System` String
: Defines the coordinate system to be used, one of: `Cartesian 1D`, `Cartesian 2D`, `Cartesian 3D`, `Polar 2D`, `Polar 3D`, `Cylindric`, `Cylindric Symmetric` and `Axi Symmetric`.

`Timestepping Method` String
: Possible values of this parameter are `Newmark` (an additional parameter `Newmark Beta` must be given), `BDF` (`BDF Order` must be given). Also as a shortcut to `Newmark`-method with values of `Beta`= $0.0, 0.5$, 1.0 the keywords `Explicit Euler`, `Crank-Nicolson`, and `Implicit Euler` may be given respectively. The recommended choice for the first order time integration is the BDF method of order 2.

`BDF Order` Integer
: Value may range from 1 to 5.

`Newmark Beta` Real
: Value in range from 0.0 to 1.0. The value 0.0 equals to the explicit Euler integration method and the value 1.0 equals to the implicit Euler method.

`Solver` solver id
The solver section defines equation solver control variables. Most of the possible keywords – related to linear algebra, for example – are common for all the solvers and are explained elsewhere.

`Equation` String [Heat Equation]
: The name of the equation. If it is `Heat Equation` then an old logic will define the following keyword.

`Procedure` File "HeatSolve" "HeatSolver"
: The name of the procedure for the heat equation. This must be given as stated here.

`Variable` String [Temperature]
: This may be of any name for temperature as long at it is used consistently elsewhere. The default name is `Temperature`.

`Nonlinear System Convergence Tolerance` Real
: The criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations is small enough

$$||T_i - T_{i-1}|| < \epsilon ||T_i||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Max Iterations` Integer
: The maximum number of nonlinear iterations the solver is allowed to do.

`Nonlinear System Newton After Iterations` Integer
: Change the nonlinear solver type to Newton iteration after a number of Picard iterations have been performed. If a given convergence tolerance between two iterations is met before the iteration count is met, it will switch the iteration type instead. In the heat equation the Picard iterations means that the radiation term is factorized to linear and third-power terms.

`Nonlinear System Newton After Tolerance` Real
: Change the nonlinear solver type to Newton iteration, if the relative change of the norm of the field variable meets a tolerance criterion:

$$||T_i - T_{i-1}|| < \epsilon ||T_i||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Relaxation Factor` Real
: Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achieve convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:

$$T_i^{'} = \lambda T_i + (1 - \lambda) T_{i-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

`Steady State Convergence Tolerance` Real
: With this keyword a equation specific steady state or coupled system convergence tolerance is given. All the active equation solvers must meet their own tolerances before the whole system is deemed converged. The tolerance criterion is:

$$||T_i - T_{i-1}|| < \epsilon ||T_i||,$$

where ϵ is the value given with this keyword.

`Stabilize` Logical
: If this flag is set true the solver will use stabilized finite element method when solving the heat equation with a convection term. If this flag is set to `False` RFB (Residual Free Bubble) stabilization is used instead (unless the next flag `Bubbles` is set to `False` in a problem with Cartesian coordinate system). If convection dominates stabilization must be used in order to successfully solve the equation. The default value is `False`.

`Bubbles` Logical
: There is also a residual-free-bubbles formulation of the stabilized finite-element method. It is more accurate and does not include any ad hoc terms. However, it may be computationally more expensive. The default value is `True`. If both `Stabilize` and `Bubbles` or set to `False`, no stabilization is used. Note that in this case, the results might easily be nonsensical.

`Smart Heater Control After Tolerance` Real
: The smart heater control should not be activated before the solution has somewhat settled. By default the smart heater control is set on when the Newtonian linearization is switched on for the temperature equation. Sometimes it may be useful to have more stringent condition for turning on the smart heater control and then this keyword may be used to give the tolerance.

`Apply Limiter` Logical
: The generic soft limiters may be applied for the heat equation equation. They could for example, account for the effects of phase change under circumstances where it may be assumed that the temperature does not go over the phase change temperature. With this flag active the minimum and maximum limiters are accounted.

Radiating sources ("radiators") are activated using the following keywords (see also the "Boundary Conditions" section). Also some of the view factor shadowing subdivision/accuracy keywords are used when resolving the shadowing between the radiation sources and participating boundary surfaces.

`Radiator Coordinates(n,3)` Real
: Activate n radiating sources by giving their coordinates in space.

`Radiator Power` Real
: Give power of each of the radiating sources introduced with "Radiator Coordinates" -keyword. This may be a vector also. However, if the powers have a functional dependence, on e.g. time, rather use separate keyword `Radiator Power i` for each point source.

`Radiator Temperature` Real
: This keyword has only an effect when the spectral radiosity model is used! Otherwise the temperature of the radiator does not have any effect. Also this may be a vector. However, if the powers have a functional dependence, on e.g. time, rather use separate keyword `Radiator Temperature i` for each point source.

`Compute Radiator Factors` Logical
: Force computation of the radiator factors even if the factors have been previously computed and stored to still existing file. One might want to set this flag when changing the computational mesh, for example. Note that you should be able to recompute radiator factors also manually before start of the simulation simply by invoking the binary `Radiators`. `False` is the default, unless the factors don't exist.

`Update Radiator Factors` Logical
: The recomputation of the radiator factors is activated by setting the value of this flag to `True`. Note that you should be able to recompute radiator factors also manually before start of the simulation simply by invoking the binary `Radiators`. `False` is the default.

In some cases the geometry of the radiation boundaries change. This may require the recomputation of the view factors. For that purpose also dynamic computation of the factors is enabled and it is controlled by the keywords below. The view factors are also automatically computed if no view factors can be loaded from files.

`Update View Factors` Logical
: The recomputation of the view factors is activated by setting the value of this flag to `True`. Note that you should be able to recompute view factors also manually before start of the simulation simply by invoking the binary `ViewFactors`. `False` is the default.

`Minimum View Factor` Real
: This keyword determines the cut-off value under which the view factors are omitted. Neglecting small values will not only save memory but also will make the matrix used for solving the Gebhart factors less dense. This consequently will enable more efficient sparse matrix strategies in solving the Gebhart factors. The value for this parameter might be of the order 10e-8.

`View Factors Geometry Tolerance` Real
: The view factors take a lot of time to compute. Therefore during the iteration a test is performed to check whether the geometry has changed. If the relative maximum change in the coordinate values is less than the value given by this parameter the view factors are not recomputed and the old values are used.

`View Factors Fixed After Iterations` Integer
: Sometimes the iteration changes the geometry of the radiation boundaries as an unwanted side-effect. Then the geometry on the radiation boundary may be set fixed after some iterations. In practice this is done by adding suitable Dirichlet conditions in the boundary conditions.

`View Factors Fixed Tolerance` Real
: This keywords defines the coupled system tolerance for the heat equation after which the recomputation of view factors is omitted. Typically this should be defined by a geometry tolerance but if the temperature solver follows the changes in geometry this may be a good control as well.

`Viewfactor Symmetry x` Logical
: We may enforce view factors to be symmetric. The code for computing view factors creates the

full mesh internally but all other computations may then be done assuming symmetric geometry. Also keywords `Viewfactor Symmetry y` and `Viewfactor Symmetry z` exist correspondingly.

`Viewfactor Divide` Integer
: For axisymmetric view factor computation gives the number of divisions for each element. The default is 1.

`Viewfactor Combine Elements` Logical
: There may be a significant amount of saved time if in the axisymmetric view factor computation the elements that are aligned and share a common node are united. The shadowing loop will then only be performed over these macroelements.

`Viewfactor Number Of Rays` Integer
: In 2D and 3D cases the shadowing is resolved by casting rays between elements. This is the number of rays sent between elements when computing view factors. It is also used for "radiators" when resolving shadowing between boundary elements and the radiating sources. The default is 1.

`Viewfactor Area Tolerance` Real
: In 2D and 3D cases the shadowing is resolved by casting rays between elements. This setting gives a size of the boundary patch, which is not to be divided into smaller pieces even if shadowing is undetermined for the patch (the number of hits is larger than zero or fewer than number of rays). If the area tolerance is used stop the recursive division, the viewfactor is multiplied by $hits/maxrays$. This keyword is also used for "radiators" when resolving shadowing between boundary elements and the radiating sources. The default is 1.

`Viewfactor Factor Tolerance` Real
: In 2D and 3D cases the shadowing is resolved by casting rays between elements. This setting gives the maximum of the computed factor which is accepted for a boundary patch even if shadowing is undetermined for the patch (the number of hits is larger than zero or fewer than number of rays). shadowing between boundary elements and the radiating sources. The default is 1.

For radiation problems we may use either `Gebhart factors` or radiosity model. Gebhart factors results to better convergence of the nonlinear system but comes with a higher computational cost than the radiosity model. By default Gebhart factors are computed only once. They may however be updated if the emissivity is a function of temperature. Also there are some other keywords to control their computation.

`Update Gebhart Factors` Logical
: If the emissivities depend on the solution the Gebhart factors may need to be recomputed. This is activated by setting giving this flag value `True`. `False` is the default. Gebhart factors are computed fully internally in ElmerSolver.

`Minimum Gebhart Factor` Real
: The Gebhart factors make part of matrix dense. By neglecting the smallest Gebhart factors the matrix structure for the heat equation may become significantly sparser and thus the solution time may drop. The value for this parameter might also be of the order 10e-8.

`Implicit Gebhart Factor Fraction` Real
: In computing heat transfer problems with radiation in an implicit manner the matrix structure becomes partially filled. This affects the performance of the linear equation solvers and also increases the memory requirements. On the other hand explicit treatment of radiation slows down the convergence significantly. This keyword allows that the largest Gebhart factors are treated in an implicit manner whereas the smallest are treated explicitly. The value should lie in between zero (fully explicit) and one (fully implicit).

`Matrix Topology Fixed` Logical
: If the Gebhart factors change the matrix structure of the heat equation may also have to be changed unless this flag is set to `False`. Then all factors that do not combine with the matrix structure are omitted.

`Gebhart Factors Fixed After Iterations` `Integer`
: Sometimes the emissivity depends on temperature but recomputing it every time may be costly. By this keyword the recomputation may be limited to the given number of visits to the heat equation solver.

`View Factors Fixed Tolerance` `Real`
: This keywords defines the coupled system tolerance for the heat equation after which the recomputation of view factors is omitted. Typically this should be defined by a geometry tolerance but if the temperature solver follows the changes in geometry this may be a good control as well.

`Gebhart Factors Fixed Tolerance` `Real`
: This keywords defines the coupled system tolerance for the heat equation after which the recomputation of Gebhart factors is omitted. The temperature dependence of emissivity is typically not so strong that small temperature changes would result to a need to recompute the Gebhart factors as well.

The other model for considering radiation is to use radiosity vectors. These do not mess with the matrix structure and thereby enable the use of all linear solver strategies, usually with good success. There is also a spectral version available. Hence the radiosity models should often be the method of choice.

`Radiosity Model` `Logical`
: Use radiosity model instead the default one based on Gebhart factors.

`Spectral Model` `Logical`
: Spectral model is a variation of the radiosity model where heat radiation to different temperature intervals are treated separately. If this flag is set to `True`, the radiosity model is activated automatically.

`Spectral Dt` `Real`
: This is a parameter related to the spectral radiation model. It describes the width of temperature intervals. There is a cost of adding more intervals so probably an value of, say 20 `K`, could be suitable. It is desirable that within the interval the emissivity is not changing too much.

Both the Gebhart factors and the radiosity model require solution of linear systems. For Gebhart factors the number of linear systems is the same as number of radiating surface elements. These remain constant for the whole simulation unless the relevant part of the geometry changes and an update of the factors is needed. For constant-emissivity radiosity, the number of linear systems is one for each nonlinear iteration (two when using Newton linearization), while for the spectral model the number of linear systems per nonlinear iteration is the same as number of active temperature intervals, i.e. typically some tens.

Because these linear systems are solved often more times than the linear system related to the heat equation it is often important to optimize the speed of these computations. This is especially true when using the Gebhart factors model, where an matrix inverse is computed by solving the same linear system n times - direct solver, such as MUMPS or UMFPACK, is often faster than the iterative solvers, as the LU-decomposition can be saved and reused. For the radiosity (and spectral) model iterative solvers should remain faster. Most linear system keywords can basically be used here. Here are just few ones explained.

`radiation: Linear System Keyword`
: Sometimes the default routines for computing Gebhart factors are not suitable. For that purposes the user may specify any linear system strategy using namespace `radiation:` and the standard linear system keywords. The solver instance is created on the fly and the keywords are passed to it without the suffix.

`radiation: Linear System Solver` `String`
: If the radiation factors are solved from a sparse matrix equation also the type of solver may be selected. Often the iterative strategies (as opposed to direct solvers) provided better speed with lower memory consumption. This keyword may be used to choose between iterative or direct solvers.

`radiation: Linear System Symmetric` True
: Makes the computation of Gebhart factors symmetric. This can only be done if all emissivities are less than one. When the system is symmetric, there are more economical linear solvers available.

`radiation: Linear System Positive Definite` True
: If the linear system is made symmetric, it is also positive definite. Also this may be used by some linear solvers for better efficiency.

`Equation` eq id

The equation section is used to define a set of equations for a body or set of bodies.

`Heat Equation` Logical
: If set to `True`, solve the heat equation.

`Convection` String
: The type of convection to be used in the heat equation, one of: `None`, `Computed`, `Constant`.

`Phase Change Model` String
: One of: `None`, `Spatial 1`, `Spatial 2` and `Temporal`. Note that when solidification is modelled, the enthalpy-temperature- and viscosity-temperature-curves must be defined in the material section.

`Body Forces` bf id

The body force section may be used to give additional force terms for the equations. The following keywords are recognized by the base solver:

`Heat Source` Real
: A heat source h for the heat equation may be given with this keyword. Note that by default the heating is given per unit mass, not unit volume.

`Friction Heat` Logical
: Currently redundant keyword, the frictional heating h_f is automatically added.

`Joule Heat` Logical
: If set `True`, triggers use of the electromagnetic heating. This keywords accounts for the heating of many different solvers; electrostatics, magnetostatics, and induction equation.

`Smart Heater Control` Logical
: Sometimes the predescribed heat source does not lead to the desired temperature. Often the temperature is controlled by a feedback and therefore a similar heater control in the simulation may give more realistic results. This flag makes sets the smart heater control on for the given body force.

`Integral Heat Source` Real
: This keyword activates a normalization of the `Heat Source` so that the integral heating power is the desired objective.

`Temperature Lower Limit` Real
: The lower limit for temperature that is enforced iteratively when the soft limiters are applied.

`Temperature Upper Limit` Real
: The upper limit for temperature that is enforced iteratively when the soft limiters are applied.

There are four optional keywords related to the Pennes' Bioheat equation term that model the perfusion process.

`Perfusion Rate` Real
: The rate of the perfusion w. Activates the perfusion process.

`Perfusion Reference Temperature` Real
: Temperature T_b of the perfusion fluid.

`Perfusion Density` Real
: Density ρ_b of the perfusion fluid.

`Perfusion Heat Capacity` Real
: Heat capacity c_b of the perfusion fluid.

`Initial Condition` ic id
: The initial condition section may be used to set initial values for temperature.

`Temperature` Real

`Material` mat id
: The material section is used to give the material parameter values. The following material parameters may be effective when heat equation is solved.

`Density` Real
: The value of density is given with this keyword. The value may be constant, or variable. For the compressible flow, the density is computed internally, and this keyword has no effect.

`Enthalpy` Real
: Note that, when using the solidification modelling, an enthalpy-temperature curve must be given. The enthalpy is derived with respect to temperature to get the value of the effective heat capacity.

`Viscosity` Real
: Viscosity is needed if viscous heating is taken into account. When using the solidification modelling, a viscosity-temperature curve must be given. The viscosity must be set to high enough value in the temperature range for solid material to effectively set the velocity to zero.

`Heat Capacity` Real
: The value of heat capacity in constant pressure c_p is given with this keyword. The value may be constant, or variable. For the phase change model, this value is modified according to rules given in the theory section.

`Heat Conductivity` Real
: The value of heat conductivity k is given with this keyword. The value may be a constant or variable.

`Convection Velocity i` Real
: Convection velocity `i`$= 1, 2, 3$ for the constant convection model.

`Compressibility Model` Real
: This setting may be used to set the compressibility model for the flow simulations. Choices are `Incompressible` and `Perfect Gas`. If set to the latter there may be mechanical work performed by the heating. Then also the settings `Reference Pressure` and `Specific Heat Ratio` must also be given.

`Reference Pressure` Real
: With this keyword a reference level of pressure may be given.

`Specific Heat Ratio` Real
: The ratio of specific heats (in constant pressure versus in constant volume) may be given with this keyword. The default value of this setting is $5/3$, which is the appropriate value for monoatomic ideal gas.

`Emissivity` Real
: Emissivity of the radiating surface, required for radiation model is present. If the emissivity is not found in the radiating boundary only then will it be looked at the material properties of the parent elements. Often locating the emissivity here makes the case definition more simple.

`Transmissivity` Real
: For the diffuse gray radiation model also transmissivity of the surface may be provided. It gives the part of the energy that is lost as it passes through the wall. By default transmissivity is zero.

`Absorptivity` Real
: For radiosity models we may use different model for emissivity and absorptivity. Otherwise absorptivity is assumed to be the same as emissivity.

`Radiator Absorptivity` Real
: Radiators may have a different spectra not compatible with the theory of black body radiation. Then the temperature of the radiator does not result to the same absorptivity as temperature of a black body object. This keywords enables the user to give different absorptivity for the radiator spectra when using the spectral model.

`Boundary Condition` bc id
: The boundary condition section holds the parameter values for various boundary condition types. In heat equation we may set the temperature directly by Dirichlet boundary conditions or use different flux conditions for the temperature. The natural boundary condition of heat equation is zero flux condition.

`Temperature` Real

`Heat Flux BC` Logical
: Must be set to `True`, if heat flux boundary condition is present.

`Heat Flux` Real
: A user defined heat flux term.

`Heat Transfer Coefficient` Real
: Defines the parameter α in the heat flux boundary condition of the type

$$-k\frac{\partial T}{\partial n} = \alpha(T - T_{ext}).$$

`External Temperature` Real
: Defines the variable for ambient temperature T_{ext} in the previous equation.

`Radiation` String
: The type of radiation model for this boundary, one of: `None`, `Idealized`, `Diffuse Gray`.

`Radiation Boundary` Integer
: If there are many closures with radiation boundary conditions that do not see each other the view factors may be computed separately. This keyword is used to group the boundaries to independent sets. The default is one.

`Radiation Boundary Open` Logical
: The closures may be partially open. Then no normalization of the view factors is enforced. The missing part of the radiation angle is assumed to be ideal radiation. Therefore if this option is enforced also the parameter `External Temperature` must be given.

`Radiation External Temperature` Real
: In case the external temperature related to the heat transfer coefficient is different than that related to the radiation they cannot be given with the same keyword. For this purpose an alternative keyword is provided for radiation problems. This is used instead if it is present.

`Emissivity` Real
: Emissivity of the radiating surface, required for radiation model is present. If the emissivity is not found here it will be searched at the parent elements.

`Transmissivity` Real
: If the transmissivity is not found here it will be searched at the parent elements.

`Radiator BC` Logical
: This boundary is getting radiated by "radiators". The surface normal is determined similarly to diffuse gray radiation model (see "Radiation Target Body" -keyword). Default 1 "False".

`Radiation Target Body` Integer
: This flag may be used to set the direction of the outward pointing normal. This is used when computing view factors. If the value of emissivity is given in the `Material` section, then the normal is assumed to point outwards from the material having the property. If the value of emissivity is given in the `Boundary Condition` section, then the direction of normal is

ambiguous. This keyword may then be used to give the direction of the normal by specifying the material to which the normal points to. Value -1 means that the normal is pointing outwards to non-existing material (this is also the default). Hence, this keyword should be given on internal ambiguous boundaries or on external boundaries where we are radiating into the domain.

`Smart Heater Boundary` `Logical`
: If the smart heater is activated the point for monitoring the temperature is the point with maximum x-coordinate on the boundary where this keyword is set `True`. Alternatively the logical variable `Phase Change` is looked for.

`Smart Heater Temperature` `Real`
: The desired temperature for the smart heater system is set by this keyword. Alternatively the real variable `Melting Point` may be used.

Appendix: Internal Radiation Boundary Conditions

We describe the internal radiation boundary condition in this appendix. These are boundary boundary condition where the body sees itself and the conditions becomes much more complicated than with interaction of ideal space.

Diffuse Gray Radiation Boundary Conditions using Gebhart Factors

On outer boundaries of a solid or liquid body, where the temperature is not fixed, the heat flux must be specified

$$-\kappa\frac{\partial T}{\partial n}=q. \tag{1.21}$$

The flux q may depend on temperatures of other boundaries of the system, if radiative exchange of heat is present. The model used here is the diffuse gray radiation model, which means that a surface radiates energy to every direction with equal intensity, and that the intensity is also independent of the wavelength of the radiation. Using these assumptions we may write the heat flux for a given point on a surface, resulting from the radiation, as

$$q(\vec{x},T)=\sigma\left(\varepsilon(\vec{x})T^4(\vec{x})-\int\varepsilon(\vec{y})G(\vec{y},\vec{x})T^4(\vec{y})dA_y\right). \tag{1.22}$$

where σ is the Stefan-Boltzmann constant, $\varepsilon(\vec{x})$ is surface emissivity, and the integral is over all the surfaces of the model. The function G is the so called Gebhart factor. The Gebhart factor may be computed using the equation

$$G(\vec{y},\vec{x})-\int F(\vec{y},\vec{x})(1-\varepsilon(\vec{z})G(\vec{z},\vec{x})dAz=F(\vec{y},\vec{x})\varepsilon(\vec{x}). \tag{1.23}$$

where the function F is defined as follows

$$F(\vec{x_i},\vec{x_j})=\frac{\cos\phi_i\cos\phi_j}{\pi r^2}H_{ij}. \tag{1.24}$$

The angles are between the normals of the surfaces and the line connecting the points and H_{ij} the visibility. The function F is the so called view factor.

In practice, the surfaces of the model are divided to, say, total of N finite parts, the boundary elements. This discretization is then used to compute the view factors and the Gebhart factor for these parts. The Gebhart factors may be computed from the equation

$$G=F(I-(I-E)F)^{-1}E, \tag{1.25}$$

where F is the view fa tor matrix, and E a diagonal matrix of surface emissivities. Writing the discrete version of the boundary condition we get

$$-\kappa_k\frac{\partial T_k}{\partial n_k}=\sigma\varepsilon_k\left(T_k^4-\frac{1}{A_k\varepsilon_k}\sum_1^N G_{ik}\varepsilon_i T_i^4 A_i\right). \tag{1.26}$$

This equation is still nonlinear, and has to be linearized in order to solve the equation on a computer. To illustrate the point here, we will first show how to linearize the idealized radiation boundary condition instead of the diffuse gray radiation boundary condition. The idealized boundary condition reads

$$-\kappa\frac{\partial T}{\partial n}=\sigma\varepsilon(T^4-T_{ext}^4). \tag{1.27}$$

We may write this somewhat differently as in

$$-\kappa\frac{\partial T}{\partial n}=\sigma\varepsilon(T^3+T^2T_{ext}+TT_{ext}^2+T_{ext}^3)(T-T_{ext}). \tag{1.28}$$

If we now take $T=\mathcal{T}$, the temperature from the previous iteration, in first of the product terms, we have an expression linear in T. This is somewhat strange linearization, not quite a Newton method, but effective in being more insensitive to the initial guess of the temperature field than the Newton method, while still having a much better convergence rate than the fixed point method. The Newton linearization is also easy to produce by writing down the Taylor series expansion of the equation around T and retaining only the constant terms and the terms linear in T:

$$-\kappa\frac{\partial T}{\partial n}=\sigma\varepsilon(4\mathcal{T}^3T-3\mathcal{T}^4-T_{ext}^4). \tag{1.29}$$

This linearization has a better convergence rate than the previous one, but with the expense of being more sensitive to the initial guess. These two linearizations may be used in succession, first taking a few steps with the former linearization, and then switch to the latter when the convergence is on the way.

The linearization of the diffuse gray radiation boundary condition is very similar. We may, for example, use the temperatures from the previous iteration to compute an "external" temperature

$$T_{ext}^4\approx\frac{1}{A_k\varepsilon_k}\sum_1^N G_{ik}\varepsilon_i\mathcal{T}_i^4A_i \tag{1.30}$$

and use either of the previous equations. The convergence rates of these methods are of first order, and usually they are used only to provide an initial guess for the full Newton iteration. The full Newton method may be written as

$$-\kappa_k\frac{\partial T_k}{\partial n_k}=\sigma\varepsilon_k\left[4\mathcal{T}^3T-3\mathcal{T}^4-\sum_1^N G_{ik}\varepsilon_i(4\mathcal{T}_i^3T_i-3\mathcal{T}_i^4)A_i\right]. \tag{1.31}$$

The Gebhart factors computed in the way presented here are assumed to be constant within elements, while the temperatures from the previous iteration are known at the element nodal points. Terms of type $\alpha G_{ik}T_k^n$ are to be integrated over the element i. This is done by a one point integration rule and requires values of the powers of the temperature held at the element center. To conserve the heat flux, the powers of the temperature held will first have to be computed on nodal points, and only after that interpolated to the element center, rather than first interpolating the temperatures and computing the powers afterwards.

Diffuse Radiation Boundary Condition Using Radiosity

When the radiation heat transfer among the surface elements depends on the spectral properties of the materials the Gebhart factor approach becomes cumbersome. Also, otherwise the Gebhart factors suffer from huge computational cost as each row in Gebhart factor computation requires solution of a linear system where all the surface elements participate.

An alternative approach is to use a concept called "radiosity" which is the total radiative power leaving the surface. For our purposes we rather end up with "irradiance" i.e. the radiation coming to the surface but the literature is more developed to treat radiosity and hence we use that too here. We follow the Ch. 5 in [1].

Let us study the energy balance for a boundary element i. The emissive power is assumed to be M_i, the incoming irradiance E_i then radiosity J_i is defined as

$$J_i=M_i+r_iE_i \tag{1.32}$$

Assuming that material parameters ε_i, α_i and r_i are constant radiosity vector J may be solved from a linear system (equation (5.166) in the reference with $r_i=1-\varepsilon_i$)

$$\sum_j^N[\delta_{ij}-r_iF_{ij}]J_j=\varepsilon_i\sigma T_i^4. \tag{1.33}$$

or for our purposes in more convenient matrix form

$$(rF-1)J=-\varepsilon\sigma T^4. \tag{1.34}$$

After solution of radiosity J we could compute irradiance from $E = FJ$ and absorbed flux from αFJ which after some manupulation yields

$$q_a = \sigma \frac{\varepsilon\alpha}{1-\alpha} T^4 + \frac{\alpha}{1-\alpha} J \tag{1.35}$$

where we have defined the reflectivity using absorptivity, $r = 1 - \alpha$. The emitted power is still

$$q_e = -\sigma\varepsilon T^4 \tag{1.36}$$

Splitting the fluxes into two is convenient since q_e is the standard radiative flux to open space depending only on the surface temperature while q_a is the absorbed radiation that may depend on the spectral nature of the radiation.

To boost convergence we also compute approximate estimate for dq_a/dT that is used for Newton's linearization. The linearization only depends on the temperature of the element itself and hence the convergence rate of the nonlinear system is on par with the Gebhart factor method.

Radiosity for Non-Gray Boundaries

When the emissivity and absorptivity depend on temperature we need to leave the condition of grayness i.e. that different photons interact in the same manner. The different wavelengths could be treated separately. However, here we assume that each radiating surface emits black body radiation associated to its own temperature. Also, we split the temperature into discrte temperatures $T_k = k\Delta T$, $k = 1, 2, 3, \ldots$. Each source is then weighted by

$$q_k(T) = \max(0, 1 - \text{abs}(T/\Delta T - k)). \tag{1.37}$$

Now we sum up the contribution to q_a one temperature at a time,

$$\left[r(T_k)F - 1\right] J_k = -q_k(T)\varepsilon\sigma T^4. \tag{1.38}$$

and

$$q_a = \sum_k \left[q_k(T)\sigma \frac{\varepsilon\alpha}{1-\alpha} T^4 + \frac{\alpha}{1-\alpha} J_k \right] \tag{1.39}$$

In summation of the contribution of different temperatures the parameters α and ε must be evaluated at the temperature of the interval.

Computing the View Factors in 3D

A view factor between two surface patches (boundary elements) of a geometrical model (mesh) is defined as

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i}{\cos\phi_j} \pi r^2 H_{ij} dA_j dA_i, \tag{1.40}$$

where A_i and A_j are the areas of the patches i and j respectively, ϕ_i and ϕ_j the angles between the normals of the surfaces at given points and the line connecting the points and r the distance between the two points. The most problematic thing about the definition of the view factor integral is the evaluation of the visibility function H_{ij} . The value of the function H_{ij} is one, if there are no obstacles between the two points, and zero otherwise. The view factors are computed for every pair of surface patches in the model, so that we eventually get a $N \times N$ matrix F.

Noting that

$$A_i F_{ij} = A_j F_{ji} \tag{1.41}$$

allows only one half of the factors to be actually computed (or alternatively using the redundant information to make the view factor computation more accurate).

In three dimensional cases, the method the Elmer view factor computation program uses to evaluate the integral 1.40 is direct numerical integration combined with an adaptive subdivision procedure, where the elements are subdivided to smaller and smaller pieces, until user specified criteria are fulfilled. An area weighted average of the factors of the parts of the subdivided patches is then computed. The subdivision of patches is stored in hierarchical storage, so that it doesn't have to be repeated for every pair of patches. This subdivision is also used by the ray tracer, which is used for computing the visibility function.

First of the criteria to terminate the integration subdivision procedure is fulfilled when the areas of the subdivided elements are small enough. This criterion prevents the subdivision to continue forever in areas where the integrand is actually discontinuous (any sharp corner, for example). The second criterion to terminate the subdivision is fulfilled when the view factors themselves have become small enough. This criterion prevents unnecessary subdivision of areas which are either far away from each other or oriented so that the projections of their areas to each other are small.

The function H_{ij} is evaluated by sending a number of rays from one of the patches to another, and checking whether they hit some other patch of the model before hitting the target. This is potentially a very time consuming task, as in the worst case, for every pair of boundary elements, all the other boundary elements of the model must be checked for obstructing the view between the pair. The ray tracer module is designed to reduce the time spent in computing the intersections, by making a hierarchical volume subdivision of the model volume and assigning each element of the model to some of these subdivided volumes in a fixed hierarchy level. The ray tracer is then able to check whether the ray will hit the bounding boxes of the volumes, then the second level of bounding boxes, etc. before eventually going through the elements assigned to a volume in the last hierarchy level. There might be only a couple of elements assigned to that volume.

When the visibility between two surface patches is in doubt (some of the rays are obstructed, some not) an additional subdivision cycle is made. When the subdivision is eventually terminated, H_{ij} is computed as $H_{ij} = 1 - n_b/n$ where n_b is the number of the rays blocked and n the total number of rays sent.

Normalizing the View Factors

The view factors computed by, for example, the kind of procedure described in the previous section, are only approximate. For the radiative exchange of heat in finite element computations, it is sometimes important to guarantee that energy is conserved. The energy conservation requires, that in a closed system

$$\sum_j^N F_{ij} = 1. \tag{1.42}$$

Together with the symmetry equation this condition may be used to normalize the factors so that energy will be conserved. Note that this condition only holds for fully closed surfaces.

Lets begin with defining a symmetric matrix

$$S_{ij} = \frac{1}{2}(A_i F_{ij} + A_j F_{ji}). \tag{1.43}$$

This matrix we can relatively easily scale so that the row sums (and as we should preserve the symmetry, also column sums) of the matrix are equal to corresponding surface patch areas, by solving a set of equations:

$$\sum_j^N d_i d_j S_{ij} = A_i, \quad i = 1, \ldots, N. \tag{1.44}$$

where the d_i:s are scale factors. The equation set is quadratic in d_i:s and may be solved by Newton iteration. The resulting linear system is diagonally dominant and very stable, so there should be no problems in solving the equation. After the scaling is known we may compute the normalized factors by

$$F_{ij} = d_i d_j S_{ij}/A_i \tag{1.45}$$

Bibliography

[1] H. D. Baehr and K. Stephan. *Heat and Mass Transfer.* Springer Berlin, Heidelberg, 1998.

Model 2

Navier-Stokes Equations

Module name: FlowSolve
Module subroutines: FlowSolver
Module authors: Juha Ruokolainen
Document authors: Juha Ruokolainen, Peter Råback

2.1 Introduction

In solid and liquid materials heat transfer and viscous fluid flow are governed by heat and Navier-Stokes equations, which can be derived from the basic principles of conservation of mass, momentum and energy. Fluid can be either Newtonian or non-Newtonian. In the latter case the consideration in Elmer is limited to purely viscous behaviour with the power-law model.

In the following we present the governing equations of fluid flow, heat transfer and stresses in elastic material applied in Elmer. Also the most usual boundary conditions applied in computations are described.

2.2 Theory

The momentum and continuity equations can be written as

$$\rho\left(\frac{\partial\vec{u}}{\partial t}+(\vec{u}\cdot\nabla)\vec{u}\right)-\nabla\cdot\overline{\overline{\sigma}}=\rho\vec{f}, \tag{2.1}$$

and

$$\left(\frac{\partial\rho}{\partial t}+(\vec{u}\cdot\nabla)\rho\right)+\rho(\nabla\cdot\vec{u})=0, \tag{2.2}$$

where $\overline{\overline{\sigma}}$ is the stress tensor. For Newtonian fluids

$$\overline{\overline{\sigma}}=2\mu\overline{\overline{\varepsilon}}-\frac{2}{3}\mu(\nabla\cdot\vec{u})\overline{\overline{I}}-p\overline{\overline{I}}, \tag{2.3}$$

where μ is the viscosity, p is the pressure, $\overline{\overline{I}}$ the unit tensor and $\overline{\overline{\varepsilon}}$ the linearized strain rate tensor, i.e.

$$\varepsilon_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right). \tag{2.4}$$

The density of an ideal gas depends on the pressure and temperature through the equation of state

$$\rho=\frac{p}{RT}, \tag{2.5}$$

where R is the gas constant:

$$R = \frac{\gamma - 1}{\gamma} c_p. \tag{2.6}$$

The specific heat ratio γ is defined as

$$\gamma = \frac{c_p}{c_v}, \tag{2.7}$$

where c_p and c_v are the heat capacities in constant pressure and volume, respectively. The value of γ depends solely on the internal molecular properties of the gas.

An incompressible flow is characterized by the condition ρ=constant, from which it follows that

$$\nabla \cdot \vec{u} = 0. \tag{2.8}$$

Enforcing the constraint (2.8) in (2.1), (2.2) and (2.3), the equations reduce to the Navier-Stokes equations

$$\begin{aligned} \rho\left(\frac{\partial \vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\vec{u}\right) - \nabla\cdot(2\mu\bar{\bar{\varepsilon}}) + \nabla p &= \rho\vec{f}, & (2.9)\\ \nabla\cdot\vec{u} &= 0. & (2.10)\end{aligned}$$

Compressible flows are modelled by the equations (2.1)-(2.7). Then, it is possible to replace the state equation (2.5) by

$$\rho = \frac{1}{c^2} p, \tag{2.11}$$

where $c = c(p, T, \dots)$ is the speed of sound. The equation (2.11) can be used with liquid materials as well.

Most commonly the term $\rho\vec{f}$ represents a force due to gravity, in which case the vector $\vec{f}$ is the gravitational acceleration. It can also represent, for instance, the Lorentz force when magnetohydrodynamic effects are present.

For isothermal flows the equations (2.9) and (2.10) describe the system in full. For thermal flows also the heat equation needs to be solved.

For thermal incompressible fluid flows we assume that the Boussinesq approximation is valid. This means that the density of the fluid is constant except in the body force term where the density depends linearly on temperature through the equation

$$\rho = \rho_0(1 - \beta(T - T_0)), \tag{2.12}$$

where β is the volume expansion coefficient and the subscript 0 refers to a reference state. Assuming that the gravitational acceleration $\vec{g}$ is the only external force, then the force $\rho_0\vec{g}(1 - \beta(T - T_0))$ is caused in the fluid by temperature variations. This phenomenon is called Grashof convection or natural convection.

One can choose between transient and steady state analysis. In transient analysis one has to set, besides boundary conditions, also initial values for the unknown variables.

2.2.1 Boundary Conditions

For the Navier-Stokes equation one can apply boundary conditions for velocity components or the tangential or normal stresses may be defined.

In 2D or axisymmetric cases the Dirichlet boundary condition for velocity component u_i is simply

$$u_i = u_i^b. \tag{2.13}$$

A value u_i^b can be constant or a function of time, position or other variables. In cylindrical cases the Dirichlet boundary condition for angular velocity u^θ is

$$u^\theta = \omega, \tag{2.14}$$

where ω is the rotation rate.

In axisymmetric geometries one has to set $u_r = 0$ and $\partial u_z/\partial r = 0$ on the symmetry axis.

If there is no flow across the surface, then

$$\vec{u} \cdot \vec{n} = 0 \tag{2.15}$$

where $\vec{n}$ is the outward unit normal to the boundary.

Surface stresses can be divided into normal and tangential stresses. Normal stress is usually written in the form

$$\sigma_n = \frac{\gamma}{R} - p_a \tag{2.16}$$

where γ is the surface tension coefficient, R the mean curvature and p_a the atmospheric (or external) pressure. Tangential stress has the form

$$\vec{\sigma}_\tau = \nabla_s \gamma, \tag{2.17}$$

where ∇_s is the surface gradient operator.

The coefficient γ is a thermophysical property depending on the temperature. Temperature differences on the surface influence the transport of momentum and heat near the surface. This phenomenon is called Marangoni convection or thermocapillary convection. The temperature dependence of the surface tension coefficient can be approximated by a linear relation:

$$\gamma = \gamma_0(1 - \vartheta(T - T_0)), \tag{2.18}$$

where ϑ is the temperature coefficient of the surface tension and the subscript 0 refers to a reference state. If a Boussinesq hypothesis is made, i.e., the surface tension coefficient is constant except in (2.17) due to (2.18), the boundary condition for tangential stress becomes

$$\vec{\sigma}_\tau = -\vartheta \gamma_0 \nabla_s T. \tag{2.19}$$

In equation (2.16) it holds then that $\gamma = \gamma_0$. The linear temperature dependence of the surface tension coefficient is naturally only one way to present the dependence. In fact, the coefficient γ can be any user defined function in Elmer. One may also give the force vector on a boundary directly as in

$$\overline{\overline{\sigma}} \cdot \vec{n} = \vec{g}. \tag{2.20}$$

2.2.2 Linearization

As is well known, the convective transport term of the Navier-Stokes equations and the heat equation is a source of both physical and numerical instability. The numerical instability must be compensated somehow in order to solve the equations on a computer. For this reason the so called stabilized finite element method ([2],[1]) is used in Elmer to discretize these equations.

The convection term of the Navier-Stokes equations is nonlinear and has to be linearized for computer solution. There are two linearizations of the convection term in Elmer:

$$(\vec{u} \cdot \nabla)\vec{u} \approx (\vec{\mathcal{U}} \cdot \nabla)\vec{u} \tag{2.21}$$

and

$$(\vec{u} \cdot \nabla)\vec{u} \approx (\vec{\mathcal{U}} \cdot \nabla)\vec{u} + (\vec{u} \cdot \nabla)\vec{\mathcal{U}} - (\vec{\mathcal{U}} \cdot \nabla)\vec{\mathcal{U}}, \tag{2.22}$$

where $\vec{\mathcal{U}}$ is the velocity vector from the previous iteration. The first of the methods is called Picard iteration or the method of the fixed point, while the latter is called Newton iteration. The convergence rate of the Picard iteration is of first order, and the convergence might at times be very slow. The convergence rate of the Newton method is of second order, but to successfully use this method, a good initial guess for velocity and pressure fields is required. The solution to this problem is to first take a couple of Picard iterations, and switch to Newton iteration after the convergence has begun.

2.2.3 Arbitrary Lagrangian-Eulerian (ALE) coordinates

For problems involving deformations the transient Navier-Stokes equation must be solved using Arbitrary Lagrangian-Eulerian (ALE) frame of reference. Assume that the mesh velocity during the nonlinear iteration is $\vec{c}$. Then the convective term yields

$$((\vec{u}-\vec{c})\cdot\nabla)\,\vec{u} \approx ((\vec{\mathcal{U}}-\vec{c})\cdot\nabla)\vec{u}. \tag{2.23}$$

This results naturally to Picard iteration. For Newton iteration the additional two terms remains the same since the mesh velocities in there cancel each other.

2.2.4 Non-Newtonian Material Models

There are several non-Newtonian material models. All are functions of the strainrate $\dot{\gamma}$. The simple power law model has a problematic behavior at low shear rates. The more complicated models provide a smooth transition from low to high shearrates.

Power law

$$\eta = \begin{cases} \eta_0\dot{\gamma}^{n-1} & \text{if } \dot{\gamma} > \dot{\gamma}_0, \\ \eta_0\dot{\gamma}_0^{n-1} & \text{if } \dot{\gamma} \leq \dot{\gamma}_0. \end{cases} \tag{2.24}$$

where η_∞ is constant, $\dot{\gamma}_0$ is the critical shear rate, and n is the viscosity exponent.

Carreau-Yasuda

$$\eta = \eta_\infty + \Delta\eta\,(1+(c\dot{\gamma})^y)^{\frac{n-1}{y}}\ , \tag{2.25}$$

where η_∞ is the high shearrate viscosity $\dot{\gamma}\to\infty$ provided that $n<1$. For shearrates approaching zero the viscosity is $\eta_0 = \eta_\infty + \Delta\eta$. $\Delta\eta$ is thus the maximum viscosity difference between low and high shearrate. This model recovers the plain Carreau model when the Yasuda exponent $y=2$.

Cross

$$\eta = \eta_\infty + \frac{\Delta\eta}{1+c\dot{\gamma}^n}, \tag{2.26}$$

where again η_∞ is the high shearrate viscosity.

Powell-Eyring

$$\eta = \eta_\infty + \Delta\eta\frac{\text{asinh}(c\dot{\gamma})}{c\dot{\gamma}}. \tag{2.27}$$

All the viscosity models can be made temperature dependent. The current choice is to multiply the suggested viscosity with a factor $\exp(d(1/(T_o+T)-1/T_r))$, where d is the exponential factor, T_o is temperature offset (to allow using of Celcius), and T_r the reference temperature for which the factor becomes one.

2.2.5 Flow in Porous Media

A simple porous media model is provided in the Navier-Stokes solver. It utilizes the Darcy's law that states that the flow resistance is proportional to the velocity and thus the modified momentum equation reads

$$\rho\left(\frac{\partial\vec{u}}{\partial t}+(\vec{u}\cdot\nabla)\vec{u}\right) - \nabla\cdot\overline{\overline{\sigma}} + r\vec{u} = \rho\vec{f}, \tag{2.28}$$

where r is the porous resistivity which may also be an orthotropic tensor. Usually the given parameter is permeability which is the inverse of the resistivity as defined here. No other features of the porous media flow is taken into consideration. Note that for large value of r only the bubble stabilization is found to work.

2.2.6 Rotating coordinates

In rotating coordinate system around origin one may define the angular velocity vector, $\vec{\Omega}$. The rotation introduces additional forces that may be evaluated from the following

$$\frac{d\vec{u}_{inertial}}{dt} = \frac{d\vec{u}_{rotating}}{dt} + 2\vec{\Omega} \times \vec{u}_{rotating} + \vec{\Omega} \times (\vec{\Omega} \times \vec{x}). \tag{2.29}$$

In numerical implementations the following Lagrange's formula is used

$$\vec{\Omega} \times (\vec{\Omega} \times \vec{x}) = (\vec{\Omega} \cdot \vec{x})\vec{\Omega} - (\vec{\Omega} \cdot \vec{\Omega})\vec{x}. \tag{2.30}$$

which results to the following form of the Navier-Stokes equation in rotating coordinates

$$\rho \left(\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} \right) - \nabla \cdot \overline{\overline{\sigma}} + 2\rho\vec{\Omega} \times \vec{u} = \rho(\vec{\Omega} \cdot \vec{\Omega})\vec{x} - \rho(\vec{\Omega} \cdot \vec{x})\vec{\Omega} + \rho\vec{f}, \tag{2.31}$$

It should be noted that now also the boundary conditions need to be given in the rotational coordinate system.

2.2.7 Coupling to Electric Fields

In electrokinetics the fluid may have charges that are coupled to external electric fields. This results to an external force that is of the form

$$\vec{f_e} = -\rho_e \nabla\phi, \tag{2.32}$$

where ρ_e is the charge density and ϕ is the external electric field. The charge density may also be a variable. More specifically this force may be used to couple the Navier-Stokes equation to the Poisson-Boltzmann equation describing the charge distribution in electric doubly layers. Also other types of forces that are proportional to the gradient of the field may be considered.

2.2.8 Coupling to Magnetic Fields

If the fluid has free charges it may couple with an magnetic field. The magnetic field induced force term for the flow momentum equations is defined as

$$\vec{f_m} = \vec{J} \times \vec{B}, \tag{2.33}$$

Here $\vec{B}$ and $\vec{E}$ are the magnetic and electric fields, respectively. The current density $\vec{J}$ is defined as

$$\vec{J} = \sigma(\vec{E} + \vec{u} \times \vec{B}). \tag{2.34}$$

2.3 Keywords

`Constants`

`Gravity Size 4 Real [x y z abs]`
The above statement gives a real vector whose length is four. In this case the first three components give the direction vector of the gravity and the fourth component gives its intensity.

`Solver solver id`
Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation String [Navier-Stokes]`
The name of the equation. If it is `Navier-Stokes` then an old logic will define the following keyword.

`Procedure File "FlowSolve" "FlowSolver"`
The name of the procedure for the Navier-Stokes equation. This should be given as stated here.

`Variable` `String Flow Solution[Velocity:3 Pressure:1]`
This is the default name of velocity field in 3D, for 2D replace modify the number of velocity components. User could give this mainly if needing several flow fields within one simulation.

`Flow Model` `String [Full][No convection][Stokes]`
Flow model to be used. The default is to include both convection and time derivative terms in the model. The "No convection" model switches off the convection terms, and the "Stokes" model both the convection terms and the (explicit) time derivative terms.

`Nonlinear System Convergence Tolerance` `Real`
this keyword gives a criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations is small enough

$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Max Iterations` `Integer`
The maximum number of nonlinear iterations the solver is allowed to do.

`Nonlinear System Newton After Iterations` `Integer`
Change the nonlinear solver type to Newton iteration after a number of Picard iterations have been performed. If a given convergence tolerance between two iterations is met before the iteration count is met, it will switch the iteration type instead.

`Nonlinear System Newton After Tolerance` `Real`
Change the nonlinear solver type to Newton iteration, if the relative change of the norm of the field variable meets a tolerance criterion:

$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Relaxation Factor` `Real`
Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achieve convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:

$$u_i^{'} = \lambda u_i + (1 - \lambda) u_{i-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

`Steady State Convergence Tolerance` `Real`
With this keyword a equation specific steady state or coupled system convergence tolerance is given. All the active equation solvers must meet their own tolerances before the whole system is deemed converged. The tolerance criterion is:

$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$

where ϵ is the value given with this keyword.

`Stabilize` `Logical`
If this flag is set true the solver will use stabilized finite element method when solving the Navier-Stokes equations. Usually stabilization of the equations must be done in order to successfully solve the equations. If solving for the compressible Navier-Stokes equations, a bubble function formulation is used instead of the stabilized formulation regardless of the setting of this keyword. Also for the incompressible Navier-Stokes equations, the bubbles may be selected by setting this flag to `False`.

`Div Discretization` `Logical`
In the case of incompressible flow using the this form of discretization of the equation may lead to more stable discretization when the Reynolds number increases.

`Gradp Discretization` `Logical`
: Whit this form of discretization pressure Dirichlet boundary conditions can be used (and pressure level must be fixed by such a condition). Also the mass flux is available as a natural boundary condition.

`Equation` `eq id`
: The equation section is used to define a set of equations for a body or set of bodies:

`Navier-Stokes` `Logical`
: if set to `True`, solve the Navier-Stokes equations.

`Magnetic Induction` `Logical`
: If set to `True`, solve the magnetic induction equation along with the Navier-Stokes equations.

`Convection` `String [None, Computed, Constant]`
: The convection type to be used in the heat equation, one of: `None`, `Computed`, `Constant`. The second choice is used for thermal flows.

`Body Force` `bf id`
: The body force section may be used to give additional force terms for the equations.

`Boussinesq` `Logical`
: If set true, sets the Boussinesq model on.

`Flow BodyForce i` `Real`
: May be used to give additional body force for the flow momentum equations, `i=1,2,3`.

`Lorentz Force` `Logical`
: If set true, triggers the magnetic field force for the flow momentum equations.

`Potential Force` `Logical`
: If this is set true the force used for the electrostatic coupling is activated.

`Potential Field` `Real`
: The field to which gradient the external force is proportional to. For example the electrostatic field.

`Potential Coefficient` `Real`
: The coefficient that multiplies the gradient term. For example, the charge density.

`Angular Velocity` `Real`
: The angular velocity $\vec{\Omega}$ used for rotating coordinate systems. The size is always expected to be three.

`Initial Condition` `ic id`
: The initial condition section may be used to set initial values for the field variables. The following variables are active:

`Pressure` `Real`

`Velocity i` `Real`
: For each velocity component `i`$= 1, 2, 3$.

`Kinetic Energy` `Real`
: For the k-ε turbulence model.

`Kinetic Energy Dissipation` `Real`

`Material` `mat id`
: The material section is used to give the material parameter values. The following material parameters may be set in Navier-Stokes equation.

`Density` `Real` The value of density is given with this keyword. The value may be constant, or variable. For the of compressible flow, the density is computed internally, and this keyword has no effect.

`Viscosity` `Real`
: The relationship between stress and strain velocity. When using the solidification modelling, a viscosity-temperature curve must be given. The viscosity must be set to high enough value in the temperature range for solid material to effectively set the velocity to zero.

`Reference Temperature` `Real`
: This is the reference temperature for the Boussinesq model of temperature dependence of density.

`Heat Expansion Coefficient` `real`
: For the Boussinesq model the heat expansion coefficient must be given with this keyword. Default is 0.0.

`Applied Magnetic Field i` `Real`
: An applied magnetic field may be given with these keywords with `i=1,2,3`.

`Compressibility Model` `String`
: This setting may be used to set the compressibility model for the flow simulations. Currently the setting may be set to either `Incompressible`, `Perfect Gas` and `ArtificialCompressible`. If perfect gas model is chosen the settings `Reference Pressure` and `Specific Heat Ratio` must also be given. The artificial compressibility model may be used to boost convergence in fluid-structure-interaction cases. The default value of this setting is `Incompressible`.

`Reference Pressure` `Real`
: with this keyword a reference level of pressure may be given. This setting applies only if the `Compressibility Model` is set to the value `Perfect Gas`.

`Specific Heat Ratio` `Real`
: The ratio of specific heats (in constant pressure versus in constant volume) may be given with this keyword. This setting applies only if the `Compressibility Model` is set to value `Perfect Gas`. The default value of this setting is $5/3$, which is the appropriate value for monoatomic ideal gas.

For the k-ε turbulence model the model parameters may also be given in the material section using the following keywords

`KE SigmaK` `Real [1.0]`

`KE SigmaE` `Real [1.3]`

`KE C1` `Real [1.44]`

`KE C2` `Real [1.92]`

`KE Cmu` `Real [0.09]`

Non-Newtonian material laws are also defined in material section. For the power law the constant coefficient is given by the keyword `Viscosity`.

`Viscosity Model` `String`
: The choices are `power law`, `carreau`, `cross`, `powell eyring` and `thermal carreau`. If none is given the fluid is treated as Newtonian.

`Viscosity Exponent` `Real`
: Parameter n in the models power law, Carreau, Cross

`Viscosity Difference` `Real`
: Difference $\Delta\eta$ between high and low shearrate viscosities. Applicable to Carreau, Cross and Powell-Eyring models.

`Viscosity Transition` `Real`
: Parameter c in the Carreau, Cross and Powell-Eyring models.

`Critical Shear Rate` `Real [0.0]`
: Optional parameter $\dot{\gamma}_0$ in power law viscosity model.

`Nominal Shear Rate` `Real [0.0]`
: Optional parameter in the power law viscosity model that gives the shearrate that returns the plain Newtonian viscosity.

`Yasuda Exponent` `Real`
: Optional parameter y in Carreau model. The default is 2. If activated the model is the more generic Yasuda-Carreau model.

`Viscosity Temp Offset` `Real`
: Parameter T_o in the thermal viscosity dependence. When using Celsius instead of Kelvin this would be 273.15, for example.

`Viscosity Temp Ref` `Real`
: Parameter T_r in the thermal viscosity dependence. This should be set so that unity factor is obtained when $T_r = T_o + T$.

`Viscosity Temp Exp` `Real`
: Exponential parameter d in the thermal viscosity dependence.

Porosity is defined by the material properties

`Porous Media` `Logical`
: If this keyword is set `True` then the porous model will be active in the material.

`Porous Resistance` `Real`
: This keyword may give a constant resistance or also an orthotropic resistance where the resistance of each velocity component is given separately.

`Boundary Condition` `bc id`
: The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The one related to Navier-Stokes equation are

`Velocity i` `Real`
: Dirichlet boundary condition for each velocity component `i`$= 1, 2, 3$.

`Pressure` `Real`
: Absolute pressure.

`Normal-Tangential Velocity` `Real`
: The Dirichlet conditions for the vector variables may be given in normal-tangential coordinate system instead of the coordinate axis directed system using the keywords

`Flow Force BC` `Logical`
: Set to `true`, if there is a force boundary condition for the Navier-Stokes equations.

`Surface Tension Expansion Coefficient` `Real`
: Triggers a tangential stress boundary condition to be used. If the keyword `Surface Tension Expansion Coefficient` is given, a linear dependence of the surface tension coefficient on the temperature is assumed. Note that this boundary condition is the tangential derivative of the surface tension coefficient

`Surface Tension Coefficient` `Real`
: Triggers the same physical model as the previous one except no linearity is assumed. The value is assumed to hold the dependence explicitly.

`External Pressure` `Real`
: A pressure boundary condition directed normal to the surface.

`Pressure i` `Real`
: A pressure force in the given direction `i`$= 1, 2, 3$.

`Free Surface` `Logical`
: Specifies a free surface.

`Free Moving` Logical
: Specifies whether the regeneration of mesh is free to move the nodes of a given boundary when remeshing after moving the free surface nodal points. The default is that the boundary nodes are fixed.

The k-ε turbulence model also has its own set of boundary condition keywords (in addition to the Dirichlet settings):

`Wall Law` Logical
: The flag activates the (Reichardts) law of the wall for the boundary specified. the default is 9.0.

`Boundary Layer Thickness` Real
: The distance from the boundary node of the meshed domain to the physical wall.

Bibliography

[1] L.P. Franca and S.L. Frey. *Computer methods in Applied Mechanics and Engineering*, 99:209–233, 1992.

[2] L.P. Franca, S.L. Frey, and T.J.R. Hughes. *Computer methods in Applied Mechanics and Engineering*, 95:253–276, 1992.

Model 3

Advection-Diffusion Equation

Module name: AdvectionDiffusion
Module subroutines: AdvectionDiffusionSolver
Module authors: Juha Ruokolainen, Ville Savolainen, Antti Pursula
Document authors: Ville Savolainen, Antti Pursula

3.1 Introduction

Advection-diffusion equation (sometimes called diffusion-convection equation) describes the transport of a scalar quantity or a chemical species by convection and diffusion. The difference in the nomenclature usually indicates that an advected quantity does not have an effect on the velocity field of the total fluid flow but a convected quantity has. Advection-diffusion equation is derived from the principle of mass conservation of each species in the fluid mixture. Advection-diffusion equation may have sources or sinks, and several advection-diffusion equations may be coupled together via chemical reactions.

Fick's law is used to model the diffusive flux. Diffusion may be anisotropic, which may be physically reasonable at least in solids. If the velocity field is identically zero, the advection-diffusion equation reduces to the diffusion equation, which is applicable in solids.

Heat equation is a special case of the advection-diffusion (or diffusion-convection) equation, and it is described elsewhere in this manual.

3.2 Theory

3.2.1 Governing Equations

The advection-diffusion equation may, in general, be expressed in terms of relative or absolute mass or molar concentrations. In Elmer, when the transported quantity is carried by an incompressible fluid (or it is diffused in a solid), relative mass concentration $c_i = C_i/\rho$ for the species i is used (C_i is the absolute mass concentration in units $\mathrm{kg/m^3}$, and ρ the total density of the mixture). We have used the approximation valid for dilute multispecies flows, i.e., $0 \leq c_i \ll 1$. The advection-diffusion equation is now written as

$$\rho \left(\frac{\partial c_i}{\partial t} + (\vec{v} \cdot \nabla) c_i \right) = \rho \nabla \cdot (D_i \nabla c_i) + S_i, \tag{3.1}$$

where $\vec{v}$ is the advection velocity, D_i the diffusion coefficient and S_i is a source, sink or a reaction term. The diffusion coefficient may be a tensor.

For a compressible fluid, the concentration should be expressed in absolute mass units, and the advection-diffusion equation reads

$$\frac{\partial C_i}{\partial t} + (\nabla \cdot \vec{v}) C_i + (\vec{v} \cdot \nabla) C_i = \nabla \cdot (D_i \nabla C_i) + S_i. \tag{3.2}$$

For a situation, where the quantity is transported through a phase change boundary, it is convenient to scale the absolute mass formulation by the respective solubilities of the different phases. Such a case is for example the surface of a liquid, where the transported quantity is evaporated into a gaseous material. The scaled concentration variable satisfies the equilibrium boundary condition on the phase change boundary automatically, and thus the advection-diffusion equation can be solved for both materials simultaneously. The scaling is following

$$x_i = \frac{C_i}{C_{i,max}}, \tag{3.3}$$

where x_i is the concentration of species i relative to its maximum solubility in the current material in absolute mass units. The maximum solubility has to be a constant (temperature independent) for the absolute mass formulation of the advection-diffusion equation to remain unchanged.

It is also possible to include temperature dependent diffusion (Soret diffusion). This introduces an additional term on the right-hand side of the equation:

$$\nabla \cdot (\rho D_{i,T} \nabla T), \tag{3.4}$$

where $D_{i,T}$ is the thermal diffusion coefficient of species i. The coefficient $D_{i,T}$ has to be given in the units m^2/Ks regardless of the units used for concentration.

The velocity of the advecting fluid, $\vec{v}$, is typically calculated by the Navier-Stokes equation and read in from a restart file. All quantities can also be functions of, e.g., temperature that is given or solved by the heat equation. Several advection-diffusion equations for different species i may be coupled and solved for the same velocity field.

Given volume species sources S_i can be prescribed. They are given in absolute mass units, i.e., $\mathrm{kg/m^3s}$. If the equation is scaled to maximum solubility, the source term can be given in absolute mass units, or in scaled units, $S_{i,sc} = S_i/C_{i,max}$, which is the default.

3.2.2 Boundary Conditions

For each species one can apply either a prescribed concentration or a mass flux as boundary conditions.

Dirichlet boundary condition reads as

$$c_i = c_{i,b}, \tag{3.5}$$

or

$$C_i = C_{i,b}, \tag{3.6}$$

depending on the units. If the concentration is scaled to maximum solubility, the Dirichlet boundary conditions have to be given also in scaled values, $x_i = C_{i,b}/C_{i,max}$. In all variations, the boundary value can be constant or a function of time, position or other variables.

One may specify a mass flux $\vec{\jmath}_i$ perpendicular to the boundary by

$$\vec{\jmath}_i \cdot \vec{n} = -D_i \frac{\partial C_i}{\partial n} = g. \tag{3.7}$$

In relative mass units, this may be written as

$$\vec{\jmath}_i \cdot \vec{n} = -\rho D_i \frac{\partial c_i}{\partial n} = g. \tag{3.8}$$

Thus the units in the flux boundary condition are always $\mathrm{kg/m^2}s$ except when the equation is scaled to maximum solubility. In that case the default is to give flux condition in scaled units, $g_{sc} = g/C_{i,max}$, although the physical units are also possible.

The mass flux may also be specified by a mass transfer coefficient β and an external concentration C_{ext}

$$-D_i \frac{\partial C_i}{\partial n} = \beta(C_i - C_{i,ext}). \tag{3.9}$$

On the boundaries where no boundary condition is specified, the boundary condition $g = 0$ is applied. This zero flux condition is also used at a symmetry axis in 2D, axisymmetric or cylindrical problems.

The equilibrium boundary condition on phase change boundaries under certain conditions is that the relative amounts of the transported quantity are equal on both sides of the boundary,

$$\frac{C_i^{(1)}}{C_{i,max}^{(1)}} = \frac{C_i^{(2)}}{C_{i,max}^{(2)}}, \tag{3.10}$$

where the superscripts (1) and (2) refer to different sides of the boundary. This boundary condition is automatically satisfied if the equation is scaled with the maximum solubilities $C_{i,max}^{(j)}$.

However, the scaling causes a discontinuity into the mass flux of the species through the phase change surface. The solver compensates this effect as long as such a boundary is flagged in the command file by the user.

3.3 Keywords

`Simulation`

The simulation section gives the case control data:

`Simulation Type` String

Advection-diffusion equation may be either `Transient` or `Steady State`.

`Coordinate System` String

Defines the coordinate system to be used, one of: `Cartesian 1D`, `Cartesian 2D`, `Cartesian 3D`, `Polar 2D`, `Polar 3D`, `Cylindric`, `Cylindric Symmetric` and `Axi Symmetric`.

`Timestepping Method` String

Possible values of this parameter are `Newmark` (an additional parameter `Newmark Beta` must be given), `BDF` (`BDF Order` must be given). Also as a shortcut to `Newmark`-method with values of `Beta=0.0,0.5, 1.0` the keywords `Explicit Euler`, `Crank-Nicolson`, and `Implicit Euler` may be given respectively. The recommended choice for the first order time integration is the BDF method of order 2.

`BDF Order` Integer

Value may range from 1 to 5.

`Newmark Beta` Real

Value in range from 0.0 to 1.0. The value 0.0 equals to the explicit Euler integration method and the value 1.0 equals to the implicit Euler method.

`Solver` solver id

The solver section defines equation solver control variables. Most of the possible keywords – related to linear algebra, for example – are common for all the solvers and are explained elsewhere.

`Equation` String [Advection Diffusion Equation Varname]

The name of the equation, e.g., `Advection Diffusion Equation Oxygen`.

`Variable` String Varname

The name of the variable, e.g., `Oxygen`.

`Procedure` File "AdvectionDiffusion" "AdvectionDiffusionSolver"

The name of the file and subroutine.

`Nonlinear System Convergence Tolerance` Real

The criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations k is small enough

$$||u_k - u_{k-1}|| < \epsilon ||u_k||,$$

where ϵ is the value given with this keyword, and u is either c_i or C_i.

`Nonlinear System Max Iterations` Integer
: The maximum number of nonlinear iterations the solver is allowed to do.

`Nonlinear System Relaxation Factor` Real
: Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achieve convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:

$$u_k^{'} = \lambda u_k + (1 - \lambda) u_{k-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

`Steady State Convergence Tolerance` Real
: With this keyword a equation specific steady state or coupled system convergence tolerance is given. All the active equation solvers must meet their own tolerances for their variable u before the whole system is deemed converged. The tolerance criterion is:

$$||u_i - u_{i-1}|| < \epsilon ||T_i||,$$

where ϵ is the value given with this keyword.

`Stabilize` Logical
: If this flag is set true the solver will use stabilized finite element method when solving the advection-diffusion equation with a convection term. If this flag is set to `False`, RFB (Residual Free Bubble) stabilization is used instead (unless the next flag `Bubbles` is set to `False` in a problem with Cartesian coordinate system). If convection dominates, some form of stabilization must be used in order to successfully solve the equation. The default value is `False`.

`Bubbles` Logical
: There is also a residual-free-bubbles formulation of the stabilized finite-element method. It is more accurate and does not include any ad hoc terms. However, it may be computationally more expensive. The default value is `True`. If both `Stabilize` and `Bubbles` or set to `False`, no stabilization is used. This choice may be enforced in a problem with Cartesian coordinates, but the results might be nonsensical. Both `Stabilize` and `Bubbles` should not be set to `True` simultaneously.

`Equation` eq id
: The equation section is used to define a set of equations for a body or set of bodies.

`Advection Diffusion Equation Varname` Logical
: If set to `True`, solve the advection-diffusion equation.

`Convection` String
: The type of convection to be used in the advection-diffusion equation, one of: `None`, `Computed`, `Constant`.

`Concentration Units` String
: If set to `Absolute Mass`, absolute mass units are used for concentration. Recommended for a compressible flow. Also possible to select `Mass To Max Solubility` which causes the absolute mass formulation of the equation to be scaled by the maximum solubilities of each material.

`Body Forces` bf id
: The body force section may be used to give additional force terms for the equations. The following keyword is recognized by the solver:

`Varname Diffusion Source` Real
: An additional volume source for the advection-diffusion equation may be given with this keyword. It may depend on coordinates, temperature and other variables, such as concentration of other chemical species, and thus describe a source, a sink or a reaction term. Given in absolute mass units or, in case of scaling, in the scaled units.

`Physical Units` `Logical True`
: With this keyword, the source term can be given in absolute mass units regardless of scaling.

`Initial Condition` `ic id`
: The initial condition section may be used to set initial values for the concentration c_i, C_i or x_i.

`Varname` `Real`

`Material` `mat id`
: The material section is used to give the material parameter values. The following material parameters may be effective when advection-diffusion equation is solved.

`Convection Velocity i` `Real`
: Convection velocity `i`$= 1, 2, 3$ for the constant convection model.

`Density` `Real`
: The value of density of the transporting fluid is given with this keyword. The value may be constant, or variable. For compressible flow, the density of the transporting fluid is computed internally, and this keyword has no effect.

`Compressibility Model` `String`
: This setting may be used to set the compressibility model for the flow simulations. Choices are `Incompressible` and `Perfect Gas`. If set to the latter, the density is calculated from the ideal gas law. Then also the settings `Reference Pressure`, `Specific Heat Ratio` and `Heat Capacity` must be given.

`Reference Pressure` `Real`
: With this keyword a reference level of pressure may be given.

`Specific Heat Ratio` `Real`
: The ratio of specific heats (in constant pressure versus in constant volume) may be given with this keyword. The default value of this setting is $5/3$, which is the appropriate value for monoatomic ideal gas.

`Heat Capacity` `Real`
: For the compressible flow, specific heat in constant volume.

`Varname Diffusivity` `Real`
: The diffusivity D given by, e.g., `Oxygen Diffusivity`. Can be a constant or variable. For an anisotropic case, may also be a tensor D_{ij}.

`Varname Soret Diffusivity` `Real`
: The thermal diffusivity coefficient D_T given by, e.g., `Oxygen Soret Diffusivity`. Can be a constant or variable.

`Varname Maximum Solubility` `Real`
: The maximum solubility of the species in absolute mass units. Has to be a constant value.

`Boundary Condition` `bc id`
: In advection-diffusion equation we may set the concentration directly by Dirichlet boundary conditions or use mass flux condition. The natural boundary condition is zero flux condition.

`Varname` `Real`

`Mass Transfer Coefficient` `Real`

`External Concentration` `Real`
: These two keywords are used to define flux condition that depends on the external concentration and a mass transfer coefficient. This condition is only applicable to absolute mass formulation of the equation (see keywords for Equation block).

`Varname Flux` `Real`
: A user defined mass flux term in absolute mass units or, in case of scaling, in the scaled units.

`Physical Units` `Logical True`
: With this keyword, the flux boundary condition can be given in absolute mass units regardless of scaling. Note that this keyword does NOT affect the Dirichlet boundary condition nor the mass transfer coefficient bc.

`Varname Solubility Change Boundary` `Logical True`
: This keyword marks the boundary over which the maximum solubility changes. Has to be present for the mass flux continuity to be preserved.

`Normal Target Body` `Integer bd id`
: In a solubility change boundary, this keyword can be used to control on which side the mass flux compensation is done. Basically, this can be done on either side but there can be some effect on the accuracy or on the speed of the solution. Recommended is to give as normal target the body with less dense mesh, or the direction of average species transport. If normal target body is not specified, the material with smaller density is used.

Model 4

Advection-Reaction Equation

Module name: AdvectionReaction
Module subroutines: AdvectionReactionSolver
Module authors: Mikko Lyly, Juha Ruokolainen, Thomas Zwinger
Document authors: Thomas Zwinger

4.1 Introduction

Advection-reaction equation describes the transport of a passive scalar quantity, c, by a fluid. The advected quantity is assumed not to have an effect on the velocity field. Besides a reaction rate, advection-reaction equation may have sources or sinks. If no reaction rate and source are given, this equation may be used to trace passive scalars through a given flow-field. If a constant source of unity value is given, the equation also may be used to evaluate the time a passive tracer has remained in the flow field.

4.2 Theory

4.2.1 Governing Equations

The advective transport of a scalar c can be written as

$$\frac{\partial c}{\partial t} + \vec{v} \cdot \nabla c + \Gamma c = S, \tag{4.1}$$

where $\vec{v}$ is the advection velocity, Γ is the reaction rate and S is a source/sink, depending on the sign.

Due to the absence of any diffusion, (4.1) has to be solved applying the Discontinuous Galerkin (DG) method. Elmer implements the particular method as presented in [1]. In order to evaluated jumps across partition boundaries in parallel computations, DG requires the utilization of halo-elements for domain decomposition (see ElmerGrid manual for details).

4.2.2 Limiters

If the scalar has a lower, $c_{\min} \leq c$ and/or an upper limit $c \leq c_{\max}$ limit (where the limit can be also a function of another variable), the variational form of (4.1) becomes a variational inequality. In order to obtain a consistent solution a method using Dirichlet constraints within the domain is applied. The exact procedure is the following:

1. construct the linear system: $Ac = S$, with the system matrix A and the solution vector c on the left-hand side and the force vector S on the right-hand side

2. set nodes as *active* if the constraint is violated

3. for *active* nodes the matrix and force vector are manipulated such that effectively a Dirichlet condition $c = c_{\text{max/min}}$ is applied
4. the manipulated system is solved: $\tilde{A}\tilde{c} = \tilde{S}$
5. a residual is obtained from the un-manipulated system: $R = A\tilde{c} - S$
6. an *active* node is reset if the residual is $R < 0$ (for lower limit) and $R > 0$ (for upper limit)

The whole algorithm is iterated (within the non-linear iteration loop) until the limit given in `Nonlinear System Convergence Tolerance` is reached. In the converged solution the residual represents the needed accumulation/volume flux (on matrix level, hence not in physical units) needed in order to obtain the limited solution. Consequently, the system not necessarily is volume conserving if the Dirichlet method is applied.

4.2.3 Boundary Conditions

At boundaries, a Dirichlet boundary condition reads as

$$c = c_b. \tag{4.2}$$

By nature of the applied DG method, the condition above only applies at inflow boundaries, i.e., if

$$\vec{v} \cdot \vec{n}_b < 0, \tag{4.3}$$

where $\vec{n}_b$ is the outwards facing surface normal of the boundary.

On the boundaries where no boundary condition is specified, the boundary condition $c = 0$ is applied upon inflow.

4.3 Keywords

`Simulation`
The simulation section gives the case control data:

`Simulation Type String`
Advection-reaction equation may be either `Transient` or `Steady State`.

`Coordinate System String`
Defines the coordinate system to be used, one of: `Cartesian 1D`, `Cartesian 2D`, `Cartesian 3D`, `Polar 2D`, `Polar 3D`, `Cylindric`, `Cylindric Symmetric` and `Axi Symmetric`.

`Timestepping Method String`
Possible values of this parameter are `Newmark` (an additional parameter `Newmark Beta` must be given), `BDF` (`BDF Order` must be given). Also as a shortcut to `Newmark`-method with values of `Beta=0.0,0.5, 1.0` the keywords `Explicit Euler`, `Crank-Nicolson`, and `Implicit Euler` may be given respectively. The recommended choice for the first order time integration is the BDF method of order 2.

`BDF Order Integer`
Value may range from 1 to 5.

`Newmark Beta Real`
Value in range from 0.0 to 1.0. The value 0.0 equals to the explicit Euler integration method and the value 1.0 equals to the implicit Euler method.

`Solver solver id`
The solver section defines equation solver control variables. Most of the possible keywords – related to linear algebra, for example – are common for all the solvers and are explained elsewhere.

`Equation` `String Advection-Diffusion`
: The name of the equation, it can be arbitrary but unique.

`Discontinuous Galerkin` `Logical`
: needs to be set to `True`. This is currently also enforced internally.

`Variable` `String Variable_name`
: The name of the variable, e.g., `Tracer`.

`Procedure` `File "AdvectionReaction" "AdvectionReactionSolver"`
: The name of the file and subroutine.

`Nonlinear System Convergence Tolerance` `Real`
: The criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations k is small enough

$$||c_k - c_{k-1}|| < \epsilon ||c_k||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Max Iterations` `Integer`
: The maximum number of nonlinear iterations the solver is allowed to do.

`Steady State Convergence Tolerance` `Real`
: With this keyword a equation specific steady state or coupled system convergence tolerance is given. All the active equation solvers must meet their own tolerances for their variable c before the whole system is deemed converged. The tolerance criterion is:

$$||c_i - c_{i-1}|| < \epsilon ||c_i||,$$

where ϵ is the value given with this keyword.

`Limit Solution` `Logical`
: Assumes the variational inequality method to apply, if set to `true`.

`Compute Nodal Average` `Logical`
: The user may choose to average the Discontinuous Galerkin field to nodes. This was historically needed to be able to visualize the fields using `ElmerPost` but `Paraview` can also handle cell fields. If this flag is set on then `Varname_nodal` is the name used for the resulting field.

`Equation` `eq id`
: The equation section is used to define a set of equations for a body or set of bodies.

`Convection` `String`
: The type of convection to be used in the advection-reaction equation, one of: `None`, `Computed`, `Constant`.

`Body Forces` `bf id`
: The body force section may be used to give additional force terms for the equations. The following keyword is recognized by the solver:

`Variable_name Source` `Real`
: defines the volumetric source for variable c

`Initial Condition` `ic id`
: The initial condition section may be used to set initial values for the scalar c.

`Variable_name` `Real`

`Material` `mat id`
: The material section is used to give the material parameter values. The following material parameters may be effective when advection-diffusion equation is solved.

`Convection Velocity i` `Real`
: Convection velocity `i`$= 1, 2, 3$ for the constant convection model.

`Variable_name Upper Limit` `Real`
: The upper limit, c_{max}, for variable `Variable_name`. Only used if keyword `Limit Solution` for the solver is set to `true`

`Variable_name Lower Limit` `Real`
: The upper limit, c_{min}, for variable `Variable_name`. Only used if keyword `Limit Solution` for the solver is set to `true`

`Variable_name Gamma` `Real`
: defines the reaction rate, Γ

`Boundary Condition` `bc id`

`Variable_name` `Real` sets the value for c at inflow boundaries

Bibliography

[1] F. Brezzi and E. Marini, .L. D.and Süli. Discontinuous Galerkin methods for first-order hyperbolic prob lems. *Math. Models Methods Appl. Sci.*, 14(12):1893–1903, 2004.

Model 5

Reynolds Equation for Thin Film Flow

Module name: ReynoldsSolver
Module subroutines: ReynoldsSolver, ReynoldsHeatingSolver
Module authors: Peter Råback
Document authors: Peter Råback

5.1 Introduction

The flow of fluids is in the continuum level usually described by the Navier-Stokes equations. For narrow channels this approach is an overkill and usually not even necessary. Neglecting the inertial forces and assuming fully developed laminar velocity profiles the flow equations may be reduced in dimension resulting to the Reynolds equation.

The current implementation of the Reynolds equation is suitable for incompressible and weakly compressible liquids as well as for isothermal and adiabatic ideal gases. The nonlinear terms for the compressible fluids are accounted for. The fluid is assumed to be newtonian i.e. there is a direct connection between the strain rate and stress. The equation may be solved either in steady state or in a transient mode.

There is an additional solver for postprocessing purposes that computes the local heat generation field using the Galerkin method. It also computes the integrals over the force and heating fields over the whole area.

5.2 Theory

The underlying assumption of the Reynolds equation is that the flow in the channel is fully developed and has thus the Hagen-Poiseuille parabolic velocity profile. Accounting also for the movement of the planes and leakage through perforation holes the pressure may be solved from the equation

$$\nabla \cdot \left(\frac{\rho h^3}{12\eta} \nabla p \right) - Y \rho p = \frac{1}{2} \nabla \cdot (\rho h \vec{v}_t) + h \frac{\partial \rho}{\partial t} + \rho v_n, \tag{5.1}$$

where ρ is the density, η is the viscosity, p is the pressure and h is the gap height, v_t is the tangential velocity, and v_n is the velocity in direction of the surface normal [1, 5]. Holes may be homogenized using the flow admittance Y which gives the ratio between pressure drop and mean flow velocity through the hole.

The exact form of the Reynolds equation depends on the material law for density, $\rho(p)$. The absolute value of density does not play any role and therefore we may study just the functional forms. For gases we solve for the pressure variation from the reference pressure P_0 rather than for the absolute pressure. The different functional forms for some idealized material laws are the following:

$$\begin{array}{rcl} \rho \propto & (P_0 + p) & \text{isothermal ideal gas} \\ \rho \propto & (P_0 + p)^{1/\gamma} & \text{adiabatic ideal gas} \\ \rho \propto & 1 & \text{incompressible} \\ \rho \propto & e^{p/\beta} & \text{weakly compressible.} \end{array}$$

Here $\gamma = C_p/C_V$ is the specific heat ratio and β the bulk modulus. In discretization of the equations it is also useful to derive the functional dependencies of the density derivatives in respect to pressure,

$$\begin{array}{rcl} \rho_p \propto & 1 & \text{isothermal ideal gas} \\ \rho_p \propto & (1/\gamma)(P_0 + p)^{1/\gamma - 1} & \text{adiabatic ideal gas} \\ \rho_p \propto & 0 & \text{incompressible} \\ \rho_p \propto & \rho/\beta & \text{weakly compressible.} \end{array}$$

In order to improve convergence of the iteration of the nonlinear system some terms including differentials of density may be expressed implicitly using pressure. This way equation (5.1) may be written in the following form:

$$\nabla \cdot \left(\frac{\rho h^3}{12\eta} \nabla p \right) - Y \rho p - \rho_p h \frac{\partial p}{\partial t} - \frac{1}{2} \rho_p h \vec{v}_t \cdot \nabla p = \frac{1}{2} \rho \nabla \cdot (h \vec{v}_t) + \rho v_n. \tag{5.2}$$

The surface velocity $\vec{v}$ may also be given in normal Cartesian coordinate system. Then the normal and tangential components may easily be obtained from

$$\begin{array}{rcl} v_n & = & \vec{v} \cdot \vec{n} \\ \vec{v}_t & = & \vec{v} - v_n \vec{n}. \end{array}$$

The normal velocity and gap height are naturally related by

$$v_n = \frac{\partial h}{\partial t}. \tag{5.3}$$

In transient case the user should make sure that this relationship is honored.

5.2.1 Flow admittances of simple geometries

The flow admittance, Y, occurring in the Reynolds equation may sometimes be solved analytically for simple hole geometries from the steady-state Stokes equation. Generally Y depends on the history but here we assume that it is presents the steady-state situation of the flow [2, 5]. This means that inertial and compressibility effects are not accounted for. For cylindrical holes the admittance then yields,

$$Y = \frac{D^2}{32\eta b}, \tag{5.4}$$

where D is the diameter of the holes and b is the length of the hole. In case of a narrow slot with width W the admittance is given by

$$Y = \frac{W^2}{12 b \eta}. \tag{5.5}$$

5.2.2 Gas rarefaction effects

Generally the Reynolds equation could also be used to model nonnewtonian material laws. The current implementation is limited to the special case of rarefied gases. The goodness of the continuum assumption η depends on the Knudsen number, K_n, which is defined by

$$K_n = \frac{\lambda}{h}, \tag{5.6}$$

where λ is the mean free path of the molecules and h is the characteristic scale (here the gap height). In this solver only the dependence with pressure is taken into account from the formula

$$\lambda = \frac{1}{1 + p/P_0}\lambda_0. \tag{5.7}$$

When the Knudsen number is very small ($K_n \ll 1$) the gas may be considered as a continuous medium. When the Knudsen number is in the transition regime ($K_n \approx 1$) we may take the gas rarefaction effect into account by an effective viscosity. This accounts for the slip conditions of the flow in the channel by decreasing the viscosity value. An approximation given by Veijola [4] is

$$\eta = \frac{\eta_0}{1 + 9.638 K_n^{1.159}}. \tag{5.8}$$

It s relative accuracy is 5 % in the interval $0 < K_n < 880$.

5.2.3 Boundary conditions for the Reynolds equation

The Reynolds equation may have different boundary conditions. The natural boundary condition that is obtained by default is

$$\frac{\partial p}{\partial n} = 0. \tag{5.9}$$

This condition may be used at symmetry and closed boundaries.

If the aspect ratio of the resonator is large then the pressure variation at the open sides is small compared to the values far from boundaries. Then may set Dirichlet boundary conditions ($p = 0$) for the pressure. However, if the aspect ratio is relatively small the open side effects should be taken into account. The pressure variation at the side is not exactly zero while also the open space has a flow resistance. The pressure derivative at the boundary is approximated by

$$\frac{\partial p}{\partial n} = \frac{p}{L}, \tag{5.10}$$

where L is the effective added length of the open sides [3]. If gas rarefaction is not accounted for then $L = 0.8488h$, otherwise

$$L = 0.8488(1.0 + 2.676 K_n^{0.659})h. \tag{5.11}$$

5.2.4 Postprocessing

When the equation has been solved the solution may be used to compute some data for postprocessing purposes. The local volume flux in the lateral direction may be obtained from

$$\vec{q} = -\frac{h^3}{12\eta}\nabla p + h\vec{v}_t. \tag{5.12}$$

The total force acting on the surface is

$$\vec{F} = \int_A \left(p\vec{n} + \frac{\eta}{h}\vec{v}_t\right) dA, \tag{5.13}$$

where the first term is due to pressure driven flow and the second one due to sliding driven flow. Also the heating effect may be computed. It consist of two parts: pressure driven flow and sliding flow. The local form of this is

$$h = \frac{h^3}{12\eta}|\nabla p|^2 + \frac{\eta}{h}|\vec{v}_t|^2. \tag{5.14}$$

Therefore the total heating power of the system is

$$Q = \int_A q\, dA. \tag{5.15}$$

It should be noted that if the velocity field $\vec{v}$ is constant then the integral quantities should fulfill the condition $Q = \vec{F} \cdot \vec{v}$.

Note that the above implementation does not take into account the leakage through perforation holes nor the compressibility effects of the fluids.

5.3 Keywords

The module includes two different solvers. `ReynoldsSolver` solves the differential equation (5.2) while `ReynoldsHeatingSolver` solves the equation (5.14) and computes the integrals. The second solver only makes sense when the pressure field has already been computed with the first one. The second solver uses the same material parameters as the first one.

Keywords for ReynoldsSolver

`Solver` `solver id`

`Equation` `String ReynoldsSolver`
: A describing name for the solver. This can be changes as long as it is used consistently.

`Procedure` `File "ReynoldsSolver" "ReynoldsSolver"`
: Name of the solver subroutine.

`Variable` `String FilmPressure`
: The name of the variable may be freely chosen as far as it is used consistently also elsewhere.

`Variable DOFs` `Integer 1`
: Degrees of freedom for the pressure. This should be 1 which is also the default value.

`Procedure` `File "ReynoldsSolver" "ReynoldsSolver"`
: The name of the module and procedure. These are fixed.

`Apply Limiter` `Logical`
: The generic soft limiters may be applied for the Reynolds equation in order to mimic the effects of cavitation. With this flag active the minimum and maximum limiters are accounted.

`Nonlinear System Convergence Tolerance` `Real`
: The transient equation is nonlinear if the relative displacement or pressure deviation is high. The iteration is continued till the relative change in the norm falls under the value given by this keyword.

`Nonlinear System Max Iterations` `Integer`
: This parameter gives the maximum number of nonlinear iterations required in the solution. This may be set higher than the typical number of iterations required as the iteration procedure should rather be controlled by the convergence tolerance.

`Material` `mat id`

`Gap Height` `Real`
: Height of the gap where the fluid is trapped. If the case is transient the user should herself make sure that also this variable has the correct dependence on time.

`Surface Velocity i` `Real`
: The velocity of the moving body may be given in either Cartesian coordinates, or in ones that are already separated to normal and tangential directions. In the first case the velocity components are given with this keyword with `i=1,2,3`.

`Tangent Velocity i` `Real`
: For setting the tangential velocity (i.e. sliding velocity) use this keyword with `i=1,2,3`.

`Normal Velocity` `Real`
: Normal velocity is the velocity in the direction of the surface normal. Typically a negative value means contraction.

`Viscosity` `Real`
: Viscosity of the gas.

`Viscosity Model` `String`
: The choices are `newtonian` and `rarefied`. The first one is also the default.

`Compressibility Model` `String`
: The choices are `incompressible`, `weakly compressible`, `isothermal ideal gas`, and `adiabatic ideal gas`.

`Reference Pressure` `Real`
: Reference pressure is required only for the ideal gas laws.

`Specific Heat Ratio` `Real`
: This parameter is only required for adiabatic processes. For ideal monoatomic gases the ratio is $5/3$. Only required for the adiabatic compressibility model.

`Bulk Modulus` `Real`
: The parameter β in the weakly compressible material model.

`Mean Free Path` `Real`
: If the viscosity model assumes rarefied gases the mean free path of the gas molecules in the reference pressure must be given.

`Flow Admittance` `Real`
: The steady-state flow admittance resulting from perforation, for example.

`Body Force` `bf id`

`FilmPressure Lower Limit` `Real`
: The lower limit for the pressure that will be iteratively be enforced when the soft limiters are active.

`Boundary Condition` `bc id`

`FilmPressure` `Real`
: Sets the boundary conditions for the pressure. Usually the deviation from reference pressure is zero at the boundaries.

`Open Side` `Logical`
: The open end effect may be taken into account by setting this keyword `True`.

Keywords for ReynoldsPostprocess

This solver uses largely the same keywords that are already defined above. Only the Solver section has its own keyword settings. This solvers should be active in the same bodies than the `ReynoldsSolver`.

`Solver` `solver id`

`Equation` `String ReynoldsPostprocess`
: A describing name for the solver. This can be changes as long as it is used consistently.

`Procedure` `File "ReynoldsSolver" "ReynoldsPostProcess"`
: Name of the solver subroutine.

`Reynolds Pressure Variable Name` `String`
: The name of the field that is assumed to provide the pressure field. The default is `FilmPressure`. Note that the `Variable` of this equation need not to be defined since it is automatically set when any of the field computation is requested.

`Calculate Force` `Logical`
: Calculate the forces resulting from the pressure distribution computed with the Reynolds equation. The name of the field is obtained by adding the suffix `Force`.

`Calculate Flux` `Logical`
: Calculate the fluxes resulting from the pressure distribution computed with the Reynolds equation. The name of the field is obtained by adding the suffix `Flux`.

`Calculate Heating` `Logical`
: Calculate the heating efficiency from the pressure distribution computed with the Reynolds equation. The name of the field is obtained by adding the suffix `Heating`.

`Calculate Force Dim` Integer
: By default the dimension of the force field is the mesh dimension plus one. Sometimes the pressure lives on a 1D line of a 2D mesh. Then this keyword may be used to suppress the dimension of force to two.

`Calculate Flux Dim` Integer
: As the previous keyword but for the flux.

Bibliography

[1] B.J. Hamrock, S.R. Schmid, and B.O. Jacobson. *Fundamentals of fluid film lubrication*. Marcel Dekker, second edition, 2004.

[2] P. Rï£¡back, A. Pursula, V. Junttila, and T. Veijola. Hierarchial finite element simulation of perforated plates with arbitrary hole geometries. In *NANOTECH 2003, San Francisco, 23-27 February 2003*, volume 1, pages 194–197.

[3] T. Veijola. *APLAC 7.60 Reference Manual*, January 2002. Electromechanical Macro Models.

[4] T. Veijola, H. Kuisma, J. Lahdenperï£¡, and T. Ryhï£¡nen. Equivalent-circuit model of the squeezed gas film in a silicon accelerometer. *Sensors and Actuators A*, pages 239–248, 1995.

[5] T. Veijola and P. Rï£¡back. A method for solving arbitrary mems perforation problems with rare gas effects. In *NANOTECH 2005, Anaheim, May 8-12, 2005*, volume 3, pages 561–564.

Part II

Models of Solid Mechanics

Model 6

Linear Elasticity

Module name: StressSolve
Module subroutines: StressSolver
Module authors: Juha Ruokolainen
Document authors: Juha Ruokolainen, Thomas Zwinger

6.1 Introduction

This module computes displacement field from the Navier equations. The Navier equations correspond to linear theory of elastic deformation of solids. The material may be anisotropic and stresses may be computed as a post processing step, if requested by the user. Thermal stresses may also be requested.

6.2 Theory

The dynamical equation for elastic deformation of solids may be written as

$$\rho \frac{\partial^2 \vec{d}}{\partial t^2} - \nabla \cdot \tau = \vec{f}, \tag{6.1}$$

where ρ is density, $\vec{d}$ is the displacement field, $\vec{f}$ given volume force, and τ the stress tensor. Stress tensor is given by

$$\tau^{ij} = C^{ijkl}\varepsilon_{kl} - \beta^{ij}(T - T_0), \tag{6.2}$$

where ε is the strain and quantity C is the elastic modulus. The elastic modulus is a fourth order tensor, which has at the most 21 (in 3D, 10 in 2D) independent components due to symmetries. In Elmer thermal stresses may be considered by giving the heat expansion tensor β and reference temperature of the stress free state T_0. The temperature field T may be solved by the heat equation solver or otherwise. The linearized strains are given simply as:

$$\varepsilon = \frac{1}{2}(\nabla\vec{d} + (\nabla\vec{d})^T). \tag{6.3}$$

6.2.1 Material laws

For isotropic materials the elastic modulus tensor may be reduced to two independent values, either the Lame parameters or Young's modulus and Poisson's ratio. The stress tensor given in terms of the Lame parameters is:

$$\tau = 2\mu\varepsilon + \lambda\nabla\cdot\vec{d}I - \beta(T - T_0)I, \tag{6.4}$$

where μ and λ are the first and second Lame parameters respectively, β the heat expansion coefficient, and I is the unit tensor. The Lame parameters in terms of Young's modulus and Poisson's ratio read

$$\lambda = \frac{Y\kappa}{(1+\kappa)(1-2\kappa)}, \quad \mu = \frac{Y}{2(1+\kappa)} \tag{6.5}$$

except for plane stress situations ($\tau_z = 0$) where λ is defined as

$$\lambda = \frac{Y\kappa}{(1-\kappa^2)}. \tag{6.6}$$

Quantities Y and κ are the Young's modulus and Poisson's ratio respectively.

For anisotropic materials, the stress-strain relations may be given in somewhat different form:

$$\tau_V = E\varepsilon_V, \tag{6.7}$$

where τ_V and ε_V are the stress and strain vectors respectively. The 6×6 matrix E (in 3D, 4×4 in 2D) is the matrix of elastic coefficients. The stress and strain vectors are defined as

$$\tau_V = \left(\tau_x \;\; \tau_y \;\; \tau_z \;\; \tau_{xy} \;\; \tau_{yz} \;\; \tau_{xz}\right)^T \tag{6.8}$$

and

$$\varepsilon_V = \left(\varepsilon_x \;\; \varepsilon_y \;\; \varepsilon_z \;\; 2\varepsilon_{xy} \;\; 2\varepsilon_{yz} \;\; 2\varepsilon_{xz}\right)^T. \tag{6.9}$$

In 2D the stress vector is

$$\tau_V = \left(\tau_x \;\; \tau_y \;\; \tau_z \;\; \tau_{xy}\right)^T \tag{6.10}$$

and the strain vector

$$\varepsilon_V = \left(\varepsilon_x \;\; \varepsilon_y \;\; \varepsilon_z \;\; 2\varepsilon_{xy}\right)^T. \tag{6.11}$$

When plane stress computation is requested $\tau_z = 0$, otherwise $\varepsilon_z = 0$. Cylindrically symmetric case is identical to the 2D case, the components are given in the order of r, z, and ϕ. The matrix E is given as input for the anisotropic material model of Elmer.

6.2.2 Viscoelastic Maxwell model

Assuming incompressibility, a viscoelastic Maxwell model is introduced [1] by evolving the viscoelastic stress tensor, τ_{ve},

$$\frac{\partial \tau_{ve}}{\partial t} = \frac{\partial \tau}{\partial t} + \frac{\mu}{\nu}\left(\tau_{ve} - \Pi I\right), \tag{6.12}$$

with μ denoting the dynamic viscosity of the material as well as Π and I standing for the isotropic part of the Cauchy stress tensor and the unit tensor, respectively. τ still denotes the elastic contribution of the stress tensor, which in case of incompressibility is altered to

$$\tau = \Pi I + 2\mu\epsilon. \tag{6.13}$$

If the viscoelastic model is used in the context of solid-Earth deformation, a restoring force (second term below), called Gravitational Pre-stress Advection (GPA), has to be introduced to prevent a loaded Earth to collapse [1]. The quasi-stationary momentum balance then reads

$$\nabla \cdot \tau - \underbrace{\rho g}_{=c_{GPA}} \nabla\left(\vec{e}_z \cdot \vec{d}\right) = 0, \tag{6.14}$$

where g is the acceleration by gravity and $\vec{e}_z$ the vertical direction (aligned with gravity). $c_{GPA} = \rho g$ is the local coefficient for gravitational pre-stress advection, changing between different layers of the Earth.

6.2.3 Modal, harmonic and stability analysis

In addition to steady state and time dependent equations, modal, harmonic and stability analysis may be considered. In modal analysis the Fourier transform of the homogeneous form of the dynamical equation is

$$\rho\omega^2\vec{\phi} = \nabla \cdot \tau(\vec{\phi}), \tag{6.15}$$

or

$$\omega^2 \int_\Omega \rho\phi_k\psi_k \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega, \tag{6.16}$$

where ω is the angular frequency and $\vec{\phi}$ is the corresponding vibration mode.

When modal analysis of pre-stressed solids are considered, we first perform a steady analysis to compute stress tensor, here denoted by σ_{ij}, and solve the variational equation

$$\omega^2 \int_\Omega \rho\phi_k\psi_k \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega + \int_\Omega \sigma_{ij} \frac{\partial\phi_k}{\partial x_i}\frac{\partial\psi_k}{\partial x_j} \, d\Omega. \tag{6.17}$$

The last term on the right-hand side represents here the geometric stiffness due to external loads, thermal stresses etc.

In stability analysis the buckling modes $\vec{\phi}$ are obtained from

$$-\lambda \int_\Omega \sigma_{ij} \frac{\partial\phi_k}{\partial x_i}\frac{\partial\psi_k}{\partial x_j} \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega, \tag{6.18}$$

where λ is the margin of safety with respect to bifurcation (the current load can be multiplied by factor λ before stability is lost).

The equations may be interpreted as generalized eigenproblems and solved with standard techniques.

6.2.4 Rayleigh damping

Damping may be taken into consideration using viscous damping or Rayleigh damping, in which it is assumed that the damping matrix C is proportional to the mass M and stiffness matrices K, or

$$C = \alpha M + \beta K \tag{6.19}$$

The identification of suitable damping coefficients α and β may be a difficult task.

6.2.5 Boundary conditions

For each boundary either a Dirichlet boundary condition

$$d_i = d_i^b \tag{6.20}$$

or a force boundary condition

$$\tau \cdot \vec{n} = \vec{g} \tag{6.21}$$

must be given. The default boundary condition is the natural boundary condition which implies that $\vec{g} = 0$.

The user may give spring k or damping λ coefficients on the boundary. These enable the introduction of the force term in the form

$$\vec{g} = k\vec{d} + \lambda\frac{\partial\vec{d}}{\partial t} \tag{6.22}$$

which may be solved implicitly maintaining the linear form of the equation.

6.2.6 Model lumping

For linear structures it is possible to create a lumped model that gives the same dependence between force and displacement as the original distributed model,

$$F = KD \tag{6.23}$$

where $F = (F_x\,F_y\,F_z\,M_x\,M_y\,M_z)^T$ and $D = (D_x\,D_y\,D_z\,\phi_x\,\phi_y\,\phi_z)^T$. However, the lumped model is not uniquely defined as it depends on the force or displacement distribution used in the model lumping. In the current model lumping procedure the lumping is done with respect to a given boundary. The lumped force and momentum are then integrals over this boundary,

$$F_i = \int_A f_i\,dA. \tag{6.24}$$

Lumped displacements and angles are determined as the mean values over the boundary,

$$D_i = \frac{1}{A}\int_A d_i\,dA. \tag{6.25}$$

Therefore the methodology works best if the boundary is quite rigid in itself.

There are two different model lumping algorithms. The first one uses pure lumped forces and lumped moments to define the corresponding displacements and angles. In 3D this means six different permutations. Each permutation gives one row of the inverse matrix K^{-1}. Pure lumped forces are obtained by constant force distributions whereas pure moments are obtained by linearly varying loads vanishing at the center of area. Pure moments are easily achieved only for relatively simple boundaries which may limit the usability of the model lumping utility.

The second choice for model lumping is to set pure translations and rotations on the boundary and compute the resulting forces on the boundary. This method is not limited by geometric constraints. Also here six permutations are required to get the required data. In this method the resulting matrix equation is often better behaving in comparison with the model lumping by pure forces which may be another reason to favour this procedure.

6.3 Keywords

`Solver` `solver id`

Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation` `String [StressSolver]`

A describing name for the solver. This can be changed but it must be given,

`Procedure` `File "StressSolve" "StressSolver"`

Name of the solver subroutine.

`Eigen Analysis` `Logical`

Modal or stability analysis may be requested with this keyword.

`Eigen System Values` `Integer`

The number of the lowest eigen states must be given with this keyword, if modal or stability analysis is in effect.

`Harmonic Analysis` `Logical`

Time-harmonic analysis where the solution becomes complex if damping is defined. The solution algorithm assumes that the diagonal entries in the matrix equation dominates.

`Frequency` `Real`

The frequency related to the harmonic analysis. If the simulation type is `scanning` this may a scalar function, otherwise it is assumed to be a vector of the desired frequencies.

`Displace Mesh` Logical
: Should the mesh be deformed by the displacement field. The default is `True` except for eigen and harmonic analysis.

`Stability Analysis` Logical
: If set to `true`, then eigen analysis is stability analysis. Otherwise modal analysis is performed.

`Geometric Stiffness` Logical
: If set to `true`, then geometric stiffness is taken into account in modal analysis.

`Calculate Strains` Logical
: Computes the strain tensor of the solution.

`Calculate Stresses` Logical
: If set to `true` the stress tensor will be computed. Also von Mises will be computed by default.

`Calculate Principal` Logical
: Computes the principal stress components.

`Calculate Pangle` Logical
: Calculate the principal stress angles.

`Model Lumping` Logical
: If model lumping is desired this flag should be set to `True`.

`Model Lumping Filename` File
: The results from model lumping are saved into an external file the name of which is given by this keyword.

`Fix Displacement` Logical
: This keyword defines whether the displacements or forces are set and thereby chooses the model lumping algorithm.

`Constant Bulk System` Logical
: For some type of analysis only the boundary conditions change from one subroutine call to another. Then the original matrix may be maintained using this logical keyword. The purpose is mainly to save time spent on matrix assembly.

`Update Transient System` Logical
: Even if the matrix is defined constant it may change with time. The time may also be pseudo-time and then for example the frequency could change with time thus making the harmonic system different between each timestep. This keyword has effect only if the previous keyword is also defined to be true.

`Maxwell Material` Logical
: If set to true, viscoelastic material model is computed. In case of incompressibility of the material the keyword `Incompressible` has to be activated. The user has to supply a value for the viscosity in the `Material` section in this case.

`Incompressible` Logical
: Enables computation of incompressible material in connection with the viscoelastic Maxwell material. This demands to declare a pressure variable as an additional degree of freedom in the solver. For a two-dimensional problem, that would read as `Variable = String "t[d:2 p:1]"`, with `d` declaring the displacements and `p` the pressure (the isotropic part of Cauchy stress tensor).

`Equation` eq id
: The equation section is used to define a set of equations for a body or set of bodies:

`Stress Analysis` Logical
: if set to `True`, solve the Navier equations.

`Plane Stress` Logical
: If set to `True`, compute the solution according to the plane stress situation $\tau_{zz} = 0$. Applies only in 2D.

`Body Force` `bf id`
: The body force section may be used to give additional force terms for the equations.

`Stress Bodyforce 1` `Real`

`Stress Bodyforce 2` `Real`

`Stress Bodyforce 3` `Real`
: The keywords may be used to give volume force.

`Stress Bodyforce 1 im` `Real`

`Stress Bodyforce 2 im` `Real`

`Stress Bodyforce 3 im` `Real`
: The keywords may be used to give volume force for the imaginary part. May be applied only to harmonic solution of the equation.

`Stress Load` `Real`
: Keyword for defining stress load for the body.

`Strain Load` `Real`
: Keyword for defining strain load for the body.

`Gravitational Prestress Advection` `Logical`
: Switches the additional restoring term of pre-stress advection used in solid-Earth deformation

`GPA Coeff` `Real`
: Sets the factor of the gravitational pre-stress advection, c_{GPA}

`Initial Condition` `ic id`
: The initial condition section may be used to set initial values for the field variables. The following variables are active:

`Displacement i` `Real`
: For each displacement component `i`$= 1, 2, 3$.

`Material` `mat id`
: The material section is used to give the material parameter values. The following material parameters may be set in Navier equations.

`Density` `Real` The value of density is given with this keyword. The value may be constant, or variable.

`Poisson Ratio` `Real`
: For isotropic materials Poisson's ratio must be given with this keyword.

`Youngs Modulus` `Real`
: The elastic modulus must be given with this keyword. The modulus may be given as a scalar for the isotropic case or as 6×6 (3D) or 4×4 (2D and axisymmetric) matrix for the anisotropic case. Although the matrices are symmetric, all entries must be given.

`Rayleigh Damping` `Logical`
: Apply Rayleigh damping.

`Rayleigh Damping Alpha` `Real`

`Rayleigh Damping Beta` `Real`
: The parameters of Rayleigh damping.

`Pre Stress` `Real`
: One may give prestress as an input to the solver.

`Pre Strain` `Real`
: One may give prestrain as an input to the solver.

`Heat Expansion Coefficient` `Real`
: If thermal stresses are to be computed this keyword may be used to give the value of the heat expansion coefficient. May also be given as 3×3 tensor for 3D cases, and 2×2 tensor for 2D cases.

`Reference Temperature` `Real`
: If thermal stresses are to be computed this keyword may be used to give the value of the reference temperature of the stress free state.

`Rotate Elasticity Tensor` `Logical`
: For anisotropic materials the principal directions of anisotropy do not always correspond to the coordinate axes. Setting this keyword to `True` enables the user to input Young's modulus matrix with respect to the principal directions of anisotropy. Otherwise Young's modulus should be given with respect to the coordinate axis directions.

`Material Coordinates Unit Vector 1(3)` `Real [1 0 0]`

`Material Coordinates Unit Vector 2(3)` `Real [0 0.7071 0.7071]`

`Material Coordinates Unit Vector 3(3)` `Real [0 -0.7071 0.7071]`
: The above vectors define the principal directions of the anisotropic material. These are needed only if `Rotate Elasticity Tensor` is set to `True`. The values given above define the direction of anisotropy to differ from the coordinate axes by a rotation of 45 degrees about x-axis, for example.

`Mesh Velocity 1` `Real`

`Mesh Velocity 2` `Real`

`Mesh Velocity 3` `Real`
: Keywords for giving the mesh velocity

`Boundary Condition` `bc id`
: The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The one related to Navier equations are

`Displacement i` `Real`
: Dirichlet boundary condition for each displacement component `i`$= 1, 2, 3$.

`Normal-Tangential Displacement` `Logical`
: The Dirichlet conditions for the vector variables may be given in normal-tangential coordinate system instead of the coordinate axis directed system. The first component will in this case be the normal component and the components $2, 3$ two orthogonal tangent directions.

`Normal Force` `Real`
: A force normal to the boundary is given with this keyword.

`Force i` `Real`
: A force in the given in coordinate directions `i`$= 1, 2, 3$.

`Force i Im` `Real`
: An imaginary part of the force in the given in coordinate directions `i`$= 1, 2, 3$. Applies only to harmonic simulation.

`Normal Force Im` `Real`
: A imaginary part of the force normal to the boundary is given with this keyword. Applies only to harmonic simulation.

`Damping` `Real`
: Introduces a force proportional to velocity with the given coefficient. Also `Damping i` and `Damping ij` may be given.

`Spring` `Real`
: Introduces a force proportional to displacement with the given coefficient. Also `Spring i` and `Spring ij` may be given.

`Stress Load` Real
Keyword for defining stress load for the boundary.

`Model Lumping Boundary` Logical True
When using the model lumping utility the user must define which boundary is to be loaded in order to determined the lumped model.

Bibliography

[1] T. Zwinger, GA. Nield, J. Ruokolainen, and M.A. King. A new open-source viscoelastic solid earth deformation module implemented in Elmer (v8.4). *Geoscientific Model Development*, 13:1155–1164, 2020.

Model 7

Finite Elasticity

Module name: ElasticSolve
Module subroutines: ElasticSolver
Module authors: Mikko Lyly, Juha Ruokolainen, Mika Malinen
Document authors: Mika Malinen

7.1 Introduction

This chapter is concerned with the equations which describe finite deformations of elastic solids. As the region of space occupied by the body at time t is not known in advance, it is not convenient to handle the equations in the form that expresses the field equations on the deformed configuration. Therefore the associated problem is formulated here by employing the reference configuration which equals to the region occupied by the body before the deformation.

7.2 Field equations

Let Ω denote the reference configuration, so that the region of space occupied by the body at the time t is given by

$$\Omega_t = \mathsf{x}(\Omega, t),$$

with $\mathsf{x}(\cdot, t)$, for fixed t, a deformation of Ω. If we define the displacement $\boldsymbol{u}(\mathbf{p}, t)$ of the material point $\mathbf{p} \in \Omega$ by

$$\boldsymbol{u}(\mathbf{p}, t) = \mathsf{x}(\mathbf{p}, t) - \mathbf{p},$$

the basic system of field equations describing finite deformations of the body Ω may then be written as

$$\begin{aligned} &\rho_0 \ddot{\boldsymbol{u}} - \mathbf{Div}\, \boldsymbol{S} = \boldsymbol{b}_0, \\ &\boldsymbol{S} = \boldsymbol{F}\bar{\boldsymbol{\Sigma}}(\boldsymbol{C}), \\ &\boldsymbol{F} = \boldsymbol{I} + \boldsymbol{\nabla u}, \quad \boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F}, \end{aligned} \tag{7.1}$$

where ρ_0 gives the density when the body is in the reference position, the tensor field $\boldsymbol{S}$ is referred to as the first Piola-Kirchhoff stress, and $\boldsymbol{b}_0 = \boldsymbol{b}_0(\mathbf{p}, t)$ defines a body force. The response function $\bar{\boldsymbol{\Sigma}}(\boldsymbol{C})$ generally characterizes the second Piola-Kirchhoff stress as a function of the right Cauchy-Green tensor $\boldsymbol{C}$.

In the basic case it is assumed that either

$$\bar{\boldsymbol{\Sigma}}(\boldsymbol{C}) = \frac{\lambda}{2}[\mathrm{tr}(\boldsymbol{C} - \boldsymbol{I})]\boldsymbol{I} + \mu(\boldsymbol{C} - \boldsymbol{I}) \tag{7.2}$$

or, when the neo-Hookean material is assumed,

$$\bar{\boldsymbol{\Sigma}}(\boldsymbol{C}) = \frac{\lambda}{2}[\det \boldsymbol{C} - 1]\boldsymbol{C}^{-1} + \mu(\boldsymbol{I} - \boldsymbol{C}^{-1}), \tag{7.3}$$

with λ and μ the Lame material parameters. We note that a common way to express (7.2) uses the material strain tensor

$$\boldsymbol{E} = 1/2(\boldsymbol{C} - \boldsymbol{I}), \tag{7.4}$$

so that the constitutive law (7.2) may be written as

$$\boldsymbol{\Sigma} = \lambda \mathrm{tr}(\boldsymbol{E})\boldsymbol{I} + 2\mu\boldsymbol{E}.$$

To treat the incompressible neo-Hookean material associated with the limit case $\lambda \to \infty$ (or equivalently $\nu \to 1/2$, with ν the Poisson ratio), we introduce an auxiliary field p defined by

$$(1/\lambda)p = -\frac{1}{2}[\det \boldsymbol{C} - 1] \tag{7.5}$$

to replace (7.3) by

$$\bar{\boldsymbol{\Sigma}}(\boldsymbol{C}, p) = -p\boldsymbol{C}^{-1} + \mu(\boldsymbol{I} - \boldsymbol{C}^{-1}). \tag{7.6}$$

To handle the case of a nearly incompressible material (the value of ν close to 0.5) computationally, the field p is taken to be an additional unknown which is solved under the constraint (7.5). It is notable that in the limit case of incompressible material p is unique only up to a constant if the displacement is prescribed over the entire boundary of the body (a special care is then needed to ensure that the condition $\det \boldsymbol{C} = \det \boldsymbol{F}^2 = 1$ is respected by the boundary conditions).

The solver also offers some utilities to handle more general constitutive laws so that a user-defined material model in the form of an Abaqus user subroutine (UMAT) may be included. In this case the constitutive law has to be defined differently by employing the Cauchy stress $\boldsymbol{\sigma}$ whose material description $\boldsymbol{\sigma}_m(\mathbf{p}, t) \equiv \boldsymbol{\sigma}(\mathbf{x}(\mathbf{p}, t), t)$ is related to the first Piola-Kirchhoff stress by

$$\boldsymbol{S} = (\det \boldsymbol{F})\boldsymbol{\sigma}_m \boldsymbol{F}^{-T}. \tag{7.7}$$

A user-defined subroutine should define a stress response function $\bar{\boldsymbol{\sigma}}$ to specify how the Cauchy stress depends on a strain field $\hat{\boldsymbol{E}}(\boldsymbol{F})$ and an additional N-tuple of state variable fields denoted by $\mathbf{q} = (q_1, \ldots, q_N)$. One may then consider a generic constitutive law of the type

$$\boldsymbol{\sigma}_m(\mathbf{p}, t) = \bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F})(\mathbf{p}, t), \mathbf{q}(\mathbf{p}, t)). \tag{7.8}$$

The user-defined subroutine should also define the derivative of the Cauchy stress at the current state with respect to the strain variable for performing the nonlinear solution procedure. That is, in the differentiation the stress response function is treated as the composition

$$\boldsymbol{F} \mapsto \bar{\boldsymbol{\sigma}}(\cdot, \mathbf{q}) \circ \hat{\boldsymbol{E}}(\boldsymbol{F}), \tag{7.9}$$

with $\mathbf{q}$ being taken as a parameter, so that the Fréchet differential of (7.9) is obtained by using the chain rule as

$$\boldsymbol{U} \mapsto D\bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F}), \mathbf{q})[D\hat{\boldsymbol{E}}(\boldsymbol{F})[\boldsymbol{U}]]. \tag{7.10}$$

The user-defined subroutine must thus return a representation of $D\bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F}), \mathbf{q})$ so that the differential (7.10) can be generated on the side where the user-defined subroutine is called. In this connection the strain field $\hat{\boldsymbol{E}}(\boldsymbol{F})$ need not necessarily be the material strain (7.4), as the Hencky (logarithmic) strain or the standard linearized strain can also be used. Including a user-defined material model requires some programming skills and the best place to check for the current functionality is the solver code. A template subroutine for a user-defined material model can also be found in the file `UMATLib.F90` that is located in the same directory as the code of the solver module.

7.3 Boundary conditions

Boundary conditions may be obtained by prescribing the displacement and surface traction on complementary parts Γ_1 and Γ_2 of the boundary $\partial\Omega$, respectively. The displacement boundary condition is simply of the form

$$\boldsymbol{u} = \hat{\boldsymbol{u}}(\mathbf{p}, t), \tag{7.11}$$

with $\hat{\boldsymbol{u}}$ a prescribed vector field on $\Gamma_1 \times [0, T]$.

Handling surface traction is more involved. First, assume that the surface traction vector $\boldsymbol{s}$ on the deformed surface $\mathbf{x}(\Gamma_2, t)$ is normal to the tangent plane of the deformed boundary surface, so that

$$\boldsymbol{s}(\mathbf{x}, t) = g(\mathbf{x}, t)\boldsymbol{m}(\mathbf{x}),$$

where $\boldsymbol{m}(\mathbf{x})$ is the unit normal on the deformed configuration, $\mathbf{x} \in \mathbf{x}(\Gamma_2, t)$ for any t, and $g(\mathbf{x}, t)$ is a given scalar function. This can be shown to be equivalent to specifying the values of $\boldsymbol{S}\boldsymbol{n}$ such that

$$\boldsymbol{S}\boldsymbol{n} = \hat{g}(\det \boldsymbol{F})\boldsymbol{F}^{-T}\boldsymbol{n} \quad \text{on } \Gamma_2 \times [0, T], \tag{7.12}$$

where $\boldsymbol{n} = \boldsymbol{n}(\mathbf{p})$ is the normal vector to the boundary $\partial\Omega$ and $\hat{g} = \hat{g}(\mathbf{p}, t) = g(\mathbf{x}(\mathbf{p}, t), t)$. The constraint (7.12) gives rise to a nonlinear force term which is handled in the computational solution iteratively by using a lagged-value approximation.

The surface traction $\boldsymbol{s}$ may also be specified by giving its components with respect to the frame of reference such that

$$\boldsymbol{s}(\mathbf{x}(\mathbf{p}, t), t) = \hat{\boldsymbol{s}}(\mathbf{p}, t), \tag{7.13}$$

with $\hat{\boldsymbol{s}}(\mathbf{p}, t)$ a given vector. While the condition (7.13) specifies the actual force per unit area of the deformed surface, it also possible to specify directly the pseudo-traction $\boldsymbol{s}^0 = \boldsymbol{S}\boldsymbol{n}$ which gives the actual force per unit undeformed area. If the pseudo-traction is specified on Γ_2 as

$$\boldsymbol{s}^0(\mathbf{p}, t) = \hat{\boldsymbol{s}}^0(\mathbf{p}, t), \tag{7.14}$$

the total force exerted across Γ_2 is then given by the surface integral

$$\int_{\Gamma_2} \hat{\boldsymbol{s}}^0 \, d\Gamma.$$

If the alternate (7.13) is used, the total force is obtained by

$$\int_{\Gamma_2} \hat{\boldsymbol{s}}(\det \boldsymbol{F})\sqrt{\boldsymbol{n} \cdot (\boldsymbol{F}^{-1}\boldsymbol{F}^{-T})\boldsymbol{n}} \, d\Gamma$$

where the additional scalar term in the integrand relates to the area change during the deformation.

7.4 Linearization: The basic constitutive laws

To handle the model computationally, the constitutive law $\boldsymbol{S} = \hat{\boldsymbol{S}}(\boldsymbol{F}) = \boldsymbol{F}\boldsymbol{\Sigma}(\boldsymbol{F})$, with $\boldsymbol{\Sigma}(\boldsymbol{F}) = \bar{\boldsymbol{\Sigma}}(\boldsymbol{F}^T\boldsymbol{F})$, has to be linearized also. This can be done in terms of the derivative $D\hat{\boldsymbol{S}}(\boldsymbol{F})[\boldsymbol{U}]$ by using the Newton approximation

$$\hat{\boldsymbol{S}}(\boldsymbol{F}_{k+1}) \approx \hat{\boldsymbol{S}}(\boldsymbol{F}_k) + D\hat{\boldsymbol{S}}(\boldsymbol{F}_k)[\boldsymbol{F}_{k+1} - \boldsymbol{F}_k].$$

We then have

$$\hat{\boldsymbol{S}}(\boldsymbol{F}_{k+1}) \approx \hat{\boldsymbol{S}}(\boldsymbol{F}_k) + \boldsymbol{F}_k D\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\boldsymbol{F}_{k+1} - \boldsymbol{F}_k] + (\boldsymbol{F}_{k+1} - \boldsymbol{F}_k)\boldsymbol{\Sigma}(\boldsymbol{F}_k).$$

In view of $\boldsymbol{F}_{k+1} - \boldsymbol{F}_k = \boldsymbol{\nabla}\boldsymbol{u}_{k+1} - \boldsymbol{\nabla}\boldsymbol{u}_k$, this leads to the linearization

$$\begin{aligned}\hat{\boldsymbol{S}}(\boldsymbol{F}_{k+1}) &\approx \hat{\boldsymbol{S}}(\boldsymbol{F}_k) + \boldsymbol{F}_k D\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\boldsymbol{\nabla}\boldsymbol{u}_{k+1} - \boldsymbol{\nabla}\boldsymbol{u}_k] + (\boldsymbol{\nabla}\boldsymbol{u}_{k+1} - \boldsymbol{\nabla}\boldsymbol{u}_k)\boldsymbol{\Sigma}(\boldsymbol{F}_k) \\ &= \hat{\boldsymbol{S}}(\boldsymbol{F}_k) - \boldsymbol{F}_k D\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\boldsymbol{\nabla}\boldsymbol{u}_k] - \boldsymbol{\nabla}\boldsymbol{u}_k\boldsymbol{\Sigma}(\boldsymbol{F}_k) + \boldsymbol{F}_k D\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\boldsymbol{\nabla}\boldsymbol{u}_{k+1}] + \boldsymbol{\nabla}\boldsymbol{u}_{k+1}\boldsymbol{\Sigma}(\boldsymbol{F}_k).\end{aligned} \tag{7.15}$$

In the case of (7.2) the derivative of the response function $\boldsymbol{\Sigma}$ is given by

$$D\boldsymbol{\Sigma}(\boldsymbol{F})[\boldsymbol{\nabla}\boldsymbol{v}] = \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{F}^T\boldsymbol{\nabla}\boldsymbol{v} + \boldsymbol{\nabla}\boldsymbol{v}^T\boldsymbol{F})\boldsymbol{I} + \mu(\boldsymbol{F}^T\boldsymbol{\nabla}\boldsymbol{v} + \boldsymbol{\nabla}\boldsymbol{v}^T\boldsymbol{F}),$$

while

$$\begin{aligned}D\boldsymbol{\Sigma}(\boldsymbol{F})[\nabla\boldsymbol{v}] =&\lambda(\det\boldsymbol{F})^2\mathrm{tr}(\nabla\boldsymbol{v}\boldsymbol{F}^{-1})\boldsymbol{C}(\boldsymbol{F})^{-1}+\\ &[\mu-\frac{\lambda}{2}(\det\boldsymbol{F}-1)(\det\boldsymbol{F}+1)]\boldsymbol{C}(\boldsymbol{F})^{-1}[\boldsymbol{F}^T\nabla\boldsymbol{v}+\nabla\boldsymbol{v}^T\boldsymbol{F}]\boldsymbol{C}(\boldsymbol{F})^{-1}\end{aligned}$$

for the neo-Hookean material obeying (7.3). In the computation of the associated tangential stiffness matrix, which result from substituting the approximation (7.15) into the discrete version of the weak formulation of (7.1), the following self-adjointness property

$$\boldsymbol{F}_kD\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\nabla\boldsymbol{u}_{k+1}]\cdot\nabla\boldsymbol{v}+\nabla\boldsymbol{u}_{k+1}\boldsymbol{\Sigma}(\boldsymbol{F}_k)\cdot\nabla\boldsymbol{v}=\boldsymbol{F}_kD\boldsymbol{\Sigma}(\boldsymbol{F}_k)[\nabla\boldsymbol{v}]\cdot\nabla\boldsymbol{u}_{k+1}+\nabla\boldsymbol{v}\boldsymbol{\Sigma}(\boldsymbol{F}_k)\cdot\nabla\boldsymbol{u}_{k+1}$$

is also used.

When a nearly incompressible neo-Hookean material is considered, we need to linearize both the constitutive law $\boldsymbol{S}=\hat{\boldsymbol{S}}(\boldsymbol{F},p)=\boldsymbol{F}\boldsymbol{\Sigma}(\boldsymbol{F},p)$, with $\boldsymbol{\Sigma}(\boldsymbol{F},p)=\bar{\boldsymbol{\Sigma}}(\boldsymbol{F}^T\boldsymbol{F},p)$, and the constraint

$$\varphi(\boldsymbol{F},p)=0 \tag{7.16}$$

where

$$\varphi(\boldsymbol{F},p)=\varepsilon p+1/2(\det\boldsymbol{F})^2-1/2 \tag{7.17}$$

with

$$\varepsilon=1/\lambda. \tag{7.18}$$

The Newton updates related to solving (7.16) are given by

$$(\det\boldsymbol{F}_k)^2\,\mathrm{tr}[(\boldsymbol{F}_{k+1}-\boldsymbol{F}_k)\boldsymbol{F}_k^{-1}]+\varepsilon(p_{k+1}-p_k)=-\varphi(\boldsymbol{F}_k,p_k). \tag{7.19}$$

To obtain the Newton linearization of the stress response function, we note in particular that the derivative of the function $\boldsymbol{G}(\boldsymbol{F},p)=-p\boldsymbol{C}(\boldsymbol{F})^{-1}$ is given by

$$D\boldsymbol{G}(\boldsymbol{F},p)[(\boldsymbol{U},h)]=-h\boldsymbol{C}(\boldsymbol{F})^{-1}+p\boldsymbol{C}(\boldsymbol{F})^{-1}[\boldsymbol{F}^T\boldsymbol{U}+\boldsymbol{U}^T\boldsymbol{F}]\boldsymbol{C}(\boldsymbol{F})^{-1}.$$

It then follows that

$$D\boldsymbol{\Sigma}(\boldsymbol{F},p)[(\boldsymbol{U},h)]=-h\boldsymbol{C}(\boldsymbol{F})^{-1}+(p+\mu)\boldsymbol{C}(\boldsymbol{F})^{-1}[\boldsymbol{F}^T\boldsymbol{U}+\boldsymbol{U}^T\boldsymbol{F}]\boldsymbol{C}(\boldsymbol{F})^{-1}. \tag{7.20}$$

7.5 Linearization: The case of a user-defined material model

Other constitutive laws can be defined by including a user-defined material model in the form of an Abaqus user subroutine (UMAT). In contrast to the other constitutive laws, such subroutine is expected to return the Cauchy stress corresponding to the current estimate of the strain increment with respect to the previous converged solution (the solution before the time/load increment) as well as the derivative of the stress response function for computing the differential (7.10).

It should be noted that, despite relying on the Cauchy stress, the weak formulation is still generated from the equilibrium equations expressed in terms of the first Piola-Kirchhoff stress. The stress power calculations are thus done in terms of an energetically conjugate pair of the first Piola-Kirchhoff stress and the rate of change of the deformation gradient. To this end, in view of (7.7)–(7.9), we consider the mapping

$$\boldsymbol{F}\mapsto\hat{\boldsymbol{S}}(\boldsymbol{F})=(\det\boldsymbol{F})\{\bar{\boldsymbol{\sigma}}(\cdot,\mathbf{q})\circ\hat{\boldsymbol{E}}(\boldsymbol{F})\}\boldsymbol{F}^{-T} \tag{7.21}$$

and use (7.10) to obtain the Fréchet differential

$$\begin{aligned}D\hat{\boldsymbol{S}}(\boldsymbol{F})[\boldsymbol{U}]&=(\det\boldsymbol{F})\{D\bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F}),\mathbf{q})[D\hat{\boldsymbol{E}}(\boldsymbol{F})[\boldsymbol{U}]]\}\boldsymbol{F}^{-T}\\ &+(\det\boldsymbol{F})\mathrm{tr}(\boldsymbol{U}\boldsymbol{F}^{-1})\bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F}),\mathbf{q})\boldsymbol{F}^{-T}-(\det\boldsymbol{F})\bar{\boldsymbol{\sigma}}(\hat{\boldsymbol{E}}(\boldsymbol{F}),\mathbf{q})\boldsymbol{F}^{-T}\boldsymbol{U}^T\boldsymbol{F}^{-T}.\end{aligned} \tag{7.22}$$

In the differentiation the state variables are thus treated as parameters. In the case of a user-defined material model the equation (7.22) provides the key ingredient for linearization.

7.6 Stress and strain computation

In addition to solving for the displacement, the solver can produce the strain and stress fields associated with the solution. In this connection the strain tensor is defined by (7.4). In the stress computation the material description of the usual Cauchy stress $\boldsymbol{\sigma}$ is produced. That is, we measure the surface force per unit area in the deformed configuration and write $\boldsymbol{\sigma}_m(\mathbf{p}, t) = \boldsymbol{\sigma}(\mathbf{x}(\mathbf{p}, t), t)$. We note that this stress is related to one of the Piola-Kirchhoff stresses as

$$\boldsymbol{\sigma}_m = (\det \boldsymbol{F})^{-1} \boldsymbol{S} \boldsymbol{F}^T = (\det \boldsymbol{F})^{-1} \boldsymbol{F} \bar{\boldsymbol{\Sigma}}(\boldsymbol{C}) \boldsymbol{F}^T. \tag{7.23}$$

7.7 Keywords

`Simulation`

In specifying the keywords for the simulation section, note that all coordinate systems are not supported.

`Coordinate System` String

The coordinate system may be `Cartesian 2D`, `Cartesian 3D` or `Axi Symmetric`.

`Material` mat id

The following keywords relate to giving the material parameters for the finite elasticity solver.

`Density` Real

This keyword is used for defining the density field ρ_0 corresponding to the reference configuration.

`Poisson Ratio` Real

The values of the scalar Lame material parameters depend on the Poisson ratio as in the case of the linear elasticity solver. The Poisson ratio is given by using this keyword.

`Youngs Modulus` Real

The values of the scalar Lame material parameters depend on Young's modulus as in the case of the linear elasticity solver. This keyword specifies the value of Young's modulus.

`UMAT Subroutine` File

The value of this keyword consists of two string parameters. The first parameter specifies the name of a compiled file containing the definition of a user-defined material model in the UMAT form. The second parameter defines the name of the user-defined subroutine. If this keyword is used, all material models must be defined in the UMAT form.

`Number of Material Constants` Integer

The value of this keyword defines the number of material constants that are passed to the UMAT subroutine.

`Material Constants` Real

The values of the material constants passed to the UMAT subroutine.

`Number of State Variables` Integer

This keyword is used to declare the number of state variables.

`Name` String

This keyword may be used to define the name argument of the UMAT subroutine.

`Equation` eq id

`Plane Stress` Logical

If the coordinate system is chosen to be `Cartesian 2D`, this keyword may be used to activate nonlinear plane stress analysis. In the case of plane stress the definition of the Lame parameter λ is altered such that the plane stress components are directly obtained in terms of the plane strain components. The strain E_{33} can then be expressed as $E_{33} = -\nu/(1-\nu)(E_{11} + E_{22})$.

`Solver` `solver id`

`Equation` `String [ElasticSolver]`
: A describing name for the solver. This can be changed but it must be given,

`Procedure` `File "ElasticSolve" "ElasticSolver"`
: Name of the solver subroutine.

`Neo-Hookean Material` `Logical`
: By default the constitutive law (7.2) is employed. Switching to the neo-Hookean material model (7.3) can be performed by giving the value `True` for this keyword.

`Mixed Formulation` `Logical`
: This keyword is used to handle incompressible or nearly incompressible material obeying the neo-Hookean constitutive law. If the value `True` is given for this keyword, the field p is taken to be an additional unknown which is solved under the constraint (7.5). In this case the solver assumes that the mesh files correspond to the lowest-order finite elements (the lowest-order pressure approximation together with the second-order displacement approximation is then constructed by default). In addition, the default names for the displacement variable $\boldsymbol{u}$ and pressure variable p are then `Disp` and `Pres`, respectively.

`Calculate Strains` `Logical`
: If the value `True` is given for this keyword, the strains are also computed. The strain components are output into an ordered six-tuple as $(E_{xx}\ E_{yy}\ E_{zz}\ E_{xy}\ E_{yz}\ E_{xz})$. However, in the axially symmetric simulation only four components are produced as $(E_{xx}\ E_{zz}\ E_{yy}\ E_{xy})$, with the convention $x = r$ and $z = \theta$.

`Calculate Stresses` `Logical`
: If the value `True` is given for this keyword, the Cauchy stress (7.23) is also computed. The stress components are output into an ordered six-tuple in the same way as the strain.

`Calculate Principal` `Logical`
: If the strain or stress computation is activated, this keyword can be used to activate the computation of principal components.

`Calculate PAngle` `Logical`
: This keyword can be used to activate the computation of the principal angles for the stress tensor. If the value `True` is given for this keyword, then the computation of principal components is also activated.

`Initialize State Variables` `Logical`
: If the material model is defined in terms of a user-defined material model (UMAT), an extra call of the UMAT subroutine can be done to obtain the state variables in the initial state (stress-free initial condition is supposed).

`Body Force` `bf id`

This section may be used to define body forces.

`Inertial Bodyforce j` `Real`
: This keyword may be used to give the component j of the body force $\boldsymbol{b}(\mathbf{x}, t)$ in order to define $\boldsymbol{b}_0 = \rho_0(\mathbf{p})\boldsymbol{b}(\mathbf{x}(\mathbf{p}, t), t)$. It is noted that in this case $\boldsymbol{b}(\mathbf{x}, t)$ defines the body force per unit mass. The density changes are then considered correctly , i.e. the condition $\rho(\mathbf{x}(\mathbf{p}, t), t) \det \boldsymbol{F}(\mathbf{p}, t) = \rho_0(\mathbf{p})$ is respected.

`Stress Bodyforce j` `Real`
: This keyword may be used to give the component j of the body force $\boldsymbol{b}(\mathbf{x}, t)$ in order to define $\boldsymbol{b}_0 = (\det \boldsymbol{F})\boldsymbol{b}(\mathbf{x}(\mathbf{p}, t), t)$. It is noted that $\boldsymbol{b}(\mathbf{x}, t)$ is now the body force per unit volume of the deformed body, so this type of force is appropriate for specifying true volumetric forces, whatever they might be.

`Boundary Condition` `bc id`

The Dirichlet conditions (7.11) for the displacement variable of the solver can be given in the standard manner. Other options for defining boundary conditions are explained in the following.

`Normal Surface Traction` `Real`

A surface force which is normal to the deformed boundary and gives force per unit area of the deformed surface may be given with this keyword.

`Surface Traction k` `Real`

By default this keyword may be used to give the actual force per unit area of the deformed surface. The value of this keyword then specifies the component k of $\hat{\boldsymbol{s}}$ in (7.13). If the keyword command `Pseudo-Traction = True` is also given, then the values of this keyword command are used to determine the components of the pseudo-traction vector $\hat{\boldsymbol{s}}^0$ which gives the actual force per unit undeformed area.

`Pseudo-Traction` `Logical`

If this keyword has the value `True`, then the surface force is defined via the pseudo-traction condition; see the explanation of the keyword `Surface Traction k` below.

`Spring` `Real`

This keyword can be used to generate a reaction force which is aligned with the normal direction of the undeformed configuration and which is proportional to the displacement in the normal direction.

`Spring i` `Real`

This keyword is similar to the keyword `Spring` but here the spring coefficients are defined with respect to the coordinate axes.

`FSI BC` `Logical`

If this keyword has the value `True`, then the Navier–Stokes flow solution is used to determine the surface force generated by the flow.

Model 8

Shell Equations of Classical Elasticity

Module name: ShellSolver
Module subroutines: ShellSolver
Module authors: Mika Malinen
Document authors: Mika Malinen

8.1 Introduction

This chapter is concerned with the equations which describe deformations of thin elastic shells. Here a shell refers to a curved three-dimensional body which can be described in terms of its mid-surface and thickness (the extent of the shell in the direction of normal to the mid-surface). When the placement of the shell in its reference configuration is described by using a system of normal coordinates (that is, two coordinate curves on the mid-surface are perpendicular to the third coordinate curve), simplifications to solving 3-D elasticity equations can be sought via the process of dimensional reduction, so that unknowns that depend only on the two curvilinear coordinates associated with the shell mid-surface can be employed. Classical shell theory [3] is dedicated to the study of such models by assuming that the exact parametrization of the shell mid-surface is known in advance.

However, classical shell theory cannot often be applied in a straightforward manner in connection with finite element modelling, since in practice the mapping giving the mid-surface is not usually available in an explicit form. To offer generality, the shell solver described here is also able to create a computational surface model by assuming that information about the surface position and the director vector (the unit normal to the exact mid-surface) are given at the nodes of a background mesh. Elementwise approximations of the mid-surface are then created such that the surface position and the normal to the approximate surface agree with the data given at the nodes. It should be noted that the approximate mid-surface obtained in this way generally gives a more accurate description of the surface position than what would be obtained by using the standard strategy where straightforward Lagrange interpolation merely based on the nodal position data is employed. Here the approximation of the director field is then derived via straightforward differentiation of the mapping giving the approximate surface (it should be noted that this approximation is consistent with the given data at the nodes). As an alternate to the internal surface reconstruction, a higher-order nodal mesh can be used to derive the necessary geometric data without expecting director data.

In the first place each (physical) element S of the approximate mid-surface is originally parametrized in terms of the rectangular Cartesian coordinates of points of a usual reference element (a square or an equilateral triangle). However, the solver considered performs additional computation in order to find a convenient elementwise reparameterization by lines of curvature coordinates, so that we may write $S = \boldsymbol{\varphi}_K(K)$ where $\boldsymbol{\varphi}_K : K \subset \mathbb{R}^2 \to \mathbf{E}^3$ and the rectangular Cartesian coordinates y^α of a point $\mathbf{y}$ in K correspond to lines of curvature coordinates on S (for the basic concepts related to lines of curvature, see [2]). Then each point on the surface can naturally be associated with an orthogonal system of basis vectors which offer a convenient starting point for representing vector-valued fields over the surface. Since the basis is orthogonal, the components of a vector field have intuitive physical significance and tensor calculations related to the shell equations are greatly simplified in comparison with the case of general curvilinear coordinates.

It is mentioned that this solver can also be adapted to handle special cases where the exact parametrization of the shell mid-surface by lines of curvature coordinates can be given globally. In this case, the approach of classical shell theory is followed, so the mathematical formulation is done over a two-dimensional (planar) reference domain.

8.2 Discrete shell model

The shell models we consider employ kinematic assumptions that enable the approximation of 3-D elasticity equations without additional assumptions about the state of stress. The simplest assumption conforms with using solid finite elements which have nodes located on the upper and lower surfaces of the shell, together with auxiliary degrees of freedom to enable a normal strain field that depends linearly on the normal coordinate. Conventional 2-D shell models can then be derived by imposing the condition of vanishing normal stress, but use of a refined shell model can also be considered in this setting.

8.2.1 Preliminaries

We may now think of a physical surface element $S = \boldsymbol{\varphi}_K(K)$ to be associated with the physical solid element $\Omega_S \subset \mathbf{E}^3$ which is the image of the set $S \times [-d/2, d/2]$ under a mapping of normal coordinates

$$(\mathbf{p}, y^3) \mapsto \mathbf{p} + y^3(\boldsymbol{a}_3 \circ \boldsymbol{\varphi}_K^{-1})(\mathbf{p}), \tag{8.1}$$

with d being the shell thickness and the surface function $\mathbf{p} \mapsto (\boldsymbol{a}_3 \circ \boldsymbol{\varphi}_K^{-1})(\mathbf{p})$ giving the unit normal to the mid-surface at a point $\mathbf{p} \in S$ in terms of the vector field $\boldsymbol{a}_3 : K \to \mathbb{R}^3$. Since the normal coordinates $(\mathbf{p}, y^3) \in S \times [-d/2, d/2]$ thus identify a point in the physical element Ω_S, it is natural to approximate the elementwise restriction of the displacement vector field of the shell

$$\boldsymbol{u} : S \times [-d/2, d/2] \to \mathbb{R}^3, \quad (\mathbf{p}, y^3) \mapsto \boldsymbol{u}(\mathbf{p}, y^3), \tag{8.2}$$

in a systematic manner such that

$$\boldsymbol{u}(\boldsymbol{\varphi}_K(\mathbf{y}), y^3) \equiv \hat{\boldsymbol{u}}(\mathbf{y}, y^3) \equiv \boldsymbol{v}^{(0)}(\mathbf{y}) - y^3\boldsymbol{v}^{(1)}(\mathbf{y}) - \frac{1}{2}(y^3)^2\boldsymbol{v}^{(2)}(\mathbf{y}), \tag{8.3}$$

with the vector fields $\boldsymbol{v}^{(k)} : K \to \mathbb{R}^3$ in two variables being taken as unknowns. The mathematical domain of definition for three-dimensional shell variables will thus be the set $\Omega_K = K \times [-d/2, d/2]$ whose points are mapped to the points of the physical space $\mathbf{E}^3$ as

$$(\mathbf{y}, y^3) \mapsto \boldsymbol{\Theta}(\mathbf{y}, y^3) \equiv \boldsymbol{\varphi}_K(\mathbf{y}) + y^3\boldsymbol{a}_3(\mathbf{y}). \tag{8.4}$$

This representation of geometry follows by writing an alternate referential description of the normal coordinates representation defined by (8.1).

Each physical point $\boldsymbol{\Theta}(\mathbf{y}, y^3)$ of the shell can be associated with three vectors

$$\boldsymbol{g}_k(\mathbf{y}, y^3) = D\boldsymbol{\Theta}(\mathbf{y}, y^3)[\hat{\boldsymbol{e}}_k] = \partial_k\boldsymbol{\Theta}(\mathbf{y}, y^3), \tag{8.5}$$

with $\hat{\boldsymbol{e}}_k$ being the orthonormal basis vectors associated with the mathematical domain of definition, to give a covariant basis for the translation space $\mathbb{R}^3$ of $\mathbf{E}^3$. Similarly, by restricting to the mid-surface, we define a set of surface basis vectors $\boldsymbol{a}_i : K \to \mathbb{R}^3$, which give the covariant basis at $\mathbf{p} = \boldsymbol{\varphi}_K(\mathbf{y})$, via

$$\boldsymbol{a}_\alpha(\mathbf{y}) = D\boldsymbol{\varphi}_K(\mathbf{y})[\hat{\boldsymbol{e}}_\alpha] = \partial_\alpha\boldsymbol{\varphi}_K(\mathbf{y}), \quad \boldsymbol{a}_3(\mathbf{y}) \cdot \boldsymbol{a}_\alpha(\mathbf{y}) = 0. \tag{8.6}$$

The covariant components of the metric surface tensor $\boldsymbol{A}$ (the first fundamental form) are now given by

$$A_{\alpha\beta}(\mathbf{y}) = \boldsymbol{a}_\alpha(\mathbf{y}) \cdot \boldsymbol{a}_\beta(\mathbf{y}). \tag{8.7}$$

We also define $B_{\alpha\beta} : K \to \mathbb{R}$ by

$$B_{\alpha\beta}(\mathbf{y}) = \boldsymbol{a}_3(\mathbf{y}) \cdot \partial_\alpha\boldsymbol{a}_\beta(\mathbf{y}) = -\boldsymbol{a}_\alpha(\mathbf{y}) \cdot \partial_\beta\boldsymbol{a}_3(\mathbf{y}). \tag{8.8}$$

They give the covariant components of the second fundamental form $\boldsymbol{B}$ of the surface at $\mathbf{p} = \boldsymbol{\varphi}_K(\mathbf{y})$. We shall also need the contravariant basis vectors $\boldsymbol{a}^i$ of the surface satisfying the orthogonality conditions

$$\boldsymbol{a}_i(\mathbf{y}) \cdot \boldsymbol{a}^j(\mathbf{y}) = \delta_i^j, \tag{8.9}$$

with δ_i^j being the Kronecker's symbol.

When lines of curvature coordinates are used, the two sets of basis vectors are related by

$$\boldsymbol{g}_1(\mathbf{y}, y^3) = \mu_1^1(\mathbf{y}, y^3)\boldsymbol{a}_1(\mathbf{y}), \quad \boldsymbol{g}_2(\mathbf{y}, y^3) = \mu_2^2(\mathbf{y}, y^3)\boldsymbol{a}_2(\mathbf{y}), \quad \boldsymbol{g}_3(\mathbf{y}, y^3) = \boldsymbol{a}_3(\mathbf{y}) \tag{8.10}$$

where μ_α^β are the mixed components of a diagonal (shifter) tensor. We then have

$$\mu_1^1(\mathbf{y}, y^3) = 1 + y^3/R_1(\mathbf{y}), \quad \mu_2^2(\mathbf{y}, y^3) = 1 + y^3/R_2(\mathbf{y}) \tag{8.11}$$

with

$$\frac{1}{R_1(\mathbf{y})} = -\frac{B_{11}(\mathbf{y})}{A_{11}(\mathbf{y})} = -B_1^1(\mathbf{y}) \quad \text{and} \quad \frac{1}{R_2(\mathbf{y})} = -\frac{B_{22}(\mathbf{y})}{A_{22}(\mathbf{y})} = -B_2^2(\mathbf{y}) \tag{8.12}$$

being the principal curvatures. The sign convention is here chosen such that the principal radii of curvature $R_\alpha > 0$ if the normal vector is directed away from the centre of curvature.

Two different kinematic assumptions in the form (8.3) can be chosen. The most general version of (8.3) leads to handling nine scalar fields as unknowns. The simplest kinematic assumption, which does not necessitate introducing an additional assumption about the state of stress, however corresponds to the choice where the part $\boldsymbol{v}^{(2)}$ can be expressed simply as

$$\boldsymbol{v}^{(2)}(\mathbf{y}) = [\boldsymbol{v}^{(2)}(\mathbf{y}) \cdot \boldsymbol{a}_3(\mathbf{y})]\boldsymbol{a}^3(\mathbf{y}), \tag{8.13}$$

i.e. only one scalar field describes the second-order part with respect to y^3. The resulting kinematic assumption thus involves seven scalar fields (for original contributions that use this restriction, see historical notes in [3] and papers by Reissner and Naghdi mentioned there). The assumption that only one quadratic component in y^3 is included is motivated by having the ability to expand all components of linearized strain tensor up to the first-order terms in y^3. In practice, especially in the context of linear theory, this choice allows the derivation of a variational formulation which does not necessitate solving the part (8.13) as tightly coupled with the other unknowns, i.e. its values can be found afterwards when $\boldsymbol{v}^{(k)}$, $k = 0, 1$, have first been solved.

8.2.2 The measure of strain

The measure of three-dimensional strain we employ is based on the Green-St Venant strain tensor field $\boldsymbol{E}(\hat{\boldsymbol{u}}) : \Omega_K \to \mathrm{Sym}$ associated with the elementwise restriction $\hat{\boldsymbol{u}}$ of the displacement field. Its components measure the change of the metric tensor associated with the displacement field and are defined such that

$$2\bar{E}_{ij}(\hat{\boldsymbol{u}})(\cdot) = \boldsymbol{g}_i(\cdot) \cdot \partial_j \hat{\boldsymbol{u}}(\cdot) + \partial_i \hat{\boldsymbol{u}}(\cdot) \cdot \boldsymbol{g}_j(\cdot) + [\partial_i \hat{\boldsymbol{u}}(\cdot)] \cdot [\partial_j \hat{\boldsymbol{u}}(\cdot)]. \tag{8.14}$$

In practice, we shall perform a change of basis in order to express the strain tensor field as

$$\boldsymbol{E}(\hat{\boldsymbol{u}})(\mathbf{y}, y^3) = E_{ij}(\hat{\boldsymbol{u}})(\mathbf{y}, y^3)\boldsymbol{a}^i(\mathbf{y}) \otimes \boldsymbol{a}^j(\mathbf{y}),$$

so that the components are then expressed with respect to the surface basis vectors depending only on the two curvilinear coordinates of the mid-surface. Alternatively, the most basic representation of strain follows by switching to a local orthonormal basis $\{\boldsymbol{e}_1(\mathbf{y}), \boldsymbol{e}_2(\mathbf{y}), \boldsymbol{e}_3(\mathbf{y})\} = \{\boldsymbol{e}^1(\mathbf{y}), \boldsymbol{e}^2(\mathbf{y}), \boldsymbol{e}^3(\mathbf{y})\}$ obtained as

$$\boldsymbol{e}_\alpha(\mathbf{y}) = \frac{\boldsymbol{a}_\alpha(\mathbf{y})}{A_\alpha(\mathbf{y})}, \quad \boldsymbol{e}_3(\mathbf{y}) = \boldsymbol{a}_3(\mathbf{y}), \quad \text{with } A_\alpha(\mathbf{y}) = |\boldsymbol{a}_\alpha(\mathbf{y})|, \tag{8.15}$$

and writing then

$$\boldsymbol{E}(\hat{\boldsymbol{u}})(\mathbf{y}, y^3) = \hat{E}_{ij}(\hat{\boldsymbol{u}})(\mathbf{y}, y^3)\boldsymbol{e}^i(\mathbf{y}) \otimes \boldsymbol{e}^j(\mathbf{y}).$$

The components of these representations can be shown to obey the following transformation rules (here the summation convention is adopted so that a repeated index simultaneously appearing as a subscript and as a superscript in a term imply summation over all possible values, with a Greek index being however allowed to have values in $\{1, 2\}$)

$$\hat{E}_{\alpha\beta}(\hat{\boldsymbol{u}}) = \frac{E_{\alpha\beta}(\hat{\boldsymbol{u}})}{A_\alpha A_\beta} = \frac{(\mu^{-1})^\lambda_\alpha (\mu^{-1})^\nu_\beta \bar{E}_{\lambda\nu}(\hat{\boldsymbol{u}})}{A_\alpha A_\beta}, \quad \hat{E}_{\alpha 3}(\hat{\boldsymbol{u}}) = \frac{E_{\alpha 3}(\hat{\boldsymbol{u}})}{A_\alpha} = \frac{(\mu^{-1})^\nu_\alpha \bar{E}_{\nu 3}(\hat{\boldsymbol{u}})}{A_\alpha},$$
$$\hat{E}_{33}(\hat{\boldsymbol{u}}) = E_{33}(\hat{\boldsymbol{u}}) = \bar{E}_{33}(\hat{\boldsymbol{u}}).$$

By letting $\mathbf{U} = (v_1, v_2, \ldots)$ denote an n-tuple of 2-D scalar fields which determine $\hat{\boldsymbol{u}}$, we next expand the components of the strain tensor in powers of the normal coordinate y^3 to obtain the first-order approximation

$$E_{ij}(\hat{\boldsymbol{u}}(\mathbf{U}))(\mathbf{y}, y^3) = \gamma_{ij}(\mathbf{U})(\mathbf{y}) + \epsilon_{ij}(\mathbf{U})(\mathbf{y}) - y^3\kappa_{ij}(\mathbf{U})(\mathbf{y}) - y^3\chi_{ij}(\mathbf{U})(\mathbf{y}) \tag{8.16}$$

where the 2-D fields $\gamma_{ij}(\mathbf{U}) : K \to \mathbb{R}$ and $\kappa_{ij}(\mathbf{U}) : K \to \mathbb{R}$ are linear with respect to $\mathbf{U}$, while $\epsilon_{ij}(\mathbf{U}) : K \to \mathbb{R}$ and $\chi_{ij}(\mathbf{U}) : K \to \mathbb{R}$ are nonlinear. If the shell undergoes only small deflections, linearization can additionally be performed by omitting the nonlinear terms to write then

$$E_{ij}(\hat{\boldsymbol{u}}(\mathbf{U})) = \gamma_{ij}(\mathbf{U}) - y^3\kappa_{ij}(\mathbf{U}).$$

We note that the tangent plane components $\gamma_{\alpha\beta}(\mathbf{U})$ and $\kappa_{\alpha\beta}(\mathbf{U})$ constitute the so-called membrane strain and bending strain tensors, while $\gamma_{\alpha 3}(\mathbf{U})$ are the transverse shear strains.

When we simplify the notation by setting

$$\boldsymbol{v}^{(0)} \equiv \boldsymbol{v}, \quad \boldsymbol{v}^{(1)} \equiv \boldsymbol{\beta}, \quad \boldsymbol{v}^{(2)} \equiv \boldsymbol{\psi}, \tag{8.17}$$

the components of the linearized strain tensors have the following component-free representations (the component-freeness should be understood with respect to the displacements)

$$\begin{aligned}
2\gamma_{\alpha\beta}(\mathbf{U}) &= \boldsymbol{a}_\alpha \cdot \partial_\beta \boldsymbol{v} + \partial_\alpha \boldsymbol{v} \cdot \boldsymbol{a}_\beta, \\
2\gamma_{\alpha 3}(\mathbf{U}) &= \boldsymbol{a}_3 \cdot \partial_\alpha \boldsymbol{v} - \boldsymbol{a}_\alpha \cdot \boldsymbol{\beta}, \\
\gamma_{33}(\mathbf{U}) &= -\boldsymbol{\beta} \cdot \boldsymbol{a}_3, \\
\kappa_{\alpha\alpha}(\mathbf{U}) &= \boldsymbol{a}_\alpha \cdot \partial_\alpha \boldsymbol{\beta} - \kappa_\alpha \gamma_{\alpha\alpha}, \\
2\kappa_{12}(\mathbf{U}) &= \boldsymbol{a}_1 \cdot \partial_2 \boldsymbol{\beta} + \partial_1 \boldsymbol{\beta} \cdot \boldsymbol{a}_2 - \kappa_2 \boldsymbol{a}_1 \cdot \partial_2 \boldsymbol{v} - \kappa_1 \partial_1 \boldsymbol{v} \cdot \boldsymbol{a}_2, \\
2\kappa_{\alpha 3}(\mathbf{U}) &= \boldsymbol{a}_3 \cdot \partial_\alpha \boldsymbol{\beta} - \kappa_\alpha \boldsymbol{a}_3 \cdot \partial_\alpha \boldsymbol{v} + \boldsymbol{\psi} \cdot \boldsymbol{a}_\alpha, \\
\kappa_{33}(\mathbf{U}) &= \boldsymbol{\psi} \cdot \boldsymbol{a}_3,
\end{aligned} \tag{8.18}$$

with $\kappa_1 \equiv B^1_1$ and $\kappa_2 \equiv B^2_2$. On the other hand, the nonlinear parts of the strain tensors can be expressed as

$$\begin{aligned}
2\epsilon_{\alpha\beta}(\mathbf{U}) &= \partial_\alpha \boldsymbol{v} \cdot \partial_\beta \boldsymbol{v}, \\
2\epsilon_{\alpha 3}(\mathbf{U}) &= -\boldsymbol{\beta} \cdot \partial_\alpha \boldsymbol{v}, \\
\epsilon_{33}(\mathbf{U}) &= 1/2\boldsymbol{\beta} \cdot \boldsymbol{\beta}, \\
2\chi_{\alpha\beta}(\mathbf{U}) &= \partial_\alpha \boldsymbol{v} \cdot \partial_\beta \boldsymbol{\beta} + \partial_\alpha \boldsymbol{\beta} \cdot \partial_\beta \boldsymbol{v} - (\kappa_1 + \kappa_2)\partial_\alpha \boldsymbol{v} \cdot \partial_\beta \boldsymbol{v}, \\
2\chi_{\alpha 3}(\mathbf{U}) &= -\boldsymbol{\beta} \cdot \partial_\alpha \boldsymbol{\beta} + \boldsymbol{\psi} \cdot \partial_\alpha \boldsymbol{v} + \kappa_\alpha \boldsymbol{\beta} \cdot \partial_\alpha \boldsymbol{v}, \\
\chi_{33}(\mathbf{U}) &= -\boldsymbol{\beta} \cdot \boldsymbol{\psi}.
\end{aligned} \tag{8.19}$$

It should be noted that under our kinematic assumptions the expression for the linearized normal strain is precisely

$$\gamma_{33}(\mathbf{U}) - y^3\kappa_{33}(\mathbf{U}) = -\beta_3 - y^3\psi_3, \tag{8.20}$$

so we cannot generally proceed beyond linear terms in y^3 in the expansions of strains. This generally motivates our choice to truncate the expressions for strains as done in (8.16). In addition, the terms of type $B^\alpha_\beta \gamma_{\alpha k}(\mathbf{U})$ in the expressions for $\kappa_{ij}(\mathbf{U})$ are expected to have little impact on the strain energy of a thin shell and could therefore be omitted to obtain simplified expressions.

It is notable that the derivatives of $\boldsymbol{\psi}$ do not occur in the expressions for strains when only the terms up to the first order with respect to y^3 are taken into account. When weak solutions of shell equations are sought, the components of $\boldsymbol{\psi}$ are thus seen to be exceptional in that less regularity can be supposed. In addition, in the context of linear theory together with the restriction (8.13), $\boldsymbol{\psi}$ enters only through its normal component ψ_3 in the expression for $\kappa_{33}(\mathbf{U})$. In this case, a variational formulation of the shell problem may be obtained such that it does not necessitate solving ψ_3 as tightly coupled with the other unknowns, i.e. its values can be found afterwards when $\boldsymbol{v}^{(k)}$, $k = 0, 1$, have first been solved. In the context of nonlinear theory such simplification cannot be attained in a fully consistent manner and using the nine-component shell model may be a more natural choice. If the nine-component model is not employed and nonlinearities are taken into account, the shell solver neglects the incomplete part $\boldsymbol{\psi} \cdot \partial_\alpha \boldsymbol{v}$ in the expression for $\chi_{\alpha 3}(\mathbf{U})$, so that a formulation in terms of $\boldsymbol{v}^{(k)}$, $k = 0, 1$, can be obtained.

8.2.3 The principle of virtual work

To simplify the statement of the principle of virtual work, we now shorten the expressions for the strain components by omitting the splitting into the linear and nonlinear parts, so that

$$\boldsymbol{E}(\hat{\boldsymbol{u}}(\mathbf{U})) = \boldsymbol{\varepsilon}(\mathbf{U}) - y^3 \boldsymbol{\rho}(\mathbf{U}) \tag{8.21}$$

with

$$\begin{aligned} \varepsilon_{ij}(\mathbf{U}) &\equiv \gamma_{ij}(\mathbf{U}) + \epsilon_{ij}(\mathbf{U}), \\ \rho_{ij}(\mathbf{U}) &\equiv \kappa_{ij}(\mathbf{U}) + \chi_{ij}(\mathbf{U}). \end{aligned} \tag{8.22}$$

Currently the shell solver can handle only a nonlinear extension of the standard constitutive law for an isotropic material characterized by Young's modulus E and Poisson's ratio ν. That is, we assume that

$$\hat{\Sigma}^{ij}(\cdot) = \frac{\nu E}{(1+\nu)(1-2\nu)}[\hat{E}_{11}(\hat{\boldsymbol{u}})(\cdot) + \hat{E}_{22}(\hat{\boldsymbol{u}})(\cdot) + \hat{E}_{33}(\hat{\boldsymbol{u}})(\cdot)]\delta_{ij} + \frac{E}{1+\nu}\hat{E}_{ij}(\hat{\boldsymbol{u}})(\cdot) \tag{8.23}$$

where the scalar fields $\hat{\Sigma}^{ij} : \Omega_K \to \mathbb{R}$ are the components of the second Piola-Kirchhoff stress with respect to the orthonormal basis. This model is expected to be feasible when the stretches of the shell remain relatively small, while rigid body deformations of arbitrary magnitude are possible.

The statement of the three-dimensional principle of virtual work can now be put into the form

$$D\mathcal{U}(\mathbf{U})[\mathbf{V}] = \mathcal{Q}(\mathbf{U}, \mathbf{V})$$

where $D\mathcal{U}(\mathbf{U})[\mathbf{V}]$ gives the derivative of strain energy and $\mathcal{Q}(\mathbf{U}, \cdot)$ is a linear functional determined by loads. We note that the linear functional does not always depend on the solution $\mathbf{U}$. In that case we could simplify the statement of the principle of virtual work as

$$D\mathcal{U}(\mathbf{U})[\mathbf{V}] = \mathcal{L}(\mathbf{V}),$$

with $\mathcal{L}(\cdot)$ being a linear functional independent of $\mathbf{U}$. The strain energy is expressed elementwise as

$$\mathcal{U}_K(\mathbf{U}) = \int\limits_{\Omega_K} W(\boldsymbol{E}(\hat{\boldsymbol{u}}(\mathbf{U}))(\mathbf{y}, y^3)) \sqrt{g(\mathbf{y}, y^3)} \mathrm{d}\mathbf{y}\mathrm{d}y^3, \tag{8.24}$$

with $W : \mathrm{Sym} \to \mathbb{R}$ giving the strain-energy density and $g(\mathbf{y}, y^3)$ denoting the determinant of three-dimensional metric tensor. The contribution to the principle of virtual work can then be expressed as

$$\int\limits_{\Omega_K} \boldsymbol{\mathcal{R}}(\boldsymbol{E}(\hat{\boldsymbol{u}}(\mathbf{U}))(\mathbf{y}, y^3)) \cdot D\boldsymbol{E}(\hat{\boldsymbol{u}}(\mathbf{U}))[\mathbf{V}](\mathbf{y}, y^3) \sqrt{g(\mathbf{y}, y^3)}\, \mathrm{d}\Omega_K = \mathcal{Q}_K(\mathbf{U}, \mathbf{V}) \tag{8.25}$$

for all kinematically admissible $\mathbf{V}$, with the second Piola-Kirchhoff stress field $\boldsymbol{\Sigma}(\cdot) = \boldsymbol{\mathcal{R}}(\boldsymbol{E}(\hat{\boldsymbol{u}}(\mathbf{U}))(\cdot))$ being found as the derivative $\boldsymbol{\mathcal{R}}(\boldsymbol{E}) \cdot \boldsymbol{H} = DW(\boldsymbol{E})[\boldsymbol{H}]$.

A truly two-dimensional variational formulation follows by using (8.21) in connection with (8.24) and performing the integration over the thickness of the shell. In the integration, we have chosen to neglect terms of $O(d/R_\alpha)$ in order to simplify the final statement of the 2-D version of the principle of virtual work. When the restriction (8.13) is employed, after some reduction the strain energy over K may be expressed as

$$\begin{aligned}
\mathcal{U}_K(\mathbf{U}) =& \frac{Ed}{2(1-\nu^2)} \int\limits_K \Big\{\nu[\varepsilon_{11}(\mathbf{U}) + \varepsilon_{22}(\mathbf{U})]^2 + (1-\nu) \sum_{\alpha,\beta=1}^{2} [\varepsilon_{\alpha\beta}(\mathbf{U})]^2\Big\}\sqrt{a}\, dK \\
&+ \frac{Ed(1-\nu)}{2(1+\nu)(1-2\nu)} \int\limits_K \Big\{\varepsilon_{33}(\mathbf{U}) + \frac{\nu}{1-\nu}[\varepsilon_{11}(\mathbf{U}) + \varepsilon_{22}(\mathbf{U})]\Big\}^2 \sqrt{a}\, dK \\
&+ \frac{Ed}{1+\nu} \int\limits_K [(\varepsilon_{13}(\mathbf{U}))^2 + (\varepsilon_{23}(\mathbf{U}))^2]\sqrt{a}\, dK \\
&+ \frac{Ed^3}{24(1-\nu^2)} \int\limits_K \Big\{\nu[\rho_{11}(\mathbf{U}) + \rho_{22}(\mathbf{U})]^2 + (1-\nu) \sum_{\alpha,\beta=1}^{2} [\rho_{\alpha\beta}(\mathbf{U})]^2\Big\}\sqrt{a}\, dK \\
&+ \frac{Ed^3}{12(1+\nu)} \int\limits_K [(\rho_{13}(\mathbf{U}))^2 + (\rho_{23}(\mathbf{U}))^2]\sqrt{a}\, dK
\end{aligned} \tag{8.26}$$

with a being the determinant of the metric surface tensor.

8.2.4 Formulation in terms of the Cartesian components

The component-free expressions for strains (the component-freeness should be understood with respect to the displacements) enable the approximation of shell equations directly in terms of the orthogonal Cartesian components of the vector fields (8.17). The variables of the shell solver are therefore the Cartesian components with respect to the global frame.

8.2.5 Strain reduction operators vs. high-order approximations

Efficient and reliable finite element discretization of shell equations is a challenging task and unsettled questions still remain. Shell problems can exhibit different types of asymptotic behaviour when the shell thickness tends to zero [1]. A great challenge for a finite element designer is to work out a formulation which works for all possible asymptotic scenarios. An ultimate challenge would be to accompany the method with a mathematical error analysis covering the full versatility of shell problems.

For the above reasons applying standard finite elements is not an option unless basis functions of high degree (the p-version of FEM) are used, as standard low-order methods are not suitable for approximating fields with negligible membrane and transverse shear strains (such case corresponds a bending-dominated asymptotic behaviour). This relates to a computational trouble known as finite element locking. To obtain better low-order methods, the shell solver of Elmer employs strain reduction operators which are applied to the membrane and transverse shear strains and designed, in the first place, to relax constraints that arise in the case of bending-dominated problems. Currently the strain reduction operators have been worked out only for the lowest-order approximation (a 3-node triangle or 4-node quadrilateral) under the restriction (8.13).

Approximation with hierarchic p-elements can also be applied, but also in this case the computational surface model is based on the surface reconstruction derived from information about the surface position and the director vector at the nodes of a background mesh. At the moment the internal surface reconstruction is based on mapped polynomials of degree 3, so the accuracy is not expected to be optimal for all element orders, since geometric errors eventually dominate the error. Numerical over-stiffness due to locking can however be treated effectively by using finite elements of a sufficiently high order, so even a non-optimal approximation combination might improve the solution if the dominating error is caused by locking. Note that the high-order version does not apply any strain reduction operators to alleviate locking, so with this approach increasing the polynomial order is the only way to handle locking. The p-approximation does not necessitate using the restriction (8.13).

8.3 Specifying surface data

In order to improve the approximation of the shell mid-surface, the shell solver needs information about the director vector. As alternatives the nodal director data can be read from a file, or it can be associated with an ordinary Elmer variable "Director" that has been solved before executing the shell solver. It should be noted that parallel versions of file formats used in connection with reading from files do not exist yet, so at the moment parallel computation is possible only when the director field is made available as the Elmer variable "Director".

The nodal director data can be formatted into special data files in two ways. The first option is to write a file mesh.director which lists the director at the nodes in a similar way as nodes are defined in the file mesh.nodes. That is, the contents of the file mesh.director should be organized as

```
n1 dx dy dz
n2 dx dy dz
 ...
nn dx dy dz
```

The first integer is the identification number for the node followed by three real numbers which are the components of the director with respect to the global coordinate frame. The file should be located in the same place as the standard mesh files.

The second option is to provide a file mesh.elements.data which should define the nodal director in elementwise manner and associate the name 'director' with this data. Thus, if just the director data is given, the contents of the file should be arranged as

```
element: element_id_1
director: dx_1 dy_1 dz_1 ... dx_n dy_n dz_n
end
element: ...
...
end
```

Here the nodewise ordering of the director data on lines starting with director: must correspond to that of the mesh.elements file. Also this file should be located in the same place as the standard mesh files. It is noted that a file mesh.elements.data is considered first in priority. Optionally, given a file mesh.director, the solver can write the director data as elementwise property to a file whose format conforms with a file mesh.elements.data. The elementwise data contained in mesh.elements.data may generally be discontinuous over adjacent finite elements.

One way to create the Elmer variable "Director" is to utilize the solver NormalSolver that uses the background mesh for the computation of the normal vector. This approach can compromise the accuracy of the geometry model but allows parallel computation. For special cases where the dependency of the director on the global coordinates is known, a slight modification of the NormalSolver module might be enough for obtaining both an accurate approximation of the director and having the option to run the shell solver in parallel.

8.4 Combined analysis with beam sections

If the mesh contains also one-dimensional elements, these lower-dimensional elements can be used to define additional stiffeners by treating them as elastic beams. The combined model can be assembled in the case of both linear and nonlinear analysis, but the deformation of beam sections is always computed according to the linear theory as described in the Model 10 chapter of this manual.

The model parameters of the beam model are named as described in the Model 10 chapter and their values are read from material sections, so elementwise organized data cannot yet be given. In particular one of the principal directions of the beam cross section (the y_3-axis) can be given by specifying the value of the keyword Director.

The shell model does not inherently recognize a moment around the shell director (normal) of the mid-surface, while the beam formulation which is used employs the full resultant couple (moment) vector with three components. To cope with this discrepancy, the beam stiffness matrix is manipulated before assembling into the global stiffness matrix in such a way that the resultant couple vector cannot have a component with respect to the director vector of the beam. The beam and shell directors are now defined independently, but it is natural to construct the combined model so that the shell and beam directors agree in places where the domains of the two models intersect.

8.5 Keywords

`Simulation`

`Coordinate System` `String Cartesian 3D`
The coordinate system should be selected to be three-dimensional, although basis functions for computation correspond to 2-D finite elements.

`Material` `mat id`
The following keywords relate to specifying the shell thickness and material parameters.

`Shell Thickness` `Real`
The thickness d of the shell is specified with this keyword.

`Poisson Ratio` `Real`
Poisson's ratio is given by using this keyword.

`Youngs Modulus` `Real`
This keyword specifies the value of Young's modulus.

`Density` `Real`
This keyword is used for defining the density of the material.

`Rayleigh Damping Alpha` `Real`
This specifies a coefficient to activate mass-proportional damping.

`Solver` `solver id`

`Equation` `String`
A describing name for the solver.

`Procedure` `File "ShellSolver" "ShellSolver"`
The name of the solver subroutine.

`Variable` `String Deflection[U:3 DNU:3]`
The name of the solver variable can be chosen freely (but it is must be used consistently elsewhere). The default variable is `Deflection[U:3 DNU:3]` which is thus suitable for the model derived under the restriction (8.13). Then the first three components of the solver variable define the mid-surface displacement field $\boldsymbol{v}^{(0)}$ (the default variable name `U`), while the rest are related to the vector $\boldsymbol{v}^{(1)}$ (the default variable name `DNU`). The variables always correspond to the orthogonal Cartesian components with respect to the global coordinate frame.

`Variable DOFs` `Integer`
The value of this can be either 6 or 9. The value 9 can be used only when p-elements are applied. The value 6 is given by default.

`Large Deflection` `Logical`
By default the nonlinear equations are solved. With the value being `False`, the linearized strain tensor is employed.

`Eigen Analysis` `Logical`
Eigenanalysis based on the linearized model can be done if this keyword is given the value `True`.

`Displace Mesh` `Logical`
If this keyword is given the value `True`, the mesh is mapped to represent the deformed configuration. The formulation of the total Lagrangian type is nevertheless used, so the shell problem is posed over the undeformed configuration. In the case of eigenanalysis this keyword is not supported.

`Nonlinear System Convergence Tolerance` Real

The ratio of the 2-norm of the nonlinear system residual to the 2-norm of the initial right-hand side vector is always used as the stopping criterion for the nonlinear iteration. This keyword specifies the stopping tolerance for the Newton iteration to solve the nonlinear system.

`Linear System Convergence Tolerance` Real

It should be noted that each linearized problem solved during the nonlinear iteration gives an increment $\boldsymbol{\delta}\mathbf{U}^{k+1} = \mathbf{U}^{k+1} - \mathbf{U}^{k}$ to the previous iterate. Therefore, except for the case of solving the linear shell model (`Large Deflection = False`), a rather mild stopping tolerance may often be used for the linear systems without affecting the progress of the nonlinear iteration.

`Mesh Reparameterization` Logical

If the value `True` is given, the solver does not apply the internal surface reconstruction which depends on given director data, but it expects a third-order nodal mesh of the physical mid-surface. In this case, all geometric information is derived directly from the mesh, so no user-supplied information about the shell director is needed. At the moment no numerical tricks are then applied to handle numerical over-stiffness (locking), but the basic third-order approximation may give reasonable results if the shell is not very thin.

`Skip Surface Reconstruction` Logical

If the value `True` is given, the solution of some special cases can be performed such that the mathematical formulation is given over a two-dimensional (planar) reference domain which is utilized to give a global parametrization of the mid-surface by lines of curvature coordinates. In this case, the approximation should be done by using p-elements.

`Strain Reduction Operator` Integer

This keyword specifies the choice of strain reduction operators. If this keyword is not given, the solver switches to a method which has been found to give the best results for benchmark cases considered during the development.

`Body Force` bf id

`Normal Pressure` Real

The value of this keyword should give the sum of normal tractions applied to the upper and lower faces of the shell at $y^3 = \pm d$. If the shell model is nonlinear, this is the normal surface force per unit area of the deformed surface. In this case the normal to the deformed mid-surface and the area are computed by using the previous iterate.

`Boundary Condition` bc id

The Dirichlet conditions for the components of $\boldsymbol{v}^{(0)}$ and $\boldsymbol{v}^{(1)}$ can be given in the standard manner. In addition, the resultant force vector $\boldsymbol{N}$ and the resultant couple vector $\boldsymbol{M}$ can be specified to give a mechanical load on a curve c which lies on the mid-surface in its reference configuration and may be represented as $c = \mathbf{f}(I)$, with $I \subset \mathbb{R}$. This gives rise to a contribution

$$\int_I \boldsymbol{N}(\mathbf{f}(s)) \cdot \boldsymbol{v}^{(0)}(\mathbf{f}(s))\,\mathrm{d}c(\mathbf{f}(s)) + \int_I \boldsymbol{M}(\mathbf{f}(s)) \cdot \boldsymbol{v}^{(1)}(\mathbf{f}(s))\,\mathrm{d}c(\mathbf{f}(s))$$

to the linear functional of the shell problem. Note that by default the components of all data vectors are defined with respect to the global coordinate frame.

`U i` Real

If the default variable name is used, then, with `i=1,2,3`, Dirichlet BCs for the components of the mid-surface displacement $\boldsymbol{v}^{(0)}$ can be given.

`DNU i` Real

If the default variable name is used, then, with `i=1,2,3`, Dirichlet BCs for the components of $\boldsymbol{v}^{(1)}$ can be given.

`Resultant Force i` Real

This keyword may be used to give the components of the resultant force $\boldsymbol{N}$ measured per unit length of a curve c on the mid-surface.

`Resultant Couple i` Real
: This keyword may be used to give the components of the resultant couple $\boldsymbol{M}$ measured per unit length of a curve c on the mid-surface.

`Dead Loads` Logical
: By default giving constant values for $\boldsymbol{N}$ or $\boldsymbol{M}$ creates “dead loads” whose orientation is fixed with respect to the global frame. Giving the value `False` for this keyword alters the meaning of the resultant force and couple loads such that their orientation follows the deformation of lines of curvature, with the direction of the first component following the deformation of the coordinate curve corresponding to the smallest curvature in the undeformed configuration. Then `Resultant Force 3` generally gives an edge-load in the direction of normal to the deformed mid-surface.

`Spring i` Real
: With `i=1,2,3`, this keyword may be used to create a resultant force which is proportional to the displacement along the `i`*th* coordinate direction. With `i=4,5,6`, one may generate a resultant couple which is proportional to the (`i-3`)*th* component of $\boldsymbol{v}^{(1)}$ with respect to the global frame.

`Mass i` Real
: This keyword, with `i=1,2,...,6`, may be used to assemble an additional mass or a moment of inertia. The first three components are related to translational DOFs, while the remaining components are related to the DOFs which determine $\boldsymbol{v}^{(1)}$; cf. the indexing convention used in connection with `Spring i`.

Bibliography

[1] D. Chapelle and K.J. Bathe. *The Finite Element Analysis of Shells - Fundamentals*. Springer, second edition, 2011.

[2] P.G. Ciarlet. *An Introduction to Differential Geometry with Applications to Elasticity*. Springer, Dordrecht, the Netherlands, 2005.

[3] P.M. Naghdi. Foundations of elastic shell theory. In *Progress in Solid Mechanics, Vol. 4 (I.N. Sneddon, R. Hill, Eds) North-Holland*, pages 1–90, 1963.

Model 9

Plate Equations of Linear Elasticity

Module name: Smitc
Module subroutines: SmitcSolver
Module authors: Mikko Lyly, Jani Paavilainen
Document authors: Mikko Lyly, Peter Råback

9.1 Introduction

The linear elastic plate elements of Elmer are based on the shear deformable model of Reissner and Mindlin.The finite element discretization is performed using the so called stabilized MITC-plate elements, which are free from numerical locking.

9.1.1 Reissner-Mindlin model

The displacement $\vec{u} = (u_x, u_y, u_z)$ of a Reissner-Mindlin plate (thin or moderately thick linearly elastic body which in its undeformed reference configuration occupies the three dimensional region $\Omega \times (-\frac{t}{2}, \frac{t}{2})$, where Ω is the midsurface and t the thickness) is obtained from the kinematic equations

$$u_x(x,y,z) = -\theta_x(x,y) \cdot z \tag{9.1}$$

$$u_y(x,y,z) = -\theta_y(x,y) \cdot z \tag{9.2}$$

$$u_z(x,y,z) = w(x,y) \tag{9.3}$$

where θ_x and θ_y are components of the rotation vector $\underline{\theta} = (\theta_x, \theta_y)$ and w is the transverse deflection of the mid-surface, see Figure 1.

The functions w and $\underline{\theta} = (\theta_x, \theta_y)$ are determined from the condition that they minimize the total potential energy

$$\frac{1}{2}\int_\Omega \underline{\underline{\kappa}} : \underline{\underline{m}}\, d\Omega + \int_\Omega \underline{\gamma} \cdot \underline{q}\, d\Omega - \int_\Omega pw\, d\Omega \tag{9.4}$$

where p is the transverse pressure load, $\underline{\underline{\kappa}} = \frac{1}{2}(\underline{\underline{\nabla\theta}} + \underline{\underline{\nabla\theta}}^T)$ is the curvature of the mid-surface, $\underline{\gamma} = \underline{\nabla}w - \underline{\theta}$ is the transverse shear strain, $\underline{\underline{m}} = \mathcal{E} : \underline{\underline{\kappa}}$ is the bending moment, and $\underline{q} = \mathcal{G} \cdot \underline{\gamma}$ the transverse shear force vector. The fourth order tensor E and second order tensor $\mathcal{G}$ define the bending and shear rigidities of the cross section, respectively. For linearly elastic materials we have $\mathcal{G} \cdot \underline{\gamma} = Gt\underline{\gamma}$ and

$$\mathcal{E} : \underline{\underline{\kappa}} = K[\underline{\underline{\kappa}} + \frac{\nu}{1-\nu}(tr\underline{\underline{\kappa}})\underline{\underline{I}}] \tag{9.5}$$

where $K = Et^3/[12(1-\nu^2)]$ is the bending stiffness, E is Young's modulus, G shear modulus, and ν Poisson ratio. The design of the tensors $\mathcal{E}$ and $\mathcal{G}$ for orthotropic and perforated materials is discussed in section 9.3.

The minimizer of the energy satisfies the equilibrium equations

$$\underline{\nabla} \cdot \underline{\underline{m}} + \underline{q} = 0 \tag{9.6}$$

$$-\nabla \cdot \underline{q} = p \tag{9.7}$$

9.1.2 Surface tension

When surface tension is present, the following term is added to the energy:

$$\frac{1}{2} \int_{\Omega} \underline{\nabla} w \cdot \mathcal{T} \cdot \underline{\nabla} w \; d\Omega \tag{9.8}$$

where $\mathcal{T}$ is a second order tensor representing the given normal force (usually $\mathcal{T} = T\underline{\underline{I}}$, where T is constant). The equilibrium equation (9.7) is then rewritten as

$$-\nabla \cdot (\underline{q} + \mathcal{T} \cdot \nabla w) = p \tag{9.9}$$

9.1.3 Boundary conditions

The following boundary conditions can be applied in the Reissner-Mindlin plate model:

- Soft fixed edge: $w = 0$ and $\underline{\theta} \cdot \underline{n} = 0$
- Hard fixed edge: $w = 0$ and $\underline{\theta} = \underline{0}$
- Soft simply supported edge: $w = 0$
- Hard simply supported edge: $w = 0$ and $\underline{\theta} \cdot \underline{t} = 0$
- Free edge: $\underline{\underline{m}} \cdot \underline{n} = 0$ and $(\underline{q} + \mathcal{T} \cdot \underline{\nabla} w) \cdot \underline{n} = 0$

The boundary conditions can of course be non-homogeneous as well. For fixed and simply supported edges the prescribed values of w, $\underline{\theta}$, $\underline{\theta} \cdot \underline{n}$, and $\underline{\theta} \cdot \underline{t}$, are taken into account on matrix level after finite element discretization. On the free part of the edge, the non-homogeneous case is treated by adding the following terms in the energy:

$$\int_{\Gamma_{free}} q_n w \; d\Gamma + \int_{\Gamma_{free}} \underline{m}_n \cdot \underline{\theta} \; d\Gamma \tag{9.10}$$

where $q_n = \underline{q} \cdot \underline{n}$ and $\underline{m}_n = \underline{\underline{m}} \cdot \underline{n}$ are prescribed functions.

9.1.4 Kirchhoff plates

When the thickness of the plate is small ($t << \mathrm{diam}(\Omega)$), the Reissner-Mindlin model can be considered as a penalty approximation of the classical plate model of Kirchhoff. The Kirchhoff model is obtained from (9.1)-(9.9) by enforcing the constraint $\underline{\gamma} = \underline{0}$. The governing equations are then reduced to

$$K\Delta\Delta w - T\Delta w = p \tag{9.11}$$

9.1.5 Transient and natural mode analysis

A transient plate model is obtained by adding the inertia term $\rho t \ddot{w}$ on the left-hand side of (9.7), (9.9), and (9.11). Here ρ is the density of the material. The natural vibration frequencies and mode shapes are then obtained by taking $p = 0$ and solving the Fourier transformed equations.

9.2 Finite element implementation

The direct minimization of (9.4) using the standard Galerkin finite element method fails due to the well known numerical locking phenomena (the method is unable to deal with the Kirchhoff constraint $\underline{\gamma} = \underline{0}$, which becomes valid when t is small). In order to avoid locking, Elmer utilizes the so called SMITC (Stabilization and Mixed Interpolation of Tensorial Components) elements, which are known to be optimally convergent and work well under all conditions [4].

The linear element of the SMITC-family was first introduced by Brezzi, Fortin and Stenberg in [2]. The method is defined by replacing the shear energy term in (9.4) by the following numerical modification:

$$\int_\Omega \underline{\gamma}_h \cdot \underline{q}_h \, d\Omega \tag{9.12}$$

where $\underline{\gamma}_h$ is called the reduced shear strain (sometimes also referred to as the assumed or substitute shear) and $\underline{q}_h = (t^2 + \alpha h^2)^{-1}\mathcal{G} \cdot \underline{\gamma}_h$ the reduced shear force. Here h is the mesh size (the diameter of the biggest element) and $\alpha > 0$ is a numerical stabilization parameter (typically $\alpha = 0.15$).

The reduced shear $\underline{\gamma}_h$ is defined elementwise such that

$$\underline{\gamma}_{h|K} = (a_K - b_K y, a_K + c_K x) \tag{9.13}$$

for any element K. The parameters a_K, b_K, and c_K, are determined from the conditions

$$\int_E (\underline{\gamma} - \underline{\gamma}_h) \cdot \underline{t} \, ds = 0 \tag{9.14}$$

for every edge E of K. Here $\underline{t}$ is the counterclockwise tangent to E.

It has been shown [3] that the linear SMITC-element is equivalent to the T3BL (Triangle, 3 nodes, Linked Interpolation) element of Xu, Auricchio and Taylor [8, 1], the anisoparametrically interpolated MIN3 element of Tessler and Hughes [7], and the TRIA3 element of MacNeal [5]. We refer to [3] for a more detailed discussion.

9.3 Elastic parameters for perforated plates

In microelectromechanical systems the plate structures are often perforated in order to reduce the squeezed-film damping effect. This has also an effect on the elasticity equation. If there are so many holes that it is not feasible to treat them individually their effect may be homogenized over the whole structure. In practice this means that the original elastic parameters are replaced by effective parameters that take into account the holes. This method was reported by Pedersen et al. [6] and implemented into the solver by Jani Paavilainen.

In the homogenization effective parameters for an orthotropic plate are defined so that the unperforated model approximates the perforated plate. The basic idea is to set the analytical expressions of the deformation energies of the perforated and unperforated plates equal. This method is inherently limited to simple geometries where analytical expressions may be found. So far, only square holes have been implemented in the solver.

The unit cell of a perforated plate may be assumed to consist of one small square plate with side $b - 2a$, and of four beams of length a as shown in Figure 9.1. Using approximate formulas an analytical formula for the deformation energy of the perforated plate is obtained. This has to be equal to the deformation energy of an unperforated orthotropic membrane. From this condition we get a set of equations from which the effective parameters may be solved.

The elasticity tensor has three independent components, $C_{11} = C_{22}, C_{12} = C_{21}$, and C_{44}. The expressions for these are [6],

$$C_{11} = C_{22} = \frac{E}{b^2}\left\{\frac{b(b-2a)}{1-\nu^2} + \frac{a(b-2a)^2}{b}\right\} \tag{9.15}$$

$$C_{12} = C_{21} = \frac{\nu E(b-2a)}{b(1-\nu^2)} \tag{9.16}$$

$$C_{44} = \frac{E}{4b^2(1+\nu)}\left\{2b(b-2a) + \frac{12Ka(b-2a)}{bh^3}\right\}. \tag{9.17}$$

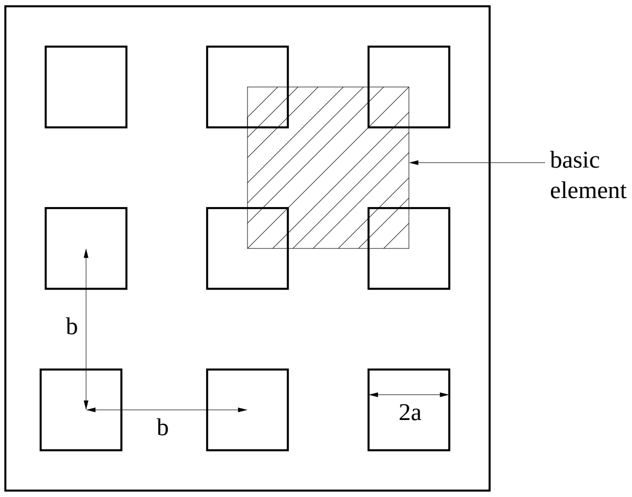

Figure 9.1: The basic element of the perforated plate consisting of five rectangular beams

where K is a constant[1], defined as

$$K = \begin{cases} \frac{1}{3}\left(1 - 0.63\frac{b-2a}{h}\right)(b-2a)^3 h, & \text{jos } h > b - 2a \\ \frac{1}{3}\left(1 - 0.63\frac{h}{b-2a}\right)(b-2a)h^3, & \text{jos } h < b - 2a. \end{cases} \tag{9.18}$$

The midplane tension of the perforated plate may be reduced to lateral stresses of the orthotropic plate by a simple scaling,

$$T = \sqrt{(1 - 4a^2/b^2)}\, T_0, \tag{9.19}$$

where is the tension T_0 of the perforated plate. Using this reduced tension and the modified material parameters of equations (9.15), (9.16) and (9.17) the orthotropic plate mimics the behavior of the perforated plate when looking at macroscopic quantities. However, the model is not suitable for approximating maximum stresses around the holes, for example.

9.4 Keywords

`Solver` `solver id`

`Equation` `String SmitcSolver`

`Procedure` `File "Smitc" "SmitcSolver"`
The procedure which includes the linear plate model.

`Variable` `String Deflection`
This may be of any name as far as it is used consistently also elsewhere.

`Variable DOFs` `Integer 3`
Degrees of freedom for the deflection. The first degree is the displacement and the two following ones are its derivatives in the direction of the coordinate axis.

`Eigen Analysis` `Logical`
Also the eigenvalues and eigenmodes of the elasticity equation may be computed. This is done automatically by calling a eigensolver after the original equation has been solved. The default is `False`.

[1]In article [6] there is an error in the definition of K. In the article there is an expression $(b - 2a)/h^3$, which would make K discontinuous at $h = b - 2a$.

`Eigen System Values` Integer
: If the eigenvalues are computed this keyword gives the number of eigenmodes to be computed. The lowest eigenvalues are always solved for.

`Hole Correction` Logical
: If the plate is perforated the holes may be taken into account by a homogenized model. This is activated with this keyword. The default is `False`.

`Material` mat id

`Density` Real
: Density of the plate.

`Poisson Ratio` Real

`Youngs Modulus` Real
: The elastic parameters are given with the keywords `Youngs Modulus` and `Poisson ratio`.

`Thickness` Real
: Thickness of the plate.

`Tension` Real
: The plate may be pre-stressed.

`Hole Size` Real

`Hole Fraction` Real
: If `Hole Correction` is `True` the solver tries to find the size and relative fraction of the holes. If these are present the hole is assumed to be a square hole.

`Boundary Condition` bc id

`Deflection i` Real
: Dirichlet BC for the components of the deflection, i=1,2,3.

`Body Force` bf id

`Pressure` Real
: Possibility for a body forces. For coupled systems there is a possibility to have up to three forces. The two others are then marked with `Pressure B` and `Pressure C`.

`Spring` Real
: The local spring which results to a local force when multiplied by the displacement.

`Damping` Real
: The local damping which results to a local force when multiplied by the displacement velocity. The spring and damping may also be defined as material parameters.

Bibliography

[1] F. Auricchio and R.L. Taylor. A triangular thick plate finite element with an exact thin limit. *Finite Element in Analysis and Design*, 19:57–68, 1995.

[2] M. Fortin F. Brezzi and R. Stenberg. Error analysis of mixed-interpolated elements for reissner-mindlin plates. *Mathematical Models and Methods in Applied Sciences*, 1:125–151, 1991.

[3] M. Lyly. On the connection between some linear triangular reissner-mindlin plate bending elements. *Numerische Mathematik*, 85:77–107, 2000.

[4] M. Lyly and R. Stenberg. Stabilized mitc plate bending elements. In *Advances in Finite Element Techniques (M. Papadrakakis and B.H.V. Topping, eds.) Civil Comp Press.*, pages 11–16, 1994.

[5] R.H. MacNeal. Derivation of element stiffness matrices by assumed strain distribution. *Nucl. Engrg. Design*, 70:3–12, 1982.

[6] M. Pedersen, W. Olthuis, and P. Bergveld. On the mechanical behaviour of thin perforated plates and their application in silicon condenser microphones. *Sensors and Actuators*, A 54:6.

[7] A. Tessler and T.J.R. Hughes. A three-node mindlin plate element with improved transverse shear. *Comp. Meths. Appl. Mech. Engrg.*, 50:71–101, 1985.

[8] Z. Xu. A thick-thin triangular plate element. *Int. J. Num. Meths. Eng.*, 33:963–973, 1992.

Model 10

One-dimensional Equations for Elastic Beams

Module name: BeamSolver3D
Module subroutines: TimoshenkoSolver
Module authors: Mika Malinen
Document authors: Mika Malinen

10.1 Introduction

The solver described in this section can be used to solve structural beam equations. The equilibrium equations are expressed in terms of stress resultants $\boldsymbol{N}$ and $\boldsymbol{M}$ that represent forces and moments experienced by the cross section of a beam. If the cross section is considered to be an oriented surface with positive unit normal $\boldsymbol{e}_1$, the stress resultant $\boldsymbol{N}$ is defined by

$$\boldsymbol{N} = \int\limits_A \boldsymbol{\sigma}\boldsymbol{e}_1 \, \mathrm{d}A \tag{10.1}$$

where $\boldsymbol{\sigma}$ is the stress tensor and A denotes the cross section of the beam. Generally $\boldsymbol{N}$ is resolved into components with respect to an orthonormal basis $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}\}$ associated with a local frame which has its origin at the intersection of the cross section and the axis of the beam. The coordinates of points with respect to this frame are denoted by (y_1, y_2, y_3) with the y_1-axis being aligned with the axis of the beam, while the other two axes are assumed to coincide with the principal directions of the cross section. The stress resultant $\boldsymbol{M} = M_1\boldsymbol{e_1} + M_2\boldsymbol{e_2} + M_3\boldsymbol{e_3}$ has as its components the bending moments

$$M_2 = \int\limits_A y_3(\boldsymbol{\sigma}\boldsymbol{e}_1)\cdot\boldsymbol{e}_1 \, \mathrm{d}A, \quad M_3 = -\int\limits_A y_2(\boldsymbol{\sigma}\boldsymbol{e}_1)\cdot\boldsymbol{e}_1 \, \mathrm{d}A, \tag{10.2}$$

while M_1 is a torsional moment. We note that in general accurate modelling of the torsion is not a simple problem and here a rudimentary approximation will only be employed.

10.2 Governing equations

Let us assume that the axis of the beam is described by using the arc length parametrization $s \in [0, L] \mapsto \mathbf{r}(s) \in \mathbf{E}^3$. The equilibrium equations are then given by

$$\begin{aligned} -\boldsymbol{N}'(s) &= \boldsymbol{F}(s), \\ -\boldsymbol{M}'(s) - \boldsymbol{t}(s)\times\boldsymbol{N}(s) &= \boldsymbol{G}(s), \end{aligned} \tag{10.3}$$

where $\boldsymbol{F}$ and $\boldsymbol{G}$ are the densities of applied forces and moments per unit length and $\boldsymbol{t}(s) \equiv \boldsymbol{e}_1(s)$ is tangential to the beam axis. It should be noted that $\boldsymbol{F}$ and $\boldsymbol{G}$ can be considered to include inertial body forces to obtain equations for transient cases.

In the case of Timoshenko's treatment of shear deformation, suitable generalized measures of strain may be expressed as

$$\begin{aligned} \boldsymbol{\varepsilon}(s) &= \boldsymbol{u}'(s) - \boldsymbol{\theta}(s) \times \boldsymbol{t}(s), \\ \boldsymbol{\kappa}(s) &= \boldsymbol{\theta}'(s) \end{aligned} \tag{10.4}$$

where $\boldsymbol{u} : [0, L] \to \mathbb{R}^3$ is the displacement of the beam axis and the components of $\boldsymbol{\theta} : [0, L] \to \mathbb{R}^3$ are the so-called rotations. It is noted that the underlying approximation of the displacement $\boldsymbol{u}_{3\mathrm{D}}$ for a generic point $\mathbf{r}(s) + y_2\boldsymbol{e}_2(s) + y_3\boldsymbol{e}_3(s)$ of the cross sections is given componentwise by

$$\begin{aligned} \boldsymbol{u}_{3\mathrm{D}}(s; y_2, y_3) \cdot \boldsymbol{e}_1(s) &= u_1(s) - y_2\theta_3(s) + y_3\theta_2(s), \\ \boldsymbol{u}_{3\mathrm{D}}(s; y_2, y_3) \cdot \boldsymbol{e}_2(s) &= u_2(s) - y_3\theta_1(s), \\ \boldsymbol{u}_{3\mathrm{D}}(s; y_2, y_3) \cdot \boldsymbol{e}_3(s) &= u_3(s) + y_2\theta_1(s). \end{aligned} \tag{10.5}$$

The constitutive relations are written as

$$\begin{aligned} \boldsymbol{N} &= \boldsymbol{D}\boldsymbol{\varepsilon}, \\ \boldsymbol{M} &= \boldsymbol{E}\boldsymbol{\kappa}, \end{aligned} \tag{10.6}$$

where

$$\boldsymbol{D} = \mathbf{diag}(EA, GAk_2, GAk_3) \tag{10.7}$$

and

$$\boldsymbol{E} = \mathbf{diag}(GJ_T, EI_2, EI_3). \tag{10.8}$$

Here E and G are Young's modulus and the shear modulus, respectively, while J_T and I_k give a torsional constant and the second moments of area, respectively. The parameters k_j are known as the shear correction factors.

10.3 The weak formulation

A weak formulation of the beam problem stems from the identities

$$\begin{aligned} -\int_0^L \boldsymbol{N}' \cdot \boldsymbol{v}\, ds &= \int_0^L \boldsymbol{F} \cdot \boldsymbol{v}\, ds, \\ -\int_0^L \boldsymbol{M}' \cdot \boldsymbol{\psi}\, ds - \int_0^L \boldsymbol{t} \times \boldsymbol{N} \cdot \boldsymbol{\psi}\, ds &= \int_0^L \boldsymbol{G} \cdot \boldsymbol{\psi}\, ds \end{aligned} \tag{10.9}$$

that yield after integration by parts

$$\begin{aligned} \int_0^L \boldsymbol{N} \cdot \boldsymbol{v}'\, ds &= \int_0^L \boldsymbol{F} \cdot \boldsymbol{v}\, ds + \boldsymbol{N}(L) \cdot \boldsymbol{v}(L) - \boldsymbol{N}(0) \cdot \boldsymbol{v}(0), \\ \int_0^L \boldsymbol{M} \cdot \boldsymbol{\psi}'\, ds - \int_0^L \boldsymbol{t} \times \boldsymbol{N} \cdot \boldsymbol{\psi}\, ds &= \int_0^L \boldsymbol{G} \cdot \boldsymbol{\psi}\, ds + \boldsymbol{M}(L) \cdot \boldsymbol{\psi}(L) - \boldsymbol{M}(0) \cdot \boldsymbol{\psi}(0). \end{aligned} \tag{10.10}$$

In order to specify forces exerted upon the beam by the environment, we shall set

$$\boldsymbol{N}_0 = -\boldsymbol{N}(0) \quad \text{and} \quad \boldsymbol{N}_L = \boldsymbol{N}(L). \tag{10.11}$$

Similarly the bending moments applied to the beam are specified as

$$\boldsymbol{M}_0 = -\boldsymbol{M}(0) \quad \text{and} \quad \boldsymbol{M}_L = \boldsymbol{M}(L). \tag{10.12}$$

By using (10.4) and (10.6) and by including transient inertia terms, the equations (10.10) lead to the standard abstraction of the problem: Find $\mathbf{w} \equiv (\boldsymbol{u}(\cdot, t), \boldsymbol{\theta}(\cdot, t)) \in \mathbf{U}$ such that

$$a(\mathbf{w}, \mathbf{z}) = l(\mathbf{z}), \quad \forall \mathbf{z} \equiv (\boldsymbol{v}, \boldsymbol{\psi}) \in \mathbf{V} \tag{10.13}$$

with

$$a(\mathbf{w}, \mathbf{z}) = \int_0^L m\ddot{\boldsymbol{u}} \cdot \boldsymbol{v}\, ds + \int_0^L \boldsymbol{I}_m \ddot{\boldsymbol{\theta}} \cdot \boldsymbol{\psi}\, ds + \int_0^L \boldsymbol{E}\boldsymbol{\theta}' \cdot \boldsymbol{\psi}'\, ds + \int_0^L \boldsymbol{D}(\boldsymbol{u}' - \boldsymbol{\theta} \times \boldsymbol{t}) \cdot (\boldsymbol{v}' - \boldsymbol{\psi} \times \boldsymbol{t})\, ds \tag{10.14}$$

and

$$l(\mathbf{z}) = \int_0^L \boldsymbol{F} \cdot \boldsymbol{v}\, ds + \int_0^L \boldsymbol{G} \cdot \boldsymbol{\psi}\, ds + \boldsymbol{N}_L \cdot \boldsymbol{v}(L) + \boldsymbol{N}_0 \cdot \boldsymbol{v}(0) + \boldsymbol{M}_L \cdot \boldsymbol{\psi}(L) + \boldsymbol{M}_0 \cdot \boldsymbol{\psi}(0). \tag{10.15}$$

Here m is the mass per unit length and the diagonal $\boldsymbol{I}_m$ gives the (mass) moments of inertia. In addition, $\mathbf{U}$ and $\mathbf{V}$ denote the sets of kinematically admissible functions and test functions, respectively.

10.4 Keywords

`Simulation`

`Coordinate System String Cartesian 3D`
The coordinate system should be selected to be three-dimensional, as no specific orientation of the beam axis is supposed.

`Material mat id`
The following keywords relate to specifying the material parameters and the cross section data.

`Youngs Modulus Real`
This keyword specifies the value of Young's modulus E.

`Shear Modulus Real`
This keyword specifies the value of G.

`Density Real`
This keyword is used for defining the density of the material. The density is needed in transient cases to include the effects of inertial forces.

`Cross Section Area Real`
This keyword specifies the area A of the cross section.

`Principal Direction 2(3) Real`
The three components given by using this keyword should define the direction of the local y_2-axis, so that the vector $\boldsymbol{e}_2$ can be computed. This direction must be orthogonal to the beam axis that is determined by the coordinates of the element nodes. If this keyword is not specified, the default value $\boldsymbol{e}_2 = (0, 0, -1)$ is used.

`Torsional Constant Real`
To get an approximation of the torsional effects, the value of the parameter J_T can be given.

`Second Moment of Area 2 Real`
This keyword should give the value of the integral $I_2 \equiv \int_A y_3^2 \, \mathrm{d}A$.

`Second Moment of Area 3 Real`
This keyword should give the value of the integral $I_3 \equiv \int_A y_2^2 \, \mathrm{d}A$.

`Solver` `solver id`

`Equation` `String`
: A describing name for the solver.

`Procedure` `File "BeamSolver3D" "TimoshenkoSolver"`
: The name of the solver subroutine.

`Variable` `String Deflection[U:3 Theta:3]`
: The name of the solver variable can be chosen freely (but it is must be used consistently elsewhere).

`Variable DOFs` `Integer 6`
: There is no need for using this keyword as the only possible value is 6 and it is given automatically by the solver. The first three components of the solver variable define the displacement $\boldsymbol{u}$ of the beam axis (the default variable name `U`), while the rest give the vector $\boldsymbol{\theta}$ (the default variable name `Theta`). In this connection the components of both the vectors are defined with respect to the global coordinate frame.

`Body Force` `bf id`

`Body Force k` `Real`
: The value of this keyword gives the `k`th component of the applied force $\boldsymbol{F}$ with respect to the global coordinate frame.

`Boundary Condition` `bc id`

The Dirichlet conditions for the components of $\boldsymbol{u}$ and $\boldsymbol{\theta}$ can be given in the standard manner. Note that here the components of both vectors are defined with respect to the global coordinate frame.

`U i` `Real`
: If the default variable name is used, then, with `i=1,2,3`, Dirichlet BCs for the components of the displacement $\boldsymbol{u}$ can be given.

`Theta i` `Real`
: If the default variable name is used, then, with `i=1,2,3`, Dirichlet BCs for the components of the rotation $\boldsymbol{\theta}$ can be given.

Model 11

Adding pointwise springs and masses

Module name: SpringAssembly
Module subroutines: SpringAssembler
Module authors: Mika Malinen
Document authors: Mika Malinen

11.1 Introduction

The utility described in this section can be used to add nodewise specified springs and masses to structural models. This utility is generic and should be applicable to several structural models including equations of linear and nonlinear elasticity, shells, plates and beams. It should be noted that some models may be defined to have pointwise springs or masses without using this utility, provided the set of boundary elements includes point elements and the model itself can handle springs or masses. The utility considered here does not however depend on the content of the mesh file defining boundary elements, since the places of the springs and masses are now supposed to be specified in terms of the indices of the mesh nodes.

11.2 Guiding assembly procedure

The module subroutine considered can be called as an additional assembly procedure to change the stiffness and mass matrices of the target model which is modified to have the springs or masses. To achieve this, in the solver input file the solver section associated with the primary model must contain the keyword command `Assembly Solvers` (for the documentation of this generic utility command see also ElmerSolver Manual). The value of this keyword gives the integer identifier of the solver section which is utilized to perform the additional assembly procedure.

To describe what is needed, let us suppose that the primary solver which should utilize the spring definitions is associated with the variable name `"Displacement"`. If one then adds X*th* solver section into the solver input file as

```
Solver X
  Equation = "Assemble Springs"
  Exec Solver = Never
  Procedure = "SpringAssembly" "SpringAssembler"
  Displacement Variable Name = "Displacement"
End
```

and adds into the solver section of the primary solver a line

```
  Assembly Solvers(1) = X
```

then it should be possible to use boundary condition sections to add spring specifications as

```
Boundary Condition Y
  Target Nodes(...) = ...
  Spring 1 = ...
  Spring 2 = ...
  Spring 3 = ...
  ...
End
```

An additional mass (and moment of inertia when applicable) may be added in a similar way by using a keyword `Mass i`, with `i` being an integer string.

11.3 Keywords

`Solver` id

`Displacement Variable Name` String
: The value of this keyword must be the variable name of the solver which is modified to have additional springs or masses. The default value is `"Displacement"`.

`Boundary Condition` bc id

`Spring i` Real
: This keyword may be used to create a reaction which is proportional to the value of the `i`*th* global DOF (degree of freedom) of the finite element used to discretize the primary model.

`Mass i` Real
: With this keyword, the mass matrix of the model can be modified by assembling an additional diagonal mass matrix. The integer string `i` of the keyword refers to `i`*th* global DOF (degree of freedom) of the element. In the basic 3D simulation the values for all `i`$\in \{1,2,3\}$ should be the same by physical reasons as they originate from the same scalar property. With `i` > 3 an additional moment of inertia may be given for models for which it is meaningful (for example a shell model).

Part III

Models of Acoustics

Model 12

The Helmholtz Model

Module name: HelmholtzSolve
Module subroutines: HelmholtzSolver
Module authors: Juha Ruokolainen, Mikko Lyly, Mika Malinen, Peter Råback
Document authors: Juha Ruokolainen, Peter Råback

12.1 Introduction

This module solves the Helmholtz equation, which is the Fourier transform of the wave equation. In addition to the basic equation the solver may take into consideration variable density, background convections field, simple damping and special boundary conditions with other time-harmonic solvers.

12.2 Theory

For example, sound propagation in air is fairly well described by the wave equation:

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} - \nabla^2 p = 0. \tag{12.1}$$

When linear the equation may be written in frequency space as

$$k^2 P + \nabla^2 P = 0, \tag{12.2}$$

where $k = \omega/c$. This is the Helmholtz equation. The instantaneous pressure may be computed from the given field P:

$$p(t) = \Re(Pe^{i\omega t}) = \Re(P)\cos(\omega t) - \Im(P)\sin(\omega t), \tag{12.3}$$

where $i = \sqrt{-1}$ is the imaginary unity.

In Elmer the equation has an added term which is proportional to first time derivative of the field, whereupon the equation becomes

$$(k^2 - ikD)P + \nabla^2 P = 0, \tag{12.4}$$

where D is the damping factor.

12.2.1 Boundary Conditions

The usual boundary condition for the Helmholtz equation is to give the flux on the boundary:

$$\nabla P \cdot \vec{n} = g, \tag{12.5}$$

also Dirichlet boundary conditions may be set. The Sommerfeldt or far field boundary condition is as follows

$$\nabla P \cdot \vec{n} + \frac{i\omega}{Z} P = 0, \tag{12.6}$$

where the complex-valued quantity Z may be defined by the user. It is noted that incoming and outgoing waves may be approximated by setting $Z = \pm c$, respectively.

A special kind of flux condition is one with a given harmonic velocity field that is obtained from a harmonic solution of a flow or structure equation. When the velocity field $\vec{v}$ is given then the flux is obtained from

$$g = i\omega\rho\vec{v} \cdot \vec{n} \tag{12.7}$$

where ρ is the fluid density. If harmonic displacement is given instead a further term $i\omega$ appears in the equation.

12.3 Keywords

`Simulation`
This section gives values to parameters concerning the simulation as whole.

`Frequency Real`
Give simulation frequency in units of $1/\mathrm{s}$. Alternatively use the `Angular Frequency` keyword.

`Angular Frequency Real`
Give simulation frequency in units of $1/\mathrm{rad}$. Alternatively use the `Frequency` keyword.

`Solver solver id`
Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere. Note also that for the Helmholtz equation `ILUT` preconditioning works well.

`Equation String [Helmholtz]`
The name of the equation.

`Procedure File ["HelmholtzSolve" "HelmholtzSolver"]`
This keyword is used to give the Elmer solver the place where to search for the Helmholtz equation solver.

`Variable String [Pressure]`
Give a name to the field variable.

`Variable DOFs Integer [2]`
This keyword must be present, and *must* be set to the value 2.

`Bubbles Logical`
If set to `True` this keyword activates the bubble stabilization.

`Use Density Logical`
Historically the solver was able to solve only cases with constant density when it may be eliminated. If the density is however not constant this flag must be set `True`.

`Velocity Variable Name String`
If there is a `Flow Interface` then the name of the harmonic velocity variable may be specified. The default is `Flow`. Note that normal real valued velocity field is not suitable.

`Displacement Variable Name String`
If there is a `Structure Interface` then the name of the harmonic displacement variable may be specified. The default is `Displacement`. Note that normal real valued displacement field is not suitable, its complex valued eigenmode however is.

`Displacement Variable Eigenmode Integer`
If eigenmode is used for the interface this keyword is used to specify the number of the mode.

`Displacement Variable Frequency` `Logical`
: If eigenmode is used for the interface this keyword may be used to choose the frequency to be the frequency of the computed eigenmode.

`Equation` `eq id`
: The equation section is used to define a set of equations for a body or set of bodies:

`Helmholtz` `Logical`
: If set to `True`, solve the Helmholtz equation, the name of the variable must match the `Equation` setting in the `Solver` section. Alternatively use the `Active Solvers` keyword.

`Initial Condition` `ic id`
: The initial condition section may be used to set initial values for the field variables. The following variables are active:

`Pressure i` `Real`
: For each the real and imaginary parts of the solved field `i`$= 1, 2$.

`Material` `mat id`
: The material section is used to give the material parameter values. The following material parameters may be set in Helmholtz equation.

`Sound Speed` `Real`
: This keyword is used to give the value of the speed of sound.

`Sound Damping` `Real`
: This keyword is used to give the value of the damping factor D in equation 12.4.

`Density` `Real`
: If sound density is varying the density must be specified and its use must be enforced by the `Use Density` keyword.

`Convection Velocity i` `Real`
: If the pressure field is convected by a background velocity field (as in the Doppler effect) then this keyword is used to give the velocity field.

`Body Force` `bf id`

`Pressure Source i` `Real`
: The pressure sources of the real ($i = 1$) and complex ($i = 2$) parts. The use of this is rather seldom.

`Boundary Condition` `bc id`
: The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The one related to Helmholtz equations are

`Pressure i` `Real`
: Dirichlet boundary condition for real and imaginary parts of the variable. Here the values `i`$= 1, 2$ correspond to the real and imaginary parts of the unknown field.

`Wave Flux 1,2` `Real`
: Real and imaginary parts of the boundary flux. Here the values `i`$= 1, 2$ correspond to the real and imaginary parts of the boundary flux.

`Wave Impedance 1,2` `Real`
: This keyword may be used to define the real and imaginary parts of the quantity Z in (12.6). Here the values `i`$= 1, 2$ correspond to the real and imaginary parts of Z.

`Plane Wave BC` `Logical`
: Automatically sets the boundary conditions assuming outgoing plane waves if set `True`.

`Flow Interface` Logical
: Use harmonic velocity field to set the flux.

`Structure Interface` Logical
: Use harmonic displacement field to set the flux.

Model 13

The Linearized Navier–Stokes Equations in the Frequency Domain

Module name: Acoustics
Module subroutines: AcousticsSolver
Module authors: Mika Malinen
Document authors: Mika Malinen

13.1 Introduction

The basic acoustic equations such as the Helmholtz equation, which is frequently taken as the starting point in acoustic analyses, are based on the assumption of lossless flow, i.e. the effects of viscosity and heat conduction are neglected. These effects are significant, however, in thin zones near a solid boundary. In this chapter, a system of acoustic field equations taking into account the effects of viscosity and heat conduction is described. Consideration is confined to the time-harmonic solution of these equations.

13.2 Mathematical model

The acoustic field equations may be derived using the general principles of continuum mechanics and supplementing these equations by suitable constitutive equations applicable for the fluid flow. Here the linearized versions of such equations are used to derive an approximate system of field equations appropriate to the small-amplitude acoustics problem.

In the following the velocity, density, pressure and temperature fields associated with the flow are denoted by $\vec{v}$, ρ, p and T, respectively. The notations ρ_0, p_0 and T_0 are used for the values of the density, pressure and temperature at the equilibrium state.

13.2.1 The field equations

Consider the acoustic equations based on the linearized equation of motion, the constitutive equation relating the stress to the motion for a Newtonian fluid, the kinematic relation, the linearized continuity equation and the linearized energy equation

$$\begin{aligned}
\rho_0 \frac{\partial \vec{v}}{\partial t} &= \nabla \cdot \overline{\overline{\sigma}} + \rho_0 \vec{b}, \\
\overline{\overline{\sigma}} &= -p\overline{\overline{I}} + \lambda(\nabla \cdot \vec{v})\overline{\overline{I}} + 2\mu \overline{\overline{D}}(\vec{v}), \\
\overline{\overline{D}}(\vec{v}) &= \frac{1}{2}(\nabla \vec{v} + \nabla \vec{v}^T), \\
\frac{\partial \rho}{\partial t} &= -\rho_0 \nabla \cdot \vec{v}, \\
\rho_0 \frac{du}{dt} &= \kappa \Delta T - p_0 \nabla \cdot \vec{v} + \rho_0 h.
\end{aligned} \tag{13.1}$$

Here $\overline{\overline{\sigma}}$ is the stress tensor, $\vec{b}$ is the body force (per unit mass), λ and μ are parameters characterizing the viscosity of the fluid, u is the specific internal energy, κ is the heat conductivity and h is the internal supply of heat.

We supplement the system (13.1) by suitable equations of state assuming that the properties of the medium are expressible as functions of two state variables, say the temperature and density. We denote the specific entropy (entropy per unit mass) and its equilibrium value by s and s_0 and assume that the relation

$$du = T_0 ds + (p_0/\rho_0^2) d\rho \tag{13.2}$$

is valid. In addition, we approximate the equations which give the changes of pressure and specific entropy in terms of the changes of the state variables by

$$p - p_0 = \frac{(\gamma - 1)\rho_0 C_V}{T_0 \beta}(T - T_0) + \frac{(\gamma - 1) C_V}{T_0 \beta^2}(\rho - \rho_0) \tag{13.3}$$

and

$$s - s_0 = \frac{C_V}{T_0}(T - T_0) - \frac{C_V(\gamma - 1)}{T_0 \rho_0 \beta}(\rho - \rho_0), \tag{13.4}$$

where C_V is the specific heat at constant volume (per unit mass), γ is the ratio of the specific heats at constant pressure and constant volume and β is the coefficient of thermal expansion defined by

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_p. \tag{13.5}$$

Confining consideration to the time-harmonic case, the solutions of the primary unknowns are assumed to be of the form

$$\begin{aligned}
\vec{v}(x,t) &= \vec{v}(x)\exp(i\omega t), \\
\rho(x,t) &= \rho_0 + \rho(x)\exp(i\omega t), \\
T(x,t) &= T_0 + T(x)\exp(i\omega t),
\end{aligned} \tag{13.6}$$

where ω is the angular frequency. By the substitution of (13.6), the system of field equations based on (13.1)–(13.4) may be reduced to a system where the only unknown fields are the amplitudes $\vec{v}(x)$ and $T(x)$ of the disturbances of the velocity and temperature fields. The reduced system may be written as

$$\begin{aligned}
i\omega\rho_0\vec{v} + \frac{(\gamma-1)C_V\rho_0}{\beta T_0}\nabla T - (\lambda + \mu - \frac{i(\gamma-1)C_V\rho_0}{\omega T_0 \beta^2})\nabla(\nabla\cdot\vec{v}) - \mu\Delta\vec{v} &= \rho_0\vec{b}, \\
-\kappa\Delta T + i\omega\rho_0 C_V T + \frac{(\gamma-1)C_V\rho_0}{\beta}\nabla\cdot\vec{v} &= \rho_0 h.
\end{aligned} \tag{13.7}$$

It is noted that after the solution of the velocity and temperature the amplitudes $p(x)$ and $\rho(x)$ of the disturbances of the pressure and density fields can readily be obtained from the relations

$$\begin{aligned}
p &= \frac{(\gamma-1)C_V\rho_0}{\beta T_0}(T + \frac{i}{\omega\beta}\nabla\cdot\vec{v}), \\
\rho &= \frac{i\rho_0}{\omega}\nabla\cdot\vec{v}.
\end{aligned} \tag{13.8}$$

For numerical approximation the system (13.7) is rewritten as a mixed problem; to motivate this, see [2]. The mixed formulation is written as

$$\begin{aligned}\vec{v} - i\nabla\tau - i\nabla\phi + i\epsilon\nabla(\nabla\cdot\vec{v}) + i\epsilon\Delta\vec{v} &= -(i/\omega)\vec{b},\\ -\frac{i\epsilon}{(\gamma-1)\delta}\Delta\tau - \frac{1}{\gamma-1}\tau + \frac{1}{1+i\gamma k^2\epsilon\eta}\phi &= \frac{ih}{\beta T_0\omega^2},\\ i\nabla\cdot\vec{v} - \frac{\gamma k^2}{1+i\gamma k^2\epsilon\eta}\phi &= 0,\end{aligned} \tag{13.9}$$

where ϕ is an auxiliary unknown, τ is the scaled temperature defined by

$$\tau = \frac{\omega\beta}{\gamma k^2}T \tag{13.10}$$

and

$$k = \frac{\omega}{c} \qquad \epsilon = \frac{\mu}{\rho_0\omega} \qquad \delta = \frac{C_V\mu}{\kappa} \qquad \eta = \frac{\lambda}{\mu} \tag{13.11}$$

with c the adiabatic sound speed defined by the relation

$$T_0\beta^2c^2 = \gamma(\gamma-1)C_V. \tag{13.12}$$

It should be noted that although the solver of the acoustic equations is based on the formulation (13.9), the solver overwrites the approximations of τ and ϕ by the unscaled temperature and the pressure, which may be expressed as

$$p = \rho_0\omega(\tau + \frac{1}{1+i\gamma k^2\epsilon\eta}\phi). \tag{13.13}$$

It is assumed that $\beta = 1/T_0$. This value is obtained by evaluating the coefficient of thermal expansion for the equilibrium values of the state variables in the case of an ideal gas.

13.2.2 Boundary conditions

Suitable boundary conditions must be adjoined to the field equations (13.1). In a usual manner, one may specify any component of the velocity vector on the boundary. Alternatively, if the component of the velocity vector is not specified at a point on the boundary, the corresponding component of the surface force vector may be prescribed. Similarly, as a boundary condition for the energy equation one may specify either the disturbance of the temperature or zero heat flux (the default boundary condition) on the boundary.

Specifying two impedances on the boundary provides an alternative way of prescribing boundary conditions in the normal direction to the boundary. Firstly, one may specify the specific acoustic impedance Z which is defined to be the ratio of the normal component of the surface force vector (which equals to the pressure in the case of a nonviscous Newtonian fluid with no bulk viscosity) to the normal component of the velocity vector at a point on the boundary, i.e. one may specify

$$Z = \frac{\vec{n}\cdot\overline{\overline{\sigma}}\vec{n}}{\vec{v}\cdot\vec{n}},$$

where $\vec{n}$ is the outward unit normal vector to the boundary. Secondly, one may prescribe the ratio of the heat flux to the disturbance of the temperature at a point on the boundary by specifying

$$Z_T = \frac{\nabla T(x)\cdot\vec{n}}{T(x)}.$$

For example, outgoing waves may be approximated by setting $Z = -\rho_0 c$ and $Z_T = -i\omega/c$ on the outflow boundary.

Slip boundary conditions may also be used. The velocity slip boundary condition relating the tangential component of the surface force vector to the tangential velocity jump at a point on the boundary is written in the form

$$\overline{\overline{\sigma}}\vec{n}\cdot\vec{t} = -\frac{c_\sigma}{2-c_\sigma}(\frac{2(\gamma-1)C_V(T_0+T_w)}{\pi})^{1/2}\rho_0(\vec{v}\cdot\vec{t} - \vec{v}_w\cdot\vec{t}),$$

where $\vec{t}$ is a tangent vector to the boundary, c_σ is the momentum accommodation coefficient and T_w and $\vec{v}_w$ are the reference wall temperature and velocity. Here the reference wall temperature is defined to be the deviation of the wall temperature from the equilibrium temperature T_0. The similar boundary condition for the heat flux is given by

$$\kappa \nabla T \cdot \vec{n} = -\frac{c_T(\gamma+1)}{2(2-c_T)}(\frac{2(\gamma-1)C_V(T_0+T_w)}{\pi})^{1/2}\rho_0 C_V(T-T_w),$$

where c_T is the energy accommodation coefficient.

13.3 The use of block preconditioning

The finite element approximation of the system (13.9) leads usually to large linear systems which have to be solved using preconditioned iterative methods. The general preconditioners available in Elmer may not always work satisfactorily well when the size of the system becomes larger and larger. To facilitate the solution of large problems, a problem-specific strategy for solving the linear systems that arise from the discretization of (13.9) has been developed. We describe the essential features of this solution method in this section; for a full description see [1].

The solution strategy discussed here is based on using nested GCR iterations in combination with a special block-preconditioner. Given the linear system

$$KU = F$$

the standard GCR method generates a sequence of improving approximations such that each iterate $U^{(k)}$ minimizes $||F - KU^{(k)}||$ over the so-called Krylov subspace. The standard algorithm can be modified easily so that the update direction can be chosen flexibly. Obviously, an optimal update direction would be given by the current error $e^{(k)} = U - U^{(k)}$. To find an approximation to the error one may apply an iterative method to

$$Ke^{(k)} = r^{(k)},$$

where $r^{(k)} = F - KU^{(k)}$ is the residual. The preconditioned GCR algorithm which employs this idea to find the update direction can be described as follows:

```
Form an initial guess $U^{(0)}$
$r^{(0)} = F - KU^{(0)}$
$k = 0$
while (Stopping criterion is not met)
        Solve $Ks^{(k+1)} = r^{(k)}$ iteratively using at most $m$ iteration steps
        $v^{(k+1)} = Ks^{(k+1)}$
        do $j = 1, k$
            $v^{(k+1)} = v^{(k+1)} - < v^{(j)}, v^{(k+1)} > v^{(j)}$
            $s^{(k+1)} = s^{(k+1)} - < v^{(j)}, v^{(k+1)} > s^{(j)}$
        end do
        $v^{(k+1)} = v^{(k+1)}/||v^{(k+1)}||$
        $s^{(k+1)} = s^{(k+1)}/||v^{(k+1)}||$
        $U^{(k+1)} = U^{(k)} + < v^{(k+1)}, r^{(k)} > s^{(k+1)}$
        $r^{(k+1)} = r^{(k)} - < v^{(k+1)}, r^{(k)} > v^{(k+1)}$
        $k = k + 1$
end while
```

Here the inner product and norm are defined by $< v, r >= \overline{v} \cdot r$ and $\|v\| =< v, v >^{1/2}$. The GCR iteration steps used to update the approximation of U are referred to as outer iterations, while the iteration steps of the preconditioning iterative method used for solving the new search direction $s^{(k+1)}$ are referred to as inner iterations.

Here the GCR algorithm is also used as the inner iterative method. In connection with the inner iterations a special block-preconditioner is used. The preconditioning is done by solving approximately the block-triangular system of the form

$$\begin{bmatrix} A & B^* & B^* & 0 \\ 0 & C & D & 0 \\ 0 & E & G & H \\ 0 & 0 & 0 & M \end{bmatrix} \begin{bmatrix} s_v \\ s_T \\ s_\phi \\ \psi \end{bmatrix} = \begin{bmatrix} r_v \\ r_T \\ r_\phi \\ r_\phi \end{bmatrix}, \tag{13.14}$$

where s_v, s_T and s_ϕ are update directions for the errors of v, T and ϕ. In addition, ψ is an auxiliary unknown which has been introduced so as to handle the boundary conditions of the preconditioner is a consistent way. In practice, an approximate solution of (13.14) is constructed by applying iterative methods to the systems of the type

$$M\psi = r_\phi, \tag{13.15}$$

$$\begin{bmatrix} C & D \\ E & G \end{bmatrix} \begin{bmatrix} s_T \\ s_\phi \end{bmatrix} = \begin{bmatrix} r_T \\ \hat{r}_\phi \end{bmatrix} \tag{13.16}$$

and

$$As_v = \hat{r}_v, \tag{13.17}$$

where $\hat{r}_\phi$ and $\hat{r}_v$ are modified right-hand sides the computation of which requires the evaluation of certain matrix-vector products. The special solver discussed hence requires that iterations are performed on three levels.

One of the key ideas in the nested application of the GCR algorithm is that the outer iteration can be made rapidly convergent. Consequently the optimality of the outer iteration need not be sacrificed by using such techniques as restarting or truncation. A few inner iterations are usually enough to produce a useful reduction in the outer iteration residual. Therefore the maximum number of iterations the inner iterative method may take need not be large. We have found that limiting the number of inner iterations by taking $m = 5$ (this is the default value) or $m = 10$ leads often to an efficient method. In addition to specifying the maximum number of inner iterations, the user can control the residual reduction in the outer iteration process by specifying the error tolerance δ_{inner} so that the inner GCR iteration is stopped if

$$\|r^{(k)} - K\hat{s}^{(k+1)}\| < \delta_{\text{inner}}\|r^{(k)}\|, \tag{13.18}$$

where $\hat{s}^{(k+1)}$ is the approximation to $s^{(k+1)}$. The default value of δ_{inner} is 0.1.

Ideally a mild stopping criterion should be used in the solution of the linear systems of the type (13.15)–(13.17) which arise in the block-preconditioning of the inner iteration. The iterative solution of (13.15) being a cheap operation, the overall cost of the block-preconditioning is essentially determined by the solution of the systems of the type (13.16) and (13.17). These systems are solved using the preconditioned BiCGStab(l) method. In this connection the Jacobi and incomplete LU factorization preconditioners can be applied.

13.4 Utilities

The dissipative acoustics solver may be used in resolving the acoustic impedance of a system. The value of the impedance defined by

$$z_i = \frac{\int_{S_i} p\, dS}{\int_{S_i} \vec{v}\cdot(-\vec{n})\, dS} \tag{13.19}$$

may automatically be calculated for a given boundary S_i. Here this impedance will be referred to as the specific acoustic impedance of the surface (S_i).

The acoustic impedance is divided into two parts, a part in phase with velocity and a part out of phase with velocity. The value of the impedance z_i is meaningful only when the velocity on the input boundary is considered. It is though possible to calculate the response over an other boundary S_j and to compare it to the input velocity, i.e. one may compute

$$z_{ij} = \frac{\int_{S_j} p\, dS}{\int_{S_i} \vec{v}\cdot(-\vec{n})\, dS}. \tag{13.20}$$

This impedance is here called the cross specific acoustic impedance.

13.5 Keywords

The following keywords are particularly related to the acoustics solver.

`Simulation`

`Angular Frequency` `Real`
: This keyword is used to declare the angular frequency. Alternatively one may define the frequency by using the `Frequency` keyword.

`Frequency` `Real`
: This keyword is used to declare the frequency. Alternatively one may define the angular frequency by using the `Angular Frequency` keyword.

`Simulation Type` `String`
: The value of this keyword should be either `Steady State` or `Scanning`. The value `Scanning` may be used to obtain results for several frequencies by using a single sif-file.

`Coordinate System` `String`
: The coordinate system must be set to be one of the following options: `Cartesian 2D`, `Cartesian 3D` or `Axi Symmetric`.

`Solver` `solver-id`

The following keywords may be used in the solver section that contains solver parameters for the acoustics solver.

`Equation` `String`
: This keyword can be used to give a name for the discrete acoustic equations

`Procedure` `File Acoustics AcousticsSolver`
: This keyword is used to give the Elmer solver the place where to search for the acoustics solver.

`Variable` `String Flow`
: The name `Flow` is used for the solution of the acoustics equations consisting of the amplitudes of the disturbances of the velocity, temperature and pressure from the equilibrium state (note that the disturbance of the density is not computed explicitly). The acoustics solver uses a convention that if dim is the coordinate system dimension then the components $1, ..., 2 \times dim$ of `Flow` give the real and imaginary parts of velocities (`Flow.1` and `Flow.2` are the real and imaginary parts of the first velocity component, etc.). The temperature and pressure solutions come after the velocity solution.

`Variable Dofs` `Integer`
: The value of this keyword should equal to $2 \times (dim + 2)$ where dim is the coordinate system dimension.

`Element` `String`
: The use of standard finite elements in the approximation of the acoustic equations is likely to lead to an unstable method. The finite element formulation can be stabilised by using additional bubble finite element functions in the approximation of velocities. If this keyword is given the value `p:1 b:n`, with n an integer, then n additional bubble functions contained in the p-element library are used in the approximation of each velocity component.

`Bubbles in Global System` `Logical`
: This keyword should be given the value `False`, so that the additional bubble basis functions needed for the stability are eliminated via the static condensation.

`Utilize Previous Solution` `Logical`
: If a single sif-file is used to compute the solutions for several frequencies, then the previous solution can be used as an initial guess for the next iterative solution. This can be done by giving the value `True` for this keyword.

`Material` `material-id`

`Specific Heat` `Real`
: This keyword is used to define the specific heat (per unit mass) at constant volume.

`Specific Heat Ratio` `Real`
: This keyword is used to define the ratio of the specific heats at constant pressure and constant volume.

`Equilibrium Density` `Real`
: This keyword is used to declare the density at the equilibrium state.

`Equilibrium Temperature` `Real`
: This keyword is used to declare the absolute temperature at the equilibrium state.

`Heat Conductivity` `Real`
: This keyword is used to define the value of the heat conductivity.

`Viscosity` `Real`
: This keyword is used to define the value of the viscosity μ.

`Bulk Viscosity` `Real`
: The material parameter λ is determined by giving the bulk viscosity κ' defined by $\kappa' = \lambda + 2/3\mu$. If the value of this keyword is not given, the Stokes condition is assumed, i.e. the value of λ is determined by the condition $\kappa' = 0$.

`Re Heat Source` `Real`
: This keyword is used to define the real part of the heat source (per unit mass).

`Im Heat Source` `Real`
: This keyword is used to define the imaginary part of the heat source (per unit mass).

`Re Body Force i` `Real`
: This keyword is used to define the real part of the i's component of the body force vector (per unit mass).

`Im Body Force i` `Real`
: This keyword is used to define the imaginary part of the i's component of the body force vector (per unit mass).

`Boundary Condition` `bc-id`

`Re Velocity i` `Real`
: This keyword is used to prescribe the real part of the i's component of the velocity vector.

`Im Velocity i` `Real`
: This keyword is used to prescribe the imaginary part of the i's component of the velocity vector.

`Re Temperature` `Real`
: This keyword is used to prescribe the real part of the amplitude of the disturbance of temperature.

`Im Temperature` `Real`
: This keyword is used to prescribe the imaginary part of the amplitude of the disturbance of temperature.

`Re Surface Traction i` `Real`
: This keyword is used to define the real part of the i's component of the surface force vector.

`Im Surface Traction i` `Real`
: This keyword is used to define the imaginary part of the i's component of the surface force vector.

`Re Specific Acoustic Impedance` `Real`
: This keyword is used to define the real part of the ratio of the normal component of the surface force vector to the normal component of the velocity vector at a point on the boundary.

`Im Specific Acoustic Impedance` `Real`
: This keyword is used to define the imaginary part of the ratio of the normal component of the surface force vector to the normal component of the velocity vector at a point on the boundary.

`Re Specific Thermal Impedance` Real
: This keyword is used to define the real part of the ratio of the normal derivative of temperature to the disturbance of the temperature at a point on the boundary.

`Im Specific Thermal Impedance` Real
: This keyword is used to define the imaginary part of the ratio of the normal derivative of temperature to the disturbance of the temperature at a point on the boundary.

`Slip Boundary` Logical
: The value of this keyword should be set to be `True` if slip boundary conditions were given.

`Momentum Accommodation Coefficient` Real
: This keyword is used to define the momentum accommodation coefficient c_σ.

`Energy Accommodation Coefficient` Real
: This keyword is used to define the energy accommodation coefficient c_T.

`Re Reference Wall Velocity i` Real
: This keyword is used to prescribe the real part of the i's component of the reference wall velocity.

`Im Reference Wall Velocity i` Real
: This keyword is used to prescribe the imaginary part of the i's component of the reference wall velocity.

`Reference Wall Temperature` Real
: This keyword is used to define the reference wall temperature.

`Calculate Acoustic Impedance` Logical
: This keyword is used to define the boundary for which the specific acoustic impedance z_i is calculated.

`Impedance Target Boundary` Logical
: When calculating the cross impedance z_{ij}, this keyword defines the boundary S_j. The input velocity boundary (S_i) is defined using the `Calculate Acoustic Impedance` keyword.

The following keywords are related to the use of block preconditioning and may be given in the Solver section.

`Solver` solver-id

`Block Preconditioning` Logical
: The value of this keyword should be set to be `True` to enable the use of block preconditioning.

`Max Outer Iterations` Integer
: The value of this keyword defines the maximum number of outer iterations.

`Max Inner GCR Iterations` Integer
: This keyword is used to define the value of the parameter m, i.e. the maximum number of inner iterations. The default value is 5.

`Ratio of Convergence Tolerances` Real
: This keyword is used to define the stopping criterion for the outer iteration. The outer iteration is stopped when

$$\|F - KU^{(k)}\|_\infty < (\varepsilon_r \times \varepsilon)(\|K\|_\infty \|U^{(k)}\|_\infty + \|F\|_\infty).$$

Here ε_r is defined using this keyword and ε is the value of the `Linear System Convergence Tolerance` keyword. Having $\varepsilon_r \ll 1$ is desirable.

`Residual Reduction Ratio` Real
: This keyword is used to define the value of the parameter δ_{inner} in the stopping criterion (13.18). The default value is 0.1.

`Linear System Convergence Tolerance` Real
: In connection with the block-preconditioning the keyword `Linear System Convergence Tolerance` defines the stopping criterion used in connection with the iterative solution of (13.16) and (13.17). In this connection the stopping criterion of the type

$$\|\hat{r}_v^{(k)} - As_v^{(k)}\| < \varepsilon\|\hat{r}_v^{(k)}\|$$

is used. Here ε is the value of this keyword. It is noted that the solution accuracy of (13.15) need not be specified by the user.

`Velocity Convergence Tolerance` Real
: The systems (13.16) and (13.17) may solved with different degrees of accuracy. Instead of using the `Linear System Convergence Tolerance` keyword one may specify the solution accuracy for (13.17) by using this keyword.

`Schur Complement Convergence Tolerance` Real
: The systems (13.16) and (13.17) may solved with different degrees of accuracy. Instead of using the `Linear System Convergence Tolerance` keyword one may specify the solution accuracy for (13.16) by using this keyword.

`Linear System Max Iterations` Integer
: In connection with the block-preconditioning the `Linear System Max Iterations` keyword is used for defining the maximum number of iteration steps which can be taken in the iterative solution of (13.15)–(13.17).

`Velocity Assembly` Logical
: The coefficient matrix A in (13.17) corresponds to the (1,1) block of the coefficient matrix K. As the elements of A can be extracted from K, the assembly of A can be avoided if a diagonal preconditioning is used in the iterative solution of (13.17). If an incomplete factorization preconditioner is used, the matrix A is assembled explicitly. In this case the value `True` must be given for this keyword.

`ILU Order for Schur Complement` Integer
: The value of this keyword defines the fill level for the incomplete LU factorization preconditioner that is applied in the iterative solution of the linear systems of the type (13.16).

`ILU Order for Velocities` Integer
: The value of this keyword defines the fill level for the incomplete LU factorization preconditioner that is applied in the iterative solution of the linear systems of the type (13.17). If this keyword is not given, then a diagonal preconditioning is used. This keyword has an effect only when the `Velocity Assembly` keyword is given the value `True`.

Bibliography

[1] M. Malinen. Boundary conditions in the schur complement preconditioning of dissipative acoustic equations. *SIAM J. Sci. Comput.*, 29:1567–1592, 2007.

[2] M. Malinen, M. Lyly, P. Råback, A. Kärkkäinen, and L. Kärkkäinen. A finite element method for the modeling of thermo-viscous effects in acoustics. In P. Neittaanmäki et. al., editor, *Proceedings of the 4th European Congress on Computational Methods in Applied Sciences and Engineering*, 2004.

Model 14

Wave Equation

Module name: WaveSolver
Module subroutines: WaveSolver
Module authors: Juha Ruokolainen, Peter Råback, Mika Malinen
Document authors: Mika Malinen

14.1 Introduction

This module can be used to solve a generalized version of the wave equation in time domain. It provides an alternative to the Helmholtz equation which follows from the wave equation by transforming to the frequency domain.

14.2 Theory

This solver can be used to handle the following generalized version of the wave equation:

$$-\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} + \Delta p + \frac{\eta}{c^2}\Delta(\frac{\partial p}{\partial t}) - \frac{\alpha}{c^2}\frac{\partial p}{\partial t} = f, \tag{14.1}$$

with c and f being the sound speed and a source function, respectively. In addition, η and α are model parameters, which can also be set to be zero to obtain the standard wave equation.

In the case of sound waves the equation (14.1) can be obtained from the equation of motion and the continuity equation by linearization. By assuming an isentropic fluid and irrotational flow, we then come to considering the velocity-pressure system

$$\begin{aligned} \rho_0\frac{\partial \boldsymbol{v}}{\partial t} - \hat{\eta}\nabla(\nabla\cdot\boldsymbol{v}) + \hat{\alpha}\boldsymbol{v} - \nabla P &= \rho_0\boldsymbol{b}, \\ \frac{1}{\rho_0 c^2}\frac{\partial P}{\partial t} + \nabla\cdot\boldsymbol{v} &= 0, \end{aligned} \tag{14.2}$$

where ρ_0 is the density of the fluid at the equilibrium state, $\hat{\eta}$ is a parameter related to the fluid viscosity and $\hat{\alpha}$ is a reaction parameter. If we use the continuity equation to eliminate the divergence term from the equation of motion and set $\eta = \hat{\eta}/\rho_0$, $\alpha = \hat{\alpha}/\rho_0$ and $p = P/\rho_0$, the velocity-pressure system can also be expressed as

$$\begin{aligned} \frac{\partial \boldsymbol{v}}{\partial t} + \frac{\eta}{c^2}\nabla(\frac{\partial p}{\partial t}) + \alpha\boldsymbol{v} - \nabla p &= \boldsymbol{b}, \\ \frac{1}{c^2}\frac{\partial p}{\partial t} + \nabla\cdot\boldsymbol{v} &= 0. \end{aligned} \tag{14.3}$$

The elimination of the velocity from the time-differentiated continuity equation

$$-\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} - \nabla\cdot(\frac{\partial \boldsymbol{v}}{\partial t}) = 0 \tag{14.4}$$

then yields (14.1), with $f \equiv \nabla \cdot \boldsymbol{b}$.

Boundary conditions. Basically boundary conditions of three types can be given. In addition to a basic Dirichlet constraint, one can specify the Neumann constraint

$$-\nabla p \cdot \boldsymbol{n} - \frac{\eta}{c^2}\nabla(\frac{\partial p}{\partial t})\cdot \boldsymbol{n} = g. \tag{14.5}$$

In the absence of body forces and reaction terms ($\boldsymbol{b} = \boldsymbol{0}$ and $\alpha = 0$), the value of g is related to the normal component of the acceleration via

$$g = -\frac{\partial \boldsymbol{v}}{\partial t}\cdot \boldsymbol{n}.$$

The final option is to set

$$-\nabla p \cdot \boldsymbol{n} - \frac{\eta}{c^2}\nabla(\frac{\partial p}{\partial t})\cdot \boldsymbol{n} = \frac{1}{c}\frac{\partial p}{\partial t} \tag{14.6}$$

in order to approximate outgoing waves, especially when η is small.

14.3 Keywords

`Solver` `solver id`

The following keywords are especially related to the wave equation solver.

`Equation` `String`

This keyword can be used to give a name for the equation handled by the solver.

`Procedure` `File "WaveSolver" "WaveSolver"`

This specifies the name of the solver subroutine.

`Variable` `String "Excess Pressure"`

The name of the solver variable can be chosen freely.

`Variable DOFs` `Integer`

The value of this keyword must be 1.

`Time Derivative Order` `Integer`

There is no need for using this keyword as the only possible value is 2 and it is given automatically by the solver.

`Initial Condition` `ic-id`

The initial condition section may be used to set initial values for the field variable. The form of the commands depends on the name given for the solver variable.

`Material` `mat id`

The material section is used to give the values of the material parameters.

`Sound Speed` `Real`

This keyword is used to give the value of the sound speed c.

`Sound Damping` `Real`

This keyword is used to give the value of the parameter η.

`Sound Reaction Damping` `Real`

This keyword is used to give the value of the parameter α.

`Body Force` `bf id`

Although volumetric sources may be rare, there is an option to specify such a source.

`Sound Source` `Real`
: This keyword can be used to specify the source function $f \equiv \nabla \cdot \boldsymbol{b}$.

`Boundary Condition` `bc id`
: Special keywords are used to specify Neumann and outflow constraints, while a Dirichlet constraint can be given in a standard manner.

`Excess Pressure` `Real`
: The form of the command to specify a Dirichlet constraint naturally depends on the name given for the solver variable.

`Source Acceleration` `Real`
: This keyword can be used to specify the source function g.

`Outflow Boundary` `Logical`
: This keyword can be used to activate the use of the constraint (14.6). An alternate keyword for the same purpose is `Plane Wave BC`.

Model 15

Large-amplitude Wave Motion in Air

Module names: CompressibleNS
Module subroutines: CompressibleNS
Module authors: Mika Malinen
Document authors: Mika Malinen

15.1 Introduction

This module contains a monolithic solver for the compressible Navier–Stokes equations subject to the ideal gas law. It can be used to model the fully nonlinear wave propagation in the time domain.

15.2 Mathematical model

The acoustic wave motion in a fluid is generally characterized by the compressional Navier–Stokes equations. If the medium obeys the ideal gas law, so that the fluid pressure p satisfies

$$p = R\rho T, \tag{15.1}$$

the Navier–Stokes system may be reduced to consist of the equation of motion

$$\rho[\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v}] - \mu \Delta \vec{v} - (\mu + \lambda)\nabla(\nabla \cdot \vec{v}) + R\rho \nabla T + RT\nabla \rho = \vec{b}, \tag{15.2}$$

the energy equation

$$\rho C_V(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T) - K\Delta T + R\rho T \nabla \cdot \vec{v} = 0, \tag{15.3}$$

and the continuity equation

$$\frac{\partial \rho}{\partial t} + \vec{v} \cdot \nabla \rho + \rho \nabla \cdot \vec{v} = 0. \tag{15.4}$$

Here $\vec{v}$, ρ and T are the fluid velocity, density and temperature, respectively, and the material properties are expressed in terms of the viscosity parameters μ and λ, the heat conductivity K and the specific heat C_V, with $R = (\gamma - 1)C_V$.

If ρ_0 and T_0 are the equilibrium values of density and temperature, we may then write

$$\rho = \rho_0 + \delta, \qquad T = T_0 + \tau, \tag{15.5}$$

so that δ and τ give the disturbances in ρ and T. To solve the coupled system consisting of (15.2)–(15.4) the fully implicit time integration is employed. At each time level a nonlinear iteration is thus applied. Given nonlinear iterates $\vec{v}_k$, τ_k and δ_k, new approximations are generated via

$$\begin{aligned}&(\rho_0+\delta_k)[(\vec{v}_t)_{k+1}+(\vec{v}_k\cdot\nabla)\vec{v}_{k+1}]-\mu\Delta\vec{v}_{k+1}-(\mu+\lambda)\nabla(\nabla\cdot\vec{v}_{k+1})\\&\quad+R(\rho_0+\delta_k)\nabla\tau_{k+1}+R(T_0+\tau_k)\nabla\delta_{k+1}=\vec{b},\\&(\rho_0+\delta_k)C_V[(\tau_t)_{k+1}+\vec{v}_k\cdot\nabla\tau_{k+1})-K\Delta\tau_{k+1}+R(\rho_0+\delta_k)(T_0+\tau_k)\nabla\cdot\vec{v}_{k+1}=0,\\&(\rho_t)_{k+1}+\vec{v}_k\cdot\nabla\delta_{k+1}+(\rho_0+\delta_k)\nabla\cdot\vec{v}_{k+1}=0,\end{aligned}\tag{15.6}$$

with the time derivatives approximated using suitable finite difference schemes. It is recommended that the BDF(2) method is used for the time discretization. It is also noted that the pressure is not approximated directly, so it has to be computed separately using (15.1) and (15.5).

It should be noted that the solver is tailored to the case of the lowest-order continuous temperature and density approximation. To obtain stable finite element solutions the velocity discretization must be enhanced by using elementwise bubble functions or by rising the polynomial order of the velocity approximation. Therefore a special element type definition in the solver input file should be given.

15.3 Keywords

The keywords that are related especially to this solver are described in the following.

`Simulation`

`Coordinate System` String
: The coordinate system must be set to be one of the following options: `Cartesian 2D`, `Cartesian 3D` or `Axi Symmetric`.

`Solver` solver id

`Equation` String
: A name to the equation may be given by using this keyword.

`Procedure` File "CompressibleNS" "CompressibleNS"
: This keyword is used to give the Elmer solver the place where to search for the compressible Navier–Stokes solver.

`Variable` String
: A name to the solver variable should be given by using this keyword.

`Variable DOFs` Integer
: The value of this keyword should equal to $dim + 2$ where dim is the coordinate system dimension. The field variables are organized in such a way that the first dim components correspond to the velocity solution and the temperature and density fluctuations come after the velocity.

`Element` String
: The user has to specify what strategy is used for enhancing the velocity approximation by giving the element type definition. If the command `Element = "p:2"` is given, then the velocity is approximated using the shape functions of the second order elements. The element type definition `Element = "p:1 b:1"` can be given to enhance the velocity approximation with one bubble function.

`Material` mat id

`Equilibrium Density` Real
: The equilibrium density ρ_0 should be specified by using this keyword.

`Equilibrium Temperature` Real
: The equilibrium temperature T_0 should be specified by using this keyword.

`Specific Heat` Real
: The value of this keyword specifies C_V.

`Specific Heat Ratio` Real
: The value of this keyword specifies γ.

`Heat Conductivity` Real
: The heat conductivity K should be defined by using this keyword.

`Viscosity` Real
: The viscosity parameter μ should be defined by using this keyword.

`Bulk Viscosity` Real
: The viscosity parameter λ is taken to be $\lambda = \kappa - 2/3\mu$, with κ the value of this keyword.

`Body Force` bc id

`Body Force i` Real
: This keyword defines the i's component of the body force.

Part IV

Models of Electromagnetism

Model 16

Electrostatics

Module name: StatElecSolve
Module subroutines: StatElecSolver
Module authors: Leila Puska, Antti Pursula, Peter Råback
Document authors: Peter Råback, Antti Pursula

16.1 Introduction

The macroscopic electromagnetic theory is governed by Maxwell's equations. In Elmer it is possible to solve the electrostatic potential in linear dielectric material and in conducting medium. The dielectric case is described in this Chapter. For static currents, refer to Chapter 17. Based on the potential, various field variables as well as physical parameters, such as capacitance, can be calculated.

16.2 Theory

Maxwell's equations are here written as

$$\begin{aligned} \nabla \cdot \vec{D} &= \rho && (16.1)\\ \nabla \cdot \vec{B} &= 0 && (16.2)\\ \nabla \times \vec{E} &= -\frac{\partial \vec{B}}{\partial t} && (16.3)\\ \nabla \times \vec{H} &= \vec{J} + \frac{\partial \vec{D}}{\partial t} && (16.4)\end{aligned}$$

For linear materials the fields and fluxes are simply related, $\vec{B} = \mu\vec{H}$ and $\vec{D} = \varepsilon\vec{E}$, where the permittivity $\varepsilon = \varepsilon_0\varepsilon_r$ is defined through the permittivity of vacuum ε_0 and the relative permittivity of the material ε_r.

In a stationary case the electric field may be expressed with a help of an electric scalar potential ϕ,

$$\vec{E} = -\nabla\phi. \tag{16.5}$$

Assuming linear material law and using the equation (16.1) gives

$$-\nabla \cdot \varepsilon\nabla\phi = \rho. \tag{16.6}$$

This is the electrostatic equation for non-conducting media.

The energy density of the field is

$$e = \frac{1}{2}\vec{E} \cdot \vec{D} = \frac{1}{2}\varepsilon(\nabla\phi)^2. \tag{16.7}$$

Thus the total energy of the field may be computed from

$$E = \frac{1}{2} \int_\Omega \varepsilon (\nabla \phi)^2 d\Omega. \tag{16.8}$$

If there is only one potential difference Φ present then the capacitance C may be computed from

$$C = \frac{2E}{\Phi^2}. \tag{16.9}$$

16.2.1 Boundary Conditions

For electric potential either Dirichlet or Neumann boundary condition can be used. The Dirichlet boundary condition gives the value of the potential on specified boundaries. The Neumann boundary condition is used to give a flux condition on specified boundaries

$$-\varepsilon \nabla \phi \cdot \vec{n} = g. \tag{16.10}$$

The flux may be defined *e.g.* by the surface charge density: $g = \sigma$.

In case there is a object in infinite space it is of course not possible to extent the volume over it. Instead a spherically symmetric approximation may be used. It results to a flux given by

$$g = \varepsilon \phi \frac{\vec{r} \cdot \vec{n}}{r^2}. \tag{16.11}$$

This may be implemented as an additional term to the matrix so that the linear nature of the problem is maintained.

Conductors are often covered by thin oxidation layers which may contain static charges. The effect of these charges can be taken into account by Robin type of boundary condition which combines the fixed potential value on the conductor and the flux condition due to the static charges

$$g = -\frac{\varepsilon_h}{h}\phi + \frac{1}{2}\rho h + \frac{\varepsilon_h}{h}\Phi_0 \qquad \text{on the boundary,} \tag{16.12}$$

where ε_h and h are the permittivity and the thickness of the oxidation layer respectively, ρ is the static charge density of the layer, and Φ_0 is the fixed potential on the conductor.

Note that this formulation is valid only for thin layers. For a larger layer a separate body should be added and a source defined for that.

16.2.2 Capacitance matrix

There is a possibility to compute the capacitance matrix. The algorithm takes use of the original matrix A before the initial conditions are set. Now the point charges are given by

$$q = A\phi. \tag{16.13}$$

The induced charges on a body may be computed by summing up the point charges.

If there are n different bodies the boundary conditions are permuted n times so that body i gets a potential unity while others are set to zero potential,

$$C_{ij} = \sum_{\Gamma_j} q. \tag{16.14}$$

The symmetry of the matrix is ensured afterwards by setting

$$C = \frac{1}{2}(C + C^T). \tag{16.15}$$

16.3 Notes on output control

The user can control which derived quantities (from the list of electric field, electric flux, electric energy, surface charge density and capacitance matrix) are calculated.

There are also available two choices of visualization types for the derived quantities. The node values can be calculated by taking the average of the derived values on neighbouring elements (constant weights). This results often in visually good images. The other possible choice is to weight the average with the size of the elements, which is more accurate and should be used when some other variable depends on these derived values. The latter choice is also the default.

16.4 Keywords

`Constants`

`Permittivity Of Vacuum Real [8.8542e-12]`

`Solver solver id`

`Equation String Stat Elec Solver`

`Variable String Potential`
: This may be of any name as far as it is used consistently also elsewhere.

`Variable DOFs Integer 1`
: Degrees of freedom for the potential.

`Procedure File "StatElecSolve" "StatElecSolver"`

Following are listed four keywords with default values for output control.

`Calculate Electric Field Logical [True]`

`Calculate Electric Flux Logical [True]`

`Calculate Electric Energy Logical [False]`

`Calculate Surface Charge Logical [False]`

`Calculate Capacitance Matrix Logical [False]`

`Capacitance Bodies Integer`
: In case of a capacitance matrix computation the number of bodies at different potential must be given (not accounting the ground).

`Capacitance Matrix Filename String`
: The name of the file where capacitance matrix is being saved. The default is `cmatrix.dat`.

`Constant Weights Logical [True]`
: Used to turn constant weighting on for the results.

`Potential Difference Real`
: Used to give the potential difference for which the capacitance is calculated, when capacitance matrix calculation is not performed. This keyword gives thus the voltage between the electrodes of a simple capacitor. The voltage has to be consistent with the potentials defined in boundary conditions.

`Material mat id`

`Relative Permittivity Real`

`Body Force bodyforce id`

`Charge Density Real`

`Boundary Condition` `bc id`

`Potential` `Real`
: If the name of the primary variable is potential then this sets the Dirichlet boundary condition.

`Electric Flux` `Real`
: Neumann boundary condition in terms of g.

`Surface Charge Density` `Real`
: Another way to define flux condition. Identical to the previous keyword.

`Electric Infinity BC` `Logical`
: The spherical approximation for the open boundaries extending to infinity.

The following five keywords are used if a thin oxidation layer is modeled. Note that these are only active if the `Electric Flux BC` keyword is set to `True`.

`Layer Thickness` `Real`
: Defines the thickness of the oxidation layer. This is presumed to extend on the outside the boundary.

`Layer Relative Permittivity` `Real`
: The relative permittivity of the oxidation layer.

`Layer Charge Density` `Real`
: The volume charge density in the oxidation layer.

`Electrode Potential` `Real`
: The potential on the conductor behind the oxidation layer.

`Capacitance Body` `Integer i`
: These should number from `i=1` up to `Capacitance Bodies`. The ground may be given directly with zero potential or with value 0 for this keyword. This definition is only needed in the computation of the capacitance matrix where the potential is permuted in a very specific way.

Model 17

Static Current Conduction

Module name: StatCurrentSolve
Module subroutines: StatCurrentSolver
Module authors: Leila Puska, Antti Pursula, Peter Råback
Document authors: Antti Pursula

17.1 Note

There is a vectorized and multithreaded version of the solver with almost the same functionality (some more, some less) as module `StatCurrentSolveVec`.

17.2 Introduction

The macroscopic electromagnetic theory is governed by Maxwell's equations. This module solves the electrostatic potential in a conducting medium allowing volume currents and electric power loss (the Joule heating) to be derived.

17.3 Theory

In the electro-quasistatic approximation Maxwell's equations are written as

$$\begin{aligned} \nabla\cdot\vec{D} &= \rho && (17.1)\\ \nabla\times\vec{E} &\simeq 0 && (17.2)\\ \nabla\times\vec{H} &= \vec{J}+\frac{\partial\vec{D}}{\partial t} && (17.3)\end{aligned}$$

so that the electric field may be expressed in terms of an electric scalar potential ϕ as

$$\vec{E} = -\nabla\phi. \tag{17.4}$$

In addition, the continuity equation for electric charges is easily obtained from (17.1) and (17.3):

$$\frac{\partial\rho}{\partial t}+\nabla\cdot\vec{J} = 0. \tag{17.5}$$

Ohm's law for conducting material gives the relationship between the current density and the electric field,

$$\vec{J} = \sigma\vec{E} \tag{17.6}$$

where σ is the electric conductivity. Starting from the continuity equation (17.5) and using the equations (17.6) and (17.4), we get

$$-\nabla\cdot\sigma\nabla\phi = \rho_t. \tag{17.7}$$

This Poisson equation is used to solve the electric potential. The source term is often zero, but in some cases it might be necessary. The vectorized and multithreaded version of the solver can also be used to handle a generalized equation

$$-\nabla\cdot\vec{J} - \nabla\cdot(\varepsilon\frac{\partial\vec{E}}{\partial t}) = 0, \tag{17.8}$$

where ε is the permittivity. This version is a consequence of (17.3).

The volume current density in a conductor is now calculated by

$$\vec{J} = -\sigma\nabla\phi, \tag{17.9}$$

and the density of electric power loss, which is turned into heat, by

$$h = \nabla\phi\cdot\sigma\nabla\phi. \tag{17.10}$$

The latter is often called the Joule heating. The total heating power is found by integrating the above equation over the conducting volume.

The user may also compute the nodal heating which is just the integral of the heating sampled at nodes.

17.3.1 Boundary Conditions

For the electric potential either a Dirichlet or Neumann boundary condition can be used. The Dirichlet boundary condition gives the value of the potential on specified boundaries. The Neumann boundary condition is used to give a current J_b on specified boundaries

$$J_b = \sigma\nabla\phi\cdot\vec{n}. \tag{17.11}$$

17.3.2 Power and current control

Sometimes the desired power or current of the system is known a priori. An additional control may then be applied to guide the system. When the electric potential has been computed, the heating power may be estimated from

$$P = \int_\Omega \nabla\phi\cdot\sigma\nabla\phi\, d\Omega. \tag{17.12}$$

If there is a potential difference U in the system, the effective resistance may also be computed from $R = U^2/P$ and the effective current from $I = P/U$.

The control is achieved by multiplying the potential and all derived fields by a suitable coefficient. For power control the coefficient is

$$C_P = \sqrt{P_0/P}, \tag{17.13}$$

where P_0 is the desired power. For current control the coefficient is

$$C_I = I_0/I, \tag{17.14}$$

where I_0 is the desired total current.

17.4 Note on output control

The user can control which of the derived quantities (*i.e.*, the volume current and the Joule heating) are calculated and additionally specify if the electric conductivity is also output. The latter is useful when the conductivity depends for example on temperature. This feature is available only for cases with isotropic (scalar) conductivities.

Two ways to visualize the derived quantities are available. The node values can be calculated by taking the average of the derived values over a collection of neighbouring elements (constant weights). This often results in visually good images. The other possibility is to weight the average with the size of the elements, which is more accurate and should be used when some other variable depends on these derived values. The latter choice is also the default.

17.5 Keywords

`Solver` `solver id`

- `Equation` `String Stat Current Solver`
- `Variable` `String Potential`
  This may be any name as far as it is also used consistently elsewhere.
- `Variable DOFs` `Integer 1`
  Degrees of freedom for the potential.
- `Procedure` `File "StatCurrentSolve" "StatCurrentSolver"`

The following two keywords control the calculation of derived fields.

- `Calculate Volume Current` `Logical [True]`
- `Calculate Joule Heating` `Logical [True]`
- `Constant Weights` `Logical [True]`
  Used to turn constant weighting on for the results.
- `Calculate Nodal Heating` `Logical [True]`
  Calculate nodal heating that may be used to couple the heat equation optimally when using conforming finite element meshes.
- `Power Control` `Real`
  Apply power control with the desired heating power being P_0.
- `Current Control` `Real`
  Apply current control with the desired current being I_0.

`Material` `mat id`

- `Electric Conductivity` `Real`

`Body Force` `bodyforce id`

- `Current Source` `Real`
  This enables a scalar-valued source, not used often though.
- `Joule Heat` `Logical`
  If this flag is active, the heat equation solver will automatically compute the quantity $\nabla\phi \cdot \sigma\nabla\phi$ as heat source. Then it is assumed that ϕ is named `Potential`. If there is no heat equation, this flag has no effect.

`Boundary Condition` `bc id`

- `Potential` `Real`
  Dirichlet BC for the potential.
- `Current Density BC` `Logical`
  Must be set to `True` if Neumann BC is used.
- `Current Density` `Real`
  Neumann boundary condition for the current.

Model 18

Computation of Magnetic Fields in 3D

Module name: MagnetoDynamics
Module subroutines: WhitneyAVSolver,WhitneyAVHarmonicSolver,MagnetoDynamicsCalcFields
Module authors: Juha Ruokolainen, Mika Malinen, Eelis Takala
Document authors: Mika Malinen, Juha Ruokolainen, Juhani Kataja, Eelis Takala, Peter Råback

18.1 Introduction

This module may be used to solve a version of the Maxwell equations in the $\boldsymbol{A}$-V form. The approximation of the associated vector potential variable $\boldsymbol{A}$ is here done by using vector-valued (edge) finite element basis functions, while the classic Lagrange interpolation is applied to compute the scalar potential V. The use of edge finite elements limits the applicability of this solver to 3D problems. In addition to performing the computations in the time domain, the analogous version of the equations may also be solved in the frequency domain. Several ways to handle stationary equations are also included. Furthermore, an additional solver may be called to produce nodal and elementwise approximations of derived fields after the two potentials have been obtained.

The equations over moving media are treated in the case of rigid motion. This implies that the associated velocity field is divergence-free and the velocity gradient is both constant and skew.

18.2 The transient equations

If a charge q moves in a region in which there are both electric and magnetic fields, the electromagnetic force on the charge moving with velocity $\boldsymbol{v}$ is experimentally observed to be given by the Lorentz force

$$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}). \tag{18.1}$$

Alternatively, the force exerted on a volume element $\mathrm{d}\Omega$ is therefore given by

$$\mathrm{d}\boldsymbol{F} = (\rho\boldsymbol{E} + \rho\boldsymbol{v} \times \boldsymbol{B})\mathrm{d}\Omega \tag{18.2}$$

where ρ is the charge density, per unit volume. By assuming a short relaxation time of the charge's motion as compared with the characteristic time scale of the motion of the body, we may incorporate the notion of such electromagnetic field in Ohm's law by writing it as

$$\boldsymbol{J} = \sigma(\frac{\boldsymbol{F}}{q}) \equiv \sigma\boldsymbol{E}', \tag{18.3}$$

where $\boldsymbol{J}$ is the current density and σ is the electrical conductivity. An alternative would be to define the electromagnetic field $\boldsymbol{E}'$ as the ratio of the volume force density to the charge density, but both the definitions lead to the conclusion that in order to explain the Lorentz force we must have

$$\boldsymbol{E}' = \boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}. \tag{18.4}$$

We note that since our definition of the electromagnetic field depends on the notions of force and charge which are assumed to be invariant quantities (under a change of observer), the electromagnetic field also has an invariant nature. It is tacit here that if the separation of the known electromagnetic field into its electric and magnetic components was sought via (18.4), the result would depend on the observer. However, we shall not need such separation since solving for $\boldsymbol{E}'$ is enough to find the current density.

Otherwise the content of electromagnetic equations on a region Ω is here cast into a magnetic version of Maxwell's equations

$$\begin{aligned} \mathbf{curl}\,\boldsymbol{E} &= -\frac{\partial \boldsymbol{B}}{\partial t}, && (18.5)\\ \mathbf{curl}(\frac{1}{\mu}\boldsymbol{B}) - \boldsymbol{J} &= \boldsymbol{g}, && (18.6)\\ \operatorname{div}\boldsymbol{B} &= 0, && (18.7)\\ \operatorname{div}\boldsymbol{J} &= 0, && (18.8) \end{aligned}$$

where μ is the permeability and $\boldsymbol{g}$ is a divergence-free source satisfying (for ways to ensure this condition beforehand, see the description of the Helmholtz projection in the context of stationary equations below)

$$\operatorname{div}\boldsymbol{g} = 0. \tag{18.9}$$

Writing this system in terms of the electromagnetic field defined by (18.4) via eliminating $\boldsymbol{E}$ and using Ohm's law (18.3) yields

$$\begin{aligned} \frac{\partial \boldsymbol{B}}{\partial t} - \mathbf{curl}(\boldsymbol{v}\times\boldsymbol{B}) + \mathbf{curl}\,\boldsymbol{E}' &= \mathbf{0}, && (18.10)\\ \mathbf{curl}(\frac{1}{\mu}\boldsymbol{B}) - \sigma\boldsymbol{E}' &= \boldsymbol{g}, && (18.11)\\ \operatorname{div}\boldsymbol{B} &= 0, && (18.12)\\ \operatorname{div}(\sigma\boldsymbol{E}') &= 0. && (18.13) \end{aligned}$$

The content of this system is adequate for finding the instantaneous spatial solutions $\boldsymbol{B}(\mathbf{x},t)$ and $\boldsymbol{E}'(\mathbf{x},t)$ at time t simultaneously over both static and moving parts of the model. Here the point $\mathbf{x}$ is thus specified by its coordinates with respect to a single, fixed Cartesian coordinate frame.

If an Elmer model contains moving bodies, the spatial discretization is typically done over an instantaneous configuration whose representation is obtained via a rigid motion of the previous mesh into the current configuration. In this case the time stepping machinery of Elmer by default produces an approximation to the material time derivative of a spatial field (the material point is held fixed in the differentiation) when the field is a scalar or vector field. It agrees with the usual definition of the substantial (or total) time derivative of the spatial field. However, when the field is a one-form and is approximated by using edge finite elements (here the case of the vector potential), a more subtle approach is needed to show that the computational procedure leads to an approximation of the Lie derivative [1].

Since the computational procedure does not make it straightforward to evaluate the standard partial derivatives with respect to time, the need of rewriting the equations in forms which employ other time derivatives arises as a natural endeavour. To this end, we shall consider the upper convected time derivative

$$\frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t} = \frac{\partial \boldsymbol{B}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{B} - (\boldsymbol{B}\cdot\nabla)\boldsymbol{v} + (\operatorname{div}\boldsymbol{v})\boldsymbol{B}, \tag{18.14}$$

which agrees with the Lie derivative $\boldsymbol{L_v B}$ of the vector $\boldsymbol{B}$ with respect to $\boldsymbol{v}$ when the motion is rigid so that

$$\operatorname{div}\boldsymbol{v} = 0. \tag{18.15}$$

It can be expressed in terms of the substantial time derivative

$$\frac{D\boldsymbol{B}}{Dt} \equiv \frac{\partial \boldsymbol{B}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{B} \tag{18.16}$$

as

$$\frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t} = \frac{D\boldsymbol{B}}{Dt} - (\boldsymbol{B}\cdot\nabla)\boldsymbol{v} + (\operatorname{div}\boldsymbol{v})\boldsymbol{B}. \tag{18.17}$$

By using vector identities, the definition (18.14) can also be rewritten as

$$\frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t} = \frac{\partial\boldsymbol{B}}{\partial t} - \mathbf{curl}(\boldsymbol{v}\times\boldsymbol{B}) + \boldsymbol{v}(\operatorname{div}\boldsymbol{B}). \tag{18.18}$$

This immediately implies that, under Gauss's law for the magnetic field (18.12), the content of the Faraday law (18.10) can be expressed as

$$\frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t} + \mathbf{curl}\,\boldsymbol{E}' = \mathbf{0}. \tag{18.19}$$

Although we consider (18.19) to be a convenient way to express the Faraday law for our purposes, we do not want to assert whether this useful mathematical reduction happens by chance or whether it should be considered to reflect some theoretical underpinning.

If we think of $\boldsymbol{A}$ as a one-form, the formula for the Lie derivative of $\boldsymbol{A}$ with respect to $\boldsymbol{v}$ should be written as (see, for example, [1])

$$\boldsymbol{L_v A} = \frac{\partial\boldsymbol{A}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{A} + (\nabla\boldsymbol{v})^T\boldsymbol{A}. \tag{18.20}$$

In the case of rigid motion, however, the velocity gradient is skew so that

$$(\nabla\boldsymbol{v})^T = -\nabla\boldsymbol{v}. \tag{18.21}$$

Hence the vector representations for the Lie differentiation of vectors and one-forms become the same.

When the edge finite element basis functions defined on the reference element give the pull-back of the basis functions on the physical element by the element mapping (the standard construction followed by Elmer), evaluating the rate of change of the associated degrees of freedom gives coefficients to approximate the finite element interpolant of the Lie derivative. That is, if $\varphi_k(\cdot)$ are linear functionals which define the degrees of freedom of a one-form, it follows that

$$\varphi_k(\boldsymbol{L_v A}) \doteq \frac{1}{\delta t}[\varphi_k(\boldsymbol{A}(\cdot,\tau+\delta t)) - \varphi_k(\boldsymbol{A}(\cdot,\tau))] \tag{18.22}$$

where τ is a fixed value of time and δt is a time increment. Thus writing the equations such that the Lie derivatives are employed is seen to be very natural.

In practice the solution is sought in terms of potentials. In the first place we set

$$\boldsymbol{B} = \mathbf{curl}\,\boldsymbol{A}. \tag{18.23}$$

It should be noted that this implies some ambiguity, since the resulting set of equations does not have a unique solution without imposing additional constraints on $\boldsymbol{A}(\mathbf{x},t)$. Otherwise, if $\boldsymbol{A}$ satisfies the equations, any field $\boldsymbol{A}_\phi$ having the decomposition $\boldsymbol{A}_\phi = \boldsymbol{A} + \nabla\phi$ may also be made to be a solution. The uniqueness of $\boldsymbol{A}$ could be assured for example by seeking $\boldsymbol{A}(\cdot,t)\in\boldsymbol{H}(\mathbf{curl},\Omega)\cap\boldsymbol{H}(\operatorname{div},\Omega)$ that satisfies additionally

$$\operatorname{div}\boldsymbol{A} = 0$$

on Ω and

$$\boldsymbol{A}\cdot\boldsymbol{n} = 0$$

on the boundary $\partial\Omega$.

In the case of the vector potential description (18.23) the Faraday law in the form (18.10) yields

$$\mathbf{curl}(\frac{\partial\boldsymbol{A}}{\partial t} - \boldsymbol{v}\times\mathbf{curl}\,\boldsymbol{A} + \boldsymbol{E}') = \mathbf{0},$$

so we may automate the satisfaction of the Faraday law by seeking the solution in the form

$$\frac{\partial\boldsymbol{A}}{\partial t} - \boldsymbol{v}\times\mathbf{curl}\,\boldsymbol{A} + \boldsymbol{E}' = -\nabla\psi, \tag{18.24}$$

with ψ being an unknown scalar potential. Switching to the substantial time derivative then gives

$$\frac{D\boldsymbol{A}}{Dt} - (\boldsymbol{v}\cdot\nabla)\boldsymbol{A} - \boldsymbol{v}\times\mathbf{curl}\,\boldsymbol{A} + \boldsymbol{E}' = -\nabla\psi. \tag{18.25}$$

Straightforward calculation shows that this is equivalent to

$$\frac{D\boldsymbol{A}}{Dt} - (\nabla\boldsymbol{A})^T\boldsymbol{v} + \boldsymbol{E}' = -\nabla\psi \tag{18.26}$$

and, by using the vector identity

$$(\nabla\boldsymbol{A})^T\boldsymbol{v} = \nabla(\boldsymbol{v}\cdot\boldsymbol{A}) - (\nabla\boldsymbol{v})^T\boldsymbol{A}, \tag{18.27}$$

we further obtain

$$\frac{D\boldsymbol{A}}{Dt} + (\nabla\boldsymbol{v})^T\boldsymbol{A} + \boldsymbol{E}' = -\nabla(\psi - \boldsymbol{v}\cdot\boldsymbol{A}). \tag{18.28}$$

By setting

$$\psi = V + \boldsymbol{v}\cdot\boldsymbol{A}, \tag{18.29}$$

the content of (18.28) can finally be expressed as

$$\boldsymbol{L_v A} + \boldsymbol{E}' = -\nabla V. \tag{18.30}$$

The equation (18.30) offers the possibility of eliminating the field $\boldsymbol{E}'$ from the set of primary unknowns while the additional constraint (18.13) serves the determination of V. It should be noted that the scalar potential ψ is related to the field $\boldsymbol{E}$ via

$$\frac{\partial\boldsymbol{A}}{\partial t} + \boldsymbol{E} = -\nabla\psi.$$

The relation (18.29) can be used to recover ψ after $\boldsymbol{A}$ and V have been solved.

To derive a weak version of the equations, let v be an appropriate test function for V, so that we have $v \in H^1(\Omega)$. Multiplying (18.13) with v, integrating by parts and using (18.30) bring us to the weak formulation

$$\int_\Omega \sigma(\boldsymbol{L_v A})\cdot\nabla v\,d\Omega + \int_\Omega \sigma\nabla V\cdot\nabla v\,d\Omega = -\int_{\partial\Omega}(\sigma\boldsymbol{E}')\cdot\boldsymbol{n}\,v\,dS. \tag{18.31}$$

The determination of the scalar potential V is thus joined with the possibility of specifying either V or the normal component of the current density $\sigma\boldsymbol{E}'$ on the boundary. If the normal component of the current density is specified on the entire boundary $\partial\Omega$ as

$$-(\sigma\boldsymbol{E}')\cdot\boldsymbol{n} = j_n, \tag{18.32}$$

the boundary data must satisfy the compatibility condition

$$\int_{\partial\Omega} j_n\,dS = 0.$$

On the other hand, by using (18.23) and (18.30), we may rewrite (18.11) as

$$\sigma(\boldsymbol{L_v A}) + \sigma\nabla V + \mathbf{curl}(\frac{1}{\mu}\,\mathbf{curl}\,\boldsymbol{A}) = \boldsymbol{g} \tag{18.33}$$

to obtain the weak version

$$\begin{aligned}\int_\Omega \sigma(\boldsymbol{L_v A})\cdot\boldsymbol{\eta}\,d\Omega + \int_\Omega \sigma\nabla V\cdot\boldsymbol{\eta}\,d\Omega + \int_\Omega \frac{1}{\mu}\,\mathbf{curl}\,\boldsymbol{A}\cdot\mathbf{curl}\,\boldsymbol{\eta}\,d\Omega \\ + \int_{\partial\Omega}(\frac{1}{\mu}\,\mathbf{curl}\,\boldsymbol{A})\cdot(\boldsymbol{\eta}\times\boldsymbol{n})\,dS = \int_\Omega \boldsymbol{g}\cdot\boldsymbol{\eta}\,d\Omega,\end{aligned} \tag{18.34}$$

with $\boldsymbol{\eta}$ an appropriate test function corresponding to $\boldsymbol{A}$. The weak formulations obtained from (18.34) and (18.31) generally form the basis for the $\boldsymbol{A}$-V formulation of the problem.

It should be noted that here we have avoided seeking a unique solution in the space

$$\boldsymbol{A}(\cdot,t) \in \boldsymbol{H}(\mathbf{curl},\Omega) \cap \boldsymbol{H}(\mathrm{div},\Omega)$$

as only the requirement $\boldsymbol{A}(\cdot,t) \in \boldsymbol{H}(\mathbf{curl},\Omega)$ appears to be necessary in this derivation. The Elmer implementation relies on this minimal regularity assumption so that a finite element approximation $\boldsymbol{A}_h(\cdot,t)$ is sought from an edge finite element space $\boldsymbol{X}_h \subset \boldsymbol{H}(\mathbf{curl},\Omega)$. This simplification however leads to the inconvenience that the uniqueness of the vector potential solution cannot be ensured. Thus, given a solution $(\boldsymbol{A},V)$, we can generate another solution $(\boldsymbol{A}_\phi,V_\phi) \equiv (\boldsymbol{A}+\nabla\phi, V - L_{\boldsymbol{v}}\phi)$, although the associated fields $\boldsymbol{B}$ and $\boldsymbol{E}'$ remain invariant.

The steady state solution of the evolutionary equations. Sometimes the steady state solution of the evolutionary model may be of an interest. For example the steady state solution of a case where a homogeneous disk is spun with a constant angular velocity in a region in which there is a steady magnetic field without circumferential variation can be described over a fixed domain. Then (18.24) yields

$$\boldsymbol{E}' - \boldsymbol{v} \times \mathbf{curl}\,\boldsymbol{A} = -\nabla\psi. \tag{18.35}$$

In this special case the scalar potential variable of the solver is taken to be ψ.

In this connection, we note that when a body makes a rigid rotation about the line through the origin $\mathbf{o}$ spanned by the angular velocity $\boldsymbol{\omega}$, the velocity field is described by the formula

$$\boldsymbol{v}(\mathbf{x},t) = \boldsymbol{v}(\mathbf{o},t) + \boldsymbol{\omega}(t) \times (\mathbf{x}-\mathbf{o}). \tag{18.36}$$

We also note that the term $(\nabla\boldsymbol{v})^T\boldsymbol{A}$ occurring in the expression for the Lie Derivative has the following representation in terms of $\boldsymbol{\omega}$

$$(\nabla\boldsymbol{v})^T\boldsymbol{A} = -(\nabla\boldsymbol{v})\boldsymbol{A} = -\boldsymbol{\omega}\times\boldsymbol{A}. \tag{18.37}$$

A Poynting's theorem for electromagnetic energy. The magnetic energy density is defined by

$$W_m = \frac{1}{2}\boldsymbol{B}\cdot\boldsymbol{H}. \tag{18.38}$$

Let us define a Poynting vector $\boldsymbol{S}'$ corresponding to our definition of the electromagnetic field $\boldsymbol{E}'$ by

$$\boldsymbol{S}' = \boldsymbol{E}' \times \boldsymbol{H}, \tag{18.39}$$

with $\boldsymbol{B} = \mu\boldsymbol{H}$. In view of (18.19) and (18.11), we then obtain

$$\begin{aligned}\mathrm{div}\,\boldsymbol{S}' &= \mathbf{curl}\,\boldsymbol{E}'\cdot\boldsymbol{H} - \boldsymbol{E}'\cdot\mathbf{curl}\,\boldsymbol{H} \\ &= -\frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t}\cdot\boldsymbol{H} - \boldsymbol{E}'\cdot(\sigma\boldsymbol{E}'+\boldsymbol{g}).\end{aligned} \tag{18.40}$$

On the other hand, the Lie derivative of the magnetic energy density (note that for scalar-valued functions the expressions of the Lie derivative and the total derivative are equivalent) is given by

$$\frac{\mathcal{D}}{\mathcal{D}t}(\frac{1}{2}\boldsymbol{B}\cdot\boldsymbol{H}) = \frac{\mathcal{D}\boldsymbol{B}}{\mathcal{D}t}\cdot\boldsymbol{H}. \tag{18.41}$$

We hence come to the conservation principle of the magnetic energy expressed as

$$\frac{\mathcal{D}W_m}{\mathcal{D}t} + \mathrm{div}\,\boldsymbol{S}' = -\boldsymbol{E}'\cdot(\sigma\boldsymbol{E}'+\boldsymbol{g}). \tag{18.42}$$

The right-hand side of (18.42) describes the power per unit volume of the heating effect of the electric current which is called the Joule effect.

Since the displacement current has been neglected in the magnetic version of the equations considered, the above conservation principle does not contain an electric energy density term. Elmer however calculates the total electromagnetic energy density as

$$W_m + W_e$$

with

$$W_e = \frac{1}{2}\varepsilon \boldsymbol{E}' \cdot \boldsymbol{E}'. \tag{18.43}$$

Here ε is the permittivity.

Solution in the frequency domain. The equations in the A-V form may also be solved in the frequency domain. In this case the ansatz $\boldsymbol{A}(x,t) = \boldsymbol{A}(x)\exp(i\omega t)$ and $V(x,t) = V(x)\exp(i\omega t)$ is made to obtain the analogous set of equations for determining the complex-valued amplitudes $\boldsymbol{A}(x)$ and $V(x)$. In addition, we employ (18.24) with $\psi = V$.

18.3 The stationary equations and the Helmholtz projection of a source

In the stationary case the electric field $\boldsymbol{E}'$ is simply the gradient of a scalar potential. The weak formulation based on (18.34) and (18.31) then simplifies to

$$\int\limits_{\Omega} \sigma \nabla V \cdot \boldsymbol{\eta}\, d\Omega + \int\limits_{\Omega} \frac{1}{\mu} \,\mathbf{curl}\, \boldsymbol{A} \cdot \mathbf{curl}\, \boldsymbol{\eta}\, d\Omega + \int\limits_{\partial\Omega} (\frac{1}{\mu}\,\mathbf{curl}\, \boldsymbol{A}) \cdot (\boldsymbol{\eta} \times \boldsymbol{n})\, dS = \int\limits_{\Omega} \boldsymbol{g} \cdot \boldsymbol{\eta}\, d\Omega \tag{18.44}$$

and

$$\int\limits_{\Omega} \sigma \nabla V \cdot \nabla v \, d\Omega = -\int\limits_{\partial\Omega} (\sigma \boldsymbol{E}') \cdot \boldsymbol{n}\, v\, dS. \tag{18.45}$$

As the solution of V and $\boldsymbol{A}$ can then be done sequentially by first solving for V, the basic scenario for applying the solver to stationary cases is that only the field $\boldsymbol{A}$ is solved by employing the weak formulation

$$\int\limits_{\Omega} \frac{1}{\mu}\,\mathbf{curl}\, \boldsymbol{A} \cdot \mathbf{curl}\, \boldsymbol{\eta}\, d\Omega + \int\limits_{\partial\Omega} (\frac{1}{\mu}\,\mathbf{curl}\, \boldsymbol{A}) \cdot (\boldsymbol{\eta} \times \boldsymbol{n})\, dS = \int\limits_{\Omega} \boldsymbol{g}_S \cdot \boldsymbol{\eta}\, d\Omega \tag{18.46}$$

where $\boldsymbol{g}_S$ denotes the static source.

It should be noted that the source should generally be divergence-free. The divergence-freeness of $\boldsymbol{g}_S$ may be assured by setting $\boldsymbol{g}_S = \mathcal{P}(\boldsymbol{g}_U)$ where $\boldsymbol{g}_U$ is the user-supplied source term and the Helmholtz projection $\mathcal{P}(\boldsymbol{g}_U) = \boldsymbol{g}_U - \nabla Q$, with $Q \in M \subset H^1(\Omega)$, is defined via the requirement

$$\int\limits_{\Omega} \nabla \cdot (\boldsymbol{g}_U - \nabla Q) q \, d\Omega = 0 \tag{18.47}$$

for any admissible variation q of Q. Integration by parts is used to obtain the version

$$\int\limits_{\Omega} \nabla Q \cdot \nabla q \, d\Omega = \int\limits_{\Omega} \boldsymbol{g}_U \cdot \nabla q \, d\Omega - \int\limits_{\partial\Omega} \mathcal{P}(\boldsymbol{g}_U) \cdot \boldsymbol{n} q \, dS. \tag{18.48}$$

If $\boldsymbol{g}_U$ has already been obtained from a scalar field V^s as

$$\boldsymbol{g}_U = -\sigma \nabla V^s, \tag{18.49}$$

to obtain a close resemblance of $\boldsymbol{g}_U$ and $\mathcal{P}(\boldsymbol{g}_U)$ (especially, $\mathcal{P}(\boldsymbol{g}_U) = \boldsymbol{g}_U$ when $\boldsymbol{g}_U$ is already divergence-free) it is natural to set

$$\mathcal{P}(\boldsymbol{g}_U) \cdot \boldsymbol{n} = \boldsymbol{g}_U \cdot \boldsymbol{n} \tag{18.50}$$

on the boundary where the current density was prescribed. On the remaining part of the boundary where the Dirichlet constraint was specified for V^s the homogeneous Dirichlet constraint for the field Q may be used.

However, by default a different strategy is applied by setting $Q = 0$ on parts where $|\boldsymbol{g}_U \cdot \boldsymbol{n}| \neq 0$. Then the boundary integral in (18.48) vanishes. The Helmholtz projection of the source may also be performed when the evolutionary version of the equations is handled.

Other types of source vectors may also be considered as the user may generally specify the source in the form

$$\boldsymbol{g} = \mathcal{P}(\boldsymbol{g}_U) + \mathbf{curl}\,\boldsymbol{M}^s - \sigma \nabla V^s \tag{18.51}$$

or, if the Helmholtz projection is not applied,

$$\boldsymbol{g} = \boldsymbol{g}_U + \mathbf{curl}\,\boldsymbol{M}^s - \sigma \nabla V^s. \tag{18.52}$$

Here $\boldsymbol{M}^s$ is referred to as the magnetization. It arises from using a constitutive law of the form

$$\boldsymbol{H} = \mu^{-1}\boldsymbol{B} - \boldsymbol{M}^s. \tag{18.53}$$

The last terms in (18.51) and (18.52) enable the direct generation of the source electric current density in terms of the source potential V^s without first computing $\boldsymbol{g}_U$ from (18.49).

To conclude, we note that it is also possible to solve the stationary equations such that both $\boldsymbol{A}$ and V are handled as unknowns and solved simultaneously by employing the variational equations

$$\int_\Omega \sigma \nabla V \cdot \boldsymbol{\eta}\, d\Omega + \int_\Omega \frac{1}{\mu}\mathbf{curl}\,\boldsymbol{A} \cdot \mathbf{curl}\,\boldsymbol{\eta}\, d\Omega + \int_{\partial\Omega} (\frac{1}{\mu}\mathbf{curl}\,\boldsymbol{A}) \cdot (\boldsymbol{\eta} \times \boldsymbol{n})\, dS = \int_\Omega \mathcal{P}(\boldsymbol{g}_U) \cdot \boldsymbol{\eta}\, d\Omega \tag{18.54}$$

and

$$\int_\Omega \sigma \nabla V \cdot \nabla v\, d\Omega = -\int_{\partial\Omega} (\sigma \boldsymbol{E}') \cdot \boldsymbol{n}\, v\, dS. \tag{18.55}$$

18.4 The boundary conditions

We see from the weak versions (18.34) and (18.31) that, if the Dirichlet type constraints

$$\boldsymbol{A} \times \boldsymbol{n} = \boldsymbol{a} \times \boldsymbol{n} \quad \text{and} \quad V = \hat{v} \tag{18.56}$$

are not given, we may specify the tangential components of the magnetic field $\boldsymbol{H} = (1/\mu)\boldsymbol{B}$ and the normal component of current density $\boldsymbol{J} = \sigma \boldsymbol{E}'$. These may be defined via giving $\boldsymbol{h}$ to define

$$\frac{1}{\mu}\mathbf{curl}\,\boldsymbol{A} = \boldsymbol{n} \times \boldsymbol{h} \times \boldsymbol{n} \tag{18.57}$$

and

$$-(\sigma \boldsymbol{E}') \cdot \boldsymbol{n} = j_n. \tag{18.58}$$

We note that giving a Dirichlet constraint is a useful way to guarantee the uniqueness of the scalar potential V which would otherwise be determined only up to a constant.

A Robin-like generalization of (18.57) leads to the boundary condition

$$\frac{1}{\mu}\mathbf{curl}\,\boldsymbol{A} = \alpha(\boldsymbol{A} \times \boldsymbol{n}) + \boldsymbol{n} \times \boldsymbol{h} \times \boldsymbol{n}, \tag{18.59}$$

with α being a given parameter. We however note that the case $\alpha \neq 0$ may lack in having a physical interpretation. On the other hand, a generalized version of (18.58) is written as

$$-(\sigma \boldsymbol{E}') \cdot \boldsymbol{n} = j_n - \beta V, \tag{18.60}$$

which in the stationary case reduces to the Robin boundary condition

$$\sigma \nabla V \cdot \boldsymbol{n} + \beta V = j_n. \tag{18.61}$$

In addition, a procedural technique may be applied to specify a Dirichlet constraint for $\boldsymbol{A}$ when the normal component of the magnetic flux density $\boldsymbol{B}$ is given on the boundary.

The surface impedance condition for time-harmonic cases. In the case of time-harmonic analysis it is possible to generate tangential surface currents via giving a boundary condition

$$\boldsymbol{E} \times \boldsymbol{n} = Z_S(\boldsymbol{H} \times \boldsymbol{n}) \times \boldsymbol{n} \tag{18.62}$$

where Z_S is referred to as the surface impedance and defined as

$$Z_S = \frac{1+i}{\sigma\delta}. \tag{18.63}$$

Here δ denotes the skin depth

$$\delta = \sqrt{2/(\sigma\mu\omega)}. \tag{18.64}$$

18.5 A consistently regularized formulation for magnetostatics

The possibility to seek a unique vector potential solution to the magnetostatic equations is included here as a special case via using the scalar variable to impose the divergence-free constraint on $\boldsymbol{A}$ in a weak manner. This strategy employs the weak formulation

$$\begin{gathered}\int\limits_\Omega \frac{1}{\mu} \,\mathbf{curl}\, \boldsymbol{A} \cdot \mathbf{curl}\, \boldsymbol{\eta}\, d\Omega - \int\limits_\Omega \nabla\phi \cdot \boldsymbol{\eta}\, d\Omega = \int\limits_\Omega \mathcal{P}(\boldsymbol{g}_U) \cdot \boldsymbol{\eta}\, d\Omega, \\ \int\limits_\Omega (-\boldsymbol{A} + k\nabla\phi) \cdot \nabla q\, d\Omega = 0\end{gathered} \tag{18.65}$$

with $\phi \in H^1(\Omega)/P_0(\Omega)$ (the definition of the space $H^1(\Omega)/P_0(\Omega)$ reflects the fact that ϕ can be determined up to a freely chosen constant only) and k being a regularization/stabilization parameter. The choice $k = 1$ is seen to be natural. Then, if $\Pi\boldsymbol{A} \equiv \boldsymbol{A} - \nabla\phi$ denotes the Helmholtz projection of $\boldsymbol{A}$, the content of this formulation can also be cast into the form

$$\begin{gathered}\int\limits_\Omega \frac{1}{\mu} \,\mathbf{curl}\, \boldsymbol{A} \cdot \mathbf{curl}\, \boldsymbol{\eta}\, d\Omega - \int\limits_\Omega (\boldsymbol{A} - \Pi\boldsymbol{A}) \cdot (\boldsymbol{\eta} - \Pi\boldsymbol{\eta})\, d\Omega = \int\limits_\Omega \mathcal{P}(\boldsymbol{g}_U) \cdot \boldsymbol{\eta}\, d\Omega, \\ -\int\limits_\Omega \Pi\boldsymbol{A} \cdot \nabla q\, d\Omega = 0.\end{gathered} \tag{18.66}$$

This formulation reflects how the scheme works; the second term in the left-hand side of the first equation in (18.66) acts as a penalization term to get into the right set of divergence-free solutions. That is, $\boldsymbol{A} - \Pi\boldsymbol{A} = \boldsymbol{0}$ is enforced by having this term. It is noted that with $k = 0$ the method reduces to the standard Lagrange multiplier formulation to impose the Coulomb gauge.

18.6 Keywords

Keywords for WhitneyAVSolver

Here we list the keywords that are relevant to solving the evolutionary and stationary versions of the equations by calling the solver subroutine `WhitneyAVSolver` and that may also be common to the other solvers. Such common keywords relate to specifying material parameters, body forces, and boundary conditions.

`Constants`

`Permeability of Vacuum` `Real`
This constant has the default value $4\pi \cdot 10^{-7}$.

`Material` `mat id`

The following material parameters may be used by all the solvers in the module.

`Electric Conductivity` Real
: This keyword is used to specify the electric conductivity σ. If the material is anisotropic, the electric conductivity may also be defined to be a tensor function by giving its components with respect to the global coordinate lines.

`Relative Permeability` Real
: If this keyword is used, the permeability μ can be specified in terms of the permeability of vacuum. To obtain the permeability, the value of this keyword is then internally multiplied with the permeability of vacuum. Instead of using this keyword, the keywords `Permeability` or `Reluctivity` may be used.

`Permeability` Real
: This keyword may be used to specify directly the permeability μ.

`Reluctivity` Real
: The value of this keyword specifies the reluctivity ν. The permeability is then taken to be $\mu = 1/\nu$. The reluctivity can also be taken to be an array-valued parameter in order to model magnetic anisotropy.

`Solver` solver id

`Equation` String WhitneySolver
: A describing name for the discrete model handled by this solver may be given by using this keyword. The name can be changed as long as it is used consistently.

`Procedure` File "MagnetoDynamics" "WhitneyAVSolver"
: This declaration specifies the name of the solver subroutine.

`Variable` String AV
: The name of the variable may be freely chosen provided it is used consistently also elsewhere. The associated number of degrees of freedom should always be one.

`Element`
: The default initialization routine should give a suitable element type definition automatically, so that the value of this keyword need not necessarily be given by the user.

`Use Piola Transform` Logical
: Several types of edge finite elements can be used for spatial discretization. This keyword must be given the value `True` when the basis functions for the edge element interpolation are selected to be members of the optimal edge element family or when second-order approximation is used. The elements of the optimal family provide optimal accuracy even when the physical elements are not obtained as affine images of the reference elements. The simpler basis functions which are used otherwise may not provide such accuracy for non-affine element shapes. For the documentation of the edge element basis functions see the appendices of the ElmerSolver Manual.

`Quadratic Approximation` Logical
: This keyword can be used to activate the approximation with the edge finite elements of second degree; see also the keyword `Use Piola Transform`.

`Static Conductivity` Logical
: If the stationary equations are solved such that both $\boldsymbol{A}$ and V are handled as unknowns, this keyword can be given the value `True` so that the solver can create an automated value for the `Element` keyword.

`Fix Input Current Density` Logical
: To ensure the divergence-freeness of the source term via performing the projection (18.47), the value `True` should be given for this keyword.

`Automated Source Projection BCs` Logical
: If the projection (18.47) is applied to the user-specified source $\boldsymbol{g}_U$, the solver has originally constrained the field Q automatically such that Q is chosen to satisfy the zero Dirichlet condition on parts where $|\boldsymbol{g}_U \cdot \boldsymbol{n}| \neq 0$. This feature can be disabled by giving the value `False` for this keyword in order to specify, for example, the homogeneous Dirichlet constraint for Q (known as

the field `Jfix`) on boundaries where the scalar potential V or V^s is constrained by a Dirichlet constraint.

`Use Lagrange Gauge` Logical
: This keyword must be `True` in order to switch to the regularized formulation (18.65) which has a unique vector potential solution.

`Lagrange Gauge Penalization Coefficient` Real
: This keyword defines the stabilization parameter k in (18.65).

`Use Tree Gauge` Logical
: Due to the chosen discretization, the vector potential is not sufficiently constrained to guarantee the uniqueness. Despite this, the iterative solvers are expected to be able to generate a consistent solution among many. However, to enable the solution with direct solvers this keyword is by default given the value `True` so that a special technique is applied to additionally constrain the discrete vector potential variable (in the case of iterative solvers the default value is `False`). This option is not supported when `Use Piola Transform = True` is given.

`Linear System Refactorize` Logical
: It is noted that if the refactorization of the system matrix is controlled with this keyword, the matrix factorization is anyhow recomputed automatically if the time step size differs from the previous one.

`Linear System Preconditioning` String
: Here the value `None` may give better performance than the standard ILU preconditioners since the null space of the system matrix may be non-trivial, leading to a singularity problem in connection with handling the LU decomposition.

`Linear System Iterative Method` String
: The iterative solvers BiCGStab or BiCGStab(L) may work well.

`Body Force` bf id

In the body force section the user may give various volumetric sources contained in the vector $\boldsymbol{g}$ as defined in either (18.51) or (18.52). The velocity field related to a rigid motion may also be defined for cases where this information is explicitly needed to write the equations.

`Current Density i` Real
: This keyword is used to specify the components of the source $\boldsymbol{g}_U$.

`Magnetization i` Real
: This keyword is used to specify the components of the magnetization M_i^s, with $i = 1, 2, 3$.

`Electric Potential` Real
: This keyword specifies the source electric potential V^s.

`Angular Velocity i` Real
: When the velocity is explicitly needed, this keyword may be given to define the components of the angular velocity $\boldsymbol{\omega}$. This is then used to obtain the representation for the velocity term in the equation (18.35).

`Lorentz Velocity i` Real
: This keyword may be used to define the velocity $\boldsymbol{v}(\mathbf{x}, t)$ when the steady state solution of the evolutionary model is sought. Alternatively, if the velocity is specified by the formula (18.36) and the angular velocity $\boldsymbol{\omega}$ is given, a constant value may be given in order to specify the uniform part $\boldsymbol{v}(\mathbf{o}, t)$ of the velocity.

`Boundary Condition` bc id

As explained, two versions of Dirichlet conditions are possible in connection with the A-V formulation. The first option relates to giving the value of the scalar potential, while the other version is used for prescribing the tangential components of the vector potential field. Assuming that the solver variable is `AV`, we may thus use the following keywords to specify the Dirichlet conditions:

`AV` `Real`
: This keyword is used to specify the Dirichlet condition for the scalar potential V, which is approximated by using the standard Lagrange interpolation.

`AV {e} j` `Real`
: This keyword is used to the give the vector $\boldsymbol{a}$ in (18.56) in order to prescribe the degrees of freedom corresponding to the edge element interpolation of the vector potential. The value of this keyword defines the component a_j, $j \in \{1, 2, 3\}$, with respect to the global Cartesian coordinate system.

`Jfix` `Real`
: This keyword is used to specify the Dirichlet condition for the scalar potential field Q, which is defined via (18.47).

The following keywords may be used in order to handle the flux-related boundary conditions:

`Magnetic Field Strength i` `Real`
: This keyword can be used to define the components h_i of the vector $\boldsymbol{h}$ so that the boundary conditions (18.57) and (18.59) may be imposed. It should be noted that on an interface between a permanent magnet with a given magnetization $\boldsymbol{M}_1^s$ and a material without magnetization ($\boldsymbol{M}_2^s = \boldsymbol{0}$) the boundary condition $\boldsymbol{m} \times (\nu_1 \boldsymbol{B}_1 - \nu_2 \boldsymbol{B}_2) = \boldsymbol{m} \times \boldsymbol{M}_1^s$ is imposed by default, with $\boldsymbol{m}$ being the normal vector associated with the common interface.

`Electric Current Density` `Real`
: This keyword can be used to define the electric current density j_n in the boundary conditions (18.58) and (18.60).

`Current Density` `Real`
: If the Helmholtz projection of the source is applied, this keyword may be used to specify the left-hand side in (18.50) as $\boldsymbol{g}_U \cdot \boldsymbol{n} = -j_n$, with j_n the value of this keyword.

`Magnetic Transfer Coefficient` `Real`
: The value of this keyword gives the parameter α in the boundary condition (18.59).

`Electric Transfer Coefficient` `Real`
: The value of this keyword gives the parameter β in the boundary condition (18.60).

Finally, the following keywords relate to a procedural technique to determine a tangential constraint for the vector potential $\boldsymbol{A}$ when the normal component of the magnetic flux density $\boldsymbol{B}$ is specified on the boundary.

`Magnetic Flux Density i` `Real`
: This keyword is used to specify the components of the magnetic flux density $\boldsymbol{B}$ with respect to the global Cartesian coordinate axes.

`Magnetic Flux Density {n}` `Real`
: This keyword may be used to specify directly the normal component B_n of the magnetic flux density.

Keywords for WhitneyAVHarmonicSolver

In the following the additional keywords related to solving the harmonic version are listed. Typically these are used for giving optional values which specify the imaginary parts of the parameter values. The corresponding real parts are then given by using the keyword commands already described above.

`Solver` `solver id`

`Equation` `String WhitneyHarmonicSolver`
: This gives a describing name for the discrete model handled here. The name can be changed as long as it is used consistently.

`Procedure` `File "MagnetoDynamics" "WhitneyAVHarmonicSolver"`
: The name of the solver subroutine is declared.

`Variable` `String P[Pot re:1 Pot im:1]`
: The name of the variable may be freely chosen provided it is used consistently also elsewhere. The associated number of degrees of freedom is always two. Here the real and imaginary parts are named so that they are easily recognized.

`Angular Frequency` `Real`
: The angular frequency $\omega = 2\pi f$ in the harmonic ansatz is specified.

`Material` `mat id`

`Reluctivity Im` `Real`
: The reluctivity $\nu = 1/\mu$ may be specified to be a complex-valued quantity with the imaginary part given by using this keyword. An array value may also be given in order to model magnetic anisotropy.

`Electric Conductivity Im` `Real`
: The value of this keyword may be used to specify the imaginary part of the conductivity parameter. If the material is anisotropic, the electric conductivity may also be defined to be a tensor function by giving its components with respect to the global coordinate lines.

`Body Force` `bf id`

The following keywords are used to specify the imaginary parts of the volume sources:

`Current Density Im i` `Real`

`Magnetization Im i` `Real`

`Boundary Condition` `bc id`

The following keywords relate to specifying imaginary parts in conjunction with defining boundary conditions:

`Magnetic Field Strength Im i` `Real`

`Electric Current Density Im` `Real`

`Magnetic Transfer Coefficient Im` `Real`

`Electric Transfer Coefficient Im` `Real`

`Magnetic Flux Density Im i` `Real`

`Magnetic Flux Density Im {n}` `Real`

To give the boundary condition (18.62) the following keywords may be used:

`Layer Electric Conductivity` `Real`
: This keyword defines the electric conductivity of the surface material.

`Layer Relative Permeability` `Real`
: The permeability μ of the surface conductor is specified in terms of the value of this keyword and the permeability of vacuum.

Keywords for MagnetoDynamicsCalcFields

An additional solver may finally be called to compute derived fields.

`Solver` `solver id`
: The fields to be computed are chosen in the solver section. The field `Magnetic Flux Density` is computed always, others if requested. The size of a vector field is 3, while a tensor field has the size 6. For the harmonic solution the sizes are doubled as the imaginary components are also present.

`Equation` `String CalcFields`
: A describing name for the solver is given. This can be changed as long as it is used consistently.

`Procedure` `File "MagnetoDynamics" "MagnetoDynamicsCalcFields"`
: The name of the solver subroutine is given.

`Potential Variable` `String`
: This keyword is used to specify the name of the underlying potential variable, for example `AV`. For edge elements the `Use Piola Transform` and `Quadratic Approximation` flags are inherited from the solver owning this variable.

`Angular Frequency` `Real`
: The angular frequency must be declared in this connection also as this is needed by some post-processed fields.

`Calculate Magnetic Field Strength` `Logical`
: If `True` is given, a vector field `Magnetic Field Strength` is computed.

`Calculate Electric Field` `Logical`
: If `True` is given, a vector field `Electric Field` is computed.

`Calculate Current Density` `Logical`
: If `True` is given, a vector field `Current Density` is computed.

`Impose Body Force Current` `Logical`
: If `True` is given, the body force current density given by the keyword `Current Density i` is added to above.

`Calculate Nodal Forces` `Logical`
: This keyword can be used to activate the computation of nodal forces to enable further mechanical analysis. The nodal forces are calculated by computing an opposite parameter derivative of the magnetic energy, with the placement of the body being described in terms of a one-parameter family of deformations $\mathrm{x}(\mathbf{x},\epsilon) = \mathbf{x} + \epsilon\lambda_k \boldsymbol{u}$. Here $\boldsymbol{u}$ is a unit vector along the global coordinate axis and λ_k is the Lagrange basis function associated with a mesh node $\mathbf{x}_k$. The components of the nodal forces are then obtained as

$$\boldsymbol{f}_k \cdot \boldsymbol{u} = -\int_\Omega \boldsymbol{H} \cdot (\boldsymbol{B} \cdot \nabla\lambda_k)\boldsymbol{u}\, d\Omega + \int_\Omega [\boldsymbol{H} \cdot \boldsymbol{B} - \int^{\boldsymbol{B}} \boldsymbol{H} \cdot \mathrm{d}\boldsymbol{B}]\,\mathrm{div}(\lambda_k \boldsymbol{u})\, d\Omega.$$

`Calculate Maxwell Stress` `Logical`
: If `True` is given, a tensor field `Maxwell Stress` is computed. This computation assumes that the reluctivity (and the permeability) is a real-valued parameter.

`Calculate Harmonic Loss` `Logical`
: If `True` is given, scalar fields `Harmonic Loss Linear` and `Harmonic Loss Quadratic` are computed. See Chapter 26 for more details.

`Calculate Joule Heating` `Logical`
: If `True` is given, a scalar field `Joule Heating` is computed.

`Calculate Nodal Heating` `Logical`
: If `True` is given, a scalar field `Nodal Joule Heating` is computed. In SI system the resulting unit is Watt. The nodal heating is easy to directly link as a nodal heat source to the heat equation assuming conforming meshes for the both equations.

`Calculate Nodal Fields` `Logical`
: If this is set `False`, do not compute nodal fields at all. The default is `True`.

`Calculate Elemental Fields` `Logical`
: If this is set `False`, do not compute elemental fields at all. The default is `True`. Elemental fields are nice in that they can present discontinuities in meshes.

If the harmonic loss will be computed, then the following material parameters should be provided. Note that these are not parameters needed by the primary solver since the loss estimation is just a post-processing feature. See Chapter 26 for more details.

`Material` `material id`

`Harmonic Loss Linear Coefficient` `Real`
: This keyword is used to define the material parameter C in (26.6) for the losses that are linear with frequency. Note that the coefficient may be a function of frequency itself.

`Harmonic Loss Quadratic Coefficient` `Real`
: As the previous keyword except define the material parameter C for the quadratic loss terms.

`Component` `component id`

`Calculate Magnetic Force` `Logical`
: The lumped magnetic force affecting the master bodies of the current component will be calculated if this is set true. If elemental fields are available, then airgap forces are also lumped.

`Calculate Magnetic Torque` `Logical`
: The total torque associated with the master bodies of the current component will be calculated if this is set true. If elemental fields are available, then also torque arising from airgap forces is included.

`Master Bodies` `Integer`
: This defines the integer identifiers of bodies which constitute a component.

`Master Boundaries` `Integer`
: The identifiers of boundary condition sections may also be used to constitute a component.

`Torque Origin(3)` `Real`
: This keyword is used to define a point in space which torque axis meets.

`Torque Axis(3)` `Real`
: This keyword is used to define the axis with respect to which torque is calculated.

In addition to the field computation, two scalar quantities are always computed by the solver and saved in the list of the `Simulation` section values: `Eddy current power` and `Magnetic Field Energy`. The first one is only relevant for time-dependent and harmonic cases.

Bibliography

[1] Jerrold E. Marsden and Thomas J.R. Hughes. *Mathematical Foundations of Elasticity*. Dover Publications, Inc., New York, 1994.

Model 19

Circuits and Dynamics Solver

Module name: CircuitsAndDynamics
Module subroutines: CircuitsAndDynamics, CircuitsAndDynamicsHarmonic, CircuitsOutput
Module authors: Eelis Takala, Juha Ruokolainen
Document authors: Eelis Takala

19.1 Introduction

It may become necessary to connect a magnetoquasistatic model to circuit equations when modelling electrical devices that are circuit-driven. Even if such model could in theory be handled without introducing circuits, it can considerably simplify the model setup. In Elmer the circuit equations are added into the system matrix of a magnetodynamics solver and thus they are solved in a strongly coupled manner together with the finite element equations. One should note that adding the circuit equations may hinder the parallel performance of the solver. In case this is a critical concern to the user, one should pay attention to tuning the model [9] (for example by using the so-called reduced support that can bring great benefits [8]). Here the module for adding the circuit equations to the system matrix of the magnetoquasistatic solvers is presented. Note that this solver may (and needs to) be used together with the modules `MagnetoDynamics` or `MagnetoDynamics2D`.

When the CircuitsAndDynamics module is used in a published academic work, please refer to [9]. When reduced support is used in order to achieve better parallel scalability, one may refer to [8].

19.2 Theory

In Elmer the magnetodynamic problems are currently solved by using the so-called $\boldsymbol{a}$-v formulation. This can be done in 2D or 3D and with all simulation types (i.e., steady, harmonic, or transient). The $\boldsymbol{a}$-v formulation [7, p. 124] may be presented as follows: find $\boldsymbol{a} \in \mathrm{W}^1_\mathrm{e}$ and $v \in \mathrm{W}^0_\mathrm{e}$ such that

$$(\nu\nabla\times\boldsymbol{a}, \nabla\times\boldsymbol{a}')_\Omega + (\sigma\partial_t\boldsymbol{a}, \boldsymbol{a}')_{\Omega_\mathrm{c}} + (\sigma\nabla v, \boldsymbol{a}')_{\Omega_\mathrm{c}} - (\boldsymbol{j}_0, \boldsymbol{a}')_\Omega + \langle \boldsymbol{n}\times\boldsymbol{h}, \boldsymbol{a}'\rangle_{\Gamma_h} = 0\ \forall \boldsymbol{a}' \in \mathrm{W}^1_{\mathrm{e},0} \quad (19.1)$$

and

$$(\sigma\partial_t\boldsymbol{a}, \nabla v')_{\Omega_\mathrm{c}} + (\sigma\nabla v, \nabla v')_{\Omega_\mathrm{c}} + \langle \boldsymbol{n}\cdot\boldsymbol{j}, v'\rangle_{\Gamma_c} = 0\ \forall v' \in \mathrm{W}^0_{\mathrm{e},0}, \quad (19.2)$$

where $\nu = \mu^{-1}$ is the reluctivity, $\boldsymbol{a}$ is the magnetic vector potential, v is the electric scalar potential, $\boldsymbol{j}_0$ is the source current density, $\boldsymbol{n}$ is the normal vector to the boundary Γ_h, and $\boldsymbol{a}'$ and v' are test functions. The domain Ω is the considered body, $\Omega_\mathrm{c} \subset \Omega$ consists of a conductive material, and Γ_h and Γ_c are boundaries where the magnetic field and the current density field are imposed, respectively. The edge finite element discretization [7, p. 93] generates approximations of the magnetic vector potential $\boldsymbol{a}$ and electric scalar potential v functions in discrete versions of W^1_e and W^0_e. This formulation thus requires a system matrix with two types of unknowns for expressing the magnetic vector potential and the electric potential. For more information about the formulation in Elmer see the modules `MagnetoDynamics` or `MagnetoDynamics2D`.

19.2.1 Component and Circuit network equations

The circuit equations are described as a general equation

$$\mathbf{A}\dot{\mathbf{x}} + \mathbf{B}\mathbf{x} = \mathbf{f} \tag{19.3}$$

where $\mathbf{x}$ is the circuit variable vector, $\mathbf{A}$ and $\mathbf{B}$ are the coefficient matrices and $\mathbf{f}$ is the force vector. In Elmer this equation can be described in the solver input file (SIF) in MATC language.

Circuit equations can be divided into two categories: 1) the component equations that determine the behaviour of a component, and 2) the circuit network equations that determine the relationship between the components (the Kirchhoff I and II laws).

The component equations are further divided into two subcategories: 1) the circuit element equations that can be used to define ideal components (for example Ohm's law), and 2) FE component equations that are coupled to the finite element equations (related to the finite elements which belong to the corresponding components). During the development of the CircuitsAndDynamics module a new section of keywords, called "Component", was added. Within this section the user may control the component equation by specifying which bodies belong to the component and what formulation should be used with it, together with specific component formulation input data (for example the number of turns).

The circuit network equations are further divided into two categories as well: 1) the Kirchhoff I law that is the current conservation law (every node in the network graph must conserve the current), and 2) the Kirchhoff II law that is the conservation of energy (the potential difference over every loop in the network graph must be zero).

In Elmer the circuit equations are technically divided into two categories: 1) the automatically written equations, and 2) the manually written equations. This is due to the fact that it is convenient to write the FE component equations automatically once the formulation and its specific data are known while the rest of the equations can be manually written without any difficulties. Note that the ideal components could be automated as well (for example a resistor model); however such components do not exist at the moment and the user needs to write those equations manually.

In the following sections three supported component formulations (massive, stranded, and foil winding) are presented. The type of the formulation can be chosen in SIF via "Coil Type" keyword in the "Component" section.

19.2.2 Massive coil

In some cases where the winding consists of only few turns the so-called massive inductor model [2] may be used. Then the $\boldsymbol{a}$-v formulation takes the following form: for all massive inductor indices j_{m} (multiple instances may exist in one computation)

$$(\nu\nabla\times\boldsymbol{a},\nabla\times\boldsymbol{a}')_{\Omega} + (\sigma\partial_t\boldsymbol{a},\boldsymbol{a}')_{\Omega_{\mathrm{m}}} + \sum_{i\in\Gamma_{j_{\mathrm{m}}}} V_{j_{\mathrm{m}}}(\sigma\nabla v_0^i,\boldsymbol{a}')_{\Omega_{\mathrm{m}}} = 0 \tag{19.4}$$

and

$$(\sigma\partial_t\boldsymbol{a},\nabla s^i)_{\Omega_{\mathrm{m}}} + V_{j_{\mathrm{m}}}(\sigma\nabla v_0^i,\nabla s^i)_{\Omega_{\mathrm{m}}} = I_{j_{\mathrm{m}}}, \tag{19.5}$$

where Ω_{m} is the massive inductor domain, $\Gamma_{j_{\mathrm{m}}}$ lists nodes on the electrode boundaries, $V_{j_{\mathrm{m}}}$ is the voltage and $I_{j_{\mathrm{m}}}$ is the total current through the electrodes.

The source electric potential v_0^i can be precomputed with an electrokinetic formulation (the Poisson equation) by setting it 1 at the positive electrode boundary and 0 at the negative boundary. If the electrokinetic solver is not executed, then the reduced support is used.

19.2.3 Stranded coil

In the case of a coil with a small diameter wire that is densely packed, a stranded model could be used. The main line of models takes the form [3]: for all stranded inductor index numbers j_{s}

$$(\nu\nabla\times\boldsymbol{a},\nabla\times\boldsymbol{a}')_{\Omega} + I_{j_{\mathrm{s}}}(\boldsymbol{j}_{s,j_{\mathrm{s}}},\boldsymbol{a}')_{\Omega_{\mathrm{s}}} = 0 \tag{19.6}$$

and

$$(\partial_t \boldsymbol{a}, \boldsymbol{j}_{s,j_\mathrm{s}})_{\Omega\mathrm{s}} + I_{j_\mathrm{s}}(\sigma^{-1}\boldsymbol{j}_{s,j_\mathrm{s}}, \boldsymbol{j}_{s,j_\mathrm{s}})_{\Omega\mathrm{s}} = V_{j_\mathrm{s}}, \tag{19.7}$$

where I_{j_s} is the total current in the coil and V_{j_s} is the voltage in the coil. This is the preferred method for defining the component equation for the so-called classical stranded inductor model [7, p. 286], but the resistance of the coil can also be given explicitly [6, 5]. The classical model yields

$$\int_{\Omega_\mathrm{s}} \partial_t \boldsymbol{a} \cdot \boldsymbol{w} \, \mathrm{d}\Omega_\mathrm{s} + RI_{j_\mathrm{s}} = V_{j_\mathrm{s}}, \tag{19.8}$$

where $\boldsymbol{w}$ is the wire density vector and R is the resistance of the coil.

19.2.4 Foil winding model

In the case of a foil winding with a small layer thickness that is densely packed, a foil winding model could be used. It takes the following form [4, 1]: for all foil winding index numbers j_f

$$(\nu\nabla \times \boldsymbol{a}, \nabla \times \boldsymbol{a}')_\Omega + (\sigma\partial_t \boldsymbol{a}, \boldsymbol{a}')_{\Omega_\mathrm{f}} + \sum_{i\in\Gamma_{j_\mathrm{f}}} (\sigma V_{j_\mathrm{f}}(\alpha)\nabla v_0^i, \boldsymbol{a}')_{\Omega_\mathrm{f}} = 0 \tag{19.9}$$

and

$$(\sigma\partial_t \boldsymbol{a}, V'(\alpha)\nabla s^i)_{\Omega_\mathrm{f}} + (\sigma V_{j_\mathrm{f}}(\alpha)\nabla v_0^i, V'(\alpha)\nabla v')_{\Omega_\mathrm{f}} = \frac{N_{j_\mathrm{f}}}{L_{j_\mathrm{f}}} I_{j_\mathrm{f}} \int_{\Omega_{\alpha,j_\mathrm{f}}} V'(\alpha)\, \mathrm{d}\alpha, \tag{19.10}$$

where Ω_f is the foil winding domain, Γ_{j_f} lists the nodes on the electrode boundaries, V_{j_f} is the voltage and I_{j_f} is the total current through the electrodes.

19.3 Keywords

The circuits and dynamics solver adds the circuit equations to the system matrix of the WhitneySolver or the MagnetoDynamics2D solver. Keywords are needed to both of these solvers.

19.3.1 CircuitsAndDynamics solver

`Solver` `solver id`

`Equation` `String Circuits`

`Procedure` `File "CircuitsAndDynamics" "CircuitsAndDynamics"`
The procedure which includes the addition of circuit equations to the linear system of MagnetoDynamics and MagnetoDynamics2D solvers.

`No Matrix` `Logical True`
This solver does not have it's own matrix. It only operates on the matrices of the magnetodynamics solvers.

`Component` `component id`

`Master Bodies` `Integer body ids`
Body ids of the bodies which constitute a circuit component.

`Coil Type` `String type`
If the circuit component is defined as a coil, it can be treated as a "stranded", "massive" or "foil winding".

`Resistance` `Real resistance`
If the coil type is defined as stranded, then the resistance of the coil can explicitly be defined with this keyword. In case this keyword is not given the resistance is computed by using the electric conductivity of the associated body materials.

`Electrode Area` `Real area`
: If the coil type is stranded and resistance is not explicitly given, then this keyword may be given to specify the area of the coil terminal. This is then used to compute the `Resistance` parameter of the coil. Note that if neither `Resistance` nor `Electrode Area` are given, then the area of the terminal is automatically computed. However, at the moment this feature is not yet working in 3D.

`Body Force 1`

`Source Name` `Real value`
: The circuit definitions may be used to give a freely chosen name to a source associated with a circuit component. For example, if the name `"Source Name"` is used, this command can then be used to specify the source (function).

19.3.2 CircuitsAndDynamics output

This solver saves the results of all the variables that belong to the CircuitsAndDynamics solver, namely the currents and voltages of components (also those variables that are defined by the user). These can then be saved into a text file by using the `SaveData` solver.

`Simulation`

`Max Output Level` `Integer Level`
: This determines what is shown on the standard output. Level 3 shows some information about what is happening during the execution, Level 8 shows component variable results and Level 10 shows all the Circuit Variable results. All these variables are stored even if they are not shown in the standard output. The stored variables can be saved with the SaveData routine.

`Solver` `solver id`

`Equation` `String Circuits Output`

`Procedure` `File "CircuitsAndDynamics" "CircuitsOutput"`
: The procedure which outputs the results of the added circuits (currents, voltages, resistances, etc.).

19.3.3 Additional Keywords to the MagnetoDynamics or MagnetoDynamics2D solver

`Solver` `solver id`

`Equation` `String WhitneySolver`

`Procedure` `File "MagnetoDynamics" "WhitneyAVSolver"`
: This is the WhitneySolver to whose system matrix the circuit equations are added. This procedure could be MagnetoDynamics2D as well.

`Export Lagrange Multiplier` `Logical True`
: The magnetodynamics solvers need this when circuit equations are added.

19.3.4 Circuit Definitions

The circuit definitions are written in MATC.

```
$ nof_circuits = 1
$ nof_variables = 6
$ Circuits = nof_circuits

$ !----- Circuit 1 ----
$ ! Define variable count and initialize circuit matrices
$ C.1.variables = nof_variables
$ C.1.A = zeros(nof_variables, nof_variables)
$ C.1.B = zeros(nof_variables, nof_variables)
$ C.1.Mre = zeros(nof_variables, nof_variables)
$ C.1.Mim = zeros(nof_variables, nof_variables)
$ C.1.perm = zeros(nof_variables)

$ ! Define variables
$ C.1.name.1 = "var1"
$ C.1.name.2 = "var2"

$ ! Define component variables
$ C.1.name.3 = "i_component(1)"
$ C.1.name.4 = "v_component(1)"
$ C.1.name.5 = "i_component(2)"
$ C.1.name.6 = "v_component(2)"
! The number in the parenthesis refers to
! the component id that is defined in sif.

$ ! Define Sources:
$ C.1.B(0,0) = 1
$ C.1.source.1 = "Source Name"
```

Bibliography

[1] Patrick Dular and Christophe Geuzaine. Spatially dependent global quantities associated with 2-d and 3-d magnetic vector potential formulations for foil winding modeling. *IEEE trans. magn.*, 38(2):633–636, March 2002.

[2] Patrick Dular, F. Henrotte, and W. Legros. A general and natural method to define circuit relations associated with magnetic vector potential formulations. *IEEE trans. magn.*, 35(3):1630–1633, May 1999.

[3] Patrick Dular, Nelson Sadowski, J.P.A. Bastos, and Willy Legros. Dual complete procedures to take stranded inductors into account in magnetic vector potential formulations. *IEEE trans. magn.*, 36(4):1600–1605, July 2000.

[4] H. D. Gersem and K. Hameyer. A finite element model for foil winding simulation. *IEEE trans. magn.*, 37(5):3427–3432, September 2001.

[5] C. Golovanov, Y. Marechal, and G. Meunier. 3d edge element based formulation coupled to electric circuits. *IEEE trans. magn.*, 34(5):3162–3165, September 1998.

[6] P. J. Leonard and D. Rodger. Voltage forced coils for 3d finite element method electromagnetic model. *IEEE trans. magn.*, 24(5):2579–2581, September 1988.

[7] Gerard Meunier, editor. *The Finite Element Method for Electromagnetic Modeling*. iSTE and Wiley, 2008.

[8] Eelis Takala, Evren Yurtesen, Jan Westerholm, Juha Ruokolainen, and Tommi Peussa. Using reduced support to enhance parallel strong scalability in 3d finite element magnetic vector potential formulations with circuit equations. *Electromagnetics*, 36(6):400–408, August 2016.

[9] Eelis Takala, Evren Yurtesen, Jan Westerholm, Juha Ruokolainen, and Peter Råback. Parallel simulations of inductive components with elmer finite element software in cluster environments. *Electromagnetics*, 36(3):167–185, April 2016.

Model 20

Vectorial Helmholtz for Electromagnetic Waves

Module name: VectorHelmholtz
Module subroutines: VectorHelmholtzSolver,VectorHelmholtzCalcFields
Module authors: Juhani Kataja, Juha Ruokolainen and Mika Malinen
Document authors: Juhani Kataja and Roman Szewczyk

20.1 Introduction

This module is aimed to solve the time-harmonic Maxwell equations in high-frequency regime so that the curl-curl equation is valid.

20.2 Theory

The time-harmonic Maxwell equations (in the case of a time factor $e^{-i\omega t}$, $\omega = 2\pi f$) are given by

$$\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E} = i\omega\boldsymbol{H}, \tag{20.1}$$

$$\mathbf{curl}\,\boldsymbol{H} = -i\omega\varepsilon\boldsymbol{E} + \boldsymbol{J}, \tag{20.2}$$

where complex scalars $\mu = \mu_0\mu_r$ and $\varepsilon = \varepsilon_0\varepsilon_r$ are the permeability and permittivity parameters of the model, while $\boldsymbol{E}$, $\boldsymbol{H}$ and $\boldsymbol{J}$ are electric field, magnetic field strength and impressed current distribution, respectively. The quantities μ_0, ε_0 are the permeability and permittivity of vacuum, and μ_r and ε_r are the relative permeability and permittivity.

The elimination of $\boldsymbol{H}$ from the equations (20.1) and (20.2) gives

$$\mathbf{curl}\,\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E} - \omega^2\varepsilon\boldsymbol{E} = i\omega\boldsymbol{J} \tag{20.3}$$

over a computational domain Ω. The Dirichlet and Robin boundary conditions for (20.3) are

$$\boldsymbol{E}\times\boldsymbol{n} = \boldsymbol{f}\times\boldsymbol{n} \quad \text{on } \Gamma_E, \tag{20.4}$$

$$\boldsymbol{n}\times\mathbf{curl}\,\boldsymbol{E} - \alpha\boldsymbol{n}\times(\boldsymbol{n}\times\boldsymbol{E}) = \boldsymbol{n}\times\boldsymbol{g}\times\boldsymbol{n} \quad \text{on } \Gamma_Z, \tag{20.5}$$

where the subsets Γ_E and Γ_Z cover the boundary $\partial\Omega$ of Ω, i.e., $\partial\Omega = \Gamma_E \cup \Gamma_Z$. Note that the Neumann boundary condition is achieved by setting $\alpha = 0$.

The variational form of (20.3) is as follows:

Find $\boldsymbol{E} \in \mathbf{H}_{f,E}(\mathbf{curl}, \Omega)$ such that

$$\int_\Omega (\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E} \cdot \mathbf{curl}\,\boldsymbol{v} - \omega^2 \varepsilon \boldsymbol{E} \cdot \boldsymbol{v}) d\Omega - \alpha \int_{\Gamma_Z} \mu^{-1}(\boldsymbol{E} \times \boldsymbol{n}) \cdot (\boldsymbol{v} \times \boldsymbol{n}) d\mathrm{S}$$
$$= -\int_{\Gamma_Z} \mu^{-1}(\boldsymbol{g} \times \boldsymbol{n}) \cdot (\boldsymbol{v} \times \boldsymbol{n}) d\mathrm{S} + \int_\Omega i\omega \boldsymbol{J} \cdot \boldsymbol{v} d\Omega \quad \forall \boldsymbol{v} \in H_{0,E}(\mathbf{curl}, \Omega). \tag{20.6}$$

20.2.1 Boundary condition models

Leontovich impedance boundary. The Robin boundary condition (20.5) can be utilized to implement impedance boundary approximation of well conducting medium by choosing

$$\alpha = -i\omega\mu Z_p^{-1} \quad \text{and} \quad \boldsymbol{g} = \mathbf{0}, \tag{20.7}$$

where Z_p is the surface impedance. Here Z_p can be given, e.g., as

$$Z_p = (1-i)\sqrt{\frac{\mu_c \omega}{2\sigma_c}}, \tag{20.8}$$

where σ_c and μ_c are the bulk conductivity and permeability of the wall [1].

First-order absorbing boundary condition. Now let the computational domain Ω be $B_R \setminus D$, where $B_R \subset \mathbb{R}^3$ is an open ball of radius R and $D \subset B_R$. Furthermore, suppose that $\Gamma_E \cup \Gamma_Z$ covers only ∂D, while the first-order absorbing boundary condition on ∂B_R is given by [2]

$$\boldsymbol{n} \times \mathbf{curl}\,\boldsymbol{E} = i\omega\sqrt{\varepsilon_0\mu_0}\boldsymbol{n} \times (\boldsymbol{n} \times \boldsymbol{E}). \tag{20.9}$$

This can be recovered with a Robin boundary on ∂B_R, with

$$\alpha = i\omega\sqrt{\varepsilon_0\mu_0} \quad \text{and} \quad \boldsymbol{g} = \mathbf{0}. \tag{20.10}$$

Port feed. Suppose a guided wave $\boldsymbol{E}_p$ propagates with a propagation constant β. Furthermore, suppose that $\boldsymbol{E}_p$ is the only propagating mode at the chosen frequency. The port feed model can be implemented with the Robin boundary condition by choosing

$$\alpha = i\beta \quad \text{and} \quad \boldsymbol{g} = 2i\beta\boldsymbol{E}_p. \tag{20.11}$$

Surface potential. The differential of a scalar potential ϕ defined on a boundary can be used to generate $\boldsymbol{g}$ as

$$\boldsymbol{g} = (\partial_\alpha \phi)\boldsymbol{a}^\alpha \tag{20.12}$$

where $\boldsymbol{a}^\alpha$ give the dual basis of coordinate basis vectors $\boldsymbol{a}_\alpha$ on the surface, so that

$$\int_{\Gamma_Z} \mu^{-1}(\boldsymbol{g} \times \boldsymbol{n}) \cdot (\boldsymbol{v} \times \boldsymbol{n}) d\mathrm{S} = \int_{\Gamma_Z} \mu^{-1}((\partial_\alpha \phi)\boldsymbol{a}^\alpha \times \boldsymbol{n}) \cdot (\boldsymbol{v} \times \boldsymbol{n}) d\mathrm{S}.$$

20.3 Keywords

`Constants`

The keywords of this section are used to change the values of natural constants.

`Permittivity of Vacuum Real`

The permittivity of vacuum ε_0, defaulting to $8.854187817 \cdot 10^{-12}\,\frac{\mathrm{C}}{\mathrm{Vm}}$.

`Permeability of Vacuum Real`

The permeability of vacuum μ_0, defaulting to $4\pi \cdot 10^{-7}\,\frac{\mathrm{Vs}}{\mathrm{Am}}$.

`Material` `material id`
: The material parameters ε_r and μ_r^{-1} are defined in this section.

`Relative Permittivity` `Real`
: The real part of relative permittivity ε_r.

`Relative Permittivity im` `Real`
: The imaginary part of relative permittivity ε_r.

`Relative Reluctivity` `Real`
: The real part of μ_r^{-1}.

`Relative Reluctivity im` `Real`
: The imaginary part of μ_r^{-1}.

`Electric Conductivity` `Real`
: If the electric conductivity σ is given, then $\boldsymbol{J}$ is replaced by $\boldsymbol{J} \equiv \sigma \boldsymbol{E} + \boldsymbol{J}_0$ where the part $\boldsymbol{J}_0$ is now the impressed current density.

Keywords for VectorHelmholtzSolver

`Solver` `solver id`
: The solver section defines control variables for the equation solver. Most of the possible keywords – related to linear algebra (starting with `Linear System`), for example – are common for all the solvers and are explained elsewhere.

`Equation` `String`
: A string identifying the solver. This can be changed but it must be given; for example `VectorHelmholtzSolver`

`Procedure` `File "VectorHelmholtz" "VectorHelmholtzSolver"`
: The name of the solver subroutine.

`Variable` `String`
: The identifier for the field variable to be solved. Real and imaginary parts must be provided, cf. the default value `E[E re:1 E im:1]`.

`Angular Frequency` `Real`
: The angular frequency ω.

`Use Piola Transform` `Logical`
: Utilize modern Piola transformed edge finite elements. This increases the number of DOFs on meshes containing quadrilateral element faces. If the mesh contains elements that are not affine images of the reference element, then this option should be enabled to maintain accuracy.

`Quadratic Approximation` `Logical`
: This keyword is needed when the simulation is done with the edge finite elements of second degree.

`Linear System Preconditioning Damp Coefficient` `Real`
: If present, the preconditioner is constructed from a damped system matrix $A + \kappa(S - M + B)$, where A is the original total system matrix, S is the stiffness matrix containing the **curl** terms, M is the scaled mass matrix corresponding to the discretization of the operator $-\omega^2 \varepsilon \boldsymbol{I}$, B is the matrix arising from boundary integrals and κ is the damping coefficient.

`Linear System Preconditioning Damp Coefficient im` `Real`
: The imaginary part of the damping coefficient κ.

`Mass-proportional Damping` `Logical`
: If this option is activated, the preconditioner is constructed from a damped system matrix $A - \kappa M$, where A and M are as above.

Lossless cavity models usually benefit from shifted preconditioning with $\kappa \neq 0$. Utilizing BiCGStab(l) with the ILU(0) or Vanka preconditioner and choosing damping coefficient $\kappa = i$ is likely to result in a convergent iteration. It is noted that if the mass-proportional perturbation is used, giving $\kappa = -|\eta|i$ creates the same perturbation which would result from introducing the electric conductivity $\sigma = |\eta|\omega\varepsilon$.

`Body Force` `bf id`
The impressed current $\boldsymbol{J}$ is specified in the `Body Force` section.

`Current Density` i `Real`
The i:th component of the real part of the current density $\boldsymbol{J}$.

`Current Density im` i `Real`
The i:th component of the imaginary part of the current density $\boldsymbol{J}$.

`Boundary Condition` `bc id`

`E re {e} i` `Real`
The real part of the component data f_i to evaluate DOFs corresponding to the edge basis functions.

`E im {e} i` `Real`
The imaginary part of the component data f_i to evaluate DOFs corresponding to the edge basis functions.

`Electric Robin Coefficient` `Real`
Real part of α.

`Electric Robin Coefficient im` `Real`
Imaginary part of α.

`Magnetic Boundary Load` i `Real`
The i:th component of the real part of $\boldsymbol{g} \times \boldsymbol{n}$.

`Magnetic Boundary Load` i `im` `Real`
The i:th component of the imaginary part of $\boldsymbol{g} \times \boldsymbol{n}$.

`TEM Potential` `Real`
This can be used to define the real part of the scalar potential ϕ to calculate $\boldsymbol{g}$.

`TEM Potential im` `Real`
This can be used to define the imaginary part of the scalar potential ϕ to calculate $\boldsymbol{g}$.

Keywords for VectorHelmholtzCalcFields

`Solver` `solver id`
Unique id for the post-processing solver.

`Equation` `String`
A string identifying the solver.

`Procedure` `File "VectorHelmholtz" "VectorHelmholtzCalcFields"`
The name of the postprocessing subroutine.

`Angular Frequency` `Real`
The angular frequency ω.

`Calculate Elemental Fields` `Logical`
Calculate elementwise constant approximations of the fields. Useful for discontinuous material parameters.

`Calculate Magnetic Flux Density` `Logical`
Output the magnetic flux density $\boldsymbol{B} = -i/\omega\, \mathbf{curl}\, \boldsymbol{E}$.

`Calculate Magnetic Field Strength` `Logical`
Output the magnetic field strength $\boldsymbol{H} = \mu^{-1}\boldsymbol{B}$.

`Calculate Electric field` Logical
: Output the electric field $\boldsymbol{E}$.

`Calculate Poynting Vector` Logical
: Output the Poynting vector $\frac{1}{2}\boldsymbol{E} \times \boldsymbol{H}^*$.

`Calculate Div of Poynting Vector` Logical
: Output the divergence of Poynting vector

$$\nabla \cdot \frac{1}{2}\boldsymbol{E} \times \boldsymbol{H}^* = \frac{1}{2} i\omega \left(\mu \boldsymbol{H} \cdot \boldsymbol{H}^* + \boldsymbol{E} \cdot (\varepsilon \boldsymbol{E})^*\right) - \frac{1}{2}\boldsymbol{E} \cdot \boldsymbol{J}^*$$

and also generate the result variable `Joule Heating` corresponding to the term $1/2\boldsymbol{E} \cdot \boldsymbol{J}^*$ above.

`Calculate Energy Functional` Logical
: Evaluate the left-hand side of (20.6) by using the discrete solution.

Bibliography

[1] J.D. Jackson. *Classical Electrodynamics*. John Wiley & Sons, third edition, 1999.

[2] A.F. Peterson. Absorbing boundary conditions for the vector wave equation. *Microwave and Optical Technology Letters*, 1(2):62–64, 1988.

Model 21

Electromagnetic Waves

Module name: EMWaveSolver
Module subroutines: EMWaveSolver, EMWaveCalcFields
Module authors: Juhani Kataja, Peter Råback, Juha Ruokolainen and Mika Malinen
Document authors: Mika Malinen

21.1 Introduction

The module described here is aimed at solving the time-dependent electromagnetic wave equation derived from Maxwell's equations. Here the unknown field is approximated by using vector-valued (edge) finite elements.

21.2 Theory

By assuming time-independent material parameters, the electromagnetic wave equation may be written as

$$\mathbf{curl}\,\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E} + \varepsilon\frac{\partial^2 \boldsymbol{E}}{\partial t^2} + \sigma\frac{\partial \boldsymbol{E}}{\partial t} = -\frac{\partial \boldsymbol{J}_E}{\partial t} \tag{21.1}$$

where $\mu = \mu_0\mu_r$ and $\varepsilon = \varepsilon_0\varepsilon_r$ are the permeability and permittivity, $\boldsymbol{E}$ is the electric field and $\boldsymbol{J}_E$ is an impressed current density. The quantities μ_0 and ε_0 are the permeability and permittivity of vacuum, and μ_r and ε_r are the relative permeability and relative permittivity; respectively.

The Dirichlet and Robin-like boundary conditions for (21.1) are

$$\boldsymbol{E}\times\boldsymbol{n} = \boldsymbol{f}\times\boldsymbol{n} \quad \text{on } \Gamma_E, \tag{21.2}$$

$$\boldsymbol{n}\times\mathbf{curl}\,\boldsymbol{E} + \alpha\boldsymbol{n}\times(\boldsymbol{n}\times\frac{\partial \boldsymbol{E}}{\partial t}) = \boldsymbol{g} \quad \text{on } \Gamma_Z, \tag{21.3}$$

where Γ_E and Γ_Z give a partitioning of the boundary $\partial\Omega$ of the computational domain Ω such that $\partial\Omega = \Gamma_E \cup \Gamma_Z$. Note that the Neumann boundary condition is achieved by setting $\alpha = 0$. An absorbing boundary condition can be created by choosing

$$\alpha = \sqrt{\varepsilon_0\mu_0}. \tag{21.4}$$

To obtain the variational formulation of (21.1) integration by parts is carried out. After using (21.3) we then come to the weak form

$$\begin{aligned}\int_\Omega (\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E}\cdot\mathbf{curl}\,\boldsymbol{v} + \varepsilon\frac{\partial^2 \boldsymbol{E}}{\partial t^2}\cdot\boldsymbol{v} + \sigma\frac{\partial \boldsymbol{E}}{\partial t}\cdot\boldsymbol{v})d\Omega + \alpha\int_{\Gamma_Z}\mu^{-1}(\frac{\partial \boldsymbol{E}}{\partial t}\times\boldsymbol{n})\cdot(\boldsymbol{v}\times\boldsymbol{n})d\mathrm{S} \\ = -\int_{\Gamma_Z}\mu^{-1}\boldsymbol{g}\cdot\boldsymbol{v}d\mathrm{S} - \int_\Omega \dot{\boldsymbol{J}}_E\cdot\boldsymbol{v}d\Omega\end{aligned} \tag{21.5}$$

which can be used to obtain the fully discrete equations.

21.3 Keywords

`Constants`
The keywords of this section are used to change the values of natural constants.

`Permittivity of Vacuum` Real
The permittivity of vacuum ε_0, defaulting to $8.854187817 \cdot 10^{-12}$ $(\frac{\mathrm{C}}{\mathrm{Vm}})$.

`Permeability of Vacuum` Real
The permeability of vacuum μ_0, defaulting to $4\pi \cdot 10^{-7}$ $(\frac{\mathrm{Vs}}{\mathrm{Am}})$.

`Material` material id
The material parameters ε_r, μ_r and σ are defined in this section.

`Relative Permittivity` Real
This defines the value of the relative permittivity ε_r.

`Relative Permeability` Real
This defines the value of μ_r.

`Electric Conductivity` Real
This defines the value of σ.

Keywords for EMWaveSolver

`Solver` solver id

`Equation` String
This gives a name for referring to this solver definition.

`Procedure` File "EMWaveSolver" "EMWaveSolver"
This is used to identify the solver subroutine.

`Variable` String
The name for the field variable to be solved. If this is not given, the solver defaults this to `E`.

`Use Piola Transform` Logical
Utilize modern Piola-transformed edge finite elements. This increases the number of DOFs on meshes containing hexahedral and pyramidal elements. If the mesh contains elements that are not affine images of the reference element, then this option should be enabled.

`Quadratic Approximation` Logical
This keyword can be used to switch to using vector-valued finite elements of second order.

`Body Force` bf id
The time derivative of the impressed current density $\boldsymbol{J}_E$ can be specified in the `Body Force` section.

`Current Density Rate` i Real
The i:th component of the time derivative field $\dot{\boldsymbol{J}}_E$.

`Boundary Condition` bc id

`E {e} i` Real
In the case of the default variable name this command defines a vector so that its tangential trace $\boldsymbol{f} \times \boldsymbol{n}$ is approximated by $\boldsymbol{E}_h \times \boldsymbol{n}$, with $\boldsymbol{E}_h$ the finite element interpolating function. The value of this keyword defines the components of the vector with respect to the global Cartesian coordinate system.

`Electric Damping Coefficient` Real
This specifies the value of the parameter α.

`Magnetic Boundary Load i` Real
The value of this keyword specifies the i:th component of $\boldsymbol{g}$.

Keywords for EMWaveCalcFields

A separate solver section can be written so as to create a postprocessed field `Elfield` which corresponds to the computed solution and which can be used for visualization.

`Solver` `solver id`

`Equation` `String`
: This gives a name for referring to this solver definition.

`Procedure` `File "EMWaveSolver" "EMWaveCalcFields"`
: This is used to identify the name of the postprocessing subroutine.

`Calculate Elemental Fields` `Logical`
: Calculate an elementwise fit for the primary solution. This is useful especially in cases where material parameters have discontinuities.

Model 22

Computation of Magnetic Fields in 2D

Module name: MagnetoDynamics2D
Module subroutines: MagnetoDynamics2D, MagnetoDynamics2DHarmonic, BSolver
Module authors: Juha Ruokolainen, Eelis Takala, Mika Malinen, Peter Råback
Document authors: Peter Råback, Mika Malinen

22.1 Introduction

This module may be used to solve a version of the Maxwell equations in 2D special cases (including axially symmetric problems) when the unknown is the z-component (or ϕ-component) of the vector potential. In contrast to the 3D version of the magnetodynamics solver described earlier, the standard Lagrange interpolation is here applied. In addition to performing the computations in the time domain, the analogous version of the equations may also be solved in the frequency domain. Furthermore, an additional solver may be called to produce derived fields (for example the magnetic flux density) from the computed vector potential. Also Joule losses may be computed for harmonic fields.

22.2 Theory

When the current density acts in a direction orthogonal to the plane considered, the scalar potential need not be considered as an unknown in the A-V formulation of Maxwell's equations. In the case of Cartesian coordinates the system is then fully described by the vector potential $\boldsymbol{A} \equiv A_3(x_1, x_2)\boldsymbol{e}_3$ as

$$\sigma \frac{\partial A_3}{\partial t}\boldsymbol{e}_3 + \mathbf{curl}(\frac{1}{\mu}\,\mathbf{curl}\,A_3\boldsymbol{e}_3) - \sigma(\boldsymbol{v} \times \mathbf{curl}\,A_3\boldsymbol{e}_3) = J_3\boldsymbol{e}_3 + \mathbf{curl}\,\boldsymbol{M} \tag{22.1}$$

where $J_3\boldsymbol{e}_3$ is the electric current density and the magnetization current is expressed in terms of a planar magnetization vector $\boldsymbol{M} = M_1\boldsymbol{e}_1 + M_2\boldsymbol{e}_2$. In addition, $\boldsymbol{v}$ is an optional velocity field describing a motion of a body. It should be noted that if the motion is modelled via performing a rigid motion of the previous mesh into the current configuration, the effect of motion may be incorporated in the total time derivative and $\boldsymbol{v}$ need not be specified explicitly; cf. the discussion in connection with the 3D equations.

When cylindrical coordinates (r, ϕ, z) are employed, the curl of the vector potential $\boldsymbol{A} \equiv A_\phi(r, z)\boldsymbol{e}_\phi$ has components $(-\partial_z A_\phi, 0, \partial_r A_\phi + A_\phi/r)$ with respect to the orthonormal basis $\{\boldsymbol{e}_r, \boldsymbol{e}_\phi, \boldsymbol{e}_z\}$. The field equation is then given by

$$\sigma \frac{\partial A_\phi}{\partial t}\boldsymbol{e}_\phi + \mathbf{curl}(\frac{1}{\mu}\,\mathbf{curl}\,A_\phi\boldsymbol{e}_\phi) - \sigma(\boldsymbol{v} \times \mathbf{curl}\,A_3\boldsymbol{e}_3) = J_\phi\boldsymbol{e}_\phi + \mathbf{curl}\,\boldsymbol{M}, \tag{22.2}$$

with the magnetization vector being $\boldsymbol{M} = M_r\boldsymbol{e}_r + M_z\boldsymbol{e}_z$.

The harmonic version of the equation is obtained by replacing the operator $\frac{\partial}{\partial t}$ by multiplication with $i\omega$ and solving the vector potential as a complex-valued field. In the case of a harmonic problem described in terms of Cartesian coordinates the Joule heat generation in stationary conductors may be computed from

$$h = \frac{1}{2}\sigma\omega^2|A_3|^2.$$

As the electric conductivity σ may be discontinuous over material boundaries, it is attractive to compute a field without it and carry out the multiplication within the heat solver where the source term is needed.

22.2.1 Boundary Conditions

The Dirichlet boundary condition for A_3 is simply

$$A_3 = A_3^b. \tag{22.3}$$

Since we now have

$$\mathbf{curl}\, A_3\boldsymbol{e}_3 \times \boldsymbol{n} = (\nabla A_3 \cdot \boldsymbol{n})\boldsymbol{e}_3,$$

natural boundary conditions of the type

$$\frac{1}{\mu}\frac{\partial A_3}{\partial n} = g \tag{22.4}$$

may be used as an alternative. It may be difficult to extend the Dirichlet conditions far enough. Then a spherically symmetric far-field approximation may be used. This gives a boundary condition of Robin kind which corresponds to the choice

$$g = \frac{1}{\mu}A_3\frac{\boldsymbol{r}\cdot\boldsymbol{n}}{|\boldsymbol{r}|^2} \tag{22.5}$$

in (22.4). If no boundary conditions are specified, the natural boundary condition with $g = 0$ prevails.

22.3 Keywords

Keywords for MagnetoDynamics2D

Here we list the keywords that are relevant when utilizing the module `MagnetoDynamics2D` and that may also be common to the other solvers. Such common keywords relate to specifying material parameters, body forces, and boundary conditions.

`Constants`

`Permeability of Vacuum Real`
This constant has the default value $4\pi \cdot 10^{-7}$ in SI units. In different unit system change this accordingly.

`Material mat id`

The following material parameters may be used by all the solvers in the module.

`Electric Conductivity Real`
This keyword is used to specify the electric conductivity σ.

`Relative Permeability Real`
If this keyword is used, the permeability μ can be specified in terms of the permeability of vacuum. To obtain the permeability, the value of this keyword is then internally multiplied with the permeability of vacuum. Instead of using this keyword, the keywords `Permeability` or `Reluctivity` may be used.

`Permeability Real`
This keyword may be used to specify directly the permeability μ.

`Reluctivity` Real
: The value of this keyword specifies the reluctivity ν. The permeability is then taken to be $\mu = 1/\nu$.

`Magnetization i` Real
: The components of the magnetization vector, $i = 1, 2$.

`H-B Curve` Cubic Real
: The H-B curve must be given as a cubic spline. This enables that the derivative of the curve is computed analytically from the spline coefficients.

`Solver` solver id

`Equation` String MgDyn2D
: This keyword gives a describing name for the discrete model handled by this solver. The name can be changed as long as it is used consistently.

`Procedure` File "MagnetoDynamics2D" "MagnetoDynamics2D"
: This declaration specifies the name of the solver subroutine.

`Variable` String Potential
: The name of the variable may be freely chosen provided it is used consistently also elsewhere. The associated number of degrees of freedom should always be one.

`Nonlinear System Max Iterations` Integer
: If the material laws are nonlinear, the equation may need some iterations before reaching the solution. This keyword gives the maximum number of iterations. The default is one. If a nonlinear H-B curve is given, then Newton's linearization is applied after the first iteration.

`Nonlinear System Convergence Tolerance` Real
: This keyword gives the convergence tolerance for the nonlinear iteration.

`Body Force` bf id

In the body force section the user may give various volume sources.

`Current Density` Real
: This keyword is used to specify the current density in the z/ϕ-direction.

`Lorentz Velocity i` Real
: This keyword may be used to define the optional velocity $\boldsymbol{v}$.

`Boundary Condition` bc id

`Potential` Real
: If the variable is given the name `Potential`, this keyword can be used to specify the Dirichlet condition for the vector potential.

`Infinity BC` Logical
: Sets far-field conditions for the vector potential assuming spherical symmetry at distance.

`Mortar BC` Integer
: This enforces continuity in the case of rotating boundary conditions by the mortar finite element method.

Keywords for MagnetodDynamics2DHarmonic

Here only the additional keywords related to the harmonic solver are listed. For other keywords see the definitions above.

`Material` mat id

`Electric Conductivity im` Real
: The imaginary part of the electric conductivity.

`Magnetization i Im` Real
: The imaginary components of the magnetization vector, $i = 1, 2$.

`Solver` solver id

`Equation` String MgDyn2DHarmonic
: A name for the solver.

`Procedure` File "MagnetoDynamics2D" "MagnetoDynamics2DHarmonic"
: This declaration specifies the name of the solver subroutine.

`Variable` String Potential[Potential Re:1 Potential Im:1]
: The name of the variable may be freely chosen provided it is used consistently also elsewhere. The associated number of degrees of freedom should always be two.

`Body Force` bf id

`Current Density Im` Real
: This keyword is used to specify the imaginary part of the current density.

Keywords for BSolver

An additional solver may finally be called to compute derived fields. **Note: The subroutine** `BSolver` **described here is obsolete.** It is recommended that the subroutine `MagnetoDynamicsCalcFields` within the module `MagnetoDynamics` is used for postprocessing.

`Solver` solver id

The postprocessing solver currently only solves for the magnetic field density. The size of the requested vector field is 2 when the target variable is real-valued and 4 if it is complex-valued. The user does not need to specify the output fields.

`Equation` String BSolver
: A describing name for the solver is given. This can be changed as long as it is used consistently.

`Procedure` File "MagnetoDynamics2D" "BSolver"
: The name of the solver subroutine is given.

`Target Variable` String
: This keyword is used to specify the name of the underlying potential variable; the default is `Potential`.

`Discontinuous Galerkin` Logical
: The derived fields are discontinuous if the material properties has jumps. Therefore the visualizations are more appealing if the fields are allowed to be discontinuous. Setting this flag `True` activates discontinuous Galerkin (DG) computation of the fields. Note that these fields are compatible only with certain postprocessing practices. One possible way is to use `vtu` output and ask elemental fields for saving, such as `Vector Field Elemental 1`.

`Average Within Materials` Logical
: If DG formulation for the fields is asked, this enforces averaging of the fields within materials.

`Calculate Joule Heating` Logical [True]
: The automatic computation of the Joule heating may be turned on by this keyword. The default is `False`. The keyword is only applicable for the harmonic case. The computation results to two additional variables. `Joule Heating` gives the absolute heating and `Joule Field` the field that gives the heating when multiplied by the electric conductivity. This may be needed if the electric conductivity is discontinuous making also the heating power discontinuous.

`Desired Heating Power` Real
: A constant that gives the desired total heating power in Watts. If the keyword is active, then the `Joule Heating` and `Joule Field` are multiplied by the ratio of the desired and computed heating powers.

Model 23

Magnetic Induction Equation

Module name: MagneticSolve
Module subroutines: MagneticSolver
Module authors: Juha Ruokolainen
Document authors: Ville Savolainen, Antti Pursula

23.1 Introduction

The magnetic induction equation describes interaction of a conducting liquid or gas with applied and induced magnetic fields in the low-frequency domain. The induction equation for the magnetic flux density is always coupled to the Navier-Stokes equation for the movement of the fluid. The magnetic field, in turn, causes the Lorentz force in the Navier-Stokes equation. The fluid is typically hot, and the Navier-Stokes equation is often coupled also to the heat equation.

The induction equation solver can also be used in a body without a moving fluid, i.e., when $\vec{v} = 0$ and the Navier-Stokes equation is not solved. In this case, the problem belongs to the field of magneto-quasistatics.

23.2 Theory

The magnetic induction equation may be derived from Maxwell's equations, with the displacement current in Ampère's law neglected, and the Ohm's law for conducting fluids, $\vec{\jmath} = \sigma(\vec{E} + \vec{v} \times \vec{B})$. This approximation for the behavior of electromagnetic fields in conducting, moving fluids is called magnetohydrodynamics.

The magnetic induction equation is given by

$$\frac{\partial \vec{B}}{\partial t} + \frac{1}{\sigma\mu}\nabla \times \nabla \times \vec{B} - \nabla \times (\vec{v} \times \vec{B}) = 0, \tag{23.1}$$

where σ is the electric conductivity and μ the magnetic permeability of the material. These must be specified by using the keywords `Electric Conductivity` and `Magnetic Permeability` in the `Material` section.

The force term induced by the magnetic field for the flow momentum equations is given by

$$\vec{f}_m = \vec{\jmath} \times \vec{B}, \tag{23.2}$$

and the Joule heating in the heat equation by

$$h_m = \frac{1}{\sigma} |\vec{\jmath}|^2, \tag{23.3}$$

where $\vec{\jmath}$ is the current density, calculated from the Ampère's law $\vec{\jmath} = \nabla \times \vec{H}$. These body forces are specified by the keywords `Lorentz Force` and `Joule Heat`.

The magnetic field can also be divided into external, or applied, and induced field, $\vec{B} = \vec{B}^e + \vec{B}^i$. The external magnetic field $\vec{B}^e$ is created by permanent magnets or currents outside the fluid. The external field may be given to the induction equation solver either from a restart file, e.g., as calculated by the magnetostatic solver, or defined via the sif file's keywords `Applied Magnetic Field 1`, `2` and `3`. If the restart file is used, the components of $\vec{B}^e$ are read from the variables named `magnetic flux density 1`, `2` and `3`. If both methods are used, the two applied fields are summed together. It is assumed that the sources of the external field are outside the flow region, i.e., $\nabla \times \vec{B}^e = 0$, and that the time derivative of the external field can be ignored. The time derivative $\partial\vec{B}^e/\partial t$ can, however, be specified directly by the keywords `Magnetic Bodyforce 1`, `2` and `3`. The induction equation solver gives the components of the induced magnetic field $\vec{B}^i$.

Both transient and steady-state solvers for the magnetohydrodynamical system (induction, Navier-Stokes and heat equations) are available. The magnetostatic and time-harmonic solvers for the external magnetic field are described elsewhere in the Models Manual. In some cases it is also possible that the velocity is *a priori* known, for example when studying induction in a rotating body. Then a user defined velocity can be used instead of computing the velocity from Navier-Stokes equations.

Currently the induction equation can be solved in a cylindrically symmetric or a general three-dimensional formulation.

23.2.1 Boundary Conditions

For the induction equation one can apply either Dirichlet or natural boundary conditions. In both cases, one must check that the computational domain is extended far enough to avoid numerical errors. For this reason, it is possible to solve the magneto-quasistatics problem in an adjacent body.

The Dirichlet boundary condition for a component of the induced magnetic field B_i (we have dropped now the superscript i that marked the induced field) is

$$B_i = B_i^b. \tag{23.4}$$

B_i^b can be a constant or a function of time, position or other variables. The keywords for the Dirichlet boundary conditions are `Magnetic Field 1`, `2` and `3`.

In the cylindrically symmetric case, the Dirichlet boundary condition for the azimuthal component B_ϕ is in the same units as for the other two components, i.e., in T, and not for a contravariant component. On the symmetry axis one has to set $B_r = 0$ and $B_\phi = 0$, and $\partial B_z/\partial r = 0$ is applied implicitly.

If no Dirichlet condition is specified, natural boundary condition is applied.

23.3 Keywords

`Solver` `solver id`

Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation` `String [Magnetic Induction]`

The name of the equation. It is also possible to use this solver as external procedure. Then the name of the equation must not be the above (use *e.g.* `Magnetic Field Solver`). Also the following four keywords have to be added with the values give here.

`Procedure` `File "MagneticSolve" "MagneticSolver"`

`Variable` `String Magnetic Field`

`Variable DOFs` `Integer 3`

`Exported Variable 1` `= -dofs 3 electric current`

The above four keywords are to be given only when using the solver as an external procedure.

`Nonlinear System Convergence Tolerance` `Real`

This keyword gives a criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations k is small enough

$$||\vec{B}^k - \vec{B}^{k-1}|| < \epsilon||\vec{B}^k||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Max Iterations` Integer

The maximum number of nonlinear iterations the solver is allowed to do. If neither the material parameters nor the boundary conditions are functions of the solution, the problem is linear, and this should be set to 1.

`Nonlinear System Relaxation Factor` Real

Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achieve convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:

$$\vec{B}' = \lambda \vec{B}^k + (1-\lambda)\vec{B}^{k-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

`Steady State Convergence Tolerance` Real

With this keyword a equation specific steady state or coupled system convergence tolerance is given. All the active equation solvers must meet their own tolerances for their variable u, before the whole system is deemed converged. The tolerance criterion is:

$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$

where ϵ is the value given with this keyword.

`Equation` eq id

The equation section is used to define a set of equations for a body or set of bodies:

`Magnetic Induction` Logical

If set to `True`, solve the magnetic induction equation.

`User Defined Velocity` Logical

Controls whether the velocity is given by the user or computed by another solver. Default value is `False`, which means that velocity solution of Navier-Stokes equations is used.

`Navier-Stokes` Logical

If set to `True`, solve also the Navier-Stokes equations. For magnetohydrodynamics, this is done, except when the computational region for the magnetic field is extended beyond the fluid.

`Heat Equation` Logical

If set to `True`, solve also the heat equation.

`Body Force` bf id

The body force section may be used to give additional force terms for the equations.

`Lorentz Force` Logical

If set true, triggers the magnetic field force for the flow momentum equations.

`Joule Heat` Logical

If set true, the Joule heating is added in the heat equation.

`Magnetic Bodyforce i` Real

This keyword can be used to specify explicitly the time dependence of the external field, i.e., the term $-\partial \vec{B}^e / \partial t$. This is especially useful for time-harmonic fields, where the time derivative can be calculated and expressed easily.

`Initial Condition` ic id

The initial condition section may be used to set initial values for the field variables. The following variables are active:

`Magnetic Field i` Real

For each magnetic flux density component `i`$= 1, 2, 3$.

`Material` `mat id`

The material section is used to give the material parameter values. The following material parameters may be set for the induction equation. They can be a constant or a function of a given variable.

`Magnetic Permeability` `Real`

The magnetic permeability is set with this keyword. For most fluids, the vacuum value for μ_0 can be used, and the keyword set to `1.25664e-6`.

`Electric Conductivity` `Real`

The value of the electric conductivity is set with the keyword. For example, for polythermal flows the conductivity could be a function of the temperature.

`Applied Magnetic Field i` `Real`

This keyword can be used to specify the external field, or a part of it, and its contribution to the term $\nabla \times (\vec{v} \times \vec{B}^e)$. The field may be a function of, e.g., time or position.

`MHD Velocity i` `Real`

The user defined velocity can be given with these keywords with `i=1,2,3`.

`Boundary Condition` `bc id`

The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The ones related to induction equation are

`Magnetic Field i` `Real`

Dirichlet boundary condition for each magnetic flux density component `i`$= 1, 2, 3$.

Model 24

Reduced Dimensional Electrostatics

Module name: StatElecBoundary
Module subroutines: StatElecBoundaryForce, StatElecBoundaryEnergy, StatElecBoundaryCharge, StatElecBoundarySpring
Module authors: Peter Råback
Document authors: Peter Råback

24.1 Introduction

In some applications the geometry is such that the 3D electrostatics may quite accurately be reduced to a 1D problem. This is the case for nearly aligned planes. If the angle between the planes is φ (in radians) the error of this approximation is roughly $2\varphi^2/3$. Therefore we may use an analytical solution that results directly from the distance of the planes that are in different potential. The ideal model may be further developed by taking into account perforated structures and dielectric layers.

24.2 Theory

It is assumed here that the electric field is stationary in the time-scale under study. The electric field $\vec{E}$ may be expressed with an electric scalar potential ϕ,

$$\vec{E} = \nabla\phi. \tag{24.1}$$

If there are no free charges, the scalar potential may be solved from

$$-\nabla \cdot \varepsilon\nabla\phi = 0. \tag{24.2}$$

When one dimension is much smaller than the other two we may assume that the field is one-dimensional. Then the electric field resulting from potential difference $\Phi = \Delta\phi$ is

$$\vec{E} = E\vec{n} = \frac{\Phi}{d}\vec{n}, \tag{24.3}$$

where $\vec{n}$ is the unit normal and $d(\vec{r})$ is the height of the aperture. The energy density per unit area is now,

$$e = \frac{1}{2}\varepsilon E^2 d = \frac{\varepsilon\Phi^2}{2d}. \tag{24.4}$$

which corresponds to a induced charge density on the surface

$$q = \frac{\varepsilon\Phi}{d}. \tag{24.5}$$

The force is obtained from the derivative of the energy,

$$f = \frac{\partial e}{\partial d} = -\frac{\varepsilon \Phi^2}{2d^2}, \tag{24.6}$$

and the spring constant from the derivative of the force,

$$k = \frac{\partial f}{\partial d} = \frac{\varepsilon \Phi^2}{d^3}, \tag{24.7}$$

The forces and spring constants are always aligned in the direction of the surface normal since any other direction is incompatible with the original assumptions.

24.2.1 Electrostatics of perforated structures

If there are holes or other imperfections in the structure they may be homogenized over the whole area. By computing the electric energy and force in the presence and absence of holes we get correction factors

$$e_{holes} = \alpha e_{ideal} \tag{24.8}$$

and

$$f_{holes} = \beta f_{ideal} \tag{24.9}$$

The correction terms may be precalculated for a given geometry. However, if the relative change in the aperture is large the correction terms should be modeled in some manner. we would also like to have similar expressions for the spring constant

$$k_{holes} = \gamma k_{ideal}. \tag{24.10}$$

If we assume that e_{holes} is proportional to $1/d$ then the following relations may easily be derived.

$$\beta = \alpha - \alpha' d \tag{24.11}$$

and

$$\gamma = \alpha - \alpha' d + \frac{1}{2}\alpha'' d^2, \tag{24.12}$$

where the derivation is done respect to d.

Now we are only left with the problem of finding a nice functional approximation for α. The holes in the membrane may be expressed using three dimensionless variables $\tilde{d} = d/r$, $\tilde{b} = b/r$ and $\tilde{R} = R/r$. Here r is the hole radius, b the hole depth, d the aperture and R the distance between holes. When $\tilde{R} >> 1$ and $b >> 1$ the correction depends only on $\tilde{d}$.

Numerical computations suggest that the correction $\alpha(\tilde{d})$ should approach unity as the distance $\tilde{d}$ approaches unity. On the other hand, it should approach $1 - q$ for small values of d. Here q is the area fraction of the holes.

Numerical calculations suggest that a second order rational polynomial gives quite an accurate fit to the computed results,

$$\alpha(d) = 1 - q\frac{1}{1 + a_1 d + a_2 d^2}. \tag{24.13}$$

Fully analytical formulas are now more tedious but the values for β and γ are easily calculated using the derivatives

$$\alpha(d)' = q\frac{a_1 + 2a_2 d}{(1 + a_1 d + a_2 d^2)^2}, \tag{24.14}$$

and

$$\alpha(d)'' = 2q\frac{a_2 - 2a_1^2 - 3a_1 a_2 d - 3a_2^2 d^2}{(1 + a_1 d + a_2 d^2)^3}. \tag{24.15}$$

Least squares fitting to the numerical computations suggest that for cylindrical hole $a_1 = 4.2523$, $a_2 = 0.4133$, for a rectangular slot $a_1 = 2.3198$, $a_2 = 0.2284$ and for a square hole $a_1 = 3.8434$, $a_2 = 0.3148$. When fitting the model the suggested constant term diverged up to 4 % from unity but the value one was enforced anyway because it has the nice limiting value properties.

24.2.2 Dielectric layer

If the conductor is covered with a dielectric layer we need to modify the equations. We assume that the aperture consists of two materials with permittivities ε_1 and ε_2 and thicknesses d_1 and d_2. Because the flux must be the same this means that the fields are

$$E_1 = \frac{\Phi}{d_1 + \varepsilon_1 d_2/\varepsilon_2} \tag{24.16}$$

and

$$E_2 = \frac{\Phi}{\varepsilon_2 d_1/\varepsilon_1 + d_2}. \tag{24.17}$$

Defining $d_x = d_1 + \varepsilon_1 d_2/\varepsilon_2$ these become

$$E_1 = \frac{\Phi}{d_x} \tag{24.18}$$

and

$$E_2 = \frac{\varepsilon_1}{\varepsilon_2}\frac{\Phi}{d_x}. \tag{24.19}$$

The total energy density is then

$$e = \frac{1}{2}\varepsilon_1 \Phi^2 \frac{d_1}{d_x^2} + \frac{1}{2}\frac{\varepsilon_1^2}{\varepsilon_2}\Phi^2 \frac{d_2}{d_x^2} = \frac{\varepsilon_1 \Phi^2}{2 d_x}. \tag{24.20}$$

We assume that the resonator moves so that d_1 changes and d_2 remains constant. Then the force density is

$$f = \frac{\partial e}{\partial d_1} = \frac{\partial e}{\partial d_x}\frac{\partial d_x}{\partial d_1} = -\frac{\varepsilon_1 \Phi^2}{2 d_x^2}. \tag{24.21}$$

And similarly the spring constant density

$$k = \frac{\partial f}{\partial d_1} = \frac{\varepsilon_1 \Phi^2}{d_x^3}. \tag{24.22}$$

These expressions may be used inside the integral instead of the constant field values to account for the dielectric layer. It may be noted that the equations are exactly the same as for the case without the layer except that the aperture d is replaced with the efficient aperture $d_x = d_1 + \varepsilon_1 d_2/\varepsilon_2$.

24.3 Implementation issues

This module is not a solver in itself. It only provides boundary conditions for real models. Natural models to combine with these boundary conditions are models describing deformation in solid structures. For plates the conditions are applied to the leading dimension while for generic 3D solids the conditions are applied to the boundaries. Therefore the same subroutines may be applied to either boundary or to material section. There is actually just one subroutine and the value it returns is defined by the name of the routine used to call it.

These routines here were historically developed for MEMS modeling in a different setting and were much later added to the open source publication as a lighter version.

24.4 Keywords

`Constants`

`Permittivity Of Vacuum Real [8.8542d-12]`
The default is given in SI units. In other units the constant should be changed appropriately.

`Boundary Condition bd id`

`Procedure "StatElecBoundary" "StatElecBoundaryForce"`
: Function that returns the nodal force density.

`Procedure "StatElecBoundary" "StatElecBoundaryCharge"`
: Function that returns the nodal charge density.

`Procedure "StatElecBoundary" "StatElecBoundaryEnergy"`
: Function that returns the nodal energy density.

`Procedure "StatElecBoundary" "StatElecBoundarySpring"`
: Function that returns the nodal spring density.

`Gap Height Real`
: Distance on which the 1D electrostatic model is applied for. May depend on displacement, for example, via MATC functions.

`Potential Difference Real`
: Potential difference between the plates.

`Relative Permittivity Real`
: Relative permittivity of the material between the plates.

`Layer Thickness Real`
: There may be a non-conducting layer on top of the plate. If this keyword is not defined no layer is assumed.

`Layer Permittivity Real`
: Relative permittivity of the layer.

`Hole Type String [slot / round / square]`
: The 1D electrostatics can account also for perforated structures if the depth of the hole is large compared to the width of the hole. The different hole geometries are an infinite slot, a round hole and a square hole.

`Hole Size Real`
: The size of the hole is for a round hole the radius, for a square half the side and for a slot half of the width.

`Hole Fraction Real`
: The fraction of the holes on the surface.

`Hole Depth Real`
: The depth of the holes i.e. also the thickness of the perforated plate.

Model 25

Poisson-Boltzmann Equation

Module name: PoissonBoltzmannSolve
Module subroutines: PoissonBoltzmannSolve
Module authors: Peter Råback
Document authors: Peter Råback

25.1 Introduction

The macroscopic electromagnetic theory is governed by Maxwell's equations. In steady state the electric field may usually be solved from a simple Poisson equation. However, if there are free charges in the domain that are affected by the electric field the equation is no longer valid. Also the contribution of the free charges need to be taken into consideration. If the electrostatic force is the only force affecting the distribution of the electric charges then the potential in the steady-state is given by the Poisson-Boltzmann equation [1]. This equation may find its use in microfluidics and electrochemical applications. Note that if the charge distribution is affected by the flow distribution of the carrier fluid this equation is no longer valid.

25.2 Theory

The electrostatic equation for the electric potential ϕ yields,

$$-\nabla \cdot \varepsilon \nabla \phi = \rho, \tag{25.1}$$

where ε is the permittivity of the medium and ρ is the charge density. Assuming that there is a fixed charge density and both positive or negative moving ions the charge may be written as

$$\rho = \rho_0 + e(z^- n^- + z^+ n^+) \tag{25.2}$$

where ρ_0 is interior charge distribution of fixed positions of all solute charges, and e is the unit charge of a electron, and z is the charge number of the positive or negative ions, and n is the corresponding ion density.

The electrochemical potential μ of the ions is defined by $\mu = ez\phi + k_B T \ln n$, where the first term is the electrostatic contribution and the second term comes from the entropy of the ions at the weak solution limit. In equilibrium μ_i is constant over the whole domain and thus the ion density obeys a Boltzmann distribution,

$$n = n_0 \mathrm{e}^{-ez\phi/k_B T} \tag{25.3}$$

where k_B is the Boltzmann constant. Inserting this to the Poisson equation we obtain the Poisson-Boltzmann equation that determines the potential field self-consistently,

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 + ez^- n_0^- \mathrm{e}^{-ez^- \phi/k_B T} + ez^+ n_0^+ \mathrm{e}^{-ez^+ \phi/k_B T}. \tag{25.4}$$

A special case of the equation is obtained if the charge numbers and the concentrations are equal, $z = -z^- = z^+$ and $n_0 = n_0^- = n_0^+$. Then the equation simplifies to

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 - 2ezn_0 \sinh(ez\phi/k_B T). \tag{25.5}$$

The Poisson-Boltzmann equation is obviously nonlinear. We will show the iterative procedure only for this case, the generic case is dealt similarly.

25.2.1 Iteration scheme

Defining $\alpha = 2ezn_0$ and $\beta = ez/k_B T$ the Poisson-Boltzmann equation for a symmetric electrolyte may be written as

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 - \alpha \sinh(\beta\phi). \tag{25.6}$$

The straight-forward iterative procedure treats only the left-hand-side of the equation in an implicit manner,

$$-\nabla \cdot \varepsilon \nabla \phi^{(n+1)} = \rho_0 - \alpha \sinh(\beta\phi^{(n)}). \tag{25.7}$$

The convergence of this scheme is, however, quite poor for many cases of practical interest. An improved strategy should linearize also the right-hand-side.

Making a Taylor's expansion we may approximate

$$\sinh(\beta\phi^{(n+1)}) \approx \sinh(\beta\phi^{(n)}) + \beta \cosh(\beta\phi^{(n)})(\phi^{(n+1)} - \phi^{(n)}) \tag{25.8}$$

which results to the Newton iteration scheme

$$\begin{aligned} &\left[-\nabla \cdot \varepsilon \nabla + \alpha\beta \cosh(\beta\phi^{(n)})\right] \phi^{(n+1)} \\ &= \rho_0 - \alpha \sinh(\beta\phi^{(n)}) + \alpha\beta \cosh(\beta\phi^{(n)})\phi^{(n)}. \end{aligned} \tag{25.9}$$

This scheme has good convergence properties and is usually the method of choice.

25.2.2 Boundary conditions

For electric potential either Dirichlet or Neumann boundary condition can be used. The Dirichlet boundary condition gives the value of the potential on specified boundaries. The Neumann boundary condition is used to give a flux condition on specified boundaries

$$\sigma = \varepsilon \nabla \phi \cdot \vec{n}, \tag{25.10}$$

where σ is the surface charge density.

25.2.3 Derived quantities

When the potential has been solved the electric field may be obtained as a postprocessing step from

$$\vec{E} = -\nabla \phi. \tag{25.11}$$

Charge density may be obtained as the right-hand-side of the Poisson equation,

$$\rho = \rho_0 + ez^- n_0^- \mathrm{e}^{-ez^-\phi/k_B T} + ez^+ n_0^+ \mathrm{e}^{-ez^+\phi/k_B T}. \tag{25.12}$$

which in symmetric case yields,

$$\rho = \rho_0 - 2ezn_0 \sinh(ez\phi/k_B T). \tag{25.13}$$

The energy density of the field ay be computed from

$$e = \frac{1}{2}\vec{E} \cdot \vec{D} = \frac{1}{2}\varepsilon(\nabla\phi)^2. \tag{25.14}$$

However, in a more generic treatment also the contribution of the concentration should be included in the expression of the energy.

25.3 Notes on output control

The user can control which derived quantities (*i.e.* electric field and electric energy) are calculated.

There are also available two choices of visualization types for the derived quantities. The node values can be calculated by taking the average of the derived values on neighboring elements (constant weights). This results often in visually good images. The other possible choice is to weight the average with the size of the elements, which is more accurate and should be used when some other variable depends on these derived values. The latter choice is also the default.

25.4 Keywords

`Constants`

`Permittivity Of Vacuum` Real [8.8542e-12 C^2/Nm^2]

`Boltzmann Constant` Real [1.3807e-23 J/K]

`Unit Charge` Real [1.602e-19 C]

`Equation` equation id

`Calculate Electric Energy` Logical [False]
Controls whether the electric energy density is written in results files (default False).

`Solver` solver id

`Equation` String Poisson Boltzmann Solver

`Variable` String Potential
This may be of any name as far as it is used consistently also elsewhere.

`Variable DOFs` Integer 1
Degrees of freedom for the potential.

`Procedure` File PoissonBoltzmannSolve PoissonBoltzmannSolve
Following are listed three keywords with default values for output control.

`Nonlinear System Max Iterations` Integer
The maximum number of nonlinear iterations.

`Nonlinear System Convergence Tolerance` Real
The relative error after which the iteration is terminated.

`Nonlinear System Newton After Iterations` Integer
The number of iterations after which Newton iteration is turned on. The default is zero which should usually be optimal.

`Nonlinear System Newton After Tolerance` Real
Optional parameter which gives the tolerance in error after which Newton iteration is turned on.

`Calculate Electric Field` Logical [True]

`Calculate Electric Flux` Logical [True]

`Constant Weights` Logical [True]
Used to turn constant weighting on for the results.

`Material` mat id

`Relative Permittivity` Real
The total permittivity is the product of the relative permittivity and the permittivity of vacuum.

`Reference Temperature` Real
This keyword is used to give the temperature occurring in the Boltzmann factor.

`Charge Number` Integer
: For symmetric cases the charge number. For unsymmetric cases one may give separately `Positive Charge Number` and `Negative Charge Number`.

`Ion Density` Integer
: For symmetric cases the original density of ions. For unsymmetric cases one may give separately `Positive Ion Density` and `Negative Ion Density`.

An alternative set of parameters are also possible which are particularly suitable for testing purposes. These are limited to the symmetric case where the potential normalized with the Zeta potential is solved. Then the permittivities should be set to unity and only two variables are needed to define the case.

`Poisson Boltzmann Beta` Real
: This keyword gives the ratio of parameter β to the the Zeta potential.

`Poisson Boltzmann Alpha` Real
: This keyword gives the parameter α

`Body Force` bodyforce id

`Charge Density` Real
: The fixed charge distribution that is not affected by the electric field.

`Boundary Condition` bc id

`Potential` Real

`Electric Flux BC` Logical
: Must be set to `True` if flux BC is used.

`Surface Charge` Real
: Gives the surface charge for the Neumann boundary condition.

Bibliography

[1] D. Andelman. *Handbook of Biological Physics*, chapter 12. Electrostatic Properties of Membranes: The Poisson-Boltzmann Theory. Elsevier Science, 1995.

Model 26

Loss Estimation Using the Fourier Series

Module names: FourierLoss
Module subroutines: FourierLossSolver
Module authors: Peter Råback, Mika Malinen
Document authors: Mika Malinen, Peter Råback

26.1 Introduction

The primary motivation for this solver is the estimation of electromagnetic losses by using the Steinmetz equation approach. It could have other uses as well. The main idea is to make a Fourier transformation on-the-fly and compute losses that are proportional to the frequency always when full cycle has been completed.

Given an evolutionary finite element field

$$\vec{A}_h(x,t) = \sum_{j=1}^{N} \alpha_j(t)\vec{\psi}_j(x), \tag{26.1}$$

the solver enables to replace the evolution of the scalar degrees of freedom $\alpha_j(t)$ for $t \in [t_0, t_0+T]$ by the Fourier series approximation

$$\alpha_j(t) \approx a_j^0 + \sum_{k=1}^{K} a_j^k \cos[k\omega(t-t_0)] + \sum_{k=1}^{K} b_j^k \sin[k\omega(t-t_0)], \tag{26.2}$$

where the angular frequency ω may be defined in terms of a period T as $\omega = 2\pi/T$. The coefficients a_j^k and b_j^k are given by

$$\begin{aligned} a_j^0 &= \frac{1}{T}\int_0^T \alpha_j(t'+t_0)dt', \\ a_j^k &= \frac{1}{T}\int_0^T \alpha_j(t'+t_0)\cos(2\pi k t'/T)dt', \\ b_j^k &= \frac{1}{T}\int_0^T \alpha_j(t'+t_0)\sin(2\pi k t'/T)dt'. \end{aligned} \tag{26.3}$$

In practice the time stepping algorithm gives the values of $\alpha_j(t)$ at only a discrete set of time values and linear interpolation is applied to generate $\alpha_j(t)$ at the other points.

The use of (26.2) in (26.1) now yields

$$\vec{A}_h(x,t) \approx \vec{a}_0(x) + \sum_{k=1}^{K} \vec{a}_k(x)\cos[k\omega(t-t_0)] + \sum_{k=1}^{K} \vec{b}_k(x)\sin[k\omega(t-t_0)] \tag{26.4}$$

where the fields $\vec{a}_k(x)$ and $\vec{b}_k(x)$ have the finite element expansions

$$\vec{a}_k(x) = \sum_{j=1}^{N} a_j^k \vec{\psi}_j(x) \quad \text{and} \quad \vec{b}^k(x) = \sum_{j=1}^{N} b_j^k \vec{\psi}_j(x).$$

If the field of interest is $\vec{B} = \nabla \times \vec{A}_h$, we have similarly

$$\vec{B}(x,t) \approx \nabla \times \vec{a}_0(x) + \sum_{k=1}^{K} \nabla \times \vec{a}_k(x)\cos[k\omega(t-t_0)] + \sum_{k=1}^{K} \nabla \times \vec{b}_k(x)\sin[k\omega(t-t_0)]. \tag{26.5}$$

If the problem setup is given in 3D case it is assumed that the solution is obtained using edge element basis. In the special case of 2D target field and the target variable is expected to be a scalar field $A_h(x,t)$ and $\vec{B}$ is then generated as

$$\vec{B} = \frac{\partial A_h}{\partial y}\vec{e}_x - \frac{\partial A_h}{\partial x}\vec{e}_y.$$

26.2 Loss estimation

In a typical application we have in mind the field $\vec{A}_h$ is taken to be the vector potential solution corresponding to the AV formulation of electromagnetic equations. Then the field $\vec{B} = \nabla \times \vec{A}_h$ gives the magnetic flux density which, in view of (26.5), may be approximated in the form

$$\vec{B}(x,t) \approx B_0(x) + \sum_{k=1}^{K} B_k(x)\cos[k\omega(t-t_0) - \phi_k],$$

with ϕ_k a phase angle.

The loss power associated with each simple-harmonic component may then be estimated over a body Ω by using the Steinmetz equation approach as

$$P_k = \int_{\Omega} C f_k^{\alpha} B_k^{\beta}\, d\Omega \tag{26.6}$$

where C, α and β are given data and $f_k = k\omega/(2\pi)$. The total loss P is then obtained as

$$P = \sum_{k=1}^{K} P_k. \tag{26.7}$$

The field variable P_h associated with this solver is the total loss power distribution per unit volume which is obtained from the weak formulation

$$\int_{\Omega} P_h v_h\, d\Omega = \int_{\Omega} \sum_{k=1}^{K} C f_k^{\alpha} B_k^{\beta} v_h\, d\Omega \tag{26.8}$$

where v_h denotes a suitable test function. The current implementation enables several terms with constant exponents α and β. The coefficient C, on the other hand varies among materials and may be a function of frequency.

26.3 Keywords

`Solver solver id`

`Procedure File "FourierLoss" "FourierLossSolver"`
: This keyword is used to give the Elmer solver the place where to search for the routine producing the loss estimate.

`Equation String [FourierLoss]`
: A name to the computational version of the loss equation may be given by using this keyword. The name has no effect but it should be unique.

`Target Variable String`
: The value of this keyword gives the name of the field for which the Fourier series expansion is produced. It is usually assumed that the field is magnetic vector potential. It may be either nodal (with 1 component in 2D and 3 components in 3D) or a curl-conforming vector field.

`Target Variable AV Logical`
: The user may enforce the target variable to be one resulting from the AV-solver which uses the edge degrees of freedom to express the vector potential. If the flag is not given, the existence of the AV solver is deduced from the size of the permutation vector. Because the detection may be erroneous, for example when a DG field is used, this may also be given value `False` to enforce use of a nodal field.

`Target Variable Direct Logical`
: By default it is assumed that the target variable is the magnetic vector potential and the curl operator is applied. However, with this flag it is possible to give either scalar field $|B|$ or vector field $\vec{B}$ directly. The handicap of this is that the field is not continuous. However, this may be particularly useful for testing purposes.

`Variable String`
: The primary variable is the loss component for the linear frequency dependence. The default name is `Fourier Loss` if all components are summed together and `Fourier Loss i` if each component is treated separately.

`Separate Loss Components Logical`
: By setting this flag to `True` the user may save the different loss components in Steinmetz model as separate fields for postprocessing. Otherwise they will be merged to one single field.

`Inexact Integration Logical`
: The integration may be performed most accurately by integrating the product of the linear approximation of the time-dependence and the sines and cosines analytically. Currently this exact integration is used. However, the user may replace it with inexact version even though this is not recommended.

`Simpsons Rule Logical`
: If using inexact integration, the default method is the trapezoidal integration. Instead the user may request the Simpson's rule for better accuracy. If exact integration is used, this keyword has no effect.

`Discontinuous Galerkin Logical`
: Instead of the standard Galerkin formulation the user may also choose to compute the post-processed fields using discontinuous Galerkin (DG) approximation. This enables discontinuities between elements. However, the cost is often quite large since the matrix structures related to DG are very large compared to the standard Galerkin method. The lumped quantities are computed on-the-fly so for them there is no difference.

`Average Within Materials Logical`
: This only applies when the DG method is used to compute the loss fields. When this flag is turned on the fields within same material are weakly enforced to be continuous. This results to much smoother fields while still maintaining the discontinuity over material interfaces.

`Calculate Elemental Fields` Logical
: The user may request elemental fields to be computed. These are natural since they allow discontinuous fields to be visualized. This features uses the same degrees of freedom as the DG approximation but does not include interactions between elements.

`Calculate Nodal Losses` Logical
: This may be used to calculate the losses directly in terms of power (i.e. Watts in SI units) lost in each node. For conforming meshes this provides the easiest coupling with heat equation and these losses become directly the r.h.s. source terms for the heat equation. Nodal losses are always lumped to one field even though the distributed fields would be separated.

`Fourier Series Components` Integer
: This keyword is used to define the parameter K in (26.2), i.e. how many Fourier series components are generated.

`Fourier Series Output` Logical
: If the value `True` is given, then the Fourier component fields $\vec{a}_k$ and $\vec{b}_k$ are output into the result file. Note that for edge elements this does not currently work as they cannot be directly written to a file but they should be mapped into a space that is able to be visualized.

`Angular Frequency` Real
: This keyword is used to give the angular frequency ω. Alternatively, the simulation section may be used for the same purpose.

`Frequency` Real
: Instead of giving the angular frequency, the user may specify the frequency $f = \omega/(2\pi)$ by using this keyword.

`Fourier Start Time` Real
: This keyword can be used to define the start time t_0 for performing the integration.

`Fourier Start Timestep` Integer
: This keyword can be used to define the start time t_0 such that $t_0 = t_n$, with n the timestep index given by using this keyword.

`Fourier Start Cycles` Integer
: This keyword can be used to define the start time t_0 such that $t_0 = nT$, with n the value of this keyword.

`Fourier Integrate Cycles` Integer
: By default the Fourier coefficient computation is restarted after integration over one complete period T. If this keyword is used, then the restarting occurs after integration over n periods, with n the value of this keyword. The reason for this could be that the results include some randomness that we would like to filter out by integrating over several cycles.

`Harmonic Loss Frequency Exponent(K)` Real
: The value of this keyword gives the parameters α_k in a vector format. The number of components, K, should be the same as the number terms in Steinmetz loss model. Alternatively the user may give this componentwise, i.e. `Harmonic Loss Frequency Exponent 1`, etc.

`Harmonic Loss Field Exponent(K)` Real
: The value of this keyword gives the parameters β_k in a vector format. The number of components, K, should be the same as the number terms in Steinmetz loss model. Alternatively the user may give this componentwise, i.e. `Harmonic Loss Field Exponent 1`, etc.

`Fourier Loss Filename` File
: Name for the file in which the losses will be saved. Losses will be saved so that each body for which there are any losses is written to a separate line. If the name is not given, nothing will be saved.

`Material` material id

`Harmonic Loss Coefficient i` Real
: This keyword is used to define the material parameter C in (26.6) for each simple-harmonic term

i. The highest existing value i determines the number of terms in the loss model. Note that the coefficient may only be a function of frequency itself.

Model 27

Coil Current Solver

Module name: CoilSolver
Module subroutines: CoilSolver
Module authors: Peter Råback
Document authors: Peter Råback

27.1 Introduction

Assume that we have been given a geometry of a coil. Now the coil consists of individual wires and therefore the direction of the current is basically well defined. Unfortunately it is often difficult to provide the directional information of the wires. In simple cases (such as ideal cylinder) the user may give the current in the coil using analytical functions. In general case this is however, not possible.

This solver tries to provide numerical means for setting the coil current. The idea is that the computed current density (or potential) may then be used in further simulations of magnetic fields.

Now the natural way to create the currents is to solve a static current conduction equation. Unfortunately for closed loops it is difficult to define the boundary conditions. Rather than trying to make some special mesh this solver provides a unique strategy to create the currents in two pieces using Dirichlet conditions for some nodes in the bulk.

27.2 Theory

Our starting point for defining currents in the coil is to use an equation for the static current conduction,

$$-\nabla \cdot \sigma \nabla \phi = 0. \tag{27.1}$$

where ϕ is the electric potential, and σ is the electric conductivity. For this equation we may either set Dirichlet boundary conditions

$$\phi = \phi_0 \tag{27.2}$$

or Neumann boundary conditions

$$-\sigma \frac{\partial \phi}{\partial n} = j_n. \tag{27.3}$$

If we solver the equation with the standard Galerkin method in FEM the volume current density may then be calculated using the same finite element spaces from

$$\vec{j} = -\sigma \nabla \phi. \tag{27.4}$$

Normalization of potential and current

Typically we know the total current over the cross section

$$J = \int j_n \, d\Gamma. \tag{27.5}$$

If the current density over the boundary area A is constant this gives the current density, $j_n = J_0/A$. Often it is more ideal to define Dirichlet conditions and normalize the potential afterwords to give the desired total current,

$$\phi := \frac{J_0}{J}\,\phi \tag{27.6}$$

and similarly for the current,

$$\vec{j} := \frac{J_0}{J}\,\vec{j}. \tag{27.7}$$

If we know that the cross section of the coil is constant and thereby also the current density is constant j_0 we may normalize the current density also locally taking only the direction from the initial current,

$$\vec{j} := j_0 \frac{\vec{j}}{|\vec{j}|}. \tag{27.8}$$

Now this normalization should not be done light-heartedly since after this the solution might not be divergence-free any more. When the magnitude of the solution is known *a priori* we could use also other means of obtaining the direction field.

Modified conductivity field

Now in the previous we assumed that conduction is isotropic and homogeneous. Unfortunately this means that for coils the current won't follow the wires as it tends to take the shortest path more easily. On the other hand we don't know the anisotropic conductivity since if we knew it we would also know the direction of the current and there would be nothing to compute. Therefore we look at some ways to iteratively determine the solution from the already computed field.

One idea is to introduce an additional field variable, say c, such that

$$-\nabla \cdot c\sigma\nabla\phi = 0. \tag{27.9}$$
$$c|\sigma\nabla\phi| = j_0. \tag{27.10}$$

This may be iteratively solved by solving ϕ from the 1st equation and c from the 2nd equation. Unfortunately numerical tests showed that the resulting field for c that is prone to numerical oscillations. Therefore we add some regularization to the 2nd equation by adding some diffusion

$$-\nabla \cdot D\nabla c + |\sigma\nabla\phi| c = j_0. \tag{27.11}$$

where D is the numerical diffusion. Unfortunately the equation does not necessarily have any solution. In fact real tests showed that increasing value of D will make the solution smoother but at the same time it will be more difficult to find a solution that would fulfil both equations. Still the current density may be closer to the real current density than in the homogeneous case. Just a small number of iterations will get the most of the benefits of the scaled conductivity field.

We may also look at the conductivity in an anisotropic form. One idea is to use the knowledge that in the true coil the conductivity is only in the direction of the gradient. We may therefore look at the conductivity in the form

$$\sigma = \sigma_0 \frac{\nabla\phi}{|\nabla\phi|} \tag{27.12}$$

Now without the multiplier c this would not lead to an improvement because then the direction does not have an effect as the flux is always in the direction of the gradient only. However, the combined effect of these two could improve the accuracy of the direction of the current density.

There can be also partially geometric means for setting the direction of the conductivity field. Basically the distance from the coil surface, s, defines a field the gradient of which defines a direction with zero conductivity. So we take a isotropic conductivity as a starting point and eliminate the conductivity in the direction of the normal i.e.

$$\sigma = \sigma_0 \frac{1}{\sqrt{3}} \left[I - \text{abs}(\nabla s) \right] \tag{27.13}$$

Unfortunately this formulation has some difficulties at the center of the coil where the the distance function is poorly defined.

For structured and extruded meshes one could also use the directions of the element edges to directly set the direction of the current. Starting from a rough information for the current direction the edges aligned with the current could be detected and thereafter the current direction could be refined accurately with the geometry. This method has not been implemented.

Case of closed coils

The closed coils are dealt with in two parts. The idea is to use rough fiction boundary conditions for the bulk nodes. Setting potentials to 0 and ϕ_0 on a narrow gap around the fictitious interface we can generate a current to the coil. Now this current is "good" only in the other side of the coil where the effects of the poorly defined boundary conditions have settled. The distance from the fictitious interface should be at least around three times the diameter of the coil cross section. Therefore the approach is better suited for "lean" coils. We apply the same procedure to opposite sides of the coil and normalize the coil current to be the same. The final current is an union of the two cases so that we always take the more accurate of the two computed currents fields.

Now the currents should ideally be continuous. However, the potential derived in this way can never be continuous since at some stage the potential should have a jump over the coil. Still, if the coil potential is used only element-wise so that it is always operated by a gradient we may use a special subroutine that returns always the more accurate set of nodal potentials for each element. After taking the gradient of the potentials the solution should be continuous.

Using the coil solver

The solver can be used to compute currents or potentials to be used by other solvers. The standard case could equally well be computed by the `StatCurrentSolver` but this solver includes some ways to make the current distribution more accurate to the case of coils. Unfortunately all these techniques are based on heuristics and they will most likely offer better current distributions that the standard approach.

If one tries to normalize the current density to a predefined constant value the user should be sure that the cross section of the coil is constant. Otherwise the user introduces a current source or drain that may be difficult to treat in the later steps of the simulation.

When using in conjunction with the `MagnetoDynamics` module it is advisable to perform the fixing of the current to be divergence free using the internal `Jfix` solver. Otherwise the solution might not exist for the equation.

Currently it assumed that there is just one coil, and for the closed coil it is assumed that the coil axis is in the direction of the z-axis. These limitations are not inherent to the method and could be resolved by small coding effort.

27.3 Keywords

The solver includes some internal definitions which eliminates some keywords. For example, the variable related to the solver is internally created and also the boundary conditions related to it are set internally so the user does not need to know of it.

```
Solver  solver id

    Equation  String CoilSolver
```

`Procedure` `File "CoilSolver" "CoilSolver"`
Name of the solver module and solver.

`Coil Closed` `Logical`
Is the coil closed. If it is then the potential will be computed in two parts. Note that for many coils it is currently assumed that all the coils are either open or closed. Closed coils will in effect create a secondary potential field and an auxiliary field `PotSelect` used to toggle between the two potentials.

`Coil Conductivity Fix` `Logical`
Fix the coil conductivity so that the current density would be more even.

`Cfix diffusion` `D`
iffusion coefficient for regularization of c field.

`Coil Anisotropic` `Logical`
Make the conductivity be aligned with the gradient of the potential.

`Calculate Coil Current` `Logical [True]`
Calculate the current flowing in the coil(s). The effect of the enforced currents in the coil may be given as a source term either as current, or as a enforced potential.

`Use Wall Distance` `Logical`
Use wall distance to introduce anisotropy into the coil conductivity. If wall distance is used then it is assumed that a field `Wall Distance` exists.

`Save Coil Set` `Logical`
Optionally save the `CoilSet` field.

`Save Coil Index` `Logical`
Optionally save the `CoilIndex` field. This makes sense only if there are more than one coil. Then the index gives the number of the coil.

`Normalize Coil Current` `Logical`
After the current has been computed normalize it to the desired magnitude if this flag is given.

`Nonlinear System Max Iterations` `Integer`
For the inhomogeneous cases give the number of iterations. The default is 2 for the scaled conductivity, and 3 for the scaled anisotropic conductivity.

The following keywords define one single coil. They may be located in the `Solver` section if there is just one single coil. If there are many coils they should be located in different `Component` sections.

`Desired Coil Current` `Real`
The desired coil current J_0 in the coil. The default is 1.

`Desired Current Density` `Real`
The desired coil current density j_0 in the coil. The default is 1.

`Coil Cross Section` `Real`
Cross section (area) of the coil that may be used to related total current and current density.

`Coil Center(3)` `Real`
Center of the coil in (x, y, z) Cartesian coordinates. If coil center is not given then the volumetric mid point is assumed to be the center.

`Coil Normal(3)` `Real`
The user may give the coil axis around which the coil circulates around. If not given the coil axis is found as the axis that gives the maximum inertial momentum for the coil. The tangent directions are deduced from the coil normal.

`Coil Bandwidth` `Real`
A parameter related to the width of the fictitious interface used to set the potentials in an automated way. This is relative to the total width. Default is 20% of the width of the coil.

`Narrow Interface` Logical
: This flag enforces the use of narrow strategy for the setting of Dirichlet conditions and computation of the resulting nodal charges. It is a better strategy than the wider coil bandwidth strategy particularly for thick coils. Makes the above keyword obsolete.

`Boundary Condition` bc id

`Potential` Real
: Dirichlet BC for the potential.

`Current Density BC` Logical
: Must be set to `True` if Neumann BC is used.

`Coil Start` Logical
: Defines a boundary where coil starts. Not needed if coil is closed.

`Coil End` Logical
: Defines a boundary where coil ends. Not needed if coil is closed.

The user can use the united potential also for closed coils by using a special subroutine. The calling convention would then be, for example, in the `MagnetoDynamics` module in the `Body Force` section

```
Electric Potential = Variable time
  Real Procedure "CoilSolver" "CoilPotential"
```

Here `time` is just a dummy variable. Similarly for coil potential normalized on-the-fly within each element,

```
Electric Potential = Variable time
  Real Procedure "CoilSolver" "CoilPotentialNormalized"
```

Part V

Other Physical Models

Model 28

Electrokinetics

Module name: Electrokinetics
Module subroutines: helmholtz_smoluchowski1, helmholtz_smoluchowski2, helmholtz_smoluchowski3, helmholtz_smoluchowski
Module author: Thomas Zwinger
Document author: Thomas Zwinger

28.1 Introduction

If dealing with electrolytic fluids constrained to small volumes, surface forces caused by electric surface charges in combination with externally applied electrostatic fields are sufficient strong to affect the fluid volume. If these effects are utilized to attenuate the fluid volume, we talk of *Electrokinetics*. The term *Electroosmotic Flow* (EOF) is used in connection with the attenuation of a net charge inside a originally neutral electrolyte caused by separation induced by a surface charge of a wall.

28.2 Theory

Chemical reactions between the contents of a liquid and the wall material may lead to a net charge of the containment at the wall-liquid interface. If the liquid is an electrolyte (i.e., it contains free ions), ions of opposite charge align along the wall creating the *Stern layer*. Adjacent to the Stern layer, a charge separation - called the *diffuse layer* of the initially neutral electrolyte takes place. Due to the two layer structure the whole are area of charge separation in the vicinity of a wall is called the *Electric Double layer* (EDL).

28.2.1 Electroosmotic slip velocity

Considering a symmetric electrolyte – i.e., the bulk ion density of ions with opposite valence numbers $\pm z$ are equal $n_0^+ = n_0^- = n_0$ – at a certain temperature, T, the typical width-scale of the EDL is given by the *Debye length* [1]

$$\lambda_\mathrm{D} = \left(\frac{\epsilon_\mathrm{f} \epsilon_0 \, k_\mathrm{b} \, T_0}{2 \, n_0 \, z^2 \, e_0^2} \right)^{1/2} . \tag{28.1}$$

Here e_0 stands for the unit charge and k_b denotes the Boltzmann constant. The relative permittivity of the electrolyte and the permittivity of vacuum are given by ϵ_f and ϵ_0, respectively.

The potential, Φ and the volume charge density, ρ_e, within the EDL are tightly coupled to each other by the Poisson-Boltzmann equation (25.4) (see chapter 25). In order to exactly resolve the dynamics close to the walls, (25.4) should be solved and the resulting specific electric force then be considered in the equation of motion. Nevertheless, provided the typical length scales of the flow perpendicular to the containment walls, H, strongly exceed those of the EDL – in other words, we obtain very small values for the non-dimensional group $\mathcal{L} = \lambda_\mathrm{D}/H \ll 1$ – the dynamics of the electrolyte inside the EDL does not have to be resolved at all. In this case simple considerations of a force balance between shear stress and electric force lead to a slip condition for the fluid [2]. At the boundary, the tangential velocity is set to the *Helmholtz-Smoluchowski* velocity

$$\vec{u}_{\text{tang.}} = \vec{u}_{\text{H-S}} = \frac{\vec{E}_{\text{tang.}}\epsilon_{\text{f}}\epsilon_0\zeta}{\mu_{\text{f}}}, \tag{28.2}$$

with μ_{f} standing for the local fluid viscosity. The *zeta potential*, ζ – a property depending on the electric properties of the wall material as well as the electrolyte – usually is determined experimentally. From a physical point of view it can be interpreted as the value of the solution obtained by (25.4) at the Stern layer. The tangential component, $\vec{E}_{\text{tang.}}$, of the external electric field, $\vec{E}$, is evaluated from the outward pointing surface normal $\vec{n}$, applying the following relation

$$\vec{E}_{\text{tang.}} = \vec{E} - \left(\vec{E}\cdot\vec{n}\right)\vec{n} \tag{28.3}$$

Alternatively, the resulting slip velocity may be related to the tangential field using the *Electroosmotic Mobility*, μ_{EOF}

$$\vec{u}_{\text{H-S}} = \mu_{\text{EO}}\,\vec{E}_{\text{tang.}}. \tag{28.4}$$

A combination of (28.2) and (28.4) leads to the following identity

$$\mu_{\text{EO}} = \frac{\epsilon_{\text{f}}\epsilon_0\zeta}{\mu_{\text{f}}}. \tag{28.5}$$

28.3 Limitations

- The Helmholtz-Smoluchowski velocity should not be applied if the non-dimensional group $\mathcal{L}$ defined in 28.2.1 is of unity order or larger. Then the potential- and charge density distribution as well as the dynamics of the electrolyte inside the EDL has to be resolved.
- In a strict sense, the Helmholtz-Smoluchowski theory applies only to configurations where the normal-component of the external field, $\vec{E}\cdot\vec{n}$, is small. If dealing with electric insulating wall materials – as it is usually the case in microfluidic applications – this condition is implicitly complied with.
- The assumption of a Newtonian fluid underlies the derivation of the Helmholtz-Smoluchowski velocity.
- The function `helmholtz_smoluchowski` can only be applied on boundaries of two-dimensional domains, where the tangential direction is uniquely defined.

28.4 Keywords

Keywords for helmholtz_smoluchowski

`Constants`

`Permittivity Of Vacuum` `Real [8.8542e-12` C^2/Nm^2`]`
: permittivity of vacuum, only needed if Helmholtz-Smoluchowski velocity is defined using expression (28.2)

`Equation` `equation id`

`Electric Field` `String [computed, constant]`
: the option for how to evaluate the electric field should be set to one of these values.
If set to `computed`, the function will search for `Electric Field {1,2,3}` in the list of solver variables. If set to `constant`, the function will search for `Electric Field {1,2,3}` in the section `Material material id`, where `material id` is the id-number associated with the material parameter list of the electrolyte

`Material` `material id`

If the Helmholtz-Smoluchowski velocity is defined using expression (28.2), then the following keywords have to be provided in this section

`Viscosity` `Real`
: viscosity of the electrolyte

`Density` `Real`
: volumetric density of the electrolyte

`Relative Permittivity` `Real`
: relative permittivity of the electrolyte

`Boundary Condition` `bc id`

In two-dimensional configurations the Helmholtz-Smoluchowski velocity directly can be assigned to the tangential component of the velocity field

```
Normal Tangential Velocity  Logical True
Velocity 2  = Variable Dummyargument
    Real Procedure "Electrokinetics" "helmholtz_smoluchowski"
```
Sets tangential EO slip velocity

The argument `Dummyargument` can be any existing variable, since it is not used to evaluate the velocity.
In three-dimensional configurations (and as an alternative also in two-dimensional), the velocity has to be defined for each component

```
Normal Tangential Velocity  Logical False
Velocity 1  = Variable Dummyargument
    Real Procedure "Electrokinetics" "helmholtz_smoluchowski1"
Velocity 2  = Variable Dummyargument
    Real Procedure "Electrokinetics" "helmholtz_smoluchowski2"
Velocity 3  = Variable Dummyargument
    Real Procedure "Electrokinetics" "helmholtz_smoluchowski3"
```

The argument `Dummyargument` can be any existing variable, since it is not used to evaluate the velocity.
If the Helmholtz-Smoluchowski velocity is defined using expression (28.2), then the zeta potential, ζ, for the specific boundary region has to be defined

`Zeta Potential` `Real`
: Sets the zeta-potential for this boundary

Alternatively, the user can declare the EO-mobility, as explained in (28.5)

`EO Mobility` `Real`
: Sets EO mobility for this boundary

Bibliography

[1] G.E. Karniadakis and A. Beskok. *Micro flows : fundamentals and simulation*. Springer-Verlag, New York, Berlin, Heidelberg, 2001.

[2] R.-J. Yang, L.-M. Fu, and Y.-C. Lin. Electroosmotic Flow in Microchannels. *J. Colloid and Interface Science*, 239:98–105, November 2001.

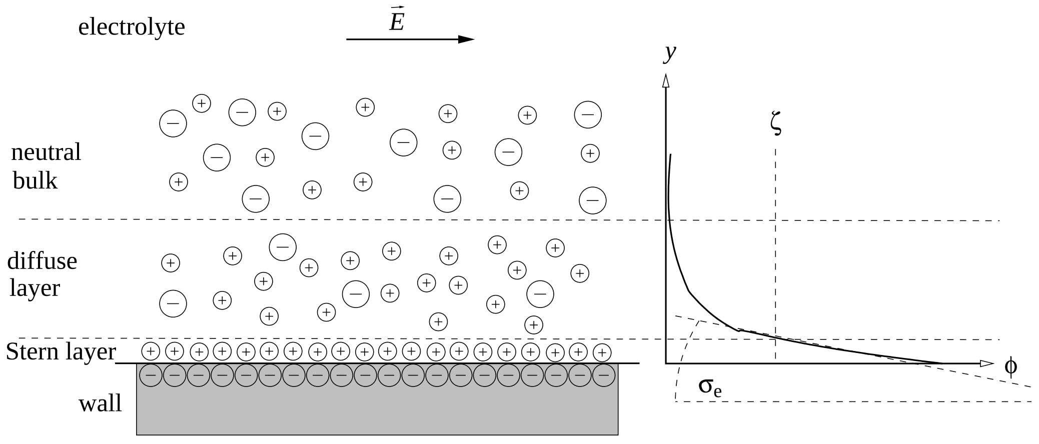

Figure 28.1: Structure of the EDL. The value of the induced potential, Φ at the Stern layer usually is referred to as the zeta-potential, ζ

Model 29

Mixed Approximation of the Poisson equation

Module name: ModelMixedPoisson
Module subroutines: MixedPoisson
Module authors: Mika Malinen
Document authors: Mika Malinen

29.1 Introduction

In some cases obtaining an accurate approximation of the flux connected with the solution of the Poisson equation may be of a particular interest. To this end, the flux may be introduced as an additional independent variable which is solved simultaneously with the associated scalar variable. Such approximations where two finite elements spaces are employed at the same time are generally known as mixed methods.

The mixed problem considered here also provides a basic example of the application of divergence-conforming finite elements. They are also known as face finite elements and offer a natural choice when just the continuity of the normal component of the solution can be assumed. It should be noted that the selection of face finite elements is not complete yet, so discretizations over simplicial (i.e., triangular or tetrahedral), quadrilateral and hexahedral meshes are only supported currently.

29.2 Governing equations and the weak formulation

Let us consider rewriting the Poisson problem

$$\begin{aligned} -\operatorname{div}(a\nabla u) &= f \quad \text{on } \Omega, \\ u &= g \quad \text{on } \Gamma_1, \\ a\nabla u \cdot \boldsymbol{n} &= q \quad \text{on } \Gamma_2 \end{aligned} \tag{29.1}$$

as

$$\begin{aligned} \boldsymbol{q} - a\nabla u &= \boldsymbol{0} \quad \text{on } \Omega, \\ -\operatorname{div}\boldsymbol{q} &= f \quad \text{on } \Omega, \\ u &= g \quad \text{on } \Gamma_1, \\ \boldsymbol{q}\cdot\boldsymbol{n} &= q \quad \text{on } \Gamma_2. \end{aligned} \tag{29.2}$$

Here a is a material parameter and the physical interpretations of the fields u and $\boldsymbol{q}$ depend on a case. For example, u may be the temperature or a potential variable, while $\boldsymbol{q}$ may represent the heat flux, electric field, current density, etc.

If $U \equiv L_2(\Omega)$ denotes the set of square-integrable scalar functions over the d dimensional body Ω, we are thus led to seeking the weak solution of the flux in the space of divergence-conforming functions defined by

$$\boldsymbol{Q} = \{\boldsymbol{v} \in L_2(\Omega)^d \mid \operatorname{div} \boldsymbol{v} \in L_2(\Omega) \text{ and } \boldsymbol{v} \cdot \boldsymbol{n} = q \text{ on } \Gamma_2\}.$$

By defining $\mu \equiv 1/a$, multiplication with test functions and integration by parts lead us to seeking $(\boldsymbol{q}, u) \in \boldsymbol{Q} \times U$ such that

$$\begin{aligned} \int_\Omega \mu \boldsymbol{q} \cdot \boldsymbol{v} \, d\Omega + \int_\Omega u \operatorname{div} \boldsymbol{v} \, d\Omega &= \int_{\Gamma_1} g(\boldsymbol{v} \cdot \boldsymbol{n}) \, d\mathrm{S}, \\ \int_\Omega \operatorname{div} \boldsymbol{q} \, w \, d\Omega &= -\int_\Omega f w \, d\Omega, \end{aligned} \tag{29.3}$$

for any $(\boldsymbol{v}, w) \in \boldsymbol{Q}_0 \times U$, with the test space of fluxes being defined by

$$\boldsymbol{Q}_0 = \{\boldsymbol{v} \in L_2(\Omega)^d \mid \operatorname{div} \boldsymbol{v} \in L_2(\Omega) \text{ and } \boldsymbol{v} \cdot \boldsymbol{n} = 0 \text{ on } \Gamma_2\}.$$

As a generalization, the scalar parameter μ can be replaced by a tensor of order 2. In addition the total time derivative may be added to the model. The weak formulation (29.3) is then replaced by

$$\begin{aligned} \int_\Omega \boldsymbol{\mu} \boldsymbol{q} \cdot \boldsymbol{v} \, d\Omega + \int_\Omega u \operatorname{div} \boldsymbol{v} \, d\Omega &= \int_{\Gamma_1} g(\boldsymbol{v} \cdot \boldsymbol{n}) \, d\mathrm{S}, \\ \int_\Omega \operatorname{div} \boldsymbol{q} \, w \, d\Omega - \int_\Omega (\frac{\partial p}{\partial t} + \boldsymbol{a} \cdot \boldsymbol{q}) w \, d\Omega &= -\int_\Omega f w \, d\Omega, \end{aligned} \tag{29.4}$$

where $\boldsymbol{\mu}$ is a second-order tensor and $\boldsymbol{a}$ is the convection velocity treated as a model parameter.

It is worth noting that the standard Neumann boundary condition is now built in the construction of the solution space, i.e., it is specified as an essential boundary condition in the mixed formulation. On the other hand the standard Dirichlet condition is handled as a natural condition. The boundary conditions of the standard and mixed formulation have thus opposite roles to another.

Finite element approximations to the solution of (29.3) are constructed in the standard manner by expressing the weak formulation in terms of finite element versions of the solution and test spaces. For the available face finite elements a natural finite dimensional version of U is based on functions which are constants elementwise. For additional details on the construction of divergence-conforming finite elements see also the Appendix "*Face and edge elements*" of the ElmerSolver Manual.

29.3 Keywords

`Material` mat id

`Material Parameter` Real
This keyword specifies the value of a.

`Material Tensor(3,3)` Real
This keyword defines the tensor $\boldsymbol{\mu}$ in the generalized model (29.4).

`Convection Velocity i` Real
The components of the convection velocity $\boldsymbol{a}$ may be given by using these keywords.

`Solver` solver id

`Equation` String
A describing name for the solver.

`Procedure` File "ModelMixedPoisson" "MixedPoisson"
The name of the solver subroutine.

`Variable`
: The name of the solver variable can be chosen freely (but it is must be used consistently elsewhere).

`Variable DOFs` Integer
: The value of this keyword should be 1.

`Second Kind Basis` Logical
: This keyword specifies whether the Brezzi-Douglas-Marini space or the Nédélec face elements of the second kind are used in two or three dimensions, respectively.

`Element` String
: A special element definition is needed to obtain a suitable set of degrees of freedom. The solver should be able to create an automated value for this keyword provided the value of the keyword `"Coordinate System"` specifies the dimension of coordinate system basis. That is, one should use the value `"Cartesian 2D"` or `"Cartesian 3D"` to specify the coordinate system.

`Body Force` bf id

`Source Field` Real
: The value of this keyword gives f.

`Boundary Condition` bc id

`Scalar Field` Real
: This keyword is used to define the data g.

`Q {f} j` Real
: If the solver variable has been given the name `Q`, this keyword defines a vector $\hat{\boldsymbol{g}}$ so that its normal trace $\hat{\boldsymbol{g}} \cdot \boldsymbol{n}$ is approximated by $\boldsymbol{q}_h \cdot \boldsymbol{n}$, with $\boldsymbol{q}_h$ the finite element solution. The value of this keyword defines the components $\hat{g}_j$, $j \in \{1, 2, 3\}$, with respect to the global Cartesian coordinate system.

Model 30

Block Preconditioning for the Steady State Navier–Stokes Equations

Module name: ParStokes, PressurePrecond, VelocityPrecond
Module subroutines: StokesSolver, PressurePrecond, VelocityPrecond
Module authors: Mika Malinen, Jonas Thies, Juha Ruokolainen
Document authors: Mika Malinen

30.1 Introduction

The discretization of the incompressible Navier–Stokes equations usually leads to large linear systems which cannot be solved efficiently with the standard iterative methods. Special preconditioning strategies should therefore be employed to obtain rapid convergence of iterations. In this section we describe a special solver for the steady state Navier–Stokes equations which has utility when the Reynolds number is moderate. The development of the solver was originally motivated by needs to solve the full Stokes equations in connection with glaciological simulations. Therefore, the possibility to utilize parallel computation has been in mind from the very beginning (the module name comes from *Par*allel *Stokes* solver).

The speciality of the solver is that it contains an in-built two-level iteration scheme to handle the linear systems arising from the linearization and discretization. Inner iterations can be associated with preconditioning and they provide search directions for the outer iterative method (GCR) applied to the primitive problem. The preconditioning strategy is here based on the idea of block preconditioning via utilizing the natural block structure of the discrete system. In practice simpler auxiliary problems need to be solved in order to find the update directions for the velocity and pressure unknowns in a decoupled manner. In this way the door is opened to utilizing other efficient methods, such as multigrid methods for the discrete Poisson problems, in connection with the solution of the more complicated problem.

The alternate flow solver contained in the `ParStokes` module basically mimics the standard Navier–Stokes (NS) solver of Elmer. However, it does not provide all features available in the standard NS solver.

30.2 The model

The equations to be solved are written as

$$\begin{aligned} \rho_0(\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} - \mathbf{div}[2\mu(\boldsymbol{D})\boldsymbol{D}(\boldsymbol{u})] + \boldsymbol{\nabla}p &= \rho_0\boldsymbol{g}, \\ -\operatorname{div}\boldsymbol{u} &= 0 \end{aligned} \tag{30.1}$$

or

$$\begin{aligned} -\mathbf{div}[2\mu(\boldsymbol{D})\boldsymbol{D}(\boldsymbol{u})] + \boldsymbol{\nabla}p &= \rho_0\boldsymbol{g}, \\ -\operatorname{div}\boldsymbol{u} &= 0 \end{aligned} \tag{30.2}$$

when the effect of convection is neglected to obtain the Stokes equations. Here $\boldsymbol{D}(\boldsymbol{u})$ is the symmetric part of the velocity gradient, the constant ρ_0 is the fluid density and μ is the fluid viscosity. For example, in the case of Glen's flow law we have $\mu(\boldsymbol{D}) = 1/2A^{-k}[I_2(\boldsymbol{D})]^{(k-1)/2}$, with A and k parameters and $I_2(\boldsymbol{D}) = 1/2(\boldsymbol{D} \cdot \boldsymbol{D})$.

Common boundary conditions may be expressed as

$$\begin{aligned} \boldsymbol{u} &= \hat{\boldsymbol{u}} \quad \text{on } \Gamma_D, \\ 2\mu(\boldsymbol{D})\boldsymbol{D}(\boldsymbol{u})\boldsymbol{n} - p\boldsymbol{n} &= \hat{\boldsymbol{s}} \quad \text{on } \Gamma_N, \\ \boldsymbol{u} \cdot \boldsymbol{n} = 0 \quad \text{and} \quad \boldsymbol{n} \times [2\mu(\boldsymbol{D})\boldsymbol{D}(\boldsymbol{u})\boldsymbol{n}] \times \boldsymbol{n} &= -\beta \boldsymbol{n} \times \boldsymbol{u} \times \boldsymbol{n} \quad \text{on } \Gamma_S. \end{aligned} \tag{30.3}$$

Here $\hat{\boldsymbol{u}}$ is the specified velocity, $\hat{\boldsymbol{s}}$ is the specified traction, and also the friction coefficient $\beta > 0$ is given as initial data.

30.3 Linearization

Either Newton's method or the strategy of Picard type can be used to linearize the effect of viscosity. The possible convection term is always linearized with the Picard method.

30.4 Discretization aspects

The solver is tailored to the case of the lowest-order continuous pressure approximation. Equal-order approximations of $(\boldsymbol{u}, p)$ are unstable, but the solver offers the following two strategies to obtain stable methods:

- P_2-P_1/Q_2-Q_1 *approximation.* The polynomial order of the velocity approximation can be taken to be of the second order. This option requires that the finite element mesh contains second-order elements based on the Lagrange interpolation.
- *A hierarchic version of bubble-stabilized methods.* The lowest-order velocity approximation may also be enhanced relative to the pressure by using elementwise bubble functions. The richness of velocity approximation depends on how many bubble basis functions are constructed. For example, the set of basis functions for the linear triangular and tetrahedral elements can be augmented with one interior bubble function by giving the element type definition `Element = "p:1 b:1"` in the Equation section. Analogous rectangular and brick elements may also be constructed, but our experience is that more than one bubble function may be necessary to obtain stability, making this strategy less attractive. It is recommended that the definition `Element = "p:1 b:4"` is generally used in three dimensional cases. The Q_2-Q_1 and P_2-P_1 approximation methods may require less computational work, especially for three-dimensional problems where the burden of numerical integration in the assembly phase is increased significantly.

It is noted that other instabilities generally arise when the flow is convection-dominated. The computational methods of the solver have not been designed to handle this scenario, i.e. the Reynolds number for the flow should be moderate.

30.5 Block preconditioning

The linearization and discretization of the flow model considered leads to solving linear systems of the form

$$\begin{bmatrix} A_k & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} U \\ P \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}, \tag{30.4}$$

where A_k is the coefficient matrix for the velocity unknowns at the nonlinear iteration step $k+1$ and B is the divergence matrix. If the linear system (30.4) is abbreviated as $Kx = b$, the outer iterative method based on the generalized conjugate residual method (GCR) can be used to generate iterates $x^{(k)}$ that minimize the residual 2-norm

$$||b - Kx^{(k)}||_2$$

over the search direction space

$$\mathcal{X}_k = x^{(0)} + \text{span}\{s^{(1)}, s^{(2)}, \dots, s^{(k)}\}$$

where $x^{(0)}$ is the initial guess. In this setting, the preconditioner is considered to be an operator P which, given the previous iterate, produces the new search direction $s^{(k+1)}$.

Here the search directions are found as approximate solutions of systems

$$\mathsf{P}s^{(k+1)} = b - Kx^{(k)}. \tag{30.5}$$

The preconditioner matrix is selected to be of the form

$$\mathsf{P} = \begin{bmatrix} T & B^T \\ 0 & Q \end{bmatrix} \tag{30.6}$$

where T approximates A and $Q = \mu^{-1}M$, with M the pressure mass matrix. By default the choice $T = A$ is used. An alternate is to select T to be the block diagonal approximation of A or the scaled vector-Laplacian matrix, which is the discrete version of the operator $-\mu\,\mathbf{div}\,\boldsymbol{\nabla}$ subject to suitable boundary conditions.

To apply the preconditioner via performing approximate solves of (30.5), the user must define methods for solving subproblems of the type

$$Q\delta P^{(k+1)} = R_P^{(k)} \quad \text{and} \quad T\delta V^{(k+1)} = R_A^{(k)} - B^T\delta P^{(k+1)}$$

or, in practice, their preconditioned versions

$$(Q\mathsf{P}_Q^{-1})\delta\hat{P}^{(k+1)} = R_P^{(k)} \tag{30.7}$$

and

$$(T\mathsf{P}_T^{-1})\delta\hat{V}^{(k+1)} = R_A^{(k)} - B^T\delta V^{(k+1)}. \tag{30.8}$$

The efficiency of the block-preconditioned solver depends heavily on how these auxiliary problems are solved (the solves associated with the coefficient matrix T are the most critical). A crucial aspect of the methodology is that these subsidiary problems can be considerably easier to solve than the original fully coupled system. They may also be solved inexactly without impairing the performance of the preconditioner. In addition, it is noted that performing highly accurate solutions of the linearized systems (30.4) is not needed in the beginning of the nonlinear iteration when the iterates are not accurate. The solver provides an option to employ adaptive stopping criteria so that the solution accuracy for (30.4) is adapted automatically based on the size of the current nonlinear error.

30.6 Defining additional solvers for subsidiary problems

If the block preconditioner is applied, the solver input file must contain two additional solver sections to enable the assembly of the subsidiary problems with the coefficient matrices T and Q. In this connection special equation names (given as the value of `Equation` keyword) have to be used. If the value of the parameter `d` defines the space dimension, these solver sections should be written as

```
Solver 1
  Equation = "Velocity Preconditioning"
  Procedure = "VelocityPrecond" "VelocityPrecond"
  Variable = "V"
  Variable DOFs = $d
  !  Potential commands to adjust the solution of velocity
  !  preconditioning problems:
   ...
End
```

```
Solver 2
  Equation = "Pressure Preconditioning"
  Procedure = "PressurePrecond" "PressurePrecond"
  Variable = "P"
  Variable DOFs = 1
  !  Potential commands to adjust the solution of pressure
  !  preconditioning problems:
   ...
End
```

The first solver section is needed for creating the velocity preconditioning system (30.8), while the second solver section corresponds to the pressure preconditioning system (30.7). Each of these sections may also contain additional keyword commands to change the default linear solver and its parameters. The default variable names `V` and `P` can be changed freely.

If a boundary condition $\boldsymbol{u} = \hat{\boldsymbol{u}}$ of Dirichlet type is specified, the variable of the velocity preconditioning equation δV (the Elmer variable `V` above) must also be constrained similarly by using the homogeneous Dirichlet condition. Usually there is no need to specify boundary conditions for the pressure preconditioning variable.

30.7 Examples

The solver described here has for example been applied to simulate flows of ice sheets. To obtain an example in this field, see the ISMIP HOM A test case

```
.../elmerfem/fem/tests/ParStokes_ISMIP_HOM_A010/
```

in the source code repository.

30.8 Keywords

`Material` `material-id`

`Density` `Real`
: This keyword is used to define the density ρ_0.

`Viscosity Model` `String`
: This keyword can be used to select a more advanced viscosity model such as Glen's law. For available options see the documentation of the standard Navier–Stokes solver.

`Viscosity` `Real`
: This keyword is used to define directly the viscosity μ.

`Constant-Viscosity Start` `Logical`
: In some cases it may be useful to start the nonlinear iteration with a constant viscosity although the intention is eventually to use another model. Starting with a constant viscosity can be avoided by giving the value `False` for this keyword.

`Solver` `solver-id`

`Equation` `String`
: This keyword declares the name of the equation.

`Procedure` `File ”ParStokes” ”StokesSolver”`
: The name of the file and procedure.

`Variable` `String`
: This keyword is used to declare the name of the solution.

`Variable DOFs` Integer
: The value of this keyword defines the number of unknown scalar fields and must hence equal to $d+1$ where d is the spatial dimensionality of the computational domain. The unknown scalar fields are always numbered in such a way that the highest running number is associated with the pressure solution.

`Convective` Logical
: If the value `True` is given, the convection term will be included so that the steady state version of the incompressible Navier–Stokes equations is solved.

`Nonlinear System Convergence Tolerance` Real
: This keyword defines the stopping criterion for the nonlinear iteration. The nonlinear iteration is terminated when the maximum number of nonlinear iterations is reached or when

$$\|b - K(x)x\|_2/\|b\|_2 < \varepsilon_N,$$

where ε_N is the value of this keyword.

`Nonlinear System Max Iterations` Integer
: This keyword defines the maximum number of nonlinear iterations.

`Nonlinear System Newton After Iterations` Integer
: If `n` is the value of this keyword, `n` Picard updates are performed before switching to Newton's method.

`Nonlinear System Newton After Tolerance` Real
: If the norm of the nonlinear residual is smaller than the value of this keyword, then the nonlinear iteration method is switched to Newton's method.

`P2-P1 Approximation` Logical
: This keyword can be used to select the P_2-P_1/Q_2-Q_1 approximation method. The finite element mesh must then contain second-order finite elements based on the Lagrange interpolation.

`Element` String
: If bubble functions are used as the stabilization strategy, then an element definition must be given to specify the number of bubble functions. The recommended choice is `"p:1 b:4"` for three dimensional simulations.

`Block Preconditioning` Logical
: If the block preconditioner is used, the value of this keyword must be `True`.

`Block Diagonal A` Logical
: This can be used to select the preconditioner matrix T to be the block diagonal approximation of the A-block.

`Use Velocity Laplacian` Logical
: This can be used to select T to be the scaled vector-Laplacian matrix. Then the keyword `Block Diagonal A` must also set to be `True`.

`Linear System Convergence Tolerance` Real
: When the block preconditioning is used, the value of this keyword defines the stopping criterion for the outer GCR method applied to (30.4). The iteration is terminated when

$$\|b - Kx^{(j)}\|_2/\|b\|_2 < \varepsilon_L,$$

where ε_L is the value of this keyword.

`Linear System Adaptive Tolerance` Logical
: The usage of adaptive stopping criteria can be activated with this keyword.

`Linear System Relative Tolerance` Real
: If the adaptive stopping criteria are employed, this keyword controls the solution accuracy for the linear systems (30.4) during the nonlinear iteration. The stopping tolerance ε_L for (30.4) is chosen to be $\varepsilon_L = \eta_R \cdot \eta_N^{(k)}$, with η_R the value of this keyword and $\eta_N^{(k)}$ the previous nonlinear error. If the value obtained in this way is smaller than the value of the keyword `Linear System Convergence Tolerance`, then this has no effect.

`Linear System Base Tolerance` Real
: The stopping tolerance for solving (30.4) can never be larger that the value of this keyword.

`Linear System Max Iterations` Integer
: When the block preconditioner is used, this keyword is used to define the maximum number of the outer GCR iterations applied to (30.4).

`Linear System GCR Restart` Integer
: The outer GCR iteration can be restarted after m iterations to avoid the increasing cost of the orthogonalization procedure. The value of this keyword specifies the parameter m. The default value is $m = 50$. Giving a larger value can be beneficial if convergence problems relating to the outer iteration are met.

`Body Force` bf-id

`Flow BodyForce i` Real
: This keyword is used to define the i's component of the body force vector $\boldsymbol{g}$.

`Boundary Condition` bc-id

`Surface Traction i` Real
: This keyword can be used to specify the components of the traction vector $\hat{\boldsymbol{s}}$. An alternate for the phrase `Surface Traction` is `Pressure`.

`Normal Surface Traction` Real
: This keyword can be used to specify a surface traction in the form $\hat{\boldsymbol{s}} = \hat{p}\boldsymbol{n}$ where $\hat{p}$ is the value of this keyword. An alternate for the phrase `Normal Surface Traction` is `External Pressure`.

`Slip Coefficient i` Real
: This defines the friction coefficient such that the friction force is proportional to the velocity in the direction `i`.

Model 31

Rotational Form of the Incompressible Navier–Stokes Equations

Module name: Stokes
Module subroutines: StokesSolver
Module authors: Mika Malinen
Document authors: Mika Malinen

31.1 Introduction

The basic incompressible flow solver of Elmer uses the standard formulation of the Navier–Stokes equations. This section describes an alternative solver based on the rotational form of the Navier–Stokes system. In addition, some iterative methods that utilize splitting strategies in the solution of the associated discrete problems are represented.

31.2 Field equations

Using the vector identity

$$(\vec{u}\cdot\nabla)\vec{u} = (\nabla\times\vec{u})\times\vec{u} + \frac{1}{2}\nabla(\vec{u}\cdot\vec{u}), \tag{31.1}$$

the Navier–Stokes system for incompressible Newtonian fluid may be written as

$$\begin{aligned} \rho\left(\frac{\partial\vec{u}}{\partial t} + (\nabla\times\vec{u})\times\vec{u}\right) - 2\mu\nabla\cdot\overline{\overline{\varepsilon}}(\vec{u}) + \nabla P &= \vec{b}, \\ \nabla\cdot\vec{u} &= 0, \end{aligned} \tag{31.2}$$

where $\overline{\overline{\varepsilon}}$ is the stretching tensor (2.4) and

$$P = p + \frac{1}{2}\rho\vec{u}\cdot\vec{u} \tag{31.3}$$

is the total (Bernoulli) pressure. The stress $\overline{\overline{\sigma}}$, which may be of interest especially near boundaries, can now be expressed as

$$\overline{\overline{\sigma}} = (-P + \frac{1}{2}\rho\vec{u}\cdot\vec{u})\overline{\overline{I}} + 2\mu\overline{\overline{\varepsilon}}(\vec{u}). \tag{31.4}$$

The system (31.2) provides an alternative starting point for finding discrete solutions. Thus, instead of approximating the conventional primitive variables $(\vec{u}, p)$, we here look for discrete solutions of $(\vec{u}, P)$. It should be noted that if the convection term is not taken into account the system (31.2) reduces to the (generalized) Stokes equations. The pressure variable then reduces to the standard pressure, i.e. $P = p$.

31.3 Boundary conditions

Either the normal velocity $\vec{u}\cdot\vec{n}$, with $\vec{n}$ the outward unit normal vector, or the normal surface force $\bar{\bar{\sigma}}\vec{n}\cdot\vec{n}$ can be prescribed on the boundary. The tangential boundary conditions can be handled systemically in a similar manner. Thus, if $\vec{t}$ is a tangent vector to the boundary, one may prescribe either the tangential velocity $\vec{u}\cdot\vec{t}$ or the tangential surface force $\bar{\bar{\sigma}}\vec{n}\cdot\vec{t}$.

A rather common way to define an outflow boundary condition for the Navier–Stokes equations is to impose the normal surface force condition

$$\bar{\bar{\sigma}}\vec{n}\cdot\vec{n} = 0, \tag{31.5}$$

which ensures the uniqueness of the pressure solution. This condition arises when the homogeneous natural boundary condition (do-nothing boundary condition) is imposed in the standard formulation of the Navier–Stokes equations. It should be noted, however, that the homogeneous natural boundary condition associated with the variational formulation of (31.2) can be written as

$$-P\vec{n} + 2\mu\bar{\bar{\varepsilon}}\vec{n} = \bar{\bar{\sigma}}\vec{n} - \frac{1}{2}\rho(\vec{u}\cdot\vec{u})\vec{n} = \vec{0}.$$

Thus, a distinction must here be made between the surface force boundary condition and the natural boundary condition. In the case of the rotational form, imposing the homogeneous natural boundary condition in the normal direction yields

$$\bar{\bar{\sigma}}\vec{n}\cdot\vec{n} = \frac{1}{2}\rho\vec{u}\cdot\vec{u},$$

which, except for the special case of irrotational steady flow of a non-viscous fluid, may be an artificial boundary condition. Nevertheless, the tangential natural boundary condition associated with the rotational form is equivalent to the condition of vanishing tangential surface force, i.e. $\bar{\bar{\sigma}}\vec{n}\cdot\vec{t} = 0$.

31.4 Linearization

The linearization of the equation of motion in (31.2) can be done by utilizing the Newton iteration. This iteration strategy is based on approximating the rotational convection term as

$$(\nabla\times\vec{u})\times\vec{u} \approx (\nabla\times\vec{u})\times\vec{a} + (\nabla\times\vec{a})\times\vec{u} - (\nabla\times\vec{a})\times\vec{a}$$

where $\vec{a}$ is the previous velocity iterate. In this connection, the nonlinear boundary condition corresponding to the outflow condition (31.5) is linearized as

$$-P + \frac{1}{2}\rho(2\vec{a}\cdot\vec{u} - \vec{a}\cdot\vec{a}) + 2\mu\bar{\bar{\varepsilon}}(\vec{u})\vec{n}\cdot\vec{n} = 0.$$

An alternative linearization strategy is to apply Picard's method. Here this method corresponds to linearizing the convection term and the outflow boundary condition as

$$(\nabla\times\vec{u})\times\vec{u} \approx (\nabla\times\vec{u})\times\vec{a}$$

and

$$-P + \frac{1}{2}\rho\vec{a}\cdot\vec{u} + 2\mu\bar{\bar{\varepsilon}}(\vec{u})\vec{n}\cdot\vec{n} = 0.$$

The convergence of the Newton method can be considerably faster than that of Picard's method. Our experience is that this can be the case, especially, when the steady solutions are sought for moderately large Reynolds numbers. However, a difficulty with the Newton method is that the iteration may not be convergent for arbitrary initial guesses. This trouble can often be avoided by performing some Picard updates before switching to Newton's method. In the case of time-accurate simulations this is usually unnecessary since suitably accurate initial guesses are often available from the previous time levels.

31.5 Discretization aspects

The solver is tailored to the case of the lowest-order continuous pressure approximation, but it does not provide any in-built technique to stabilize discrete solutions based on inherently unstable equal-order approximations of $(\vec{u}, P)$. The solver offers two strategies which can be used to obtain stable methods. First, one can use elements where the velocity approximation is augmented by using elementwise bubble functions. Second, one can utilize hierarchic versions of the second-order elements to rise the polynomial order of the velocity approximation. Both the strategies can be put into effect by utilizing the shape functions for p-elements. Some stable approximation methods are summarized as follows.

- *A hierarchic version of P_2-P_1 approximation for triangular and tetrahedral elements.* If the basic mesh consists of linear elements (element type 303 or 504), giving the element type definition `Element = "p:2"` in the Equation section switches to the P_2-P_1 approximation where the velocity approximation is enhanced by using hierarchic basis functions associated with the mid-edge nodes.
- *A hierarchic version of Q_2-Q_1 approximation for rectangular and brick elements.* Analogously to the previous case, if the basic mesh consists of bilinear or trilinear elements (element type 404 or 808), giving the element type definition `Element = "p:2"` in the Equation section switches to the Q_2-Q_1 approximation where the velocity approximation is enhanced by using hierarchic basis functions.
- *A hierarchic version of bubble-stabilized methods.* The velocity approximation may also be enhanced relative to the pressure by using elementwise bubble functions. The richness of velocity approximation depends on how many bubble basis functions are constructed. For example, the set of basis functions for the linear triangular and tetrahedral elements can be augmented with one interior bubble function by giving the element type definition `Element = "p:1 b:1"` in the Equation section. Analogous rectangular and brick elements may also be constructed, but our experience is that more than one bubble function may be necessary to obtain stability, making this strategy less attractive. The Q_2-Q_1 and P_2-P_1 approximation methods may generally require less computational work, especially for three-dimensional problems where the number of interior bubble functions can be large (notice that in the case of the time-dependent equations the interior degrees of freedom are not eliminated by using the method of static condensation).

It is noted that other instabilities may arise when the flow is convection-dominated. A potentially useful aspect of using the rotational formulation is that, as compared with the standard convection form, instabilities relating to dominating convection may be more benign.

31.6 Utilizing splitting strategies by preconditioning

Discrete Navier–Stokes problems lead usually to large linear systems which are customarily solved with iterative algorithms, in combination with preconditioning. The general preconditioning strategy used in Elmer is based on the computation of incomplete factorizations. The performance of these preconditioners is case-dependent and may not always be satisfactory.

More efficient solution algorithms for a particular problem can often be developed by exploiting the block structure of the linear system. In the following such a solution strategy will be described. Since the application of the preconditioner considered is based on solving certain simpler problems, the door is opened to utilizing other efficient methods, such as multigrid methods for the discrete Poisson problems, in connection with the solution of this more complicated problem.

The linearization and discretization of (31.2) leads to solving linear systems of the form

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} U \\ \Pi \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}, \tag{31.6}$$

where A is the coefficient matrix for the velocity unknowns and B is the divergence matrix. The solution strategy we consider is based on applying a preconditioned Krylov subspace method to (31.6). Given a previous iterate (U_k, Π_k), the preconditioning is performed via solving approximately systems of the form

$$\begin{bmatrix} A & 0 & 0 \\ B & M & 0 \\ B & H & S \end{bmatrix} \begin{bmatrix} U_{k+1} - U_k \\ \psi_{k+1} \\ \Pi_{k+1} - \Pi_k \end{bmatrix} = \begin{bmatrix} F - AU_k - B^T\Pi_k \\ -BU_k \\ -BU_k \end{bmatrix}. \tag{31.7}$$

Here M is the pressure mass matrix, while H and S are approximations of (scaled) Laplacian operators. In practice the approximate solution of this block triangular system is generated by applying linear solvers to systems with the coefficient matrices A, M and S. A crucial aspect of the methodology is that these subsidiary problems can be considerably easier to solve than the original fully coupled system. They may also be solved inexactly without impairing the performance of the preconditioner. Moreover, to our experience the performance of the preconditioner is insensitive to discretization parameters and depends only mildly on the Reynolds number, especially in the case of the evolutionary equations. The method is also suitable for finding the steady solutions via using large time step sizes.

The outer iterative method applied to the primary system (31.6) is based on GCR, while the user can specify linear solvers which are used to solve the subsidiary problems related to the preconditioning. It should be noted that boundary conditions associated with the preconditioning operators are built-in, so the user need not specify these constraints.

31.7 Restrictions

Currently, only homogeneous surface force conditions can be imposed on the boundary. If Q_2-Q_1 or P_2-$P1$ approximation is used, the boundary conditions are set by employing the linear interpolation of boundary data. As a result, optimal accuracy may not be realised. If the preconditioning is done via solving (31.7), the time discretization must be done using BDF(1) and viscosity should be constant.

If decoupled solution strategies are employed, parallel computations are possible only with the version that does not involve performing the outer Krylov iteration update.

31.8 Keywords

`Material` `material-id`

`Density` `Real`
: This keyword is used to define the density ρ.

`Viscosity` `Real`
: This keyword is used to define the viscosity μ.

`Solver` `solver-id`

`Equation` `String`
: This keyword declares the name of the equation.

`Procedure` `File "Stokes" "StokesSolver"`
: The name of the file and procedure.

`Variable` `String`
: This keyword is used to declare the name of the solution.

`Variable DOFs` `Integer`
: The value of this keyword defines the number of unknown scalar fields and must hence equal to $d + 1$ where d is the spatial dimensionality of the computational domain. The unknown scalar fields are always numbered in such a way that the highest running number is associated with the pressure solution.

`Convective` `Logical`
: If the value `"False"` is given, the convection term will be neglected so that the generalized Stokes equations are solved.

`Nonlinear Iteration Method` `String`
: This keyword defines the nonlinear iteration method. The default is the Newton method, and Picard's method can be chosen by giving the value `"Picard"`.

`Nonlinear System Convergence Tolerance` `Real`
: This keyword defines the stopping criterion for the nonlinear iteration. The nonlinear iteration is terminated when the maximum number of nonlinear iterations is reached or when

$$\left\| \begin{bmatrix} F - A(U_k)U_k - B^T\Pi_k \\ -BU_k \end{bmatrix} \right\| < TOL \left\| \begin{bmatrix} F \\ 0 \end{bmatrix} \right\|,$$

where TOL is the value of this keyword.

`Nonlinear System Max Iterations` `Integer`
: This keyword defines the maximum number of nonlinear iterations.

`Nonlinear System Newton After Iterations` `Integer`
: If `n` is the value of this keyword, `n` Picard updates are performed before switching to Newton's method.

`Nonlinear System Newton After Tolerance` `Real`
: If the norm of the nonlinear residual is smaller than the value of this keyword, then the nonlinear iteration method is switched to Newton's method.

`Nonlinear System Relaxation Factor` `Real`
: If this keyword is used, then the new nonlinear iterate is taken to be

$$(1-\lambda)(U_k, \Pi_k) + \lambda(U_{k-1}, \Pi_{k-1}),$$

where λ is the value of this keyword.

`Block Preconditioning` `Logical`
: If the block preconditioning via (31.7) is used, the value of this keyword must be `"True"`.

`Linear System Convergence Tolerance` `Real`
: When the block preconditioning is used, the value of this keyword defines the stopping criterion for the outer GCR method applied to (31.6).

`Linear System Max Iterations` `Integer`
: When the block preconditioning is used, this keyword is used to define the maximum number of the outer GCR iterations applied to (31.6). It should be noted that the GCR iteration requires that all previous iterates are saved. Especially in the case of time-accurate simulations the convergence of the preconditioned GCR method is expected to be rapid so that saving all the iterates is not expected be expensive. If the block preconditioning is used, the solver allocates computer memory based on the value of this keyword, so giving an exaggerated value should be avoided.

`Body Force` `bf-id`

`Body Force i` `Real`
: This keyword is used to define the i's component of the body force vector $\vec{b}$.

`Boundary Condition` `bc-id`

`Outflow boundary` `Logical`
: If the value `"True"` is given, then the normal outflow boundary condition (31.5) will be used. Note that this does not define the tangential boundary conditions which have to be specified separately.

If the preconditioning is done via solving (31.7), three additional solver sections need to be written to define linear solvers for subsidiary problems with the coefficient matrices A, M and S. In this connection special equation names (given as values of `Equation` keyword) have to be used. These solver sections should be written as follows.

```
Solver 1
  Equation = "Velocity Preconditioning"
  Procedure = "VelocityPrecond" "VelocityPrecond"
  Exec Solver = "before simulation"
  Variable Output = False
  Variable DOFs = $ d
  Variable = "VelocityCorrection"
  ...
End

Solver 2
  Equation = "Divergence Projection"
  Procedure = "DivProjection" "DivProjection"
  Exec Solver = "before simulation"
  Variable Output = False
  Variable DOFs = 1
  Variable = "DivField"
  ...
End

Solver 3
  Equation = "Pressure Preconditioning"
  Procedure = "PressurePrecond" "PressurePrecond"
  Exec Solver = "before simulation"
  Variable Output = False
  Variable DOFs = 1
  Variable = "PressureCorrection"
  ...
End
```

The first solver section defines a linear solver for the preconditioning system with the velocity matrix A, while the second solver section defines a solver for the system involving the pressure mass matrix M. Finally, the third section is related to the system with the coefficient matrix S arising from the discretization of the Laplacian operator. Each of these sections should also contain the standard keyword commands that actually define the linear solver. Some examples of such definitions can be found in the tests subdirectory of the fem module.

Part VI

Free Surfaces, Phase Change and Particle Dynamics

Model 32

Level-Set Method

Module name: LevelSet
Module subroutines: LevelSetSolver, LevelSetDistance, LevelSetIntegrate, LevelSetCurvature, LevelSet-Timestep
Module authors: Peter Råback, Juha Ruokolainen
Document authors: Peter Råback

32.1 Introduction

There are a number of problems involving free surfaces in continuum mechanics. There are two main strategies to solve them using the finite element method: Lagrangian and Eulerian approach. In the Lagrangian approach the free surface is solved exactly so that it is also an interface between the individual elements. This requires that the computational mesh is distorted in a way that this is possible. However, often the changes in geometry may be too drastic or even the whole topology may change and the Lagrangian approach is no longer feasible. The Eulerian approach describes the interface in a fixed mesh using some additional variable to describe the position of the interface. One possible Eulerian technique is the level-set method (LSM).

In the level-set method the free surface is given as a zero level-set of a higher dimensional variable. E.g. for 2D surfaces the level-set function is defined in 3D space. The level-set function is usually defined to be a signed distance so that inside the domain it obtains a positive value and outside a negative value. The changes in the value of the level-set function mean also that the interface changes the position.

This module includes several different subroutines that may be used when applying the level-set method. Currently there is no reinitialization strategy for 3D problems. Also some other procedures are not fully optimized for the best performance. Therefore the current implementation is best applied to quite simple 2D problems.

32.2 Theory

The interface is defined by a marker function ϕ so that at the interface $\phi = 0$, inside the fluid of interest $\phi > 0$ and elsewhere $\phi < 0$. The interface is update by solving the equation

$$\frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi = a \tag{32.1}$$

where $\vec{u}$ is the convection field and a is the normal flux on the interface. It is quite challenging to solve the differential equation above without diffusion effects playing a significant role. It is advisable to use 2nd order time-discretization schemes and short timesteps. More precisely, the Courant number $C = |\vec{u}|dt/h$ should be below unity.

It is desirable that the absolute value of function equals the shortest distance to the zero level-set. However, as the level-set function is advected this property may be gradually lost. Therefore a process called reinitialization may be evoked. In 2D the reinitialization may easily be done by geometric procedure. First the zero level-set is formed by going through all the elements and finding the line segments that make the zero level-set. Then the minimum distance of all the nodes is computed by a brute-force search. Assuming there are N nodes and M line segments, the search algorithm is $N \times M$ which is quite acceptable complexity for small cases but may become computationally costly in large cases.

The line segments may be assumed to go with the flow and thereby they form an on-the-fly Lagrangian mesh. Therefore it is also possible to advect the line segments when the velocity field is given since for any node $\vec{r} = \vec{r} + \vec{u}\,dt$. After the advection the shortest distance is computed. In the case of no advection the sign of the distance is inherited from the original level-set function. However, when the level-set is also convected the sign must be deduced from the geometric information as well. In the current implementation each line segment is given a flag telling on which side of the element the fluid of interest is located. This directional information is then used in giving the correct sign for the distance.

The volume of the fluid of interest in the level-set method may be computed over an integral that obtains a value one inside the fluid and value zero outside the fluid. The Heaviside function $H(\phi)$ has this desired property. However, as the interface does not follow the element division the numerical integration would result into spurious fluctuations depending on the position of the interface within the elements. To obtain a smooth behavior the Heaviside function must be regularized.

$$H_\alpha(x) = \begin{cases} 0, & x < -\alpha \\ f(\alpha/x) & |x| \leq \alpha \\ 1, & x > \alpha, \end{cases} \tag{32.2}$$

where the following has been implemented

$$f(t) = \frac{1}{2}\left(1 + \sin\left(t\frac{\pi}{2}\right)\right) \tag{32.3}$$

while one could also use

$$f(t) = \frac{3}{4}\left(t - t^3/3\right) + \frac{1}{2}. \tag{32.4}$$

Here α is the interface bandwidth which equals typically the size of a few elements. Now the volume (area in 2D) is obtained by the integral

$$V = \int_\Omega H_\alpha(\phi)\, d\Omega. \tag{32.5}$$

After the same regularization the area (length in 2D) may be obtained from the integral

$$A = \int_\Omega \delta_\alpha(\phi)|\nabla\phi|\, d\Omega \tag{32.6}$$

where the delta function is

$$\delta_\alpha(x) = \begin{cases} 0, & |x| > \alpha \\ \frac{1}{2\alpha}\cos\left(\frac{x}{a}\pi\right), & |x| \leq \alpha. \end{cases} \tag{32.7}$$

The information obtained by the above integrals may be used to improve the volume conservation of the level-set advection. If the initial volume V_0 is known the level-set function may be given a small correction by

$$d\phi = \frac{V_0 - V}{A}. \tag{32.8}$$

This correction has no physical basis but it may be argued that a consistently small update of the level-set function has a minor effect in overall results. It is more important that the volume is conserved since the history information of the shape of a bubble is gradually lost while the errors in volume are never forgotten. However, if the fluid of interest is divided into several parts this kind of overall correction does not have any justification since it could ruin the volume balance between the different domains.

The problems in accuracy may be partially resolved by using an optimal timestepping strategy. This may be achieved by looking at the velocity field around the active boundary. The normal velocity may be obtained by $u_n = \vec{u} \cdot \nabla\tilde{\phi}$. Registering the maximum velocity at band the timestep may be limited so that the Courant number is bound. If ds is the maximum allowed change in the position of the zero level-set the corresponding time-step is $dt = ds/\max|u_n|$.

In the Eulerian approach to the free surface problems the surface tension force must be smeared out to a volume force within a narrow band from the interface. The transformation is achieved by using a regularized delta function,

$$\int_\Gamma \sigma\kappa\, d\Gamma = \int_\Omega \sigma\kappa\delta(\phi)\nabla\phi\, d\Omega, \tag{32.9}$$

where σ is the surface tension coefficient and κ the curvature of the interface given by

$$\kappa = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}. \tag{32.10}$$

In the finite element approach the force cannot be estimated directly since it involves three derivatives of the level-set function. Therefore we must solve an additional equation for the curvature κ,

$$\kappa - c_\kappa \nabla^2 \kappa = \nabla \cdot \nabla\tilde{\phi}. \tag{32.11}$$

Here c_κ is an ad hoc diffusion coefficient that may be used to smooth the resulting curvature field. Otherwise the sharp corners may result to very large peak values of the curvature. The weak formulation of the above equation introduces surface fluxes which are evaluated from the normal derivatives of the level-set function. Once the level-set function and the corresponding curvature have been computed the surface tension may be applied as a volume force in the flow equations.

32.3 Keywords

LevelSetSolver

This subroutine uses the finite element method to solve the equation (32.1). The implementation is valid in 2D, 3D and axisymmetric problems.

`Solver` `solver id`

`Equation` `String "Level Set Solver"`

`Procedure` `File "LevelSet" "LevelSetSolver"`
The subroutine for advecting the level-set function.

`Variable` `String "Surface"`
The name of the level-set function. This may be chosen freely as long as it is used consistently elsewhere.

`Stabilize` `Logical`
Either stabilization or bubbles are used to solve the convection problem. This flag enforces the stabilization on.

`Material` `mat id`

`LevelSet Velocity i` `Real`
The velocity field that advects the level-set function. In 2D `i=1,2` and in 3D `i=1,2,3`. This may be a constant field or also something computed with the Navier-Stokes solver.

`Body Force` `bodyforce id`

`LevelSet Flux` `Real`
The flux (i.e. the normal velocity) of the level-set function.

LevelSetDistance

This solver uses the geometric information to compute the signed distance and, if desired, to advect the zero level-set at the same time. This solver does not solve an equation and hence it does not need to have a variable of its own. The solver is limited to 2D and axisymmetric cases.

`Solver` `solver id`

`Equation` `String "Level Set Distance"`

`Procedure` `File "LevelSet" "LevelSetDistance"`
: The subroutine for renormalizing (and advecting) the level-set function.

`LevelSet Variable` `String "Surface"`
: This keyword should refer to the name of the level-set variable that is used to advect the field. The default is `Surface`.

`Exported Variable 1` `String "Surface"`
: In case the level-set variable does not exist it must be introduced. This may be the case if this subroutine is also used for advecting the level-set function.

`LevelSet Convect` `Logical`
: Whether to also convect the level-set function. Default is `False`.

`Extract Interval` `Integer`
: When this function is used to extract the zero level-set function the user may choose the interval how often this is done. The default is one. Just extracting the level-set may be useful if one just wants to save the zero level-set without activating reinitialization.

`Reinitialize Interval` `Integer`
: When this function is used to reinitialize the level-set function the user may choose the interval how often this is done. The default is one but often this results to excessive smoothing of the level-set field. If reinitialization is asked the zero level-set will also be automatically extracted.

`Reinitialize Passive` `Logical`
: If this keyword is set `True` the reinitialization is not applied to the level-set field. The field is only used to extract the zero level-set and compute the corresponding signed distance but this information is not used to change the original field.

`Narrow Band` `Real`
: In case that also the convecting is done by this solver there is the possibility to introduce a narrow band which gives the distance at within the level-set function is recomputed. Default is ∞. Typically this should be larger that the level-set bandwidth α used to evaluate surface integrals.

`Filename` `File`
: The zero level-set may also be saved. It consists of a number of line segments that are defined elementwise. The results from the file may be used for visualization, for example, in MatLab. If no filename is given the zero level-set is not saved.

`File Append` `Logical`
: If the above is given this flag enforces the results to be appended on the same file rather than writing over the old results.

`Material` `mat id`

`LevelSet Velocity 1` `Real`

`LevelSet Velocity 2` `Real`
: If also convection is accounted in this solver the convection field is given by the above expressions. Currently it is not possible to give the desired surface flux as it is not uniquely defined for the line segments having different normals even at the same point.

LevelSetIntegrate

This subroutine computes the integrals (32.5) and (32.6). In addition of computing volume and surface integrals this subroutine may also be used to set the absolute level of the level-set function so that volume is conserved using equation (32.8). The implementation is valid in 2D, 3D and axisymmetric problems.

`Solver` `solver id`

- `Equation` `String Level Set Integrate`
- `Procedure` `File "LevelSet" "LevelSetIntegrate"`
  The subroutine for computing the integrals.
- `LevelSet Variable` `String "Surface"`
  This keyword gives the name of the level-set function used for computing the integrals. The default is `Surface`.
- `LevelSet Bandwidth` `Real`
  When computing the values over the domain the interface is treated a with smooth functions. How smooth the functions are depends on the value of this keyword. Typically the bandwidth should be such that the interface is extended over a few elements.
- `Conserve Volume` `Logical`
  The volume in the level-set formulation is not conserved by construction. To that end the level of the level-set function may be tuned so that conservation is enforced. The default is `False`.
- `Conserve Volume Relaxation` `Real`
  If conservation is enforced it may be done only partially as there are inaccuracies in the evaluation of the volume integrals. The default is one.
- `Initial Volume` `Real`
  If conservation is enforced the target volume is given by this keyword. Otherwise the volume from the first timestep is used as the target value.

LevelSetCurvature

This solver computes the value of the curvature give the level-set function using equation (32.11).

`Solver` `solver id`

- `Equation` `String Level Set Curvature`
- `Procedure` `File "LevelSet" "LevelSetCurvature"`
  The subroutine for computing the curvature.
- `Variable` `String "Curvature"`
  The name of the curvature variable.
- `LevelSet Variable` `String "Surface"`
  This keyword gives the name of the level-set function used for computing the integrals. The default is `Surface`.
- `Curvature Diffusion` `Real`
  Artificial diffusion may be used to control the singularities of the curvature field around sharp corners. The default is zero.
- `Curvature Coefficient` `Real`
  A constant that is used to multiply the curvature field before the solver is exited. This may be used for example to change the sign of the curvature if the material of interest is on the outside and not an the inside.
- `LevelSet Bandwidth` `Real`
  The delta function for the volume force may be applied to the curvature field also within this solver directly. This has the disadvantage that the evaluation is done at nodal points rather than

at the integration points. However, if the flow solver used may not be modified this may be the best alternative. If this keyword does not exist, no delta function is used to filter the curvature field.

`Boundary Condition` `bc id`

`Levelset Curvature BC` `Logical`
: The weak formulation of the curvature computation results to boundary integrals that should be set at all surfaces where the curvature is computed.

LevelSetTimestep

The solution of the level-set function is accurate only if the timestep is limited so that the local Courant number along the zero level-set is in the order of one or smaller. A tailored function for setting the timestep is given in this module. This solver assumes that the level-set variable is named `Surface` and that this variable is related to some solver. The velocity needed for setting the timestep should be given by the keywords `LevelSet Velocity i`, where `i=1,2,3`.

`Simulation`
: The function call and the needed parameters reside in the `Simulation` block of the command file.

```
Timestep Function
  Real Procedure "LevelSet" "LevelSetTimestep"
```

`LevelSet Courant Number` `Real`
: This keyword gives the desired Courant number of for the level-set solvers. The default for the desired Courant number is one.

`LevelSet Timestep Directional` `Logical`
: If the timestep limit is active this option may be used to account only the normal direction of the interface velocity rather that the absolute direction. Default is `False`.

Other solvers

Basically the user may give user defined material parameters where the values are computed as a function of the level-set function. Unfortunately this approach generally uses nodal points for the smearing whereas it is optimal to use the Gaussian integration points for doing this. There is one exception to this model that has been implemented for the `MaterialModels` module, namely the viscosity may be computed at Gaussian integration points.

`Material` `mat id`

`Viscosity Model` `String levelset`
: This uses the level-set methodology to smear out the viscosity between inside and outside values.

`Viscosity` `Real`
: The value of the viscosity outside the domain (negative level-set function values).

`Viscosity Difference` `Real`
: The difference between the inside and outside viscosity values.

`Levelset bandwidth` `Real`
: The bandwidth at which the viscosity is smeared out between the extreme values.

Model 33

Kinematic Free Surface Equation with Limiters

Module name: FreeSurfaceSolver
Module subroutines: FreeSurfaceSolver
Module authors: Thomas Zwinger, Peter Råback, Juha Ruokolainen, Mikko Lyly
Document authors: Thomas Zwinger

33.1 Introduction

Flows with a free surface are to be found in geophysical as well as technical applications. On large scale flows the free surface usually is governed by a kinematic boundary condition given as a partial differential equation. This equation then is solved on the specific boundary in combination with the (Navier)-Stokes equation and the mesh update solver.

33.2 Theory

The implicit equation describing the free surface is given by

$$F(\vec{x}, t) = z - h(x, y, t), \tag{33.1}$$

with the explicit position of the free surface $h(x, y, t)$. Mass conservation implies that, with respect to the velocity of the surface, $\vec{u}_{\mathrm{m}}$, F has to define a substantial surface, i.e.,

$$\frac{\partial F}{\partial t} + \vec{u}_{\mathrm{m}} \nabla F = 0. \tag{33.2}$$

The net volume flux through the free surface then is given by the projection of the difference between the fluid velocity at the free surface, $\vec{u}$ and the velocity of the free surface with respect to the surface normal

$$a_{\perp} = (\vec{u}_{\mathrm{m}} - \vec{u}) \cdot \vec{n}. \tag{33.3}$$

In Geophysical context (e.g., Glaciology), $a_{\perp}$ often is referred to as the net accumulation. With the surface unit normal defined as

$$\vec{n} = \frac{\nabla F}{\|\nabla F\|}, \tag{33.4}$$

this leads to

$$\frac{\partial F}{\partial t} + \vec{u} \nabla F = -\|\nabla F\| a_{\perp}. \tag{33.5}$$

Using the definition in (33.1), (33.5) can be rewritten in its explicit form

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} - w = \left[1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right]^{1/2} a_\perp, \tag{33.6}$$

with the components of fluid velocity vector at the free surface given as $\vec{u} = (u,\, v,\, w)^{\mathrm{T}}$. The variational formulation of (33.6) reads as

$$\int\limits_\Omega \left(\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y}\right) \varphi\, dV = \int\limits_\Omega \left\{ w + \left[1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right]^{1/2} a_\perp \right\} \varphi\, dV, \tag{33.7}$$

where the occurrence of h in the right-hand side is inserted from the previous time-step/non-linear iteration, hence linearizing the equation. In case of a horizontally moving mesh, the contribution in form of an arbitrary Lagrangian-Eulerian (ALE) formulation has to be included (by default is is omitted). With the horizontal mesh velocity components, u_{mesh} and v_{mesh}, the ALE version of equation (33.6) then reads

$$\frac{\partial h}{\partial t} + (u - u_{\text{mesh}})\frac{\partial h}{\partial x} + (v - v_{\text{mesh}})\frac{\partial h}{\partial y} - w = \left[1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right]^{1/2} a_\perp, \tag{33.8}$$

33.2.1 Limiters

In certain cases the free surface is constrained by an upper $h_{\text{max}}(x, y, t)$ and/or a lower $h_{\text{min}}(x, y, t)$ limit. For instance, the free surface of a fluid contained in a vessel cannot penetrate the vessel's walls. This adds the constraint

$$h_{\text{min}} \leq h \leq h_{\text{max}} \tag{33.9}$$

to (33.7) converting the variational formulation into a variational inequality. In order to obtain a with (33.9) consistent solution a method using Dirichlet constraints within the domain is applied. The exact procedure is the following:

1. construct the linear system: $Ah = f$, with the system matrix A and the solution vector h on the left-hand side and the force vector f on the right-hand side
2. set nodes as *active* if (33.9) is violated
3. for *active* nodes the matrix and force vector are manipulated such that effectively a Dirichlet condition $h = h_{\text{max/min}}$ is applied
4. the manipulated system is solved: $\tilde{A}\tilde{h} = \tilde{f}$
5. a residual is obtained from the un-manipulated system: $R = A\tilde{h} - f$
6. an *active* node is reset if the residual is $R < 0$ (for lower limit) and $R > 0$ (for upper limit)

The whole algorithm is iterated (within the non-linear iteration loop) until the limit given in `Nonlinear System Convergence Tolerance` is reached. In the converged solution the residual represents the needed accumulation/volume flux (on matrix level, hence not in physical units) needed in order to obtain the limited solution. Consequently, the system not necessarily is volume conserving if the Dirichlet method is applied. As the solver in principle works with second order elements, the limitation procedure only converges with only the between elements shared nodes being subject to the algorithm described in this section. This is done automatically by the code.

33.3 Constraints

The code only works in Cartesian coordinates and – by the nature of the differential equation – effectively converges only in a transient simulation. Although, technically, it also can be run in steady state simulations.

33.4 Keywords

`Solver` `solver id`

`Equation` `String "Free Surface Limited"`

`Variable` `String Varname`
The change in the free surface coordinate. This may be of any name as far as it is used consistently also elsewhere, as `Varname` is used as a preceding keyword for the exported variable of the residual, as well as for the accumulation

`Variable DOFs` `Integer 1`
Degrees of freedom for the free surface coordinate.

`Procedure` `File "FreeSurfaceSolver" "FreeSurfaceSolver"`
The following four keywords are used for output control.

`Velocity Implicitness` `Real`
Determines the level of implicitness in the velocity field. Values shall be in the interval $c_v \in [0, 1]$. The velocity is interpolated between the current and the previous time level such that $u = (1 - c_v)\, u^{n-1} + c_v\, u^n$. Thus, unity corresponds to complete implicitness (default).

`Maximum Displacement` `Real`
This limits the maximal local displacement in a time-step. If exceeded, relaxation automatically is applied in order to limit the displacement.

`Apply Dirichlet` `Logical`
Takes the variational inequality method (here referred to as Dirichlet method) into use. The user should be aware that if the method is applied (value `True`) this implies setting the `Nonlinear Max Iterations` to a value large enough for the method to converge. The default value is `False`.

`ALE Formulation` `Logical`
If set to `True`, the mesh horizontal mesh velocity is taken into account in the convection term. The default value is `False`.

`Relaxation Factor` `Real`
The changes in the free surface may be relaxed. The default is no relaxation or value 1.0

`Stabilization Method` `String`
Sets stabilization method. Either `Stabilized` or `Bubbles` can be set.

`Nonlinear System Convergence Tolerance` `Real`
This keyword gives a criterion to terminate the nonlinear iteration after the maximum change in the free surface coordinate is small enough

$$\max ||dR/(R - R_0)|| < \epsilon$$

where ϵ is the value given with this keyword.

`Exported Variable 1` `String`
The residual, which is the essential property in solving the variational inequality has to be given as an exported variable. The name is fixed by the variable name `Varname` given in the Solver section plus `Residual`. For instance, if the variable is named `FreeSurf`, the exported variable is expected to be `FreeSurf Residual`.

`Exported Variable 1 DOFs` `Integer`
As the free surface is a scalar, the value has to be set to 1.

`Use Linear Elements` `Logical`
If set to true, forces usage of linear element types despite the order of the mesh. Mind, that in case of limited elements, by default linear elements are used. The default value is `False`.

`Equation` `eq id`

`Convection` `String`
: The type of convection to be used: `None` (default), `Computed`, `Constant`. In the last case, the keyword `Convection Velocity` is expected to be found in the Material section.

`Body Force` `bf id`

`Varname Accumulation` `Real`
: sets the value for the normal accumulation/volume flux, $a_\perp$ for the variable name `varname`. If this keyword is set, the following keyword `Varname Accumulation Flux` is ignored (as those are excluding)

`Varname Accumulation Flux i` `Real`
: sets the accumulation flux in Cartesian components (`i` = 1,2,3 in 3-dimensional problem). The resulting vertical flux then is evaluated using the surface normal.

`Initial Condition` `ic id`

`Varname` `Real`
: Initiation of the free surface variable (sets initial shape of surface)

`Boundary Condition` `bc id`

`Body ID` `Integer`
: usually, the solver is run on a lower dimensional boundary of the model. Then a separate body-id has to be defined and all component of the solver (`Equation`, `Body Force`, `Equation`, `Initial Condition` and `Material`) defined accordingly.

`Varname` `Real`
: Dirichlet condition of the free surface variable (makes really sense only on dimension - 2 boundaries, e.g. lines in case of a three dimensional run)

`Mesh Update i` `Real`
: usually, the free surface evolution should have a feedback on the domain's geometry. This usually is achieved by running the MeshUpdate Solver and linking the variable of the free surface with the corresponding component of the `Mesh Update` (i=1,2,3). For instance, in a 3-dimensional case with the variable name `FreeSurf` this could read as: `Mesh Update 3 = Equals FreeSurf`

Model 34

Free Surface with Constant Flux

Module name: FreeSurfaceReduced
Module subroutines: FreeSurfaceReduced
Module authors: Peter Råback
Document authors: Peter Råback

34.1 Introduction

The determination of free surface is often an essential part of solving a fluid dynamics problem. Usually the surface is found by solving a free surface equation resulting from force balance, or by finding the free surface from zero flux condition. In some extreme cases both of these methods were found to fail and therefore an alternative approach was taken. The method can only be applied to stationary 2D or axisymmetric flows where the total flux is conserved. This is the case, for example, in many coating and drawing processes.

34.2 Theory

The determination of the free surface takes use of the conservation of mass. If the flow is stationary the mass flux through all planes cutting the flow must be same. In the following we concentrate on the axisymmetric case which has more applications than the 2D case.

In the axisymmetric case the mass flux is obtained from

$$f(R,z) = \int_{R_0}^{R} (\vec{u} \cdot \vec{n})\, r ds. \tag{34.1}$$

The free surface is set by finding a surface profile $R(z)$ such that the integral is constant for all nodes on the surface, or

$$f(R, z_j) = f(R_1, z_1) \quad \forall j \in [1, M]. \tag{34.2}$$

Note that the factor 2π has been consistently omitted since it has no bearing to the shape of the free surface.

The subroutine uses simple heuristics to determine the direction of the flow on the free surface. The first upwind node z_1 on the free surface is assumed to be fixed and the corresponding flux is f_1. The new radius is set approximately by assuming that the added or removed flow has the same velocity as the velocity on the surface. Then the corrected radius is found from

$$u_n R^{(m)}\, dR^m = f(R^{(m)}, z) - f(R_1, z_1) \tag{34.3}$$

or

$$R^{(m+1)} = R^{(m)} + \frac{f(R^{(m)}, z) - f(R_1, z_1)}{u_n R^{(m)}}. \tag{34.4}$$

After the new profile is being found the element nodes are moved to the new positions. The nodes that are not on the surface may be mapped in many different ways. The straight-forward strategy is to use linear 1D mapping. Also more generic 2D mapping may be used.

The free surface and the fluid flow must be consistent and therefore the system must be solved iteratively. When convergence of the coupled system has been obtained the suggested dR vanishes and the free surface solver does not affect the solution.

Sometimes the free surface solver overshoots and therefore it may be necessary to use relaxation to suppress the large changes of the solution.

Note that the free surface solver is simple based on mass conservation. No forces are applied on the free surface. If surface tension needs to be taken into account it may be done while solving the Navier-Stoke equation.

34.3 Applicable cases and limitations

The method has some limitations which are inherent of the method:

- Limited to steady-state simulations.
- Limited to 2D and axisymmetric cases.
- If there is back-flow within the free surface flow the correctness of the solution is not guaranteed.

Some limitations result from the current implementation:

- The free surface must be oriented so that the flow is on its negative side.
- There may be several free surfaces of this type but they must be directed the same way.
- The line integral from R_0 to R may cause some difficulties in unstructured meshes. Therefore structured meshes are favored.
- At the moment density is assumed to be constant and therefore only incompressible fluids may be considered.

34.4 Keywords

`Solver` `solver id`

> `Equation` `String "Free Surface Reduced"`
>
> `Variable` `String dx`
> The change in the free surface coordinate. This may be of any name as far as it is used consistently also elsewhere.
>
> `Variable DOFs` `Integer 1`
> Degrees of freedom for the free surface coordinate.
>
> `Procedure` `File "FreeSurfaceReduced" "FreeSurfaceReduced"`
> The following four keywords are used for output control.
>
> `Perform Mapping` `Logical`
> If this keyword is `True` the coordinate mapping is done locally by using linear 1D mapping. This is also the default. Also 2D mapping is possible by using a separate mesh update solver. Then the keyword should be set to `False`.
>
> `Nonlinear System Relaxation Factor` `Real`
> The changes in the free surface may be relaxed. The default is no relaxation or value 1.0

`Nonlinear System Convergence Tolerance` Real
: This keyword gives a criterion to terminate the nonlinear iteration after the maximum change in the free surface coordinate is small enough

$$\max ||dR/(R - R_0)|| < \epsilon$$

where ϵ is the value given with this keyword.

`Boundary Condition` bc id

`Free Surface Reduced` Logical
: Must be set to `True` for the free surface when the solver is used. The boundary must be simply continuous.

`Free Surface Number` Integer
: If more than one free surface of the reduced type is present simultaneously they must somehow be separated. This keyword is for that purpose. The surfaces should be ordered from 1 to the number or free surfaces. Value 1 is also the default if the surface is active. Note that free surfaces with different numbers should be aligned the same way and should not touch each other.

`Free Surface Bottom` Logical
: If this flag is free it sets the lower boundaries of integration when solving for the free surface. Note that this surface should not touch any of the free surfaces. A free surface is automatically a lower boundary for another free surface.

If mapping is not performed within the solver also boundary conditions for the mapping are required. Surface tension may be taken into account while solving the Navier-Stokes equation. The proper keywords for activating the surface tension are explained in the manual of the Navier-Stokes solver.

Model 35

Transient Phase Change Solver

Module name: PhaseChangeSolve
Module subroutines: TransientPhaseChange
Module authors: Peter Råback
Document authors: Peter Råback

35.1 Introduction

There are many phenomena that involve an interface between liquid and solid phase. Such problems occur, for example, in crystal growth and casting processes. This subroutine defines the position of the phase change boundary in a transient case using an Lagrangian approach.

For Lagrangian steady state phase change algorithm look at the next chapter. For Eulerian phase change algorithm look at the enthalpy method in the heat solver. Generally Lagrangian approaches are more accurate but their use is limited to rather smooth interfaces with moderate displacements.

35.2 Theory

General theory

The phase change from solid to liquid occurs at the melting point T_m. At the boundary the temperatures of the liquid and solid are therefore equal to that. The phase change results to a change in the internal energy known as the latent heat L.

The latent heat makes the diffusive heat flux over the boundary discontinuous and results to the so called Stefan condition

$$L\rho\vec{v}\cdot\vec{n} = (\kappa_s\nabla T_s - \kappa_l\nabla T_l)\cdot\vec{n}, \tag{35.1}$$

where $\vec{n}$ is the normal of the phase change boundary, $\vec{v}$ is the velocity of the phase change boundary, ρ is the density of the solid and T_s and T_l are the temperatures of the solid and liquid phases, and κ_s and κ_l are the thermal conductivities, respectively.

In steady state pulling and drawing processes the velocity of the phase change boundary should be equal to pull velocity, $\vec{v} = \vec{V}$ (bulk velocity of the solid phase).

Transient algorithm

In transient phase change problems the interface temperature is set to be at the melting point when solving the heat equation. From the solution a heat flux is then obtained from

$$\vec{q} = \kappa_s\nabla T_s - \kappa_l\nabla T_l. \tag{35.2}$$

Now this heat flux is assumed to be used for the melting of the solid phase into liquid phase. Assuming that the phase change boundary is mapped to the new position moving it only in the y-direction we get from equation (35.1) the velocity in the y-direction,

$$\rho L n_y (v_y - D_v \nabla^2 v_y) = \vec{q} \cdot \vec{n}. \tag{35.3}$$

Here an artificial diffusion D_v has been added since the algorithm otherwise is prone to numerical oscillations. In order for the diffusion not to affect the results significantly it must fulfill the condition $D_v << h^2$ where h is the size of the 1D elements.

The corresponding displacement is easily obtained from multiplication $u_y = v_y\, dt$, where dt is the timestep. However, in the current formulation may also be done using the Galerkin method to include the possibility of an additional diffusion factor. Therefore the equation is of the form,

$$\frac{\partial u_y}{\partial t} - D_u \nabla^2 u_y = v_y. \tag{35.4}$$

In continuous processes the triple point may be used to define the pull velocity so that at the point the solution of the equation vanishes. In case the pull occurs in the y-direction this means that $V_y = v_y$.

The algorithm is ideally suited for relatively small time-steps where the change in the position is small compared to the other dimensions of the problem. Otherwise the transient algorithm may result to spurious oscillations. However, often the timestep size is most severely limited by the flow computations. Therefore it may be possible to boost the convergence towards the true operation regime by multiplying the suggested change by a constant factor.

35.3 Applicable cases and limitations

The method has some limitations which are described below

- Phase change surface must be nearly aligned with either of the main axis. To be more precise the boundary must in all instances be such that for each coordinate there is only one point on the boundary.
- Applicable in 2D and 3D cases
- Melting point and density over the interface may vary is assumed to be constant over the whole interface.
- The convection velocity of the interface should be constant.
- It should be noted that the solver only gives the position of the phase change boundary. In order to modify the whole geometry a mesh update solver must be applied.

35.4 Keywords

`Solver` `solver id`

`Equation` `String "Transient Phase Change"`

`Procedure` `File "TransientPhaseChange" "TransientPhaseChange"`
The subroutine that performs the phase change analysis.

`Variable` `String PhaseSurface`
The variable for the PhaseSurface coordinate. This may be of any name as far as it is used consistently also elsewhere.

`Variable DOFs` `Integer 1`
Degrees of freedom for the free surface coordinate, the default.

`Phase Change Variable` `String`
By default the phase change analysis uses `Temperature` as the active variable. The analysis may be performed also to any other scalar variable given by this keyword

`Use Nodal Loads` `Logical`
: The most accurate method of computing the heat fluxes is to use the residual of the matrix equation. This is activated by the keyword `Calculate Loads` in the heat equation and it results to a variable named `Temperature Loads` that may be used directly to give the melting heat over the interface nodes.

`Normal Variable` `String`
: The normal of an element may be computed directly from each element segment, or it may be computed using Galerkin method in the `NormalSolver`. In the latter case the name of the normal field variable may be given by this keyword.

`Triple Point Fixed` `Logical`
: This keyword enforces the triple point to be fixed. Depending on the type of algorithm this may mean different things. In the transient algorithm this means that the interface velocity is tuned so that the velocity at the triple point is zero. Only applicable in 2D where the triple point is unique.

`Pull Rate Control` `Logical`
: The pull rate may be set so that the triple point remains at a fixed position. The feature is activated setting this keyword `True`.

`Velocity Relaxation Factor` `Real`
: The relaxation factor for the interface velocity field.

`Velocity Smoothing Factor` `Real`
: The velocity diffusion factor of the interface, D_v.

`Transient Speedup` `Real`
: The factor at which the change in the boundary position is changed in the transient case. This may be used to speedup the transient convergence.

`Nonlinear System Max Iterations` `Integer`
: In case the pull-rate control is used the phase change algorithm may have to be solved several times in order to define the consistent pull-rate. This keyword gives the maximum number of iterations.

`Nonlinear System Convergence Tolerance` `Real`
: The tolerance for terminating the transient algorithm.

`Body` `body id`

`Solid` `Logical`

`Liquid` `Logical`
: The solver requires information on which of the materials in the system is solid and which is liquid. Currently the solver assumes that both the liquid and solid is uniquely defined.

`Material` `mat id`

`Heat Conductivity` `Real`
: In a transient case the heat conductivities of the materials must be given.

`Density` `Real`
: Density is needed to obtain the latent heat in units of energy per volume.

`Latent Heat` `Real`
: The latent heat is the specific internal energy related to the phase change. The latent heat may also be a variable.

`Convection Velocity i` `Real`
: For the transient algorithm the pull velocity of the boundary may be given with this keyword.

`Boundary Condition` `bc id`

`Body Id` Integer
: The phase change solver operates usually on a boundary of a two-dimensional domain. Technically the equation on the boundary is treated in a normal finite element manner and therefore the boundary must be defined to be the body where the equation is to be solved. Usually this would be the next free integer in the list of bodies.

`Phase Change Side` Logical
: This keyword is used for the boundaries that define the edges of the phase change interface. The diffusive operators used for smoothing create a weak term in the Galerkin formulation that must be cancelled. When this flag is active the weak terms are not assembled at all for the boundary thus eliminating the need to cancel them.

Model 36

Steady State Phase Change Solver

Module name: PhaseChangeSolve
Module subroutines: SteadyPhaseChange
Module authors: Peter Råback and Juha Ruokolainen
Document authors: Peter Råback

36.1 Introduction

There are many phenomena that involve an interface between liquid and solid phase. Such problems occur, for example, in crystal growth and casting processes. This subroutine defines the position of the phase change boundary by finding the correct isotherm in a steady state simulation. The mesh is the correspondingly mapped i.e. this is a Lagrangian approach.

For transient phase change algorithms look at the next chapter. For Eulerian phase change algorithm no additional solver is required as the phase change is implemented within the heat equation of Elmer by using the enthalpy method. Generally Lagrangian approaches are more accurate but their use is limited to rather smooth interfaces with moderate displacements.

36.2 Theory

For the general theory on phase change look at the Theory section of the previous chapter.

Steady state algorithm

In steady state the algorithm is based mainly on geometrical ideas. First the heat equation for temperature T is solved by using a flux condition for the interface

$$q = L\rho\vec{V}\cdot\vec{n}. \tag{36.1}$$

Thereafter the next approximation for the phase change surface may be found by going trough each element and creating a list of line segments E_j on the isosurface. This is basically the zero level-set of the field $T - T_m$. Each line segment is defined by two coordinate $\vec{x}_{j,1}$ and $\vec{x}_{j,2}$. The surface is then updated by mapping the current phase change surface to the line segments. For the moment a N^2 algorithm is used for the mapping. For larger cases a more robust search algorithm might be implemented.

For example, if a free surface is almost aligned along the x-axis, then for a node (x_i, y_i) on the boundary the proposed change of the point i in the y-direction is

$$s_y = (y_{j,1} - y_i) + (x_i - x_{j,1})\frac{y_{j,2} - y_{i,1}}{x_{j,2} - x_{j,1}} \tag{36.2}$$

assuming that $x_i \in [x_{j,1}, x_{j,2}]$ while $s_x = 0$.

Speeding up the convergence

In many cases the simple geometrical search algorithm converges very slowly. The reason is the explicit character of the algorithm that fails to account for the change in the temperature field caused by the moving phase change boundary. This limitation may be partially overcome using suitable under- or over-relaxation. This relaxation parameter may also be tuned during the iteration using lumped quantities such as the proposed change in the volume of the phases that may be expressed as

$$U = \int_A \vec{s} \cdot \vec{n} \, dA. \tag{36.3}$$

The proposed volume changes form a series, $U^{(0)}, U^{(1)}, \ldots, U^{(m-1)}, U^{(m)}$. Assuming that the series is a geometric one we may estimate the required relaxation factor that would give the correct phase change boundary at just one iteration,

$$c^{(m)} = c^{(m-1)} \frac{U^{(m-1)}}{U^{(m-1)} - U^{(m)}}. \tag{36.4}$$

In numerical tests this formula was found occasionally to overshoot and therefore a less aggressive version is used instead,

$$c^{(m)} = c^{(m-1)} \frac{1}{2} \frac{U^{(m-1)} + U^{(m)}}{U^{(m-1)} - U^{(m)}}. \tag{36.5}$$

The use of the lumped model requires that the temperature field is described accurately enough. To ensure numerical stability the factor c should have a upper and lower limits. After the factor has been defined the suggested displacements are simply scaled with it, $\vec{s}' = c\vec{s}$.

It is also possible to accelerate the solution locally using a Newton kind of iteration. If the basic algorithm has already been applied at least twice we may estimate the sensitivity of the local temperature to the moving interface and using this information to estimate a new change,

$$s^{(m)} = \frac{T_m - T^{(m)}}{T^{(m)} - T^{(m-1)}} s^{(m-1)}. \tag{36.6}$$

This algorithm might be a better option if the phase change surface is such that there is not much correlation between the displacements at the extreme ends. However, the algorithm may be singular if the isotherms of consecutive iterations cross. Any point i where $T_i^{(m-1)} \approx T^{(m)}$ leads to problems that may be difficult to manage. This handicap may rarely limit the usability of the otherwise robust and effective scheme.

36.3 Applicable cases and limitations

The method has some limitations which are described below

- Limited to steady state cases
- Limited to 2D and axisymmetric cases.
- Phase change surface must be nearly aligned with either of the main axis. To be more precise the boundary must in all instances be such that for each coordinate there is only one point on the boundary.
- Melting point is assumed to be constant over boundary (not concentration dependent, for example).
- It should be noted that the solver only gives the position of the phase change boundary.
- When internal mesh update is used the mesh is distorted only in one coordinate direction.

36.4 Keywords

`Solver` `solver id`

It is worth noting that for this solver the problem is, or at least could be, solved accurately. All the nonlinearities of the problem reside in the coupling with the heat equation. Hence, there is no point in giving criteria on the nonlinear system level. Only the coupled level tells whether the system has truly converged.

`Equation` `String "Steady Phase Change"`

`Procedure` `File "SteadyPhaseChange" "SteadyPhaseChange"`

The subroutine that performs the phase change analysis.

`Variable` `String Surface`

The variable for the PhaseSurface coordinate. This may be of any name as far as it is used consistently also elsewhere.

`Variable DOFs` `Integer 1`

Degrees of freedom for the free surface coordinate, the default.

`Phase Change Variable` `String`

By default the phase change analysis uses `Temperature` as the active variable. The analysis may be performed also to any other scalar variable given by this keyword

`Nonlinear System Relaxation Factor` `Real`

Giving this keyword triggers the use of relaxation in the phase change solver. Using a factor below unity may sometimes be required to achieve convergence. Relaxed phase change variable is defined as follows:

$$u_i' = u_i + \lambda s_{i-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity. If using the lumped model to accelerate the solution the final relaxation factor will the product of the two.

`Nonlinear System Newton After Iterations` `Integer`

The local Newton type of iteration may be set active after a number of iterations given by this keyword.

`Nonlinear System Newton After Tolerance` `Real`

The Newton type of iteration may also be activated after a sufficiently small change in the norm. This keyword gives the limit after which Newton iteration is triggered on.

`Lumped Acceleration After Iterations` `Integer`

The phase change solver may be accelerated pointwise, or by using a lumped model to determine an optimal relaxation factor for the whole solution. This keyword activates the lumped model procedure.

`Lumped Acceleration Mode` `Integer`

This helps to toggle between different versions of the lumped acceleration. The options include values 0,1,2,3 where 0 is also the default.

`Lumped Acceleration Limit` `Real`

The lumped approach sometimes gives too high or too small relaxation factors. This may happen particularly at the very vicinity of the solution where the approximation errors have a greater effect.

`Triple Point Fixed` `Logical`

This keyword enforces the triple point to be fixed. This means that the temperature used for finding the isotherm is set to be the temperature of the triple point. This freezes the position by construction. Typically this should be combined with a temperature control that at convergence results to the triple point being at melting point.

`Internal Mesh Movement` `Logical`
: The mesh around the growth interface may be moved in two ways: using the mesh update solver based on the linear elasticity, or using the simple 1D mapping built in the solver. If this flag is set active the internal mesh movement is used.

`Passive Steps` `Integer`
: If for some reason we want to omit that the solved phase change position is updated to the mesh we may use this flag which for the given number of rounds does not apply the mesh update.

`Body` `body id`

`Solid` `Logical`

`Liquid` `Logical`
: The solver requires information on which of the materials in the system is solid and which is liquid. Currently the solver assumes that both the liquid and solid is uniquely defined.

`Material` `mat id`

`Melting Point` `Real`
: The melting point is the temperature at which the transition form solid to liquid occurs. The melting point is assumed to be constant. If the triple point is fixed the value of the melting point is not used in finding the levelset.

`Density` `Real`
: Density may be needed in the computation of the surface normals By default, the normals point out from the denser of the two materials. Also the density is needed for the computation of latent heat release.

`Latent Heat` `Real`
: The latent heat is the specific internal energy related to the phase change. The latent heat may also be a variable. It is actually not needed by the phase change solver but must be provided for the heat solver.

`Boundary Condition` `bc id`

`Phase Change` `Logical`
: The interface of the phase change is determined by this special flag.

`Phase Velocity i` `Real`
: For the steady state case the heat equation often requires the heat flux as a boundary condition. For this reason the phase velocity for each component may be determined. The keyword is not needed by the current solver.

Model 37

Particle Dynamics

Module name: ParticleDynamics
Module subroutines: ParticleDynamics
Module authors: Peter Råback, Juha Ruokolainen
Document authors: Peter Råback

37.1 Introduction

Note: this is an initial version of the dynamic particle tracker. For real applications it probably requires some additional effort.

The ability to follow single or statistical particles within a finite element can be used in a variety of applications. A common application is to follow particles along streamlines for the purpose of flow visualization. Accounting for electrostatic forces opens the field to microfluidics and accounting for the gravitational force enables applications in sedimentation, for example. If also particle-particle interaction is accounted for also granular flow phenomena may be studied.

This module depends on the many library routines related to particle transport in Elmer. In this module it is assumed that there may be particle-particle interactions. This choice fixes the time-stepping strategies of the different particles together, at least without heroic timestepping schemes. In other words, the same timestep size is applied to the whole particle set.

The particles are located in the finite element mesh using a marching routine where intersections with element boundaries are checked for. The nearest boundary on the way is crossed until there is no boundary to cross. Then the right element has been reached. The algorithm is fast when the stepsize with respect to elementsize is smaller or of the same order. Therefore for the initialization the octree-based search may be more economical and also more robust regarding geometric shapes.

The particle-particle interaction is based on the knowledge of nearest neighbours. Currently the neighbours are determined using the closeness to the nodes if the parent element. This means that the interaction distance needs to be smaller than $h/2$ where h is the mesh parameter. Further, it means that the mesh must be rather uniform.

As the name implies, this module assumes the particles to be dynamic i.e. they have an acceleration. However, the user may also use the module neglecting the inertial forces and requiring a force balance between the drag force and external forces.

37.2 Theory

Forces acting on the particle

Assume that we have a particle in position $\vec{r}$. The corresponding velocity is

$$\vec{v} = \frac{d\vec{r}}{dt} \tag{37.1}$$

Newton's second law yields

$$m\frac{d\vec{v}}{dt} = \Sigma f(\vec{r}, \vec{v}, \ldots) \tag{37.2}$$

where a number of different forces may be considered.

The gravity force acting on the particle is

$$\vec{f} = m\vec{g}, \tag{37.3}$$

where $\vec{g}$ is the acceleration due to gravity. The electrostatic force is simply proportional to the electric field

$$\vec{f_e} = q\vec{E} = q\nabla\phi \tag{37.4}$$

where q is the electric charge.

The viscous fluids cause also a force that acts on the particle

$$\vec{f_S} = -b(\vec{v} - \vec{v}_0) \tag{37.5}$$

where $\vec{v}_0$ is the velocity of the fluid. If the change is estimated to be $d\vec{r}$ then the estimate may be improved by the gradient of velocity, i.e. $\nabla\vec{v}_0 \cdot d\vec{r}$. For Stokes flow the proportionality coefficient scales with viscosity, for example for spheres $b = 6\pi\eta d$ where η is the fluid viscosity and d the radius of the sphere.

Collision model

Two particles may collide with one-another. Assume that the initial particle positions are $\vec{r}_1$ and $\vec{r}_2$. Velocity vectors are $\vec{v}_1$ and $\vec{v}_2$ and lets define $\delta\vec{r} = \vec{r}_1 - \vec{r}_2$ and $\delta\vec{v} = \vec{v}_1 - \vec{v}_2$. Now the condition for a collision is

$$|\delta\vec{r} + \delta\vec{v}\, dt| = R_1 + R_2. \tag{37.6}$$

This leads to condition for the timestep

$$dt = \frac{-b - \sqrt{b^2 - ac}}{a} \tag{37.7}$$

where $b = \delta\vec{r} \cdot \delta\vec{v}$, $a = \delta\vec{v} \cdot \delta\vec{v}$, and $c = \delta\vec{r} \cdot \delta\vec{r} - (R_1 + R_2)^2$. Collision happens if $0 < dt < Dt$.

The collision only affects the normal component. The normal vector is aligned with $\delta\vec{r}' = \delta\vec{r} + dt\,\delta\vec{v}$ i.e. $\vec{n}_r = \vec{r}'/|\vec{r}'|$. Now the normal velocity components are $v_{i,n} = \vec{v}_i \cdot \vec{n}_r$. After the collision the normal velocity component is

$$v'_{1,n} = \frac{cM_2(v_{2,n} - v_{1,n}) + M_1 v_{1,n} + M_2 v_{2,n}}{M_1 + M_2} \tag{37.8}$$

and likewise for $v'_{2,n}$. Here the parameter c is called bounciness and it varies between zero, for fully inelastic collision, to one, for fully elastic collisions. The new velocity is now

$$\vec{v}'_i = \vec{v}_i + (v'_{i,n} - v_{i,n})\vec{n}_r \tag{37.9}$$

and the new position,

$$\vec{r}'_i = \vec{r}_i + \vec{v}_i dt + \vec{v}'_i dt' \tag{37.10}$$

where $dt' = Dt - dt$.

Collisions with the wall are governed with the same equations assuming that mass of the wall is infinite.

The change in the velocity and coordinate position may be mutated to a change in velocity and force. This way the collision model is better additive with the other type of models present in the system.

Contact model

In general the contact between particles depends on their relative position, relative velocity, and relative angular velocity. Generally the contacts should include some damping (negative feedback from velocity) since otherwise the system is prone to flow up. In molecular dynamics, for example, also interaction with more than two particles should be considered. The current treatment is quite limited and we here assume that the contact results just to a spring force in the form

$$\vec{f}_k = k \max(R_1 + R_2 - |d\vec{r}|, 0)\vec{n}_r, \tag{37.11}$$

where k is the spring coefficient.

A similar particle contact model may be present with the wall but possibly with different value for the spring coefficient.

Periodic boundary conditions

It is relatively straight-forward to implement periodic boundary conditions for rectangular and hexahedral type of geometries. And for different geometries the periodic conditions seems more unlikely.

Time evolution

For particles with mass the basic update sequence of velocity and position is

$$\vec{v}_{i+1} = \vec{v} + \frac{dt}{m}\Sigma f \tag{37.12}$$
$$\vec{r}_{i+1} = \vec{r} + dt\vec{v}_{i+1} \tag{37.13}$$

while for massless particles it is assumed that the particle drag is in balance with the other forces given explicitly

$$\vec{v}_{i+1} = \frac{1}{b}\Sigma f \tag{37.14}$$
$$\vec{r}_{i+1} = \vec{r} + dt\vec{v}_{i+1} \tag{37.15}$$

The timestep dt may be given explicitly, or it may be defined from the characteristic velocity V. If the change in distance dS is given then

$$dt = \frac{dS}{V}, \tag{37.16}$$

and when the Courant number C is given

$$dt = C\frac{h}{V}, \tag{37.17}$$

where V is either the maximum absolute velocity, or the mean absolute velocity.

Also other timestepping schemes could be used but that's something for later.

37.2.1 Postprocessing

The possibility to use each particle as an integration point in data fitting problem makes it possible to couple the particles back to a continuous field. The following kinds of information could be abstracted from the particles, for example.
Kinetic energy of particles

$$E_k = \frac{1}{2}mv^2. \tag{37.18}$$

Potential energy associated to gravity field

$$E_g = m\vec{g} \cdot \vec{r}. \tag{37.19}$$

Potential energy associated to electrostatic field The corresponding potential energy is

$$E_e = q\phi. \tag{37.20}$$

Etc. In practice sufficient amount of data may not be present at every node if the data is used only after appropriate smoothing.

37.3 Keywords

`Solver` `solver id`

`Equation` `String [ParticleDynamics]`
The name of the equation.

`Procedure` `File "ParticleDynamics" "ParticleDynamics"`
The name of the procedure.

Keywords related to the allocation and initialization of the particles.

`Number of Particles` `Integer`
Number of particles to be sent. The number may be given by this keyword as an absolute number. Often a relative number, particularly in parallel computation, may be favorable.

`Particle Node Fraction` `Real`
The relative fraction of particles to nodes. The nodes may also be masked ones.

`Particle Element Fraction` `Real`
The relative fraction of particles to elements. The elements may also be masked ones.

`Coordinate Initialization Method` `String`
Initialization method for the coordinates. The options include `nodal ordered`, `elemental ordered`, `sphere random`, `box random`, `box random cubic` with their own initialization strategy.

`Initial Coordinate` `Size n, dim; Real`
The default initialization methods for coordinates.

`Initialization Condition Variable` `String`
If this is given then the particles are initialized only where this has a nonzero permutation vector.

`Initialization Mask Variable` `String`
If this is given then the particles are initialized only in elements or nodes where the variable has a positive value.

`Min Initial Coordinate i` `Real`

`Max Initial Coordinate i` `Real`
For box initialization methods set the bounding box for doing initialization.

`Particle Cell Radius` `Real`
If the initialization method is `box random cubic` then the particle is always put to a unit cell located in the given bounding box.

`Particle Cell Fraction` `Real`
If the initialization method is `box random cubic` then this keyword gives the fraction of filled cells in the initial configuration.

`Initial Sphere Radius` `Real`
If the initialization method is `sphere random` then this set the radius of the sphere.

`Initial Sphere Center` `Size 3; Real`
Sets the size of the initial sphere center.

`Velocity Initialization Method` `String`
There are many ways to initialize the velocities of the particles: `thermal random`, `even random`, `constant random`.

`Initial Velocity` `Size n, dim; Real`
The particle velocities may be also initialized only by this keyword, or this may be used to give a bulk component to the otherwise random velocity field.

`Initial Velocity Amplitude` `Real`
In many velocity initialization methods an initial velocity amplitude is needed.

`Initial Velocity Time` Real
: When initializing the velocity also the initial coordinates may be affected by determining a offset for the time used to advance the particles. This could be used, for example, to distribute the particles from an initial point using the random velocity field.

`Initial Coordinate Search` Logical
: After the initialization is done do an initial octree-based search for the initial coordinate positions. This is applicable only to serial problems.

`Reinitialize Particles` Logical
: Reinitialize the particles in the start of each time when the subroutine is called. This would make sense in some kind of scanning mode. The default is `False`.

`Particle Release Number` Integer
: If not all particles are sent at the same time. This is the absolute number of particles sent at the start of the subroutine call.

`Particle Release Fraction` Real
: If not all particles are sent at the same time. This is the fraction of particles sent at the start of the subroutine call.

`Delete Wall Particles` Logical
: Currently a hack which is used to remove particles sitting on the wall which otherwise seem to get stuck.

Keywords related to the timestepping strategy.

`Timestep Size` Real
: The internal timestep size.

`Max Timestep Size` Real
: The lower limit of the internal timestep size.

`Min Timestep Size` Real
: The upper limit of the internal timestep size.

`Timestep Distance` Real
: The distance that is travelled within one timestep based on the characteristic velocity.

`Timestep Courant Number` Real
: The desired Courant number resulting from the timestep based on the characteristic velocity. Note that currently just one element is used to compute the parameter h. A global definition of the Courant number would result to a significant increase in the computational cost.

`Max Characteristic Speed` Logical
: When computing characteristic velocity use the max norm.

`Max Timestep Intervals` Integer
: Maximum number of internal timesteps.

`Max Cumulative Time` Real
: Maximum cumulative time within one call.

`Simulation Timestep Sizes` Logical
: Alternatively, one may use the timesteps as defined by the `Timestep Sizes` of the `Simulation` section.

Keywords related to the actual physical interaction models chosen within the particles and with particles and walls.

`Particle Particle Collision` Logical
: Is there some collisions between particles.

`Particle Particle Contact` Logical
: Is there contact between particles resulting to additional forces.

`Box Particle Periodic` Logical
: Is the system periodic.

`Box Periodic Directions` Integer
: If not all directions require the periodic model this may be used to define the active directions.

`Box Particle Collision` Logical
: Is there collisions between particles and 2D or 3D box. This provided for a cheaper treatment of BCs than the generic way.

`Box Particle Contact` Logical
: Is there contact between particles and walls resulting to additional force.

`Box Contact Directions` Integer
: If not all directions require the contact model this may be used to define the active directions.

`Velocity Variable Name` String
: Name of the variable if velocity drag is present.

`Velocity Gradient Correction` Logical
: When using the drag model evaluate the drag forces using correction from the velocity gradient.

`Potential Variable Name` String
: Name of the variable if electrostatic potential is present.

`Velocity Condition Variable Name` String
: Name of the field which determines the fixed velocity conditions of the particles.

`Coordinate Condition Variable Name` String
: Name of the field which determines the fixed coordinate conditions of the particles.

Keywords related to the physical properties of the particle and to the joint physical properties of the particle-particle and particle-wall contacts.

`Particle Mass` Real
: There are a number of particle properties needed in different interaction models and particle mass is one of them. In principle these could be altered to be variables but currently they are assumed to be the same for all particles.

`Particle Radius` Real
: The particle radius used in particle-particle interaction, and in evaluating the density of the particle.

`Particle Gravity` Logical
: Should gravity be accounted for. If yes, use the gravity defined in the

`Particle Lift` Logical
: The background fluid has a density that results to a lift (buoyancy) that may be accounted for. Should gravity be accounted for. If yes, use the gravity defined in the `Constants` section.

`Particle Damping` Real
: Particle damping proportional to velocity only.

`Particle Drag Coefficient` Real
: Particle drag coefficient in fluid field.

`Particle Bounciness` Real
: Defines, when particles collide is the collision totally elastic or totally inelastic. Corresponding extreme values are 1 and 0. This relates only to collision models.

`Particle Spring` Real
: Spring constant in the force model between particles. This relates only to contact models.

`Particle Charge` Real
: The electric charge of the particle.

`Particle Decay Distance` Real
: The decay of the particle effect.

`Wall Particle Radius` Real
: In interaction with the walls different properties are given as the interaction with the wall is quite different regarding, for example, the contact shape.

`Wall Particle Spring` Real
: Spring constant in interaction with wall.

`Wall Particle Bounciness` Real
: Elasticity of collision with interaction with the wall.

Keywords related to the generation of fields from the particle data.

`Particle To Field` Logical
: Is there any coupling from particles to field needed? This leads to the need of finite element machinery. The opposite is always assumed to be true i.e. the particles are always assumed to be located in the FE mesh.

`Reinitialize Field` Logical
: When revisiting the solver should the particle field be initialized at the start.

`Particle To Field Mode` Integer
: If a field is generated from the particles, what actually should be computed.

`Particle Decay Time` Real
: This is an optional parameter that represents the characteristic time that is used to forget history data from the particle to field representation.

`Particle Decay Distance` Real
: This is an optional parameter that represents the characteristic distance that is used to forget history data from the particle to field representation.

Keywords related to saving and echoing information. No effect to the actual computations.

`Output Interval` Integer
: The internal output interval of the solver. If not given the particle data will be saved within the solver. The alternative is to save particle data with an external solver.

`Output Format` String
: Output format which may be either `table` or `vtu`.

`Table Format` Logical

`Vtu Format` Logical
: Alternative way of giving the output format. Has the nice property that several formats may be given at the same time.

`Filename Prefix` String
: The prefix of the filename used for saving. Depending on the chosen format an appropriate suffix is attached to the prefix.

`Filename Particle Numbering` Logical
: If possible in the format, use particle indexes for the numbering of files.

`Filename Timestep Numbering` Logical
: If possible in the format, use timestep indexes for the numbering of the files. This is the default in vtu format.

`Particle Info` Logical
: Optionally print out on the screen information on the number of particles and time steps taken.

`Statistical Info` Logical
: Optionally print out on the screen some statistical information on the coordinate positions and velocities. May be useful for debugging purposes, for example.

`Scalar Field i` String
: The scalar fields of the particles to be saved in vtu format. Currently options include `distance` and `dt`.

`Vector Field i` String
: The vector fields of the particles to be saved in vtu format. Currently options include `velocity` and `force`.

`Particle Save Fraction` Real
: If there is a huge number of particles it may be sufficient to use only a subset of them for visualization. This keyword gives the fraction.

`Boundary Condition` bc id

`Wall Particle Collision` Logical
: This activates the collision model between particles and generic boundaries.

`Particle Accumulation` Logical
: An optional flag that activates the possible destruction of the particles at the boundary in case conditions for accumulation are met.

`Particle Accumulation Max Speed` Real
: If this critical speed is given, then accumulate only those particles with smaller velocity.

`Particle Accumulation Max Shear` Real
: If this critical shear rate is given, then accumulate only those particles with a smaller shear rate.

`Particle Trace` Logical
: If this flag is set active then use the accumulated particles to compute a trace to a finite element field.

`Moving Wall` Logical
: The movement of the wall may be accounted for in the wall-particle collision model.

Model 38

Semi-Lagrangian Advection Using Particle Tracking

Module name: ParticleAdvector
Module subroutines: ParticleAdvector
Module authors: Peter Råback, Juha Ruokolainen
Document authors: Peter Råback

38.1 Introduction

This solver utilizes the particle tracker features of Elmer to advect scalar fields diffusion-free on the mesh. For each node of the field one particle is sent backwards in time and the field value is restored from the location where the particle is found.

For more details on the particle tracking look at the other modules utilizing the same features in a more generic way.

38.2 Theory

In particle advection we assume that the fields are transported diffusion-free carried by a velocity field $\vec{v}$. The particle are initialized at the nodes of the mesh. Thereafter each particle is followed $-\delta t$ in time i.e. the following integral is evaluated

$$\vec{r} = \vec{r}_0 + \int_0^{-\delta t} \vec{v}\, dt. \tag{38.1}$$

Currently the integral may be evaluated using first order explicit scheme or a second order Runge-Kutta scheme. In the first order scheme a quadratic correction term is available making the scheme effectively comparable with the Runge-Kutta scheme. There the following approximation is used

$$< \vec{v} >= \vec{v}_0 + \frac{1}{2}(\nabla \vec{v}_0) \cdot \vec{v}_0\, dt. \tag{38.2}$$

When the particles have been transported the field may be evaluated from

$$f(\vec{r}_0, t) = f(\vec{r}, t - \delta t) \tag{38.3}$$

The treatment of boundaries results to some additional complication. It is assumed that if the particles vanishes in the upstream boundary then the boundary values of f are used.

The particles may have some properties along the path integral. For this purpose the user may evaluate evolution over time,

$$I_t = \int c(t)\, dt \tag{38.4}$$

and over distance,

$$I_s = \int c(t)\, ds \tag{38.5}$$

38.3 Parallel operation

The parallel operation of the particle advector routine is much more complicated than the serial. The particles are followed in the partitioned mesh and if they pass the partition interface they are passed to the next partition. Finally the values must be sent back to the originating partition in order to collect the results.

The 2nd order Runge-Kutta method will probably have problems in parallel so the user should rather choose the quadratic correction method which offers similar accuracy.

38.4 Keywords

`Solver` `solver id`

`Equation` `String [ParticleAdvector]`
: The name of the equation.

`Procedure` `File "ParticleAdvector" "ParticleAdvector"`
: The name of the procedure.

`Advect Elemental` `Logical`
: Should we advect the particles on center of elements instead of nodes. This is often more robust as the velocity field is well defined within elements.

`Advect DG` `Logical`
: Initialize particles at corners of DG elements with reduced size. This may also be more robust but also significantly more costly.

`Advect IP` `Logical`
: Initialize particles at integration points. May yield optimal accuracy with high computational cost.

`Coordinate Initialization Method` `String`
: This is automatically enforced to `nodal ordered`.

`Particle Node Fraction` `Real`
: This is automatically enforced to one in order to have one particle for each node.

`Initialization Mask Variable` `String`
: If this is given then the particles are initialized only in nodes where the variable has a positive value.

`Reinitialize Particles` `Logical`
: Reinitialize particles after each time visiting the solver. This could be desired operation in transient simulation if we just want to advect for one timestep.

`Velocity Initialization Method` `String`
: This is enforced to `nodal velocity` which means that the first velocity is taken from the nodal point.

`Velocity Variable Name` `String`
: Name of the velocity field in vector form. Default is `flow solution`.

`Time Order` `Integer`
: This is defaulted to zero which means that the velocity field is used directly for the particle velocity.

`Timestep Size` `Real`
: There are several keywords related to the timestepping strategy. All of them are available also here. The negative sign is added internally. Here just the most typical ones are given. This keyword gives the internal timestep size.

`Simulation Timestep Sizes` Logical
: Alternatively, one may use the timesteps as defined by the `Timestep Sizes` of the

`Max Timestep Intervals` Integer
: Maximum number of internal timesteps.

`Max Integration Time` Real
: This keyword may be used to retire particles after following them long enough in time. This may help to save some time if there are regions with almost zero velocities or closed circulation loops.

`Particle Accurate At Face` Logical
: When hitting the well this keyword enforces the more accurate integration method which returns the correct point of exit. This is then used in the advection as the point of evaluation. When using accurate particle detection the algorithms might not always be as robust.

`Runge Kutta` Logical
: Use the 2nd order Runge-Kutta method for integration.

`Velocity Gradient Correction` Logical
: This is an alternative way of increasing the accuracy of the integral. Here the gradient of the velocity field is evaluated at the point of the particle to account for the curvature of the flow.

`Source Gradient Correction` Logical
: This is a way to increase accuracy of the path integrals by evaluating the integrands $c(t)$ and $c(s)$ using gradient correction over the step.

`Variable i` String
: Names of the variables to be advected (i=1,2,3,...). Any proper field variable of Elmer may be advected. The field may exist in advance, if not it will be created There is also a group of internal variables with fixed name: `particle status, particle number, particle distance, particle coordinate, particle coordinate_abs, particle velocity, particle velocity_abs, particle time, particle time integral, and particle distance integral` These are related to the particle tracking machinery.

`Result Variable i` String
: The default name of the advected variable is obtained by adding the prefix `Adv` to the field name. Alternatively, the user may give the name of the result variable by this keyword.

`Operator i` String
: Possible operator that may be applied to the variable. The choices are `derivative` (with respect to time), `difference`, and `cumulative`. By default no additional operator is applied.

`Norm Variable Index` Integer
: The solver may compute the change in one specific field value in order to provide information for consistency check, or convergence monitoring. This keyword sets the index related to the `Variable i` list. Default is zero i.e. no norm is computed.

`Particle Info` Logical
: Show the particle information at the end of the solver execution.

`Body Force` bf id

`Particle Distance Integral Source` T
: he integrands $c(s)$ related to the distance path integral. Existence of this keyword will activate also the path integral variable named `Particle Distance Integral`.

`Particle Time Integral Source` T
: he integrands $c(t)$ related to the time path integral. Existence of this keyword will activate also the path integral variable named `Particle Time Integral`.

`Particle Fixed Condition` Real
: We may want to freeze the particles on the body depending on the physics. This is done if this condition has a positive value.

`Boundary Condition` `bc id`

`Particle Wall` `T`
: his will mark the wall which the particle cannot go though and by default stops at.

`Particle Fixed Condition` `Real`
: We may want to freeze the particles on the boundary depending on the physics. This is done if this condition has a positive value.

Model 39

Ordinary differential equation in moving mesh

Module name: StructureFlowLine
Module subroutines: StructureFlowLine
Module authors: Peter Råback
Document authors: Peter Råback

39.1 Introduction

The purpose of this solver is to allow accurate solution of ordinary differential equations in moving mesh. The approach is limited meshes that could be extruded being drawn to the direction of the extrusion. The idea is that then in moving coordinates we have a direct correspondence between mesh parameter h, draw velocity v and timestep dt, by $dt = h/v$.

All the equation could basically also be solved in an eulerien mesh where the velocity is presented as convection term in the transport equation. The problem with this kind of approach is that such convection terms require some kind of stabilization schemes. In extreme cases when diffusion tends to zero such equations easily become hard to solve. The current approach cannot consider diffusion. Only convection, reaction and diffusion terms are possible.

39.2 Keywords

`Solver` `solver id`

`Equation` `String MarchingOde`
The name of the equation.

`Procedure` `File "MarchingODESolver" "MarchingODESolver"`
The name of the procedure.

`Variable` `String`
Name of the variable used for the marching.

`Nonlinear System Max Iterations` `Integer`
If the system is nonlinear we may solve it for each node until convergence or number of iterations has been reached.

`Nonlinear System Convergence Tolerance` `Real`
Convergence tolerance if several iterations are used..

`Draw Velocity` `Real`
Speed of the drawing process in the direction of the extrusion.

`Parabolic Model` `Logical`
: Really solve for u^2 rather than u. This treatment may then be used to eliminate the nonlinear nature of the problem.

`Apply BCs Only` `Logical`
: We might want to apply the marching routine only at the surface.

`Active Coordinate` `Integer`
: The drawing direction i.e. 1, 2 or 3. This is a keyword passed on to routines detecting extruded structures, as are some of the following ones too.

`Dot Product Tolerance` `Real`
: When determining the structure of the mesh in the active direction this tolerance is used to decide that an element edge is aligned with the direction of the action.

`Material` `bc id`

`Varname: Source` `Real`
: Source term for the field to be marched.

`Varname: Reaction Coefficient` `Real`
: Reaction term for the field to be marched.

`Varname: Time Derivative Coefficient` `Real`
: Time derivative term. If not given it is assumed to be unity.

Part VII

Mesh Adaptation, Transformation and Analysis

Model 40

Mesh Adaptation Solver

Module name: MeshSolve
Module subroutines: MeshSolver
Module authors: Juha Ruokolainen
Document authors: Juha Ruokolainen

40.1 Introduction

Moving boundaries are often encountered in different types of computations, e.g. in Fluid-Structure Interaction (FSI) problems. Moving boundaries pose the problem of mesh adaptation to the boundaries. With this solver, instead of generating the whole mesh afresh when a boundary is moved, the current mesh nodes are moved so that the mesh hopefully remains 'good'. This type of solution only applies to cases where the changes in geometry are relatively small. It is, however, often cheaper in terms of CPU time to use this module in contrast to regenerating the whole mesh.

For time dependent simulations the mesh deformation velocity is also computed. The name of this variable is `Mesh Velocity`.

40.2 Theory

The equation for elastic deformation of the mesh, given displacement of the boundaries, may be written as

$$-\nabla \cdot \tau = 0, \tag{40.1}$$

where the stress tensor τ can be expressed in terms of Lame parameters as

$$\tau = 2\mu\varepsilon(\vec{d}) + \lambda \nabla \cdot \vec{d} I. \tag{40.2}$$

Here $\vec{d}$ is the mesh displacement field, μ and λ are the first and second Lame parameters, respectively, and I is the unit tensor. The linearized strains are given as

$$\varepsilon(\vec{d}) = \frac{1}{2}(\nabla\vec{d} + (\nabla\vec{d})^T). \tag{40.3}$$

The Lame parameters in terms of Young's modulus Y and the Poisson ratio κ read

$$\mu = \frac{Y\kappa}{(1-\kappa)(1-2\kappa)}, \quad \lambda = \frac{Y}{2(1+\kappa)}. \tag{40.4}$$

Note that in this context the values of the material parameters are fictional, and may be chosen to help convergence or quality of the resulting mesh.

40.2.1 Boundary Conditions

For each boundary a Dirichlet boundary condition

$$d_i = d_i^b \tag{40.5}$$

may be given. Usually the displacement is given a priori or computed by, for example, the elasticity solvers.

40.3 Keywords

`Solver` `solver id`

Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation` `String [Mesh Update]`

The name of the equation. If different from the default name `Mesh Update`, then the following two keywords must be defined as well.

`Procedure` `File "MeshSolve" "MeshSolver"`

Name of the solver subroutine.

`Variable` `String`

Name of the variable.

`Equation` `eq id`

The equation section is used to define a set of equations for a body or set of bodies:

`Mesh Update` `Logical`

If set to `True`, solve the mesh adaptation equations.

`Material` `mat id`

The material section is used to give the material parameter values. The following material parameters may be set for the Navier equations.

`Poisson Ratio` `Real`

For isotropic materials the Poisson ratio must be given with this keyword.

`Youngs Modulus` `Real`

The elastic modulus must be given with this keyword.

`Boundary Condition` `bc id`

The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. Those related to the Navier equations are

`Mesh Update i` `Real`

Dirichlet boundary condition for each displacement component `i`$= 1, 2, 3$. The boundary displacement may be computed with some other solver. The computed displacement field may then be used in the setting in the following way:

`Mesh Update i` `Equals Displacement i`

Including such lines with `i=1,2,3` in the boundary condition setting will give the position of the updated mesh on the boundary directly in terms of the solution of the displacement solver.

40.4 Examples

40.4.1 A Simple FSI computation using MeshSolver

In this simple computation Navier-Stokes equations are solved in the domain shown in the two pictures below. On the left there is an inflow boundary, and on the right an outflow boundary. In the block inside the flow domain (the mesh is not shown for the block), the elasticity equations are solved. The block is fixed at the bottom, and is otherwise deformed by the fluid pressure and flow fields. The whole system is iterated as follows:

- Solve fluid flow,
- Solve deformation of the block,
- Solve the fluid domain mesh with MeshSolver according to the displacements of the block,

until convergence is obtained.

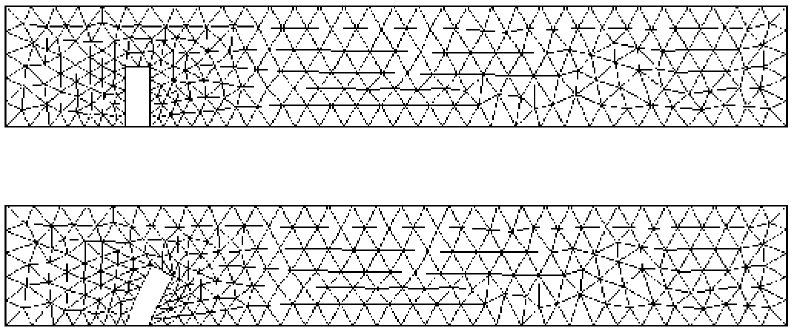
Figure 40.1: The original computational mesh (up), and the mesh of the converged solution (down) of a FSI computation.

Model 41

Nonphysical Mesh Adaptation Solver

Module name: NonphysicalMeshSolve
Module subroutines: MeshSolver
Module authors: Juha Ruokolainen, Peter Råback
Document authors: Juha Ruokolainen, Peter Råback

41.1 Introduction

This solver is a variation of the `MeshSolver` for cases where the true mesh velocity is not of concern and more liberties can be used in the mesh adaptation. Also it may be used as the mesh adaptation solver in conjunction with `MeshSolver`. For example, in shape optimization of fluid-structure interaction problems two mesh adaptation solvers may be needed simultaneously.

41.2 Theory

For the equation to be solved look at the theory section of the `MeshSolver`. In addition to that, also weak ways of giving boundary conditions is implemented, namely

$$\tau \cdot \vec{n} = kd + f + c(d - d_0) \tag{41.1}$$

where k is a spring coefficient, f is a given force, and d_0 is the target configuration. When c goes to infinity, this condition approaches the Dirichlet conditions.

41.3 Keywords

`Solver solver id`

Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Procedure File "NonphysicalMeshSolve" "MeshSolver"`

Name of the solver subroutine.

`Variable String [-dofs 3 Mesh Deform]`

The name of the displacement field. It should be different from `Mesh Update` in order to avoid conflicts in its interpretation. Here we use the name `Mesh Deform`. The dimension should be the same as that of the mesh.

`Cumulative Displacements Logical`

If the same solver is called multiple times, then this flag controls whether the displacements are added each time to the initial or previous mesh shape. The default is `False`.

`Moving Mesh` Logical
: This keyword relates to a mesh that is being moved by an outside solver such as the `MeshSolver`. The default is `True`.

`Target Field` String
: The name of the field d_0 that is used as a target when setting the boundary conditions in a weak manner.

`Nodal Penalty Factor` Real
: A coefficient that is used to set the displacements to those given by the target field in a soft manner. This is constant for each node which results to problems in mesh consistency.

`Material` mat id
: The material section is used to give the material parameter values. The following material parameters may be set for the Navier equations.

`Poisson Ratio` Real
: For isotropic materials the Poisson ratio must be given with this keyword.

`Youngs Modulus` Real
: The elastic modulus must be given with this keyword.

`Boundary Condition` bc id
: The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. Those related to the Navier equations are

`Mesh Deform i` Real
: Dirichlet boundary condition for each displacement component `i`$= 1, 2, 3$.

`Mesh Coefficient i` Real
: The spring coefficient related to the given coordinate direction, `i`$= 1, 2, 3$.

`Mesh Force i` Real
: The right-hand-side of the mesh deformation equation, `i`$= 1, 2, 3$.

`Mesh Normal Force` Real
: The right-hand-side of the mesh deformation equation in the normal direction.

`Mesh Penalty Factor` Real
: When using the soft way of setting boundary conditions, this value gives the weight function c.

Model 42

Rigid Mesh Transformation

Module name: RigidMeshMapper
Module subroutines: RigidMeshMapper
Module authors: Peter Råback
Document authors: Peter Råback

42.1 Introduction

Sometimes there is a need to transform meshes without generating a new mesh. The most simple case is that of a rigid motion where some bodies move according to prescribed rotations and translations. Typically this could be a preprocessing step in a parametric study of some problem. Then this solver may be used to perform the mesh transformation.

In addition to applying rigid transformations to bodies followed by a stretching, this solver includes also a relaxation parameter which may be used to define which fraction of the node coordinates is taken from the suggested coordinates, and which part from the original coordinates.

It should be noted that the usage of this solver is rather limited. It cannot handle cases where bodies move with respect to one another if there is a mesh between the bodies. Then the `MeshUpdate` solver should be used instead.

42.2 Theory

Given original coordinate $\vec{x}_0$ the solver applied first a rotation, then a translation, and finally a scaling operator such that the suggested new coordinates yield

$$\vec{x}_1 = \mathcal{S}(\mathcal{R}(\vec{x}_0 - \vec{o}) + \vec{t}) + \vec{o}, \tag{42.1}$$

where $\vec{t}$ is the vector of translation, $\vec{o}$ is the origin, $\mathcal{S}$ the scaling matrix, and $\mathcal{R}$ is the rotation matrix. Rotation may currently be performed only around one main axis.

Often it is desirable that the rigid transformations are performed only for some objects and while some stay fixed. Between them the transformation degree should vary smoothly. To this aim the solver may be used to compute a degree of transformation field from the Poisson equation

$$-\nabla \cdot (1 + c|\nabla\Phi|)\nabla\Phi = s \tag{42.2}$$

where c is an optional coefficient which may be used to increase the mesh rigidity around singularities. A suitable boundary condition is $\Phi = 0$ for fixed objects and $\Phi = 1$ for moving objects.

Usually the rigid mesh mapper does not have a source term. However, for different testing purposes there is also the possibility to give a source term s which may be used to distort the mesh in a continuous way. Upon request the deformation may then be normalized to unity.

When the rigid mesh mapping is applied together with the relaxation, the end result is

$$d\vec{x} = \Phi\left(\vec{x}_1 - \vec{x}_0\right). \tag{42.3}$$

42.3 Keywords

`Solver` solver id

`Equation` String [RigidMeshMapper]
The name of the equation.

`Procedure` File "RigidMeshMapper" "RigidMeshMapper"
The name of the procedure.

`Variable` String
Optionally the solver may be used to compute a relaxation field whose values are on the interval $[0, 1]$. The final displacement is then obtained as a product of the field and the suggested rigid body motion. The name is arbitrary since it is not referenced elsewhere.

`Mesh Rotation Axis Order(dim)` Integer
The user may specify the order in which the mesh is rotated around the axes with this keyword. By default first rotation is around x-axis, then around y, and finally around z. Different order will give a different end result.

`Translate Before Rotate` Logical
If this keyword is given value `True`, then the translation is carried out before rotations. The default is vice versa.

`Cumulative Displacements` Logical
The displacement resulting from this solver may be either absolute or cumulative. For example, for rotating problems if the mode is cumulative only the incremental angle is given. If the cumulative mode is not enforced, then the full angle from the start of the simulation should be given. The default is `False`.

`Calculate Mesh Velocity` Logical
If this keyword is enforced, then the solver will compute the mesh velocity resulting from rigid body deformation in a transient case. The name of the resulting vector field will be `Mesh Velocity`.

`Mesh Relax Normalize` Found
Normalize the mesh relaxation field such that the maximum value is one.

`Body Force` bf id
The mesh transformations are defined in this section.

`Mesh Translate` Real $[t_x t_y t_z]$
The translational vector which may also be given individually for each component, $i = 1, 2, 3$.

`Mesh Rotate` Real $[\alpha_x \alpha_y \alpha_z]$
The rotation around main coordinate directions. This may also be given individually for each component, $i = 1, 2, 3$. When given for each component, they may also be variables of time, for example.

`Mesh Scale` Real $[s_x s_y s_z]$
The scaling around of main directions. This may also be given individually for each component, $i = 1, 2, 3$.

`Mesh Origin` Real $[o_x o_y o_z]$
The origin used in rotation and scaling.

`Mesh Matrix(dim,dim)` Real
Give the mesh transformation matrix. This overrides all other rigid mesh mapping keywords.

`Mesh Displace i` `Real`
: An alternative for giving the mesh deformation in rigid body motion. Give separately for each component, $i = 1, 2, 3$. This is a local field that may vary between the nodes while the rigid body motion may only depend on global variables such as time.

`Mesh Relax` `Real`
: The relaxation factor determining which amount of the coordinate transformation is taken into account. This is a local field which may depend on coordinate values whereas the other above keywords must be constant for each body force.

`Mesh Relax Source` `Real`
: An optional source term for the mesh relaxation field.

`Boundary Condition` `bc id`
: The boundary conditions that define the moving and fixed walls.

`Moving Boundary` `Logical`
: Gets relaxation field multiplied by one.

`Fixed Boundary` `Logical`
: Gets relaxation field multiplied by zero.

Model 43

Structured Mesh Mapper

Module name: StructuredMeshMapper
Module subroutines: StructuredMeshMapper
Module authors: Peter Råback and Thomas Zwinger
Document authors: Peter Råback and Thomas Zwinger

43.1 Introduction

For structured meshes some operations may be done much more effectively than for generic meshes. One such operation is the mapping of mesh so that it is fitted for given top and bottom surfaces. For example, the mesh for the computational glaciology could be deduced from a uniform initial mesh when its top and bottom surfaces would be known in terms of some given fields.

43.2 Theory

The algorithm used for the mapping has two sweeps. Assume that we would like to perform mapping in direction $\vec{e}_z$. At the first sweep over all elements we would deduce pairwise information over nodes on which nodes are in `up` and `down` directions from each other. Then using this directional information recursively one can easily deduce which nodes are the `top` and `bottom` representatives of any node in the mesh. With this information any node between the top and bottom surface may be mapped as the linear combination of the top and bottom displacements. In the end the top surface of the mesh is mapped to the given top position and the bottom surface of the mesh to the given bottom position, correspondingly.

43.3 Keywords

`Solver solver id`

`Equation String [StructuredMeshMapper]`
: The name of the equation.

`Procedure File "StructuredMeshMapper" "StructuredMeshMapper"`
: The name of the procedure.

`Active Coordinate Integer`
: The direction in which the structured mapping is performed i.e. 1, 2 or 3.

`Displacement Mode Logical`
: The values may be either used either directly as absolute coordinate values, or as displacement adding them to the original coordinate values. With this keyword the latter may be chosen.

`Dot Product Tolerance` `Real`
: When determining the structure of the mesh in the active direction this tolerance is used to decide that an element edge is aligned with the direction of the action.

`Top Surface Level` `Real`
: If the value is constant, then this keyword may be used to give the top surface position.

`Top Surface Variable Name` `String`
: The top surface may be given by some auxiliary variable computed by some other solver, for example.

`Bottom Surface Level` `Real`
: If the value is constant, then this keyword may be used to give the bottom surface position.

`Bottom Surface Variable Name` `String`
: The bottom surface may be given by some auxiliary variable computed by some other solver, for example.

`Mid Surface` `Real`
: Sometimes there is a middle layer that needs to be mapped as well. Then this keyword may be used. The mapping is then done linearly in two parts.

`Correct Surface` `Logical`
: If this keyword is set to True, a minimum height (see next keyword) is applied to the extrusion.

`Minimum Height` `Real`
: Sets the constant minimum extrusion height.

`Mesh Height Map` `Real`
: Assuming that the mesh has height blow the minimum height how should the relative height to that (<1) be mapped. Allows for smoother geometry deformations avoiding large areas of constant thickness.

`Correct Surface Mask` `String`
: This optionally defines a name for a variable where the information on whether a point is part of a column that has been corrected to the minimum height. It is -1 if it has been corrected and +1 else.

`Mesh Velocity Variable` `String`
: This keyword is used to give the variable in which mesh velocity will be computed to. The mesh velocity will be really 1D only so this variable should be a scalar. If the user wants to compute a vector then the correct component of that should be given as the parameter.

`Mesh Update Variable` `String`
: This keyword is used to give the variable in which mesh coordinate will be computed to. The new coordinate will be really 1D only so this variable should be a scalar. If the user wants to compute a vector then the correct component of that should be given as the parameter.

`Displacement Mode` `Logical`
: The coordinates resulting from this solver may be either absolute or cumulative. If displacement mode is not enforced then the coordinates will be treated as absolute values. The default is `False`.

`Mesh Velocity First Zero` `Logical`
: If this keyword is set `True` then the 1st time this routine is visited the mesh velocity is enforced to zero. May be attractive if the initial geometry is not really the initial state.

`Always Detect Structure` `Logical`
: Redetect structure always when visiting solver. Does not really make sense except if the mesh has changed.

`Recompute Stabilization` `Logical`
: Mesh deformation may affect the stabilization. This flag activated the stabilization parameters (that may be used by FlowSolve or HeatSolve, for example). Using this flag enforces that the parameters are recomputed.

`Mapped Mesh Name` `File`
: We may want to save the mapped mesh. This keyword gives the output directory.

`Boundary Condition` `bc id`

`Top Surface` `Real`
: The top surface position may also be given with a boundary condition.

`Bottom Surface` `Real`
: The bottom surface position may also be given with a boundary condition.

`Mid Surface` `Real`
: Optionally one may give a mid surface that should be between the top and bottom surfaces. This is the only method for giving the mid surface since it also specifies the active nodes whereas for top and bottom they are defined even without the boundary condition.

Model 44

Free surface with streamlines

Module name: StructureFlowLine
Module subroutines: StructureFlowLine
Module authors: Peter Råback
Document authors: Peter Råback

44.1 Introduction

There are many different kinds of free surface problems. It is difficult to create a generic algorithm that would be optimal for all problems. Therefore specific cases may need specific solvers. This solver is intended for steady-state drawing and pulling problems where the mesh is structured in the direction of the forced flow. Then it is possible to follow the streamlines of the flow and map element edges with the flow. When converged the streamlines will then coincide with the element edges providing an optimal solution for the problem.

The solver can be used by itself as a free surface solver, or together with a mesh adaptation solver so that the current solver only gives the suggested displacement at the boundaries.

44.2 Theory

Assume that we want to map the coordinates $\vec{r}_i$ so that they coincide with the streamlines. Then

$$\vec{r}_{i+1} = \vec{r}_i + \vec{v}_a \frac{|dr_{i,k}|}{|v_{a,k}|} \tag{44.1}$$

where k is the active coordinate direction of the pulling or drawing process. The average velocity may be computed from

$$\vec{v}_a = \frac{1}{2}(\vec{v}_i + \vec{v}(\vec{r}_i + 1)) \tag{44.2}$$

or from

$$\vec{v}_a = \vec{v}\left(\frac{1}{2}(\vec{r}_i + \vec{r}_{i+1})\right). \tag{44.3}$$

When $\vec{r}_{i+1}$ is updated the information may be used to derive on improved estimate of $\vec{v}_a$. This requires as an operation that the velocity must be evaluated within an arbitrary position. For this purpose an octree structure for the elements is used to speedup the search. Typically even one corrector step will improve the results significantly.

44.3 Keywords

`Solver` `solver id`

`Equation` `String [StructuredFlowLine]`
: The name of the equation.

`Procedure` `File "StructuredFlowLine" "StructuredFlowLine"`
: The name of the procedure.

`Velocity Variable Name` `String`
: Name of the variable used to define the streamlines.

`Active Coordinate` `Integer`
: The drawing direction i.e. 1, 2 or 3.

`Dot Product Tolerance` `Real`
: When determining the structure of the mesh in the active direction this tolerance is used to decide that an element edge is aligned with the direction of the action.

`Displacement Mode` `Logical`
: The values may be either used either directly, or saved to a field. If this flag is set `True` the coordinate values will be changed directly. The default is `False`.

`Hard Displacement Name` `String`
: The name of the field for the suggested displacement. These values may be used in a soft way in mesh deformations in order to avoid singularities that often appear at corners.

`True Flow Line Iterations` `Logical`
: When computing the drawing shapes the first iterations lead to larger displacements and set higher demands to the numerical methods. Close to convergence the velocity may be more accurately defined at the existing flow line. This flag determines the number of the more costly iterations. The default is zero.

`Averaging Order` `Integer`
: Order of iterations in evaluating the new position. Default is one.

`Averaging Method` `Integer`
: Whether to use velocity at the average point (1), or average of the velocity (2).

`Nonlinear System Relaxation Factor` `Real`
: Relaxation may be used to relax already the suggested displacement field.

`Boundary Condition` `bc id`

`Flow Line` `Logical`
: By defining this keyword the solver is applied only to the those boundary nodes where the flag is active. This reduces the computational time required.

`Body Force` `bf id`

`Flow Line` `Logical`
: By defining this keyword the solver is applied only to the those bulk nodes nodes where the flag is active.

Model 45

Statistics of finite element mesh

Module name: ElementStats
Module subroutines: ElementStats
Module authors: Peter Råback
Document authors: Peter Råback

45.1 Introduction

This module is used to calculate provide information on the mesh. The quality of the mesh will have a great effect on the solution. This solver may help the user to determine whether the mesh is suitable for the need.

Currently three different operations are performed:

- element size
  This is the metric determinant given by the ElementInfo.
- element skew
  This is the skewness (in degrees) of the mesh, if applicable. Only quadrilaterals, pyramids, wedges and hexahedrons may be skewed.
- element ratio
  This refers to the ratio between maximum and minimum edge lengths of an element.

The operations are performed separately for bulk and boundary elements.
It is available currently only in serial.

45.2 Keywords

`Solver` solver id

`Equation` String ElementStats

`Procedure` File "ElementStats" "ElementStats"

`Create Histogram` Logical
: When analyzing the mesh the results may be shown also as a histogram with an even distribution between min and max values. This flag activates the classification of the statistics into a histogram.

`Histogram Intervals` Integer
: The number of intervals in the histogram when classifying the element properties.

Part VIII

Derived Fields and Quantities

Model 46

Streamline Computation

Module name: StreamSolver
Module subroutines: StreamSolver
Module authors: Mika Juntunen, Peter Råback
Document authors: Mika Juntunen, Peter Råback

46.1 Introduction

Streamline is a line in flow whose tangent is parallel to velocity field $\vec{u}$ of the flow in every point $\vec{x}$. It should be noted that the path of material is generally not the same as streamlines. There is also third set of closely related lines, namely streak lines. On certain streak lines lie all those flow elements that at some earlier instant passed through a certain point in domain. Of course, the streak lines are generally different than streamlines but when the flow is steady all three set of lines coincide.

Streamlines are mainly used in providing a picture of the flow field. Drawing streamlines so that neighbouring streamlines differ by the same amount, gives a picture where direction and magnitude change of flow are clearly prescribed.

46.2 Theory

We are restricted here to the incompressible, steady flow in 2D geometry. The geometry may be 3D, but it must effectively be 2D as in axis symmetric geometry.

In 2D Cartesian geometry stream function ψ is defined

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{46.1}$$

Here the geometry is (x, y) and the corresponding flow is $\vec{u} = (u, v)$. Let Ω be the domain of the flow and $\vec{v}$ a test function for the flow. Definition (46.1) leads to finite element approximation

$$\int_\Omega \nabla\psi \cdot \vec{v}\, \mathrm{d}\Omega = \int_\Omega \vec{u}^\perp \cdot \vec{v}\, \mathrm{d}\Omega \tag{46.2}$$

In axis symmetric geometry the mass conservation calculated in a different way. This leads to following definition for stream function.

$$u = \frac{1}{r}\frac{\partial \psi}{\partial r}, \quad v = -\frac{1}{r}\frac{\partial \psi}{\partial z} \tag{46.3}$$

where the cylindrical coordinates are (z, r, ϕ), velocity components are (u, v, w) and axis of symmetry is z i.e. $r = 0$. This function is sometimes called the *Stokes stream function* and it is not as informative as the stream function in Cartesian case. Of course the finite element approximation is a bit different.

$$\int_\Omega \nabla\psi \cdot \vec{v}\, \mathrm{d}\Omega = \int_\Omega \vec{u}^\perp \cdot \vec{v} r\, \mathrm{d}\Omega \tag{46.4}$$

Here the ϕ component of the flow is excluded.

From definitions (46.1) and (46.3) it is apparent that stream function is constant along the streamlines. So drawing the contours of stream function gives the streamlines.

Sometimes setting the Dirichlet node does results to local distortion of the streamline function. To circumvent this there is an alternative way to fix the level of the streamline. There a implicit penalty $c\psi$ is added to the equation which effectively defines a finite level for the streamfunction. Small values of c are to be favored in order not to distort the streamline field. Using this penalty method the magnitude of the streamline may be quite large but after scaling the values should be rather independent on the value of c.

46.3 Limitations

Some limitations of the current implementation:

- The flow field is assumed to be incompressible.
- There is no dependency on time. Solver can be used in transient cases, but it only produces the streamlines of the current flow field as if it was steady.
- Only 2D Cartesian and axis symmetric coordinate systems are implemented.
- Solver gets the velocity field from user defined variable. In Cartesian case it assumes that first degree of freedom is the x-component and the second is the y-component of the velocity. In axis symmetric case it assumes that the first degree of freedom is the r-component and the second is the z-component of the velocity field.
- User can define the node whose value is first set to zero. This *shouldn't* have affect on results if the normal stream function is used in Cartesian coordinates and Stokes stream function in axis symmetric coordinates. However, if used stream function is forced to something else, the position of the first node usually has a large effect on results. This is because the mass conservation is calculated differently.

46.4 Keywords

`Simulation`

`Coordinate System String`
The coordinate system should be set to be one of the following options: `Cartesian 2D` or `Axi Symmetric`.

`Solver solver-id`
All the keywords beginning `Linear System` can be used. They are explained elsewhere.

`Equation String`
The name you want to give to the solver, for example `StreamSolver`.

`Procedure File "StreamSolver" "StreamSolver"`
The name of the file and subroutine.

`Variable String`
The name you want to call the solution, for example `StreamFunction`.

`Variable DOFs Integer 1`
The degree of freedom of the variable. Stream function is scalar so this must be set to 1.

`Stream Function Velocity Variable String`
The name of the velocity field variable. FlowSolvers solution is called `Flow Solution` and this is also the default value.

`Stream Function First Node` Integer
: Number of the node that is set to zero. If given, non-positive values are set to 1 and too large values are set to largest possible i.e. 'the last node'. If not given, then other means are assumed to be used for setting the level.

`Stream Function Penalty` Real
: When the level of the streamline is defined by a penalty formulation then this keyword is used to define the penalty factor c. Default is zero.

`Stream Function Shifting` Logical
: Shift the smallest value to zero. Default is `True`.

`Stream Function Scaling` Logical
: Scale largest absolute value to 1. Default is `False`.

`Stokes Stream Function` Logical
: This keyword forces the stream function type regardless of the coordinate system. If the coordinate system is axis symmetric, then the default is `True`, else the default is `False`.

46.5 Example

This example computes the streamlines from a 2D incompressible flow field assuming the default name for flow field

```
Solver 4
  Exec Solver = after all
  Equation = "streamlines"
  Procedure = "StreamSolver" "StreamSolver"
  Variable = String Stream

  Linear System Solver = "Iterative"
  Linear System Iterative Method = "cg"
  Linear System Preconditioning = ILU0
  Linear System Residual Output = 10
  Linear System Max Iterations = Integer 500
  Linear System Convergence Tolerance = 1.0e-10
  Linear System Abort Not Converged = False
End
```

Model 47

Flux Computation

Module name: FluxSolver
Module subroutines: FluxSolver
Module authors: Juha Ruokolainen, Peter Råback
Document authors: Peter Råback

47.1 Introduction

This module is used to calculate the fluxes resulting usually from Poisson kind of equations. These include, for example, the heat equation, the electrostatic equation, and the pressure equation for Darcy's flow. There are also flux computation subroutines that are built in the solvers but this provides a generic approach that should be easy to combine with most solvers.

47.2 Theory

Given a potential ϕ it is often interesting to know its gradient or the resulting flux. The gradient may be computed from $\nabla\phi$. The flux resulting from a potential field is assumed to be proportional to the gradient. The proportionality coefficient c may be conductivity, permeability, diffusivity, etc., depending on the application field. It may be a scalar or a tensor of second kind. The flux may now be expressed as

$$\vec{q} = -c\nabla\phi. \tag{47.1}$$

For the heat equation the potential would thus be the temperature and the conductivity would be the heat conductivity.

The magnitude of a flux (or gradient) may be defined as

$$|q| = |\vec{q}\cdot\vec{q}| = |c\nabla\phi \cdot c\nabla\phi| \tag{47.2}$$

The computation of magnitude may be done before or after the numerical discretization giving slightly different results.

47.3 Implementation issues

The flux may be computed in many ways. Often for visualization purposes it suffices to take some nodal average of the element-wise computed fluxes. The most consistent method for flux computation is, however, using the finite element method to solve the equation (47.1). The Galerkin method creates a diagonally dominated matrix equation that may be solved easily with iterative methods even with cheap preconditioners.

The flux computation may be done component-wise so that each component q_i, where $i = 1\ldots\text{dim}$, is solved separately. This saves a significant amount of memory even though it slightly complicates the implementation. In the solver it is also possible to choose just one component as could be sometimes desirable.

The main limitation of the current version is that it does not take into account any boundary conditions. Therefore, if there is an internal boundary over which the flux is not continuous, the calculated value does not make sense.

47.4 Keywords

`Solver` `solver id`

`Equation` `String Flux Solver`

`Procedure` `File "FluxSolver" "FluxSolver"`

`Discontinuous Galerkin` `Logical`
: For discontinuous fields the standard Galerkin approximation enforces continuity which may be unphysical. As a remedy for this, the user can enforce Discontinuous Galerkin (DG) formulation. Then the result may be discontinuous, and may even be visualized as such if the postprocessing format supports it.

`Average Within Materials` `Logical`
: This keyword enforces continuity within the same material in the DG discretization using the penalty terms of the DG formulation.

`Calculate Flux` `Logical`
: This flag controls the computation of fluxes. The default is `False`.

`Calculate Flux Abs` `Logical`
: In conjunction with flux computation this flag may be used to compute the absolute value of the flux vector. It requires that the previous flag is active.

`Calculate Flux Magnitude` `Logical`
: This flag can be used to compute the magnitude of the vector field. Basically it is the same in continuous level as the previous but this requires less memory and solves the matrix equation only once. The downside is that even negative values may be introduced.

`Calculate Grad` `Logical`
: This flag turns on gradient computation. The default is `False`.

`Calculate Grad Abs` `Logical`
: In conjunction with gradient computation this flag may be used to compute the absolute value of the gradient. It requires that the previous flag is active.

`Calculate Grad Magnitude` `Logical`
: This flag can be used to compute the magnitude of the gradient field. Basically it is the same in continuous level as the previous but this requires less memory and solves the matrix equation only once. The downside is that even negative values may be introduced.

`Enforce Positive Magnitude` `Logical`
: If this is active, then the negative values of the computed magnitude fields are a posteriori set to zero.

`Target Variable` `String "Temperature"`
: This gives the name of the potential variable used to compute the gradient. By default the variable is `Temperature`.

`Flux Coefficient` `String "Heat Conductivity"`
: This gives the name of the proportionality coefficient to compute the flux. By default the coefficient is `Heat Conductivity`.

The resulting linear system is easily solved even without preconditioning. Fox example, the following linear system control may be applied.

```
    Linear System Solver  "Iterative"
    Linear System Iterative Method  "BiCGStab"
    Linear System Preconditioning  None
    Linear System Max Iterations  500
    Linear System Convergence Tolerance  1.0e-10
```

47.5 Example

This example computes the diffusive heat flux in the whole domain after the whole solution has been performed.

```
Solver 3
  Exec Solver = after all
  Equation = "flux compute"
  Procedure = "FluxSolver" "FluxSolver"
  Calculate Flux = Logical True
  Flux Variable = String Temperature
  Flux Coefficient = String "Heat Conductivity"

  Linear System Solver = "Iterative"
  Linear System Iterative Method = "cg"
  Linear System Preconditioning = ILU0
  Linear System Residual Output = 10
  Linear System Max Iterations = Integer 500
  Linear System Convergence Tolerance = 1.0e-10
End
```

Model 48

Vorticity Computation

Module name: VorticitySolver
Module subroutines: VorticitySolver
Module authors: Peter Råback
Document authors: Peter Råback

48.1 Introduction

This module is used to calculate the vorticity of vector fields. Vorticity may be of interest mainly in the post-processing of flow fields or electromagnetic fields. The default name for the vorticity is `Curl varname`.

48.2 Theory

The vorticity $\vec{w}$ of a vector field $\vec{v}$ is obtained simply as the curl of the field,

$$\vec{w} = \nabla \times \vec{v}. \tag{48.1}$$

Component-wise the equations for the vorticity read

$$w_x = \frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \tag{48.2}$$

$$w_y = \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \tag{48.3}$$

$$w_z = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}. \tag{48.4}$$

Thus, all three components exist in 3D, while in 2D and axisymmetric cases only the z-component is present

The most consistent method for computing the vorticity in conjunction with the finite elements is to solve the equations (48.4) using the Galerkin method. The resulting matrix is diagonally dominated and the linear system may be solved easily with iterative methods even with poor preconditioners. In 3D the vorticity computation may be done component-wise so that each component w_i, where $i = 1, 2, 3$, is solved separately. This saves some memory and may also save in the overall time consumption.

48.3 Keywords

```
Solver  solver id

    Equation  String Vorticity Solver
    Procedure  File "VorticitySolver" "VorticitySolver"
```

`Target Variable` String "Velocity"
: This gives the name of the vector variable used to compute the vorticity. By default the variable is `Velocity`.

`Constant Bulk Matrix` Logical
: This keyword may be used to activate the saving of the stiffness matrix if the same solver is called repeatedly. The stiffness matrix depends only on geometric information and is hence the same if the geometry is unaltered.

The solver is easily solved even without preconditioning. Fox example, the following linear system control may be applied.

```
Linear System Solver  "Iterative"

Linear System Iterative Method  "cg"

Linear System Preconditioning  None

Linear System Max Iterations  500

Linear System Convergence Tolerance  1.0e-10
```

48.4 Example

This example computes the vorticity field after each timestep. Some resources are saved by reusing the same bulk matrix.

```
Solver 5
  Exec Solver = after timestep
  Equation = "vorticity"
  Procedure = "VorticitySolver" "VorticitySolver"
  Constant Bulk Matrix = True

  Linear System Solver = "Iterative"
  Linear System Iterative Method = "cg"
  Linear System Preconditioning = ILU0
  Linear System Residual Output = 10
  Linear System Max Iterations = Integer 500
  Linear System Convergence Tolerance = 1.0e-10
  Linear System Abort Not Converged = False
End
```

Model 49

Divergence Computation

Module name: DivergenceSolver
Module subroutines: DivergenceSolver
Module authors: Peter Råback
Document authors: Peter Råback

49.1 Introduction

This module is used to calculate the divergence of vector fields. Divergence may be of interest mainly in the postprocessing to check how well incompressibility constraints are honored.

49.2 Theory

The divergence d of a vector field $\vec{v}$ is obtained simply from

$$d = \nabla \cdot \vec{v}. \tag{49.1}$$

The most consist ant method for computing the divergence in conjunction with the finite elements is to solve the equation (49.1) using the Galerkin method. The resulting matrix is diagonally dominated and may be computed easily with iterative methods even with poor preconditioners.

49.3 Keywords

`Solver` solver id

`Equation` String Divergence Solver

`Procedure` File "DivergenceSolver" "DivergenceSolver"

`Target Variable` String "Velocity"
: This gives the name of the vector variable used to compute the divergence. By default the variable is `Velocity`.

`Constant Bulk Matrix` Logical
: This keyword may be used to activate the saving of the stiffness matrix if the same solver is called repeatedly. The stiffness matrix depends only on geometric information and is hence the same if the geometry is unaltered.

The following keywords are not usually needed as they are set by the initialization procedure of the solver.

`Variable String`

By default the variable is obtained from the divergence variable by adding a prefix `Div` to the field name. Naturally the name of the resulting field may also be given as desired.

The solver is easily solved even without preconditioning. Fox example, the following linear system control may be applied.

```
Linear System Solver "Iterative"
Linear System Iterative Method "cg"
Linear System Preconditioning None
Linear System Max Iterations 500
Linear System Convergence Tolerance 1.0e-10
```

Model 50

Scalar Potential Resulting to a Given Flux

Module name: ScalarPotentialSolver
Module subroutines: ScalarPotentialSolver
Module authors: Peter Råback
Document authors: Peter Råback

50.1 Introduction

This module is an auxiliary solver that may be used to compute the scalar potential that results to a given flux. The flux is assumed to be a vector field resulting from some computation. This solver is the dual of the `FluxSolver`. Computing first the flux of a given potential and thereafter resolving for the potential that creates the flux should give approximately the original potential.

50.2 Theory

The flux resulting from a potential field is assumed to be proportional to the gradient of the field, ϕ. The proportionality factor is here called conductivity, c. The flux may therefore be expressed as

$$q = -c\nabla\phi. \tag{50.1}$$

For heat equation the potential would be the temperature and the conductivity would be the heat conductivity.

This solver handles the equation in the reverse form, i.e. given the flux the potential is solved. The weak formulation is created by choosing the test functions to be the gradients of the shape functions. This results to the standard discretization of the Poisson equation.

The potential is not defined uniquely unless the level is fixed at least at one point. Therefore the user should set a Dirichlet condition at least at one node.

50.3 Keywords

`Solver` `solver id`

`Equation` `String ScalarPotentialSolver`

`Procedure` `File "ScalarPotentialSolver" "ScalarPotentialSolver"`

`Variable` `String "Scalar Potential"`
The desired name of the resulting scalar field.

`Flux Variable` String
: This gives the name of the flux variable used to compute the source term. Note that this must be the name of a vector field such as `Velocity`.

`Flux Coefficient` String
: This gives the name of the coefficient used in the computation of the flux. For example, in thermal analysis it would be `Heat Conductivity`. If a non-existing material parameter is given, the coefficient will be assumed to be the unity, i.e. $c = 1$.

The equation is a Poisson type of equation and defaults for it are set to be `cg+ILU0`. If these do not suffice, other linear system options should be defined.

`Boundary Condition` bc id

`Scalar Potential` Real
: The defined field variable must be set to be a constant at least at one point.

`Target Nodes` Integer
: The user may also define a target node on-the-fly at which the condition is set.

Model 51

Artificial Compressibility for FSI

Module name: ArtificialCompressibility
Module subroutines: CompressibilityScale
Module authors: Peter Råback
Document authors: Peter Råback

51.1 Introduction

When fluid-structure interaction (FSI) problems are solved with a loosely coupled iteration strategy there is a risk of applying unphysical boundary conditions that lead to severe convergence problems. The reason for this is that initially the fluid domain is unaware of the constraint of the structural domain, and vice versa. If the iteration converges this discrepancy will be settled, but sometimes the initial phase is so ill posed that convergence is practically impossible to obtain [4, 3].

The problem may be approached by applying the method of artificial compressibility to the fluid-structure interaction. Previously artificial compressibility has mainly been used as a trick to eliminate the pressure from the Navier-Stokes equations or to improve the convergence of the solution procedure [2, 6, 1]. Here the compressibility is defined so that it makes the fluid imitate the elastic response of the structure.

The method is best suited for cases where there is a direct correspondence between the pressure and the volume. Inertial forces and traction forces should be of lesser importance. The method might, for example, boost up the modeling of human arteries.

51.2 Theory

51.2.1 Fluid-structure interaction

The theoretical model with some results is thoroughly presented in

We look at the time-dependent fluid-structure interaction of elastic structures and incompressible fluid. The equations of momentum in the structural domain is

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \tau + \vec{f} \text{ in } \Omega_s, \tag{51.1}$$

where ρ is the density, $\vec{u}$ is the displacement, $\vec{f}$ the applied body force and $\tau = \tau(\vec{u})$ the stress tensor that for elastic materials may be locally linearized with $\vec{u}$. For the fluid fluid domain the equation is

$$\rho \left(\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) = \nabla \cdot \sigma + \vec{f} \text{ in } \Omega_f, \tag{51.2}$$

where $\vec{v}$ the fluid velocity and σ the stress tensor. For Newtonian incompressible fluids the stress is

$$\sigma = 2\mu\varepsilon(\vec{v}) - pI, \tag{51.3}$$

where μ is the viscosity, $\varepsilon(\vec{v})$ the strain rate tensor and p the pressure. In addition the fluid has to follow the equation of continuity that for incompressible fluid simplifies to

$$\nabla \cdot \vec{v} = 0 \text{ in } \Omega_f. \tag{51.4}$$

For later use we, however, recall the general form of the continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \text{ in } \Omega_f. \tag{51.5}$$

The fluid-structure interface, Γ_{fs}, must meet two different boundary conditions. At the interface the fluid and structure velocity should be the same,

$$\vec{v}(\vec{r}, t) = \dot{\vec{u}}(\vec{r}, t), \;\; \vec{r} \in \Gamma_{fs}. \tag{51.6}$$

On the other hand, the surface force acting on the structure, $\vec{g}_s$, should be opposite to the force acting on the fluid, $\vec{g}_f$, thus

$$\vec{g}_s(\vec{r}, t) = -\vec{g}_f(\vec{r}, t), \;\; \vec{r} \in \Gamma_{fs}. \tag{51.7}$$

A widely used iteration scheme in FSI is the following: First, assume a constant geometry and solve the Navier-Stokes equation for the fluid domain with fixed boundary conditions for the velocity. Then calculate the surface forces acting on the structure. Using these forces solve the structural problem. Using the resulting displacement velocities as fixed boundary conditions resolve the fluid domain. Continue the procedure until the solution has converged.

The above described iteration usually works quite well. However, in some cases the boundary conditions (51.6) and (51.7) lead to problems. The elasticity solver is not aware of the divergence free constraint of the velocity field. Therefore the suggested displacement velocities used as boundary conditions may well be such that there is no solution for the continuity equation. A proper coupling method makes the solution possible even if the velocity boundary conditions aren't exactly correct. Further, if the Navier-Stokes equation is solved without taking into account the elasticity of the walls, the forces in equation (51.7) will be exaggerated. The pathological case is one where all the boundaries have fixed velocities. Then even an infinitely small net flux leads to infinite pressure values. A proper coupling method should therefore also give realistic pressure values even with inaccurate boundary conditions. The method of artificial compressibility meets both these requirements.

51.2.2 Artificial compressibility

When a surface load is applied to an elastic container it results to a change in the volume. In many cases of practical interest the change in volume is mainly due to a pressure variation from the equilibrium pressure that leads to zero displacements. If the structural domain is described by linear equations the change in volume dV has a direct dependence on the change in the pressure, dP, or

$$\frac{dV}{V} = c\, dP. \tag{51.8}$$

This assumption limits the use of the model in highly nonlinear cases.

The change in the volume should be the same as the net volume flux into the domain. As this cannot be guaranteed during the iteration, some other way to enable the material conservation must be used. A natural choice is to let the density of the fluid vary so that is has the same pressure response as the elastic walls,

$$\frac{d\rho}{\rho} = c\, dP, \tag{51.9}$$

where c is the artificial compressibility. This is interpreted locally and inserted to the continuity equation (51.5) while neglecting the space derivative of the density, thus

$$c\, \frac{dp}{dt} + \nabla \cdot \vec{v} = 0, \tag{51.10}$$

where dp is the local pressure change. Here the time derivative of pressure must be understood as an iteration trick. A more precise expression is

$$\frac{c}{\Delta t}\left(p^{(m)} - p^{(m-1)}\right) + \nabla \cdot \vec{v}^{(m)} = 0, \tag{51.11}$$

where m is the current iteration step related to fluid-structure coupling. When the iteration converges $p^{(m)} \to p^{(m-1)}$ and therefore the modified equation is consistent with the original one. The weak form of the equation for finite element method (FEM) may easily be written,

$$\int_{\Omega_f} (\nabla \cdot \vec{v}^{(m)})\varphi_p \, d\Omega + \frac{1}{\Delta t}\int_{\Omega_f} c\left(p^{(m)} - p^{(m-1)}\right)\varphi_p \, d\Omega = 0, \tag{51.12}$$

where φ_p is the test function.

The artificial compressibility may be calculated analytically in simple geometries. For example, for a thin cylinder with thickness h and radius R the compressibility is $c = 2R/Eh$ [5], where E is the Young's modulus, and correspondingly for a sphere $c = 3R/Eh$.

In most practical cases the elastic response of the structure cannot be calculated analytically. Then the compressibility may also be computed from equation (51.8) by applying a pressure change dP to the system,

$$c = \frac{1}{V}\frac{dV}{dP}. \tag{51.13}$$

The change in volume may be calculated by comparing it to initial volume, thus

$$c = \frac{V - V_0}{V_0}\frac{1}{dP}. \tag{51.14}$$

For small deformations $ds = \vec{u} \cdot \vec{n}$, where $\vec{n}$ is the surface normal. Therefore we may use an alternative form convenient for numerical computations,

$$c = \frac{\int_{\Gamma_{fs}} (\vec{u} \cdot \vec{n}) \, dA}{\int_{\Omega_f} dV} \frac{\int_{\Gamma_{fs}} dA}{\int_{\Gamma_{fs}} dp \, dA}. \tag{51.15}$$

This way c has a constant value over the domain.

51.2.3 Scaling artificial compressibility

If the artificial compressibility distribution is a priori defined we may use the above equations to scale the compressibility appropriately. For example, the compressibility could be given only within a limited distance from the elastic wall. and the functional behavior of $c(\vec{r})$ would be user defined. Computing compressibility becomes then just a matter of scaling,

$$c(\vec{r}) = c_0(\vec{r}) \underbrace{\frac{\int_{\Gamma_{fs}} (\vec{u} \cdot \vec{n}) \, dA}{\int_{\Omega_f} c_0(\vec{r}) dV} \frac{\int_{\Gamma_{fs}} dA}{\int_{\Gamma_{fs}} dp \, dA}}_{\text{scaling factor}}. \tag{51.16}$$

A suitable test load for computing compressibility is the current pressure load on the structure. However, for the first step the compressibility must be predefined. It is safer to over-estimate it since that leads to too small a pressure increase. Too large a pressure increase might ruin the solution of the elasticity solver and by that also the computational mesh used by the flow solver would be corrupted. Therefore some sort of exaggeration factor exceeding unity might be used to ensure convergence.

51.2.4 Elementwise artificial compressibility

If the displacement field is extended smoothly throughout the whole geometry it may be possible to define the artificial compressibility separately for each element or node. This is particularly useful for geometries where the elastic response changes significantly. The equation is now similar to (51.14),

$$c = \frac{V^e - V_0^e}{V_0^e}\frac{1}{dP}, \tag{51.17}$$

where the superscript e refers to the volume of an element. This may also be solved using finite element strategies to get nodal values for c.

51.3 Keywords

Keywords of FlowSolve

`Material` mat id
: In the material section the compressibility model and the initial artificial compressibility field is given.

`Compressibility Model` String [Artificial Compressible]
: Set the material model of the fluid.

`Artificial Compressibility` Real
: The initial value of artificial compressibility. This may also be a distributed function that is then scaled by the solver.

Keywords of solver CompressibilityScale

If the artificial compressibility is tuned so that it best imitates the elastic response, a additional solver must be used to rescale the above mentioned compressibility. The solver computes the total compressibility and the force acting on the surface. The compressibility is integrated over all volumes that are solved with the Navier-Stokes equation.

`Solver` solver id

`Equation` String CompressibilityScale
: The name of the solver.

`Procedure` File "ArtificialCompressibility" "CompressibilityScale"
: The subroutine in the dynamically linked file.

`Steady State Convergence Tolerance` Real
: How much the relative value of the compressibility may change between iterations, abs$(c_i - c_{i-1})/c_i < \varepsilon$.

`Nonlinear System Relaxation Factor` Real
: Relaxation scheme $c_i' = \lambda c_i + (1 - \lambda)c_{i-1}$ for the compressibility. By default is $\lambda = 1$.

`Boundary Condition` bc id

`Force BC` Logical
: The elastic response is calculated over the surface(s) which has this definition as `True`.

Keywords of solver CompressibilitySolver

When the compressibility is solved elementwise using this solver there has to usually be a isobaric steady-state test phase where the compressibility is defined. For this solver all the normal `Linear System` keywords also apply.

`Solver` solver id

`Equation` String CompressibilitySolver

`Procedure` File "ArtificialCompressibility" "CompressibilitySolver"

`Variable` String ac
The name of the artificial compressibility field variable.

`Displacement Variable Name` String "Mesh Update"
The name of the displacement field variable that is used to compute the the volume change.

`Displaced Shape` Logical True
Flag that defines whether the current shape is the displaced or original shape.

`Reference Pressure` Real
The value of pressure used for the test loading.

The computed field should then be given as the value in the material section.

`Material` mat id

`Artificial Compressibility` Equals ac
The initial value of artificial compressibility given by the solver.

51.3.1 Examples

The examples show a 2D square and a 3D cube being gradually filled. The fluid comes in from one wall and the opposing elastic wall makes room for the fluid so that the continuity equation is satisfied. Here the value of artificial compressibility is scaled every timestep to account for the nonlinear elasticity.

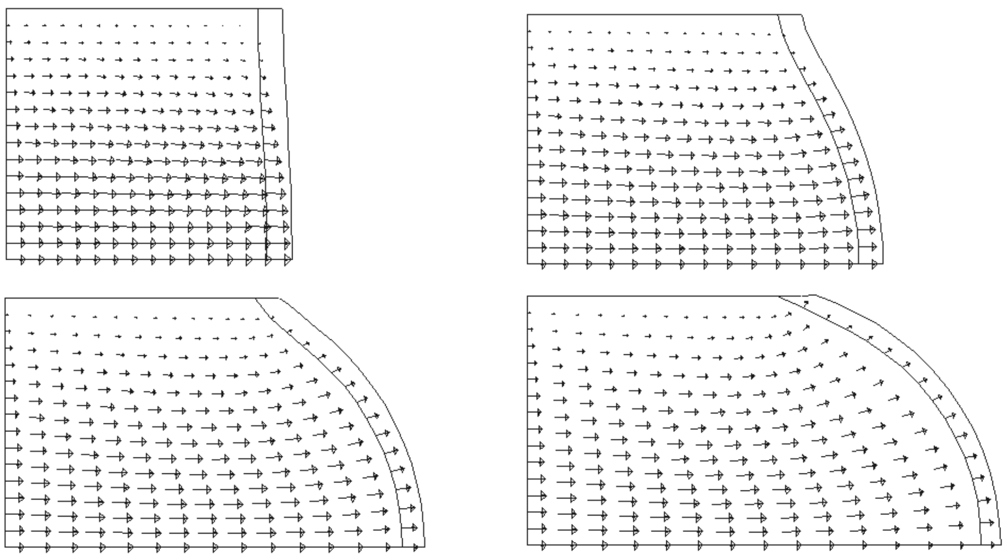

Figure 51.1: Snapshots of an elastic square being gradually filled by incompressible fluid.

Bibliography

[1] F. D. Carter and A. J. Baker. Accuracy and stability of a finite element pseudo-compressibility cfd algorithm for incompressible thermal flows. *Num. Heat Transfer, Part B*, 20:1–23, 1991.

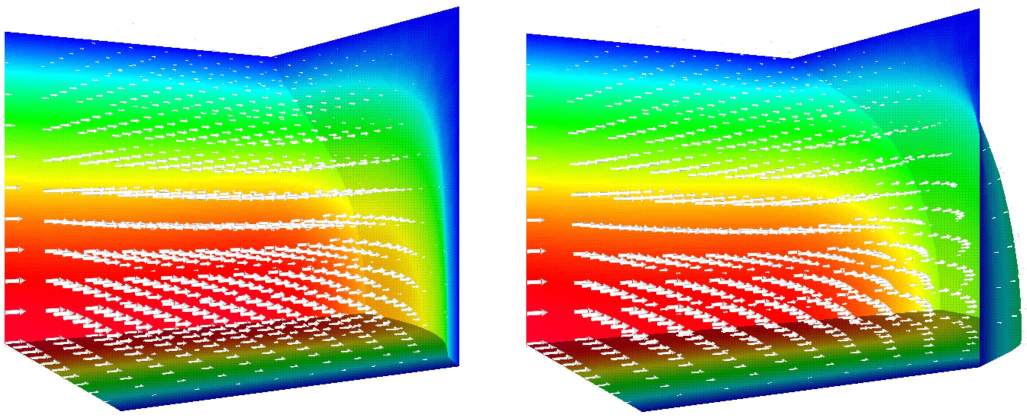

Figure 51.2: Snapshots of an elastic cube being gradually filled by incompressible fluid.

[2] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *J. Comput. Phys.*, 135:118–125, 1997.

[3] L. Formaggia, J. F. Gerbeau, F. Nobile, and A. Quarteroni. Numerical treatment of defective boundary conditions for the navier-stokes equations. *EPFL-DMA Analyse et Analyse Numerique*, 20, 2000.

[4] E. Järvinen, M. Lyly, J. Ruokolainen, and P. Råback. Three dimensional fluid-structure interaction modeling of blood flow in elastic arteries. In *Eccomas Computational Fluid Dynamics Conference*, September 2001.

[5] Kris Riemslagh, Jan Vierendeels, and Erik Dick. An efficient coupling procedure for flexible wall fluid-sructure interaction. In *Eccomas Congress on Comp. Meth. in Appl. Sci. and Eng*, September 2000.

[6] S. E. Rogers, D. Kwak, and U. Kaul. On the accuracy of the pseudocompressibility method in solving the incompressible navier-stokes equations. *Appl. Math. Modelling*, 11:35–44, 1987.

Model 52

Fluidic Force

Module name: FluidicForce
Module subroutines: ForceCompute
Module authors: Juha Ruokolainen, Antti Pursula
Document authors: Antti Pursula

52.1 Introduction

This module is used to calculate the force that a fluid flow induces on a surface. The fluidic force can be divided into two main components: force due to pressure and viscous drag force. The fluid can be compressible or incompressible and also non-Newtonian with the same limitations that there are in the Elmer Navier-Stokes Equation solver. The force calculation is based on a flow solution (velocity components and pressure) which has to be present when calling the procedure. Also the torque with respect to a given point can be requested.

52.2 Theory

The force due to fluid is calculated as a product of the stress tensor and normal vector integrated over the surface

$$\vec{F} = \int_S \overline{\overline{\sigma}} \cdot \vec{n}\, dS. \tag{52.1}$$

The stress tensor is

$$\overline{\overline{\sigma}} = 2\mu\overline{\overline{\varepsilon}} - \frac{2}{3}\mu(\nabla \cdot \vec{u})\overline{\overline{I}} - p\overline{\overline{I}}, \tag{52.2}$$

where μ is the viscosity, $\vec{u}$ is the velocity, p is the pressure, $\overline{\overline{I}}$ the unit tensor and $\overline{\overline{\varepsilon}}$ the linearized strain rate tensor, i.e.

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{52.3}$$

The torque about a point $\vec{a}$ is given by

$$\vec{\tau} = (\vec{r} - \vec{a}) \times \vec{F}(\vec{r}), \tag{52.4}$$

where $\vec{r}$ is the position vector.

52.3 Additional output

There is also a feature for saving the tangential component of the surface force i.e. the shear stress elemen-twise on the boundaries. The shear stress output is written on disk in a file which contains three columns: 1) the value of the shear stress, 2 and 3) the corresponding x and y coordinates. The shear stress is saved on all boundaries where fluidic force computation is requested. This feature is implemented only for 1D-boundaries of 2D-geometries.

52.4 Keywords

`Solver` `solver id`

- `Equation` `String Fluidic Force`
- `Procedure` `File "FluidicForce" "ForceCompute"`
- `Calculate Viscous Force` `Logical [True]`
  Setting this flag to false disables the viscous drag force, and only the surface integral of pressure is calculated.
- `Sum Forces` `Logical [False]`
  By default the solver calculates the fluidic force by boundaries. Setting this flag to True applies summing of each individual boundary force in to a resultant force which is the only force vector in output.
- `Shear Stress Output` `Logical [False]`
  Setting this flag to True activates writing shear stress values on disk.
- `Shear Stress Output File` `String [shearstress.dat]`
  Defines the name of the shear stress file.
- `Velocity Field Name` `String`
  The name of the velocity field variable. This keyword may be necessary if some other flow solver than the built-in Navier-Stokes solver of Elmer is used. Normally this keyword should be omitted.

`Material` `mat id`

- `Viscosity` `Real`

`Boundary Condition` `bc id`

- `Calculate Fluidic Force` `Logical [True]`
  The fluidic force is calculated for the surfaces where this flag is set to true.
- `Moment About(dim)` `Real`
  Coordinates for the point on which the torque is returned.

Model 53

Electrostatic force

Module name: ElectricForce
Module subroutines: StatElecForce
Module authors: Antti Pursula
Document authors: Antti Pursula

53.1 Introduction

This solver calculates the electrostatic force acting on a surface. The calculation is based on an electrostatic potential which can be solved by the electrostatic solver (see Model 16 of this Manual).

53.2 Theory

The force is calculated by integrating the electrostatic Maxwell stress tensor [1] over the specified surface. Using the stress tensor $\overline{\overline{T}}$ the total force on the surface S can be expressed as

$$\vec{F} = \int_S \overline{\overline{T}} \cdot \, d\vec{S}. \tag{53.1}$$

The components of the Maxwell stress tensor for linear medium are

$$T_{ij} = -D_i E_j + \frac{1}{2}\delta_{ij}\vec{D}\cdot\vec{E}, \tag{53.2}$$

where electric field $\vec{E}$ and electric displacement field $\vec{D}$ are obtained from the electric potential Φ

$$\vec{E} = -\nabla\Phi, \tag{53.3}$$

and

$$\vec{D} = -\varepsilon_0\varepsilon_r\nabla\Phi, \tag{53.4}$$

where ε_0 is the permittivity of vacuum and ε_r is the relative permittivity of the material, which can be a tensor.

53.3 Keywords

```
Constants

    Permittivity Of Vacuum  Real [8.8542e-12]

Solver  solver id
```

`Equation` `String Electric Force`
: The name of the equation. Not necessary.

`Procedure` `File "ElectricForce" "StatElecForce"`

`Exec Solver` `String After Timestep`
: Often it is not necessary to calculate force until solution is converged.

`Material` `mat id`

`Relative Permittivity` `Real`

`Boundary Condition` `bc id`

`Calculate Electric Force` `Logical True`
: This keyword marks the boundaries where force is calculated.

Bibliography

[1] J. Vanderlinde. *Classical electromagnetic theory*. John Wiley & Sons, 1993.

Model 54

System Reduction for Displacement Solvers

Module name: RigidBodyReduction
Module subroutines: RigidBody
Module authors: Antti Pursula
Document authors: Antti Pursula

54.1 Introduction

This module is used to reduce and simplify the computation of a displacement solver when the problem includes rigid blocks. In such a case, it is often difficult for iterative solvers to find a solution for the full system, and direct solvers become obsolete when the system is large enough. The convergence and also the speed of the solution can be substantially improved when the degrees of freedom corresponding to the nodes belonging in the rigid blocks are reduced onto the 6 DOFs (3 in 2D) of the corresponding rigid body. In the module, the reduction is achieved via a projection matrix.

Additionally, the routine automatically eliminates the degrees of freedom corresponding to the Dirichlet boundary conditions. It is also possible to request the elastic regions to be extended into the rigid blocks. There is also possibility to reorder the reduced matrix elements to decrease its bandwidth.

54.2 Theory

The module starts with normally constructed matrix equation for the unknown displacements x, $Ax = b$. Let us assume that the nodes are ordered in such a way that the first n elements of the vectors correspond to the elastic parts of the structure and the remaining m elements correspond to the rigid parts of the structure. The goal is to reduce the $(n + m) \times (n + m)$ matrix A to a $(n + \alpha k) \times (n + \alpha k)$ matrix B, where k is 3 for 2D and 6 for 3D problems and α is the number of rigid blocks present. Reductions are made also for the vectors so that finally the matrix equation reads $Bu = f$.

The relation between the unknowns is

$$x = Pu, \tag{54.1}$$

where the projection matrix P ties the nodes in the rigid bodies to the same displacements in coordinate directions and the same rotations about the coordinate axis. The rotations are defined with a coordinate system whose origin is at the center of each rigid body. For the right hand sides we can write

$$f = Qb, \tag{54.2}$$

where the matrix Q sums the forces and torques present at the nodes in rigid bodies for a resultant force and torque of the center point of the corresponding rigid body. In both mappings, the rotations are linearized so the module is valid only for cases where the rotations are small.

Using these definitions, we have

$$Ax = APu = b \tag{54.3}$$

and

$$Bu = f = Qb. \tag{54.4}$$

Combining the equations gives $Bu = QAPu$ and thus

$$B = QAP. \tag{54.5}$$

With a suitable order of the rotations one can write

$$Q = P^T \equiv C, \tag{54.6}$$

and

$$B = CAC^T. \tag{54.7}$$

The matrix C has a identity matrix block of size $n \times n$ which keeps the elastic nodes intact, and a projection block of size $\alpha k \times m$.

The reduced order solution u is transformed back to the original nodes by the same mapping

$$x = C^T u. \tag{54.8}$$

54.3 Applicable cases and limitations

The module works for

- Linear steady-state problems
- Linear transient problems
- Eigen analysis
- Quadratic eigenproblems

There are following limitations:

- Rigid blocks should not have common nodes (there should be elastic nodes in between rigid blocks)
- If a Dirichlet bc is given on a node of a rigid block then the entire rigid block is assumed to be fixed in all directions

54.4 Keywords

`Body` `body id`

`Rigid Body` `Logical`
Value `True` defines the rigid body.

`Solver` `solver id`
The module does not need a separate solver but a call in the stress analysis, or the elasticity solver in the linear mode.

`Equation` `String Stress Analysis`

`Variable` `String Displacement`

`Variable DOFs` `Integer`
It is important to give the DOFs right, either 2 or 3 depending on the dimension.

`Before Linsolve` `File "RigidBodyReduction" "RigidBody"`
: The model order reduction is performed after the matrix has been assembled but before the matrix equation has been solver. The matrix equation is modified to a smaller equation and the new equation is solved within the subroutine.

`Eigen Analysis` `Logical`
: It is possible to use the model order reduction with modal analysis, as well as with static and transient cases.

`Eigen System Values` `Integer`
: The number of eigen values to be computed.

`Eigen System Damped` `Logical`

`Eigen System Use Identity` `Logical [True]`
: The reduction is possible also with quadratic (damped) eigenproblems.

`Optimize Matrix Structure` `Logical`
: If true, the matrix structure is optimized. This feature is recommended since the reduced matrix has often very scattered structure. The optimization is performed with the Cutholl-McKee algorithm.

`Reverse Ordering` `Logical`
: This flag can be used to reverse the matrix ordering if the matrix structure is optimized, resulting in reverse Cuthill-McKee ordering.

`Extend Elastic Region` `Logical`
: If true, the elastic regions of the geometry are extended into the rigid block. This feature allows taking into account the bending in the joints between elastic and rigid parts.

`Extend Elastic Layers` `Integer`
: Defines the number of element layers that the elastic regions are extended.

`Output Node Types` `Logical`
: Writes in the ElmerPost output file a variable describing the status of each node in the geometry. The variable has value 0 for elastic nodes, -1 for rigid blocks that are fixed due to a Dirichlet boundary condition, and a positive integer for separate rigid blocks. The variable may be used to check that the reduction is performed on the right blocks, and to check how many layers the elastic regions should be extended, for example.

`Additional Info` `Logical`
: If true, additional information is written about the performed tasks during the simulation.

54.5 Examples

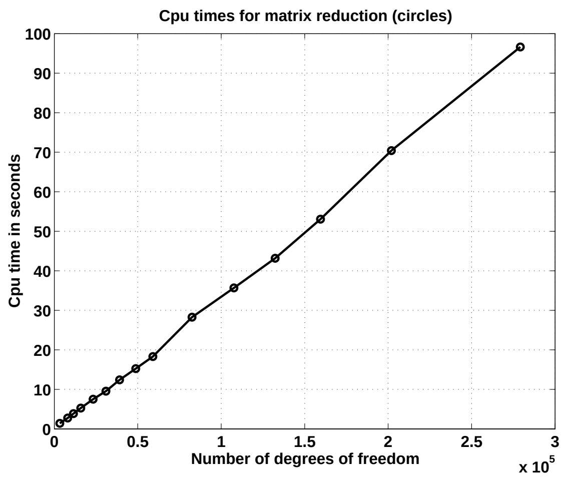

Figure 54.1: The cpu time required for the matrix reduction operations depends linearly on the degrees of freedom in the system.

54.6 Interpolate, Filter, and Find Data

Model 55

Filtering Time-Series Data

Module name: FilterTimeSeries
Module subroutines: FilterTimeSeries
Module authors: Peter Råback
Document authors: Peter Råback
Document updated: 13.2.2008

55.1 Introduction

The module includes auxiliary utilities for filtering time-series data. Supported filters include various averaging possibilities and Fourier series, for example. The solver does not introduce any new physics. However, it may be useful in analyzing time-dependent data to be used in conjunction with time-harmonic models, or in studying phenomena with different timescales (turbulence).

55.2 Theory

Mean of a function

The solver is built so that an estimate for the filtered data may be obtained at all times i.e. the normalizing is done after each timestep. As an example let's consider taking a simple mean over a period of time. The starting point is the time averaged mean,

$$< f >_T = \frac{1}{T} \int_0^T f(t)\, dt. \tag{55.1}$$

Its discrete counterpart assuming piecewise constant integration is

$$< f >_n = \frac{1}{T_n} \sum_i^n f_i\, dt_i, \tag{55.2}$$

where $T_n = \sum dt_i$. Now this may be presented inductively as

$$< f >_n = \frac{T_{n-1} < f >_{n-1} + f_n dt_n}{T_{n-1} + dt_n} \tag{55.3}$$

$$T_n = T_{n-1} + dt_n. \tag{55.4}$$

Weighted mean

It's also possible to take a weighted mean with a user defined function $g(t)$ depending on time only. Then similarly,

$$< fg >_n = \frac{T_{n-1} < fg >_{n-1} + f_n g_n dt_n}{T_{n-1} + dt_n}. \tag{55.5}$$

Fourier series

Using the weighted mean as starting point its possible to present the solution in terms of sine and cosine series. In order to obtain normalized Fourier series components the function g is internally replaced by sine and cosine functions defined as $2\sin(2\pi kwt)$ and $2\cos(2\pi kwt)$, where k is the degree of the term, and w is the user defined frequency. After each full cycle the inner product then includes the Fourier coefficients and the transient solution may hence be approximated by

$$f \approx \sum_{k=1}^{m_s} s_k \sin(2\pi kwt) + \sum_{k=1}^{m_c} c_k \cos(2\pi kwt), \tag{55.6}$$

where m_s and m_c are maximum degrees defined by the user.

Continuous average

Sometimes it may be useful that the new solution is given a relatively higher weight than the old solution. This is achieved by relaxing the weight (elapsed time) related to the old solution by

$$T_{n-1} := T_{n-1} \exp(-dt_n/\tau), \tag{55.7}$$

where τ is the time scale when decay to fraction $1/e$ is desired. If the decay time is short compared to the overall simulation time this provides a continuous mean that represents only the recent results. The fraction of the last timestep in solution will always be dt/τ.

Computing variances

It is not possible to compute the variance directly with one sweep as computing the variance from the functional values requires the knowledge of the mean. However, computing the mean of the square of the solution enables that the variance is computed a posteriori since the following holds for any field variable,

$$\sigma^2 =< (f - < f >)^2 >=< f^2 > - < f >^2 . \tag{55.8}$$

55.3 Keywords

`Solver` `solver id`

`Procedure` `File "FilterTimeSeries" "FilterTimeSeries"`

`Variable i` `String`
: The names of the variables to be filtered. There can in principle be up to 99 variables. Note that the keywords with the same `i` form a set which define one filtering. If the `Variable` is not redefined the previously defined variable with a lower `i` is used.

`Operator i` `String`
: Normally the variable is treated as its plain value. There are however different options for using the field value in a modified manner. These include `length` (L2 norm), `abs`, and `square`.

`Start Time i` `Real`
: The start time for performing the integration. Note that for Fourier series this is used to reset the zero time i.e. $t := t - t_0$.

`Stop Time i` `Real`
: The stop time for performing the integration.

`Start Timestep i` `Integer`
: Sometimes its unpractical to compute the start time. For example, the start of the simulation could include a starting strategy with a number of timesteps. Then the number of timesteps that starts the averaging may be given by this keyword. Note that this keyword also activates timestep-insensitive averaging.

`Stop Timestep i` `Integer`
: The timestep number that ends the averaging.

`Start Cycle i` `Real`
: Alternative way to give the start time for sine and cosine series. The start time is the inverse of this.

`Stop Cycle i` `Real`
: Alternative way to give the stop time for sine and cosine series.

`Start Real Time` `Real`
: Start after given real wall-clock-time, rather than physical simulation time.

`Start Real Time Fraction` `Real`
: Relative way of given start time when the `Real Time Max` keyword in `Simulation` block is given.

`Reset Interval i` `Real`
: The time interval at which the computation of a mean is reinitialized.

`Decay Time i` `Real`
: The decay time τ in computing continuous means.

`Decay Timestep i` `Real`
: The number of timestep needed to perform averaging. If the timestep given is N then the decay of the previous timesteps is done with $\exp(-1/N)$. Note that this keyword also activates timestep-insensitive averaging.

`Time Filter i` `Real`
: The function $g(t)$ that may be used in computing the mean.

`Sine Series i` `Integer`
: The number of terms in the sine series. Note that its possible to make a Fourier series only if the target variable is a scalar. Its also possible to have only one sine or cosine series at a time.

`Cosine Series i` `Integer`
: The number of terms in the cosine series.

`Frequency i` `Integer`
: If using cosine or sine series the frequency must be given.

Model 56

Data to field solver

Module name: DataToFieldSolver
Module subroutines: DataToFieldSolver
Module authors: Peter Råback
Document authors: Peter Råback

56.1 Introduction

This subroutine may be used fit data to a continuous field. In principle just a simple identity equation with some diffusion for regularization is solved. The origin of the data could be from a particle trace of a Monte-Carlo simulation, measurement data read from an external file, etc.

56.2 Theory

Galerkin method minimizes the L2 norm of the solution and is therefore a good choice also for problems where a regular field must be fitted into data. Imagine a case where we have an initial field $\hat{f}$ and want to a field f on it. Then we need to solve an equation $f = \hat{f}$ with the Galerkin method. Now if the initial data is noisy the fitting should include some regularization. The natural way to introduce regularization in this context is to add artificial diffusion such that we are really solving for

$$-\nabla \cdot D\nabla f + f = \hat{f} \tag{56.1}$$

which is a linear diffusion-reaction equation. The weak formulation for this is

$$\int D\nabla f \cdot \nabla\psi \, d\Omega - \int D\frac{\partial f}{\partial n} \, d\Gamma \int f \, d\Omega = \int \hat{f} \, d\Omega \tag{56.2}$$

Now the data could be given already in the integral form i.e. as

$$\hat{F} = \int \hat{f} \, d\Omega. \tag{56.3}$$

Also the user may know the weight w that was accumulated when the data was generated.

The data may not always be accurate or even given. Therefore using the given data in all fields might not be a good idea. We may introduce a masking field, say $p \in [0, 1]$, that picks the values that are considered as accurate. On the other hand diffusion, might not be needed for those points.

Adding these generalizations we obtain the following form

$$\int (1-p)D\nabla f \cdot \nabla\psi \, d\Omega - \int D(1-p)\frac{\partial f}{\partial n}\psi \, d\Gamma + \int p \, w \, f\psi \, d\Omega = \int p \, \hat{f}\psi \, d\Omega + p \, \hat{F}. \tag{56.4}$$

This formulation enables the use of various techniques for data fitting.

56.3 Keywords

Below are the dedicated keywords for the solver. In addition generic keywords such as many of those related to `Linear System` solution methods may be used.

`Solver` `solver id`

`Equation` `String [DataToFieldSolver]`
: The name of the equation.

`Procedure` `File "DataToFieldSolver" "DataToFieldSolver"`
: The name of the procedure.

`Variable` `String [FieldName]`
: The name of the resulting field.

`Target Variable` `String`
: Name of the scalar field to be used as input data. This is either $\hat{f}$ or $\hat{F}$

`Diffusion Coefficient` `Real`
: The value of the diffusion constant used in the regularization.

`Normalize by Nodal Weight` `Logical`
: Normalize the particle properties with nodal weight i.e. the standard mass matrix. This would be a good choice if the right-hand-side is a result of an integral i.e. $\hat{F}$.

`Normalize by Given Weight` `Logical`
: Normalize the particle properties i.e. divide the trace with the corresponding weight. Basically this corresponds to the integral over w. Here an integral data set is assumed such that it is consistent with the given weight.

`Weight Variable` `String`
: If there is a weight associated to the data the name of it should be given with this function.

`Set Constant Weight Sum` `Logical`
: If the weights are provided by the user and this flag is set `True` then the given weights are normalized so that their sum is always the same as the sum of nodal weights. The idea is that the solution will be independent on the given level of the weight, and only depend on the relative sizes.

`Mask Variable` `T`
: he data points may selectively chosen to be considered. If a mask variable is given it may have both lower and upper boundaries. Only if the mask variable is between these bounds will the data be considered. Effectively $p = 1$ in the interval, and $p = 0$ outside the interval.

`Max Mask Value` `Real`
: Maximum value for the mask interval. Default is `+HUGE`.

`Min Mask Value` `Real`
: Minimum value for the mask interval. Default is `-HUGE`, except if the maximum interval is not given either. Then the default is `0`.

`Mask Diffusion` `Logical`
: If the mask is true, should diffusion be neglected. Default is `False`.

`Boundary Condition` `bc id`

`FieldName` `Real`
: The user may set Dirichlet conditions for the field to be fitted if they are known.

`FieldName Continue` `Logical`
: Enforce boundary conditions with constant slope. The default boundary condition is otherwise the natural boundary condition with zero gradient.

Model 57

Projection to plane

Module name: ProjectToPlane
Module subroutines: ProjectToPlane,ParallelProjectToPlane
Module authors: Juha Ruokolainen and Peter Råback
Document authors: Peter Råback

57.1 Introduction

Sometimes the solution of a complex problem calls for a dimensional reduction of some field variables. A possible scenario for using the solver is in extracting some useful information from DNS or LES type of flow simulations. This module offers the subroutines needed in such a cases.

There are currently a serial and a parallel version of the subroutine which are both located in this module. The reason for the fork is that the parallel version uses techniques that are not optimal in the serial problem. Therefore the user should herself choose the correct version. Optimally these two approaches should of course be fused.

57.2 Theory

In principle the dimensional reduction is performed taking on average of a 2D (or axisymmetric) nodal point $\vec{r}_{2D}$ when it travels through the 3D mesh.

$$f_{2D}(\vec{r}_{2D}) = \frac{1}{S} \int f(\vec{r}_{3D})\, dS. \tag{57.1}$$

In practice this is implemented with the following steps

1. Create a list of faces for the 3D mesh
2. Loop though each nodal point in the 2D mesh
   (a) Loop through each face in the 3D mesh
      i. Check if there is an intersection between the integration line and face
      ii. If intersection found memorize the point of intersection
   (b) Order the intersection points in the integration direction
   (c) Take a weighted average over the ordered list, (f_i, r_i)

The algorithm is accurate for linear elements. For higher order elements it is suboptimal in accuracy. Also in axisymmetric mapping the elements should be small enough so that the curvature of the line segment is not significant. Near the origin there may be few hits and then the averaging is done by just taking a small number of values around the center axis.

57.3 Keywords

`Solver` `solver id`

`Equation` `ProjectToPlane`
: The arbitrary name of the equation.

`Procedure` `File "ProjectToPlane" "ProjectToPlane"`
: or

`Procedure` `File "ProjectToPlane" "ParallelProjectToPlane"`

`Convert From Equation Name` `String`
: The solver needs a 3D mesh which is associated determined by the association to the solver given by this keyword.

`Convert From Variable` `String`
: The variable to be converted.

`Volume Permutation` `Integer`
: The algorithm is build so that integration direction is the second coordinate (y). This is typically valid for axisymmetric cases, for example. If the integration should be performed with respect to some other direction the volume coordinates may be permuted by this keyword.

`Plane Permutation` `Integer`
: Permutation of the plane coordinates.

`Rotate Plane` `Logical`
: Should rotation be performed.

`Max Relative Radius` `Real`
: For the axisymmetric projection the outer radius may be difficult since the 3D mesh typically may have faces that do not quite extend to the surface. This is a result of finite sized linear elements. To ease this problem the user may give the maximum relative radius that is used when trying to find the point of intersection.

`Minimum Hits At Radius` `Integer`
: The number of hits needed for a accepted integration. The default is one.

`Integration Points At Radius` `Integer`
: If no minimum number of hits is achieved then a few points around the axis are used to determine the value. The default is two.

Model 58

Structured projection to plane

Module name: StructureProjectToPlane
Module subroutines: StructureProjectToPlane
Module authors: Peter Råback
Document authors: Peter Råback

58.1 Introduction

For structured meshes some operations may be done much more effectively than for generic meshes. One such operation is mapping some values within the mesh to the reduced dimensional surface of the mesh. In practice this could mean, for example, mapping the values at the bottom of mesh to the top of mesh to determine the difference in value. Or to map some isosurface values to the top of the mesh.

58.2 Theory

The algorithm used for the mapping has two sweeps. Assume that we would like to perform mapping in direction $\vec{e}_z$. At the first sweep over all elements we would deduce pairwise information over nodes on which nodes are in `up` and `down` directions from each other. Then using this directional information recursively one can easily deduce which nodes are the `top` and `bottom` representatives of any node in the mesh. With this information mapping information to top or bottom nodes, or to the whole mesh becomes extremely cheap.

58.3 Keywords

`Solver` `solver id`

`Equation` `String [StructuredProjectToPlane]`
The name of the equation.

`Procedure` `File "StructuredProjectToPlane" "StructuredProjectToPlane"`
The name of the procedure.

`Velocity Variable Name` `String`
Name of the variable used to define the streamlines.

`Active Coordinate` `Integer`
The direction in which the structured mapping is performed i.e. 1, 2 or 3.

`Project To Bottom` `Logical`
Instead of projecting to the top of the active direction, project to the bottom.

`Projection Mask Variable` `String`
: By default the projection is performed assuming that the whole mesh is active. This keyword may be used to choose the variable that chooses the active set of nodes user in the projections.

`Dot Product Tolerance` `Real`
: When determining the structure of the mesh in the active direction this tolerance is used to decide that an element edge is aligned with the direction of the action.

`Project to Everywhere` `Logical`
: By default the operators `height, depth, thickness, distance` and `index` are mapped to the whole mesh. The other operators are mapped to the top only, if not else specified. This keyword will always map all operators to the whole mesh.

`Variable i` `String`
: When applying the different projections, the variable to apply the projection to. The resultant field will obtain as the name the name of the operator followed by the initial variable name.

`Operator i` `String`
: The operator to apply to the variable or just the geometry. The choices are

- `sum`: sum over all nodes on the structured line
- `int`: integral over the structured line using trapezoidal rule
- `min`: minimum value over the line
- `max`: maximum value over the line
- `bottom`: field value at the bottom layer
- `top`: field value at the top layer
- `middle`: field value at the middle layer
- `thickness`: thickness of the object i.e. the length of the line
- `depth`: depth i.e. distance from the top surface
- `height`: height i.e. distance from the bottom surface
- `distance`: minimum distance to either top or bottom surface
- `index`: number of the layer from the bottom
- `layer below top`: field value at layer below the top
- `layer above bottom`: field value at layer above bottom
- `isosurface`: field value at the given value of the isosurface variable

`Layer Index i` `Integer`
: The number of structured layers from the top or bottom layer when appropriate. The index `i` refers to the corresponding variable and operator.

`Isosurface Variable i` `String`
: The variable used to determine the isosurface position.

`Isosurface Value i` `Real`
: The value for the isosurface position.

`Target Variable i` `String`
: By default the target variable is created using the name of the variable and operator. However, the user may also give the target variable using this keyword. Then it should be allocated before.

`Target Variable i At Bottom` `Logical`
: Create a target variable that is not at top but at bottom instead.

`Target Variable i At Middle` `Logical`
: Create a target variable that is not at top but at middle instead.

`Target Variable i Everywhere` `Logical`
: Create a target variable that is not at top but everywhere.

Model 59

Internal Cost Function Optimization

Module name: FindOptimum
Module subroutines: FindOptimum
Module authors: Peter Råback
Document authors: Peter Råback

59.1 Introduction

This solver is an auxiliary solver for optimization problems. As input it requires a cost function computed with the previous parameter values, and as output it gives the new parameters for which the cost function will be computed for. Typically the cost function depends on the solution of one or several differential equations. Based on this solution a measure of goodness for the solution is computed.

The routine is still in its development phase but is provided as a skeleton that may be further developed.

59.2 Theory

The optimization routines must be slightly modified from their standard form since the solver is not in a ruling position in respect to the simulation. Therefore its difficult to plug in existing optimization packages to this solver.

Currently the solver includes some very basic optimization routines. Of these the Simplex algorithm (Nelder-Mead) and the differential GA (Genetic algorithm) are the only ones that may be used for a number of design variables.

For just one design variables there is the choice of simple scanning, bisection search and the secant method. Secant method finds roots making it better suited for problems where the target is known i.e. design problems.

59.3 Keywords

`Simulation`

`Simulation Type` `String "scanning"`
: The natural mode used for optimization problems is `scanning`. If the problem is really time-dependent the current internal solution is not probably the optimal solution.

`Timestep Intervals` `Integer`
: The maximum number of optimization rounds is in the case of scanning defined by the timestep intervals.

`Solver` `solver id`

`Equation` `String FindOptimum`
: A describing name for the solver.

`Variable` `String OptPar`
: The name of the variable may be freely chosen as far as it is used consistently also elsewhere.

`Variable DOFs` `Integer n`
: Degrees of freedom for the pressure. Here `n` should be equal to the number of parameters.

`Variable Global` `Logical True`
: Indicates the variable is a global one i.e. not a field variable. For global variables the number of unknowns is the same as number of dofs.

`Procedure` `File "FindOptimum" "FindOptimum"`
: The name of the module and procedure. These are fixed.

`Optimization Method` `String`
: Choices are currently `random, scanning, genetic, bisect, secant` and `simplex`.

`Cost Function Name` `String`
: The name of the cost function that is a real stored in the `Simulation` list structure.

`Optimal Restart` `Logical`
: Use the previous best set of parameters for the 1st round of cost function computation.

`Optimal Finish` `Logical`
: Use the best set of parameters for the last round of cost function computation. This may be useful as the last step is often also saved.

`Best File` `File [best.dat]`
: The file were the best set of parameters is always saved.

`Guess File` `File [best.dat]`
: The file were the best set of parameters is read in case of optimal restart.

`Fixed Parameter i` `Logical`
: Is the `i`:th parameter fixed. Applies for some optimization routines.

`Min Parameter i` `Real`
: Minimum value for `i`:th parameter. Applies for some optimization routines.

`Max Parameter i` `Real`
: Maximum value for `i`:th parameter. Applies for some optimization routines.

`Initial Parameter i` `Real`
: Initial value for `i`:th parameter if not given by the `Optimal restart`

`Internal history` `Logical`
: Save the internal values within the solver.

`History File` `File`
: The name of the file where the history data is saved.

`Cost Function Target` `Real`
: If the given cost function is C use $C - C_0$ instead.

`Cost Function Absolute` `Logical`
: If the given cost function is C use $|C|$ instead.

`Cost Function Maximize` `Logical`
: If the given cost function is C use $-C$ instead.

The following keywords apply to the GA algorithm

`Population Size` `Integer [5n]`

`Population Coefficient` `Real [0.7]`

`Population Crossover` `Real [0.1]`

The following keywords apply to the simplex algorithm

`Simplex Relative Length Scale` `Real [0.01]`
The relative length scale that determines the size of the 1st simplex.

`Simplex Restart Interval` `Integer`
The restart interval after which the simulation is restarted if the convergence is poor.

`Simplex Restart Convergence Ratio` `Real`
A critical value which is used to define a poor convergence ratio.

The following keywords apply to the secant method

`Step Size` `Real`
The step size of the first computations.

`Max Step Size` `M`
aximum allowed step size.

`Relaxation Factor` `R`
elaxation used in the secant method.

This shows just a couple of examples how the design parameters could be used in the simulation. The variables may be referred in a similar manner as other global variables such as `time` or `timestep size`.

```
Body Force 1
  Heat Source = Equals OptPar 1
End

Boundary Condition 1
  Heat Flux = Equals OptPar 2
End
```

Part IX

Saving Data and Results

Model 60

Saving Scalar Values to a File

Module name: SaveData
Module subroutines: SaveScalars
Subroutine authors: Peter Råback
Document authors: Peter Råback
Document updated: January 8th 2008

60.1 Introduction

This subroutine may be used to compute derived quantities and saving scalar values to external file. The results are easily then utilized by MatLab, Excel or any other program that can read ASCII data. In addition to the number values also an additional file with the suffix `.name` is saved. It tells what variables are at each column.

60.2 Theory

The equations and algorithms needed for the computation of scalar values are relatively simple. Here some of them are introduced.

When saving statistical information there are two possibilities. We may use normal number statistics where each node is given an equal weight. Then, for example the mean becomes,

$$< f >= \frac{\sum_{i=1}^{n} f_i}{n}. \tag{60.1}$$

The other possibility is to treat the variable as a continuous function and compute the statistical values as averages over the domain. Now the mean is

$$< f >= \frac{\int f \, d\Omega}{\int d\Omega}. \tag{60.2}$$

In addition to the mean we may compute the mean deviation, $< |f- < f > | >$.and the variance $\delta f = \sqrt{< f^2 > - < f >^2}$.

It is possible to compute energy type of lumped quantities by integrating over the domain. The energy of the field f resulting from a diffusion equation is

$$E_{diff} = \frac{1}{2} \int_\Omega \nabla f \cdot c \nabla f \, d\Omega, \tag{60.3}$$

where c may a tensor or a scalar. Kinetic energy related to convection is of type

$$E_{con} = \frac{1}{2} \int_\Omega c \vec{v} \cdot \vec{v} \, d\Omega, \tag{60.4}$$

and potential type of energy

$$E_{pot} = \int_{\Omega} cf \, d\Omega. \tag{60.5}$$

Sometimes it may be interesting to compute the fluxes through surfaces. The values may be used in evaluating the accuracy of the results – what goes in should in steady state also come out. There are two different fluxes that may be computed. For convective field the flux is of type

$$F_{con} = \int_{\Gamma} cf\vec{v} \cdot \vec{n} \, d\Gamma. \tag{60.6}$$

Currently the code automatically detects the dimension of the given variable. If it is scalar f then $\vec{v}$ is assumed to be the velocity computed by Navier-Stokes equation. If the given variable is vector then it is used as the velocity field and f is not present. Here $\vec{n}$ is the surface normal.

Diffusive fluxes may be computed from

$$F_{diff} = \int_{\Gamma} c\nabla f \cdot \vec{n} \, d\Gamma, \tag{60.7}$$

where c may also be a tensor.

60.3 Implementation issues

There are many solvers that internally compute lumped quantities. By convention these are added to the list structure of the `Simulation` section in the following style

```
CALL ListAddConstReal( Model % Simulation,'res: capacitance',Capacitance)
```

By default the solver looks through quantities starting with prefix `res:` and saves them to an external file. Also the user may create new quantities to be saved in similar manner.

60.4 Keywords

`Solver` solver id

`Procedure` File "SaveData" "SaveScalars"

`Filename` String
: Name of the file where results are to be saved. If no filename is given then the results will only be echoed on output.

`Output Directory` String
: Name of the directory where results are to be saved, relative to the case directory. By default the results are saved in the case directory.

`Scalars Prefix` String
: Save constants starting with this prefix. The default is `res:`.

`Variable i` String
: The names of the variables to be saved. There can be up to 99 variables. In addition to field variables there are some special variables. The scalar variables. e.g. `Time`, are saved as is. There are also variables `CPU Time` and `CPU Memory` that may be used to save execution details.

`Target Variable i` String
: This is an optional keyword that for each entry computed by the subroutine may give an alternative name that is placed as a proper variable in the model. This variable may then be used similarly to `time` in functional expressions. Note that if the operator creates several output values the numbering of this is not the same as that of `Variables`. So check the output file for correct entries when in doubt.

`Mask Name i` String
: If the operator is such that it can use masks then this keyword may be used to override the default mask name `Save Scalars`.

`Save Points(n)` Integer
: Save the specified degrees of freedom in the n nodes specified.

`Save Coordinates(n,DIM)` Real
: Save the degrees of freedom in the nodes nearest to the given n coordinates.

`Exact Coordinates` Logical
: When this keyword is true the coordinates will be looked in an exact manner. Then the degrees of freedom are linear combinations of the node values of the element that the point belongs to.

`Moving Mesh` Logical
: If this parameter is `True` the saved points will be defined every time the subroutine is visited. The default is `False`.

`File Append` Logical
: If the results from consecutive rounds should be appended to the file this flag should be set to `True`. The default is `False`.

`Filename Numbering` Logical
: If set to true a running index is added to the filename so that the next free filename is used to save the results.

`Partition Numbering` Logical
: Optionally add the number of partitions to the filename. This makes the benchmarking more convenient since each case may use the same command file without conflicts.

`Show Norm Index` Integer
: The user may choose to output one value of the results as the norm of the solver in a similar output syntax as `ComputeChange` shows its norms. This is of course not a real norm but may be used in monitoring desired convergence measures in ElmerGUI, for example. By default no norm is shown.

`Echo Values` Logical
: When this is turned on the scalar values will also be echoed to the screen. This is the default action also when the filename is not given and hence nothing is saved to file.

`Cost Function Index` T
: he user may also choose to save a desired value in the list structure with the name `Cost Function`. This may be utilized by the `FindOptimum` solver in optimization problems.

`Parallel Reduce` Logical
: By default the output is written independently for each partition in parallel runs. Enabling this, however, the information is reduced to just one file. The reduction is done using `MPI_ALLREDUCE`: `MPI_MAX, MPI_MIN, MPI_SUM`. The default is `MPI_SUM`. Parallel operator is controlled with keyword `Parallel Operator`. These are not sufficient for all operators. By default the value of the 1st partition is written.

`Save Eigenvalues` Logical
: Save the eigenvalues found in any of the variables.

`Save Eigenfrequencies` Logical
: Save the frequencies computed from the eigenvalues found in any of the variables.

`Operator i` String
: There are different operators that may be performed on all the given variables. These include statistical operators working on the set of numbers, `max, min, max abs, min abs, mean, rms, variance` and `deviation`. Note that these operate directly on the result vector and do not employ the mesh in anyway.

  Different scalar quantities are obtained also by domain integral operators over the mesh. Operator `int` gives the integral over a variable, `int mean` the mean value, `int rms` the root mean square, and `int variance` the variance of the variable. The volume used by a given variable is obtained by operator `volume`. If a name for the coefficient, is given for the operator, the integral is taken over the coefficient. One can for example obtain the weight from a integral over `Density`. Three different energy type of energy quantities may be computed by `diffusive energy, convective energy, potential energy, volume`.

By default the statistical and integral operators are performed over the whole mesh. However, the user may apply the operator only to some parts of the mesh that are related to the logical entities of the sif file. Currently these entities are `body`, `body force`, and `material`. If the user wants to select some of these parts then the standard operator should be preceded by the name of the section. For example, there could be operators `body max` or `body force int`.

There are also a number of similar operators that only operate on the boundary. These are invoked by `boundary sum`, `boundary dofs`, `boundary mean`, `boundary max`, `boundary min`, `boundary max abs`, `boundary min abs`, `area`, `boundary int`, and `boundary int mean`. Also boundary integrals are possible using operators `diffusive flux`, `convective flux`, `boundary int`, `boundary int mean` and `area`. These require that in the boundary conditions the active boundaries are defined. Also here there may be an optional coefficient.

Some operators do not work on the solution itself but use other info related to that. Operator `dofs` simply returns the length of the variable under study. Operator `norm` returns the last computed norm of the field variable, and operators `nonlinear change` and `steady state change` return the last computed convergence measures at the nonlinear and steady state levels. Operator `nonlin iter` returns the number of nonlinear iterations, while operators `nonlin converged` and `coupled converged` which tell whether or not the simulation has converged. Note that these operators most operate on the primary variable for which the matrix equation is solved for.

Operator `bounding box` returns the minimum and maximum coordinate value of each coordinate i.e. six values in 3D mesh.

Finally for parallel runs the operator `partitions` may be useful in creating parallel scaling results in automated manner.

There is no upper bound to the number of operators or variables. If the variable is a vector the statistics is performed on its length.

`Coefficient i` String

Even though only limited number of operators are given almost any energy or flux kind of quantity may be computed since the coefficient c may be defined by the user. The idea is that the same data that is already used as a material parameter can be simple referred to by its name. The coefficient may be, `Heat Conductivity`, `Permittivity`, `Density`, for example. Usually the coefficient is the same that was used in computing the field variable under integration. For the `diffusive energy` and `diffusive flux` the coefficient may even be a matrix. This parameter is optional and the default is one.

`Parallel Operator i` String

Sometimes the default parallel reduction method is not the desired one. Therefore the user may define the parallel reduction method by this keyword. The alternatives are `min, max` and `sum`.

`Polyline Coordinates(n,DIM)` Real

This keyword may be used to create line segments that are defined by points x_1, y_1, x_2, and y_2. For each line different kinds of fluxes through the elements may be computed. This makes it possible, for example, to check the mass flux even though no boundary has a priori been defined.

`Save component results` Logical

Save results arising from component sections.

`Boundary Condition` bc id

`Save Scalars` Logical

The flag activates the computation of boundary-related information. The results are treated independently for each boundary. The keyword replaces the previously used `Flux Integrate`. Also if `Mask Name` is given this arbitrary string may be used instead.

If the user only wants a basic functionality of `SaveScalars` it is possible to let the system create automatically an instance simply by:

`Simulation`

`Scalars File` `File`
: Name of the file where the results are saved.

`scalars: Keyword` `Type`
: Any keyword with the suffix `scalars:` is passed to the solver instance without the suffix.

By default the solver is executed `after saving`. With the namespace it is possible to include all keywords (with the keyword type given) also in the simulation section. At some point the `Simulation` section will get crowded making it justified to create a separate solver instance.

60.5 Examples

In the following examples it is assumed that the `SaveScalars` solver gets the first free index.

Range of solution

The following example shows how to deduce after each converged timestep the minimum and maximum values of temperature to an external file.

```
Solver 2
  Exec Solver = After Timestep
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = "temprange.dat"

  Variable 1 = Temperature
  Operator 1 = min
  Operator 2 = max
  Operator 3 = int mean
End
```

Force resulting from flow solution

The following example shows how to compute the force resulting from the Navier-Stokes equation on the boundary 3. It is assumed that for the flow solver is set `Calculate Loads = True` and that the name of the solution vector is `Flow Solution`. Then the force acting on the boundary is obtained by summing up the nodal forces on the boundary.

```
Solver 2
  Exec Solver = After Timestep
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = "forces.dat"

  Variable 1 = Flow Solution Loads 1
  Operator 1 = boundary sum
  Variable 2 = Flow Solution Loads 2
  Operator 2 = boundary sum
End

Boundary Condition 3
```

```
  Name = no_slip
  ...
  Save Scalars = Logical True
End
```

Benchmark information

The following example shows how benchmark information from the computation of some potential equation is gathered to one file. Different runs will append the results to the same file and in parallel runs the degrees of freedom, number of partitions and consumed cpu time will be automatically gathered.

```
Solver 3
  Exec Solver = After all
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = timing.dat
  File Append = Logical True
  Variable 1 = Potential
  Operator 1 = dofs
  Operator 2 = partitions
  Operator 3 = cpu time
  Parallel Reduce = Logical True
End
```

Model 61

Saving data along lines to a file

Module name: SaveData
Module subroutines: SaveLine
Subroutine authors: Peter Råback, Ville Savolainen
Document authors: Peter Råback
Document updated: 10.6.2022

61.1 Introduction

The subroutine saves lines that pass through higher dimensional computational meshes. The data is saved in simple matrix format thereby allowing the easy postprocessing of data by MatLab, Excel or any other program that can read ASCII data. In addition to the number values also an additional file with the suffix `.name` is saved. It tells what variables are at each column.

61.2 Theory

One mildly theoretical problem in saving data comes from the fact that the data should be saved in lines that were not a priori defined. If there are relatively few points the dummy algorithm where each element is checked for including the node may be used. For the lines, however, this algorithm might become quite expensive as there may be many points that constitute the line and faster choices are needed.

Therefore we only look for intersections of element faces and the lines. Each element face is divided into triangles. The triangle has points $\vec{e}_1$, $\vec{e}_2$ and $\vec{e}_3$. The line is drawn between points $\vec{r}_1$ and $\vec{r}_2$. Therefore the line goes through the point only if

$$\vec{r}_1 + a(\vec{r}_2 - \vec{r}_1) = \vec{e}_1 + b(\vec{e}_2 - \vec{e}_1) + c(\vec{e}_3 - \vec{e}_1) \tag{61.1}$$

has a solution for which $0 \leq a, b, c \leq 1$. This results to a matrix equation

$$\begin{pmatrix} r_{2x} - r_{1x} & e_{1x} - e_{2x} & e_{1x} - e_{3x} \\ r_{2y} - r_{1y} & e_{1y} - e_{2y} & e_{1y} - e_{3y} \\ r_{2z} - r_{1z} & e_{1z} - e_{2z} & e_{1z} - e_{3z} \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} e_{1x} - r_{1x} \\ e_{1y} - r_{1y} \\ e_{1z} - r_{1z} \end{pmatrix} \tag{61.2}$$

which may be easily solved with standard methods linear algebra. Because the face element is a triangle there is an additional condition that $b + c \leq 1$.

61.3 Keywords

`Solver` `solver id`

`Procedure` File "SaveData" "SaveLine"

`Filename` String
: Name of the file where results are to be saved, the default is `sides.dat`.

`Output Directory` String
: Name of the directory where results are to be saved, relative to the case directory. By default the results are saved in the case directory.

`File Append` Logical
: If the results from consecutive rounds should be appended to the file this flag should be set to `True`. The default is `False`.

`Save Axis` Logical
: Save all the principal axis. Also keywords `Save Axis i` exist, where `i`=1,2,3 defines the axis.

`Polyline Coordinates(n,DIM)` Real
: Save the line consisting of line segments defined by two points ($n = 2$). There can be more than one set of points ($n = 2, 4, 6, \ldots$) but as a line segment is defined by two points there must be an even number of points.

`Polyline Divisions(n/2,DIM)` Integer
: The user may give the number of divisions for each polyline. This allows also the proper saving of discontinuous data. The size of this vector should be such that it is compatible with the number of lines.

`Save Isocurves` Logical
: Saves isocurves defined on 2D meshes.

`Isosurface Variable i` String
: The variable which isocurve to save, $i = 1, 2, \ldots$. This must be a scalar variable.

`Isosurface Value i` Real
: A constant value that defines the value of the isosurface variable at the isocurve to be saved. Note that for the same variable there may be several values, each with a different keyword.

`Variable i` String
: By default `SaveLine` saves all the active variables. However, it is possible to save only a specified list of variables given by this keyword where tt i=1,2,3,. . . This may be particularly useful if one wants to save a table of linear dependence, for example Temperature along x-direction, to be used as a boundary condition in consecutive Elmer runs with a different mesh.

`Save Flux` Logical
: Saves a flux resulting from a gradient of a field by the model $h = -\kappa\partial T/\partial n$. This may only be applied to existing boundaries, not lines defined by points.

`Flux Variable` String
: The name of the field variable (default T is `Temperature`).

`Flux Coefficient` String
: The diffusion constant (by default κ is `Heat Conductivity`)

`Save Mask` String
: a By default SaveLine saves only the values that are on boundary marked with `Save Line` flag. If the user wants several instances of the SaveLine subroutine, for saving different boundaries to different files, the mask name may be defined by this keyword. The correspondingly one should use the same flag in the `Boundary Condition` and Body section.

`Optimize Node Ordering` Logical
: When saving data on pre-existing boundaries, should we try to optimize the order so that a continuous line is created. This uses connectivity information that is optimal only when saving 1D data defined on lines. This is defaulted to `True` only for serial simulations.

`Break Line Loop` Logical
: This is a keyword related to finding optimal node ordering. For loops the connectivity-based

node ordering does not work properly. However, if we cut the loop at one point, the resulting connectivity enables us to create a continuous loop. So use this only when you know that your line creates a full loop.

`Parallel Reduce` `Logical`

By default the output is written independently for each partition in parallel runs. By enabling this, however, the information is reduced to just one file when using predefined lines. The data is communicated to the partition that owns most data to start with and it does the writing to the file.

`Boundary Condition` `bc id`

`Save Line` `Logical`

The flag activates the saving of the boundary condition as a line. The subroutine tries to save the finite-element lines as a chain of points to enable nice preprocessing with MatLab or similar tools. The flux may only be saved on lines defined by boundary conditions.

61.4 Examples

In the following examples it is assumed that the 1st solver is the primary solver and the `SaveLine` solver gets index two.

The following example shows how to save a line that extends from point (0,0,0) to point (1,2,3) in 3D mesh after the whole simulation has ended using 100 divisions for both line segments.

```
Solver 2
  Exec Solver = After Simulation
  Equation = SaveLine
  Procedure = "SaveData" "SaveLine"
  Filename = "line.dat"
  Polyline Coordinates(2,3) = 0.0 0.0 0.0 1.0 2.0 3.0
  Polyline Divisions(2) = 100 100
End
```

Model 62

Saving material parameters and boundary conditions

Module name: SaveData
Module subroutines: SaveMaterials, SaveBoundaryValues
Module authors: Thomas Zwinger
Document authors: Thomas Zwinger, Peter Råback
Document updated: 4.4.2011

62.1 Introduction

These subroutines creates fields from material parameter entries or boundary conditions that normally cannot accessed as a field, as they are expressions that are evaluated when needed. The `SaveMaterials` may be used to create additional field variables from the material parameters. A similar procedure `SaveBoundaryValues` stores parameters defined on boundaries as variables for the whole mesh. This can be of help if a boundary condition that is not directly accessible from the variables (like a normal component of a vector field) should be evaluated in the post-processing step.

62.2 Keywords

Keywords of subroutine SaveMaterials

`Solver` `solver id`

`Procedure` `File "SaveData" "SaveMaterials"`

`Parameter i` `String`
The user may choose a number of parameters (i=1,. . . ,99) which will be save as variables. This may be particularly handy if one wants to visualize how the parameters depend on the position over the domain. Values in bodies with the assigned material list not containing the keyword of the parameter are set to zero by default.

Keywords of subroutine SaveBoundaryValues

`Solver` `solver id`

`Procedure` `File "SaveData" "SaveBoundaryValues"`

`Variable` `String -nooutput dummyvar`
a dummy variable for the solver that does not show up

`Variable DOFs` `Integer 1`

`Parameter i` `String`
: The user may choose a number of parameters (i=1,. . . ,99) which will be save as variables. These parameters will then be stored as variables with the values assigned as they were found on the specific boundary. Bulk values and values on boundaries with the parameter not being defined are set to zero by default.

`Body Force Parameters` `Integer`
: The user may also save parameters given in body force section by giving the number of these parameters by this keyword. Note that the body force parameters must be first in the list followed by the material parameters. The default is zero.

Model 63

Result output in different formats

Module name: ResultOutputSolve
Module subroutines: ResultOutputSolver
Module authors: Peter Råback, Erik Edelmann, Mikko Lyly
Document authors: Peter Råback

63.1 Introduction

This subroutine is intended for saving data in other than the native format of Elmer – ElmerPost. The reason for using another postprocessing tool might be that some feature is missing in ElmerPost, or that the user is more acquainted with some other visualization software. Currently supported formats include GiD, Gmsh, VTK legacy, XML coded VTK file bearing the suffix VTU and Open DX.

The recommended 3rd party visualization tool of Elmer results is Paraview and the corresponding format for it is the Vtu format. The old VTK format is not recommended.

63.2 Keywords

`Solver` `solver id`

`Equation` `String "ResultOutput"`
The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure` `File "ResultOutputSolve" "ResultOutputSolver"`
The name of the file and subroutine.

`Output File Name` `File`
Specifies the name of the output file.

`Output Format` `String`
This keyword the output format of choice. The choices are `gid, gsmh, vtk, vtu`, and `dx`.

`Gid Format` `Logical`

`Gmsh Format` `Logical`

`Vtk Format` `Logical`

`Vtu Format` `Logical`

`Dx Format` `Logical`
The user may also use the above logical keywords to set which of the formats is saved. This has more flexibility in that there may be several formats that are saved simultaneously where the `Output Format` keyword may only be used to activate one solution type.

`Binary Output` Logical
: For Vtu format (no other format) the data may also be written in binary format which is significantly more compact than the ASCII format. Default is `True`.

`Ascii Output` Logical
: As the default format for Vtu is binary this keyword is more natural for enforcing ascii format. The default is `False`.

`Single Precision` Logical
: The floating numbers in Vtu format may be saved either in single (32 bits) or double (64 bits) precision. The single precision saves some disk space. Default is `False`.

`Eigen Analysis` Logical
: For GiD and Gmsh output format activates the eigenmode writing, and in Vtu format makes the numbering of the files follow the eigenmodes. Vtu format will still save the eigenmodes without this keyword but this will be done in the same file just altering the name of the field.

`Number Of EigenModes` Integer
: Maximum number of modes, when supported. Default is that all eigenmodes are saved.

`Active EigenModes` Integer
: List of active eigenmodes to be saved. Supported only in the Vtu format. By default modes are saved in order (1,2,. . .).

The following keywords related only to the GiD, Vtu and Gsmh formats. In the other formats all available degrees of freedom are saved. Also in Vtu format all dofs are saved if none of the list is given.

`Scalar Field i` String
: The scalar fields to be saved, for example `Pressure`. Note that the fields must be numbered continuously starting from one.

`Vector Field i` String
: The vector fields to be saved, for example `Velocity`

`Tensor Field i` String
: The tensor fields to be saved. The rank of tensor fields should be 3 in 2D and 6 in 3D.

Sometimes when the variables need to be explicitly listed it may be difficult to know what the actual available variables are. For this purpose there is the following keyword.

`Show Variables` Logical
: Show all the different variables on output as a list. Default is `False`.

In the Vtu output format the user may use several various masking operations to choose the elements to be saved. These cannot be used with other formats. Also the user may choose whether to save elemental or nodal fields, if applicable.

`Save Geometry Ids` Logical
: Save the index of geometric entities i.e. of the bodies and boundaries. The body index will be saved as such whereas the boundary index will get a offset of 100 (or always 10 times larger if not larger than largest body index).

`Default Body Id` Integer
: This may be used to modify the default geometry id for the bodies when saving the geometry ids in VTU format.

`Default BC Id` Integer
: This may be used to modify the default geometry id for the boundary conditions when saving the geometry ids in VTU format.

`BC Id Offset` Integer
: This may be used to modify the offset from the default one when saving the geometry ids in VTU format.

`Save Elemental Fields` Logical
: There may be some elemental fields present, for example if they have been computed with the Discontinuous Galerkin method. Then the elementwise nodal information is averaged to elemental field. Elemental fields can present discontinuities better than the nodal fields, whereas the nodal fields can represent smooth fields better.

`Save Nodal Fields` Logical
: Save the computed nodal fields. Most fields are nodal only.

`Lagrange Element Degree` Integer
: This keyword enables the visualization of results which have been obtained by applying hierarchic higher-order finite elements (the p-version of FEM). The value of this keyword defines the polynomial degree of Lagrange interpolation elements which is used in the visualization. In principle, it is reasonable to choose the value to be the same as the polynomial order of the p-solution (defined by a command "`Element = p:...`" in a solver section), but the two values can be chosen independently. At the moment this option works for the values up to 8 for quadrilaterals, triangles and hexahedra, while tetrahedra and prisms allow values up to 4. Pyramids however allow at most the second-order visualization.

`Save Linear Elements` Logical
: For higher-order nodal (Lagrange) elements the user has the option to save the data still using linear basis. This effectively saves only values at the corner nodes of each element. Often the derived data from quadratic elements is just of linear accuracy and therefore there is no real benefit in treating the actual quadratic data.

`Discontinuous Galerkin` Logical
: When dealing with elemental fields there is the option to save the field as discontinuous such that each instance of a node is recreated. This may make the resulting files huge but may still be the desired option when dealing with discontinuous fields.

`Discontinuous Bodies` Logical
: When dealing with elemental fields one can also create a minimal discontinuous set of nodes such that the discontinuous information is averaged within shared nodes body-wise. Often the discontinuity is present only over different bodies so this strategy maintains the essential discontinuous while having only a minor effect on the file size.

`Vtu Part Collection` Logical
: Save each part into a different file and include a file with .pvd suffix that acts as a container for these individual parts. The parts may be recognized by their entity name.

`Vtu Time Collection` Logical
: Save a time collection file with a .pvd suffix that includes the timestamps for each file in a transient simulation.

`Skip Halo Elements` Logical
: If halo element are used in the parallel computation this flag can be used to suppress the saving of these elements. They are basically redundant so this flag could well be set to true for typical use.

`Save Halo Elements Only` Logical
: If halo elements are used then this flag can be used to save only the halo elements. This would probably have mainly uses in debugging or demonstrating the halo elements.

`Save Bulk Only` Logical
: Save the bulk elements only.

`Save Boundaries Only` Logical
: Save the boundary elements only. This could be useful if one is interested of the results only at the boundaries. This flag would also often save considerable amount of disk space.

`Mask Variable` String
: The user may give a variable for masking. If this flag is given then only the elements where the permutation vector associated with this variable is positive for all nodes are saved.

`Mask Name` String
: The user may give a logical name of the mask. This mask will then be checked for in body force and boundary condition lists and the active elements are determined on-the-fly.

`Mask Condition` String
: The user may give a real valued condition for the mask. This mask condition will be check in body for and boundary condition lists and the results will be saved only where the value of the mask condition is positive. This could easily be used to save results only within a sphere, for example.

`Body` body id

`Geometry Id` Integer
: May be used to remap the default body index to a new value for VTU output.

`Boundary Condition` bc id

`Geometry Id` Integer
: May be used to remap the default boundary condition index to a new value for VTU output.

If the user only wants a basic functionality of `VtuOutputSolver` it is possible to let the system create automatically an instance of the solver simply by adding a `Post File` with suffix `.vtu` to the Simulation section.

`Simulation`

`Post File` File [filename.vtu]
: Name of the file where the unstructured XML based VTK results are to be saved.

`vtu: Keyword` Type
: Any keyword with the suffix `vtu:` is passed to the solver instance without the suffix.

By default the solver is executed `after saving`. With the help of the namespace it is possible to include any keyword (with the keyword type given) also in the `Simulation` section. At some point the section will get crowded making it justified to create a separate solver instance.

63.3 Example

This example saves the results of a computation in binary unstructured VTK XML format following the `Output Intervals` definition in the `Simulation` section. The variables to be saved are hand-selected.

```
Solver 3
  Exec Solver = after saving
  Equation = "result output"
  Procedure = "ResultOutputSolve" "ResultOutputSolver"
  Output File Name = "case"

  Vtu Format = Logical True
  Binary Output = Logical True     ! binary format is the default
  Single Precision = Logical True  ! double precision is the default

! In the Vtu format all fields are saved if the user does not list them explicitly.
  Scalar Field 1 = String Temperature
  Scalar Field 2 = String Pressure
  Vector Field 1 = String Velocity
End
```

Model 64

Saving data on uniform Cartesian grid

Module name: SaveGridData
Module subroutines: SaveGridData
Module authors: Peter Råback
Document authors: Peter Råback

64.1 Introduction

This subroutine is intended for saving data in a uniform grid. One possible use of the feature is to combine a boundary representation and a glyph representation (velocity vectors) in the same visualization. This routine would then save the data for the glyphs. The routine may also be used so that the resolution in two other directions is set to a high value which will then effectively create uniform line plots.

The algorithm goes through all the elements and checks whether the element could include some of the uniform grid nodes. If so then these are tested for. The algorithm should scale linearly with mesh size. Optionally one may check for duplicates to eliminate the same nodes being repeatedly saved. Currently no particular order is guaranteed for the nodes.

The routine works currently only in VTU, VTI and ASCII table formats, but other format may be possible in the future.

64.2 Keywords

`Solver` `solver id`

`Equation` `String "SaveGridData"`
: The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure` `File "SaveGridData" "SaveGridData"`
: The name of the file and subroutine.

`Filename Prefix` `File`
: Specifies the name of the output file. A appropriate suffix is added to the given name.

`Output Format` `String`
: This keyword the output format of choice. The choices are `vtu,` `vti` and `table`.

`Vtu Format` `Logical`

`Vti Format` `Logical`

`Table Format` `Logical`
: The user may also use the above logical keywords to set which of the formats is saved. This has more flexibility in that there may be several formats that are saved simultaneously where the `Output Format` keyword may only be used to activate one solution type.

`Fileindex Offset` Integer
: By default the files are numbered starting from one. However, for some restarted simulations the offset may be defined to be something else. The default is zero.

`Check for Duplicates` Logical
: This flag activates the checking of duplicates. This is usually advisable but may for peculiar geometries require a large logical table and is therefore not defaulted to be true.

`Grid dx` Real
: The grid size in direction x (i.e. `Coordinate 1`). Similarly we have `Grid dy` and `Grid dz`, if applicable. If the density in directions y or z is not defined, it is assumed to be the same as for x.

`Grid nx` Integer
: The number of cells in direction x. This is only required if the previous keyword is not given. Then the resolution is defined by the size of the bounding box. Similarly we have `Grid ny` and `Grid nz`,

`Grid Origin i` Real
: The mesh is by default located so that there is a node at $(0, 0, 0)$. If the origin should not reside here then this keyword may be used to transform the origin ($i = 1, 2, 3$).

`Min Coordinate i` Real
: The bounding box may be limited by this keyword ($i = 1, 2, 3$). If not given the minimum values of the mesh are used.

`Max Coordinate i` Real
: The bounding box may be limited by this keyword ($i = 1, 2, 3$). If not given the maximum values of the mesh are used.

`Scalar Field i` String
: The scalar fields to be saved, for example `Pressure`. Note that the fields must be numbered continuously starting from one. If no fields are given then an attempt is made to save all the relevant fields.

`Vector Field i` String
: The vector fields to be saved, for example `Velocity`

`Mask Name` String [MyMask]
: The user may provide a mask that is used to determine the active elements. If the elements are lower dimensional then it is assumed that the last coordinate is eliminated from the gridded data. So if the full mesh is 3D and a mask is given for a boundary only then the data is saved on x-y plane.

`Filename Timestep Numbering` Logical
: Use this keyword in transient case to number the files by the timesteps. Applies only to the table format.

`Filename Particle Numbering` Logical
: Use this keyword in transient case to number the files by particles indexes. Applies only to the table format.

`Boundary Condition` bc id

`MyMask` Logical True
: The mask define in the solver section may be set True in the BC or Body Force section.

64.3 Example

The following example saves the results in the VTK XML image data format at the end of simulation. A mesh parameter $h = 0.1$ is used for every direction and the lower left corner is taken as the base of the finite uniform mesh.

```
Solver 5
  Exec Solver = after all
  Equation = SaveGrid
  Procedure = "SaveGridData" "SaveGridData"

  Grid dx = Real 0.1
  Grid Origin At Corner = Logical True
  Check For Duplicates = Logical True
  Binary Output = Logical True
  Single Precision = Logical True
  Filename Prefix = String glyphs
  Vti Format = Logical True
End
```

Model 65

Isosurface extraction for reduced output

Module name: Isosurface
Module subroutines: IsosurfaceSolver
Module authors: Juha Ruokolainen, Peter Råback
Document authors: Peter Råback

65.1 Introduction

This subroutine extract isosurfaces from 3D meshes or isolines from 2D meshes. The intended use of the routine is in heavy simulations where the standard output could result to an I/O bottle-neck. If the desired isosurface is known in advance then this can be used to reduce the amount of data to be written. The solver does not itself write the data. It is expected that there is some external strategy for writing the data. It could, for example, be the `ResultOutputSolver` for VTK XML output.

65.2 Keywords

`Solver solver id`

`Equation String "Isosurface"`
The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure File "Isosurface" "IsosurfaceSolver"`
The name of the file and subroutine.

`Output Directory File`
Specifies the name of the output directory. Output file name will be determined in the normal manner.

`Isosurface Variable String`
The name of the variable for which the isosurface is determined. This must be a scalar variable.

`Isosurface Values Real`
The isosurface values used to extract the surfaces. If this is a vector then a number of isosurfaces will be defined.

`Isosurface Values Real`
The isosurface value used to extract the surfaces. For constant values this is the scalar variant of the previous keyword. However, this may also have a functional dependency on some scalar variable, such as time, for example.

`Variable i String`
Name of the variable to be outputted on the isosurface (or isoline), `i`$=1, 2, 3, \ldots$

65.3 Example

This solver extracts the 1D isolines from 2D temperature field and stores them to a mesh called `isosurf` at the three given isosurface values. If the original mesh would be 3D then the resulting isosurface mesh would be 2D. Note that if saving of results is desired then this solver should be performed prior to the saving of results.

```
Solver 8
  Exec Solver = after all
  Equation = "isoline"
  Procedure = "Isosurface" "IsosurfaceSolver"
  IsoSurface Variable = Temperature
  IsoSurface Values(3) = 0.25 0.5 0.75

  Output Directory = isoline

  Variable 1 = Temperature
  Variable 2 = stream
  Variable 3 = vorticity
End
```

Model 66

Coupling Elmer with OpenFOAM via file IO

Module name: Elmer2OpenFoamIO,OpenFOAM2ElmerIO
Module subroutines: Elmer2OpenFoamIO,OpenFOAM2ElmerFit
Module authors: Peter Råback
Document authors: Peter Råback

66.1 Introduction

These solver modules provides the possibility to transfer field values from Elmer into OpenFOAM and back. The data transfer is done via file IO and has therefore some limitations regarding speed of operation. However, the approach also has some flexibility since it does not set any additional constraints to how Elmer and OpenFOAM have been compiled.

The solvers assume a working Elmer case and OpenFOAM case directory. The solvers assume that the OpenFOAM directory `0` includes the cell centers computed with `writeCellCenters` in file `C`. For a multiblock case the mesh directory will include subdirectories and then the cell centers should be available at least in one subdirectory. Note that the Elmer field will only be mapped if the cell center file `C` is provided.

The Elmer2OpenFoamIO solver first reads the cell centers, then creates a temporal mesh structure from them, and interpolates the desired field using the standard interpolation routines within Elmer. The interpolation uses octree search tree and therefore it should be rather fast – in serial. The interpolated field is written into the same directory where cell centers `C` were found. The format is a one that can be used to initialize object from class `volScalarField`.

The coupling from OpenFOAM to Elmer is performed with the OpenFOAM2ElmerFit routine. It again read the cells centers from files `C`. The solution with a given name is not fetched from directory `0` but rather from the time step given by the user. The resulting values are not interpolated using the basis function since there is no OpenFOAM mesh available at Elmer. Rather a procedure similar to `DataToFieldSolver` is used. There diffusivity may be given as a regularization parameter. A good choice may be the physical diffusivity related to the solver. I.e. if you're reading in temperatures they may be regularized by heat conductivity.

In case the performance of the routines is not satisfactory or one needs parallel operation then the MPI coupler `EOF library` mainly written by Juris Vencels could be a better choice for the problem.

66.2 Keywords for module Elmer2OpenFoamIO

```
Solver  solver id
```

`Equation` `String [Elmer2OpenFoamIO]`
: The name of the equation.

`Procedure` `File "Elmer2OpenFoamIO" "Elmer2OpenFoamWrite"`
: The name of the procedure for writing Elmer fields to be read by OpenFOAM.

`Filename` `File`
: Full name of the target file (with the suffix).

`Target Variable` `String`
: Name of the Elmer field to be mapped for OpenFOAM. Currently only one field is supported.

`OpenFoam Directory` `File`
: Name of the OpenFOAM directory that includes the whole OpenFOAM case tree.

`OpenFoam File` `File`
: Name of the OpenFOAM file that will include the exported field from Elmer interpolated to the cell centers.

`OpenFoam Mesh i` `File`
: If the OpenFOAM mesh directory cannot be retrieved automatically for some reason the user may define the OpenFOAM meshes to be treated. This could be the case in Windows where the system commands might not be available.

66.3 Keywords for module OpenFoam2ElmerIO

`Solver` `solver id`

`Equation` `String [OpenFoam2ElmerIO]`
: The name of the equation.

`Procedure` `File "OpenFoam2ElmerIO" "OpenFoam2ElmerFit"`
: The name of the procedure for importing fields from OpenFOAM to Elmer.

`OpenFoam Directory` `File`
: Name of the OpenFOAM directory that includes the whole OpenFOAM case tree.

`OpenFoam Mesh i` `File`
: If the OpenFOAM mesh directory cannot be retrieved automatically for some reason the user may define the OpenFOAM meshes to be treated. This could be the case in Windows where the system commands might not be available.

`OpenFoam Field` `String`
: Name of the OpenFOAM field to be read from the directory tree. Currently only one scalar field is supported. Typical files are `T` (temperature) and `p` (pressure).

`OpenFoam Timestep` `Integer`
: Name timestep number to define the subdirectory where the field is read from. Zero refers to the initial state.

`Fit Coefficient` `Real`
: This constant gives the factor that is used to weight the data when performing the fitting. This has a relevance only with respect to the diffusivity used for regularization. Default value is one.

`Diffusion Coefficient` `Real`
: This constant gives the diffusivity used for regularization.

`Diffusivity Name` `String`
: The user may give an existing material parameter for the diffusivity. Then this is the name which is searched in `Material` section.

`Passive OpenFOAM Coordinate` `Integer 3`
: In case the initial file is 3D and we want to interpolate the results into a 2D mesh we may give the passive coordinate index that will be used in dimensional reduction. If not given, 3D interpolation will be assumed.

66.4 Example

This example is used to map the field `joule heating` from Elmer to the OpenFOAM initialization file `fieldSolidHS.dat` to be used in temperature distribution computations.

```
Solver 2
  Equation = ElmerToOpenFOAM
  Procedure = "Elmer2OpenFoamIO" "Elmer2OpenFoamWrite"

! The variable to be mapped
  Target Variable = joule heating

! The OpenFOAM project directory containing the mesh etc.
  OpenFOAM Directory = FILE "ofdir"

! The file to write the OpenFOAM sources to.
  OpenFOAM File = File "fieldSolidHS.dat"
End
```

Part X

Reading Data

Model 67

Read fields from Gmsh results file

Module name: GmshOutputReader
Module subroutines: GmshOutputReader
Module authors: Peter Råback
Document authors: Peter Råback

67.1 Introduction

This subroutine can read results provided in simple Gmsh ASCII output format and use the results to initialize Elmer fields. Gmsh format is handy as it is relatively simple and has supporting software that can deal with it. Also the Gmsh result format includes both the mesh geometry, topology and results. Hence it can be used to read old results and interpolate them into a new mesh

This solver is not intended to replace the fully fledged restart format of Elmer. However, this tool may be nice if we need to interpolate results in a workflow in a flexible manner. Note that the `ResultOutputSolver` includes also a Gmsh writer and may therefore be used in conjunction with this module in workflows coupled via saving and reading files.

67.2 Keywords

`Solver` `solver id`

`Equation` `String "GmshReader"`
: The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure` `File "GmshOutputReader" "GmshOutputReader"`
: The name of the file and subroutine.

`Filename` `File`
: Filename to read the results from. This file should currently be in Gmsh 2.2. format ASCII file.

`Align Coordinate` `Logical`
: When reading the mesh into ElmerSolver should we align the mesh with some coordinate direction. The possible values are 1, 2, 3 to align in maximum coordinate direction. If negative value is used then the mesh will be aligned with the minimum coordinates.

`Mask Name` `String`
: If we want to interpolate results only on part of the mesh we may use a mask. Only where the mask is present interpolation will be performed. The possible sections where the mask should exist are `Body Force` and `Boundary Condition`.

Model 68

Reload Existing Simulation Results

Module name: ReloadData
Module subroutines: ReloadSolution
Module authors: Antti Pursula
Document authors: Antti Pursula

68.1 Introduction

This subroutine is intended for repeated loading of existing results during simulation. An example of a typical application is to use previously computed fluid flow as a convection field for the transfer of a passive scalar variable. The module is implemented as a dummy solver which is defined in the command file just as the 'normal' solvers.

This module offers extended features compared to the `Restart File` option in the `Simulation` section. The module reads a new solution step from the solution file on each timestep, whereas the restart file option reads only the initial state for a simulation.

The module reads in all the available variables from the solution file. The solution file should be in the mesh directory. If the simulation takes more than a single steady state iteration per time step it is advisable to use `Exec Solver = Before Timestep` for this module.

68.2 Keywords

`Solver` `solver id`

`Equation` `String "Reload Data"`
: The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure` `File "ReloadData" "ReloadSolution"`
: The name of the file and subroutine.

`Reload Solution File` `String "flow-data.dat"`
: The name of the old solution data file, eg. flow-data.dat

`Reload Starting Position` `Integer`
: The index of the timestep where to start reading. If the keyword is not given the reading is started from the first step in the file, or from the beginning of the reload range, if specified.

`Reload Range Minimum` `Integer`

`Reload Range Maximum` `Integer`
: The beginning and the end of the reading range. The timesteps on the range are read in cyclically if the current simulation has more timesteps than what there are on the range.

`Reload Reading Intervals` `Integer`
: Defines the interval for reading in old results, defaults to 1. An integer i larger than 1 defines that results are read in only on every ith timestep. However, consecutive steps are read in regardless of the value of i.

`Continuous Reading` `Logical True`
: When set to `True` the reload solution file is kept open also between the timesteps. However, when reading is not started at the first solution step, or when the old solution is read in cyclically, it is advisable to switch this feature off. Defining `False` will slow down reading especially from large ASCII files.

Model 69

Runtime Control of the Input Data

Module name: ReloadInput
Module subroutines: ReloadInput
Module authors: Juha Ruokolainen
Document authors: Peter Råback

69.1 Introduction

This subroutine is intended for cases where the user wants to have run-time control over the solution. The control is obtained by reloading the command file (.sif-file) during the solution. This is done with an additional solver that is called similarly as any other solver during the solution process.

The most likely usage of the solver is in cases where the user realizes during the solution process that the some parameters were not optimally chosen. For example, the convergence criteria may have been set too tight for optimal performance. Then the user may set looser criteria by editing the command file during the computation. Once the new value is read the solver will apply the new criteria thereafter.

69.2 Limitations

The solver should not be used for things that need allocation. For example, the number of solvers or boundaries may not change. Also the computational mesh must remain the same.

69.3 Keywords

`Solver solver id`

`Equation String "Reload"`
The name of the equation. This is actually not much needed since there are no degrees of freedom associated with this solver.

`Procedure File "ReloadInput" "ReloadInput"`
The name of the file and subroutine.

Model 70

Reading NetCDF data into FE mesh

Module name: GridDataReader
Module subroutines: GridDataReader
Module authors: Peter Råback, Vili Forsell
Document authors: Peter Råback

70.1 Introduction

This solver provides the possibility to read in data in uniform grid into Elmer mesh. The supported format includes currently only NetCDF.

Currently following features are supported:

- Bilinear (in 2D) or trilinear (in 3D) interpolation
- Possibility to have a multiplier and offset in the target field
- Linear interpolation in time
- Multiple possibilities to define the time instant or index

The module requires the netcdf library. Because of this, it has been isolated from the main build system. The source code for XdmfWriter can be found from the source tree in `misc/netcdf2` and it should be compiled by the user as follows:

```
elmerf90 -I$HDF5/include -L$HDF5/lib -o GridDataReader.so GridDataReader.f90 -lnetcdff -lnetcdf
```

The environment variable `$NETCDF` defines the installation directory for the NetCDF-library.

70.2 Theory

In structured data the finding of grid cell where a certain node belongs to is trivial. Assume that the grid is defined by coordinates a_i and b_i so that any coordinate x_i is presented by

$$x_i = a_i n_i + b_i \tag{70.1}$$

where $n_i \in [1, N_i]$ and $i \in [1, \mathrm{DIM}]$. Now the finite element node with coordinate $\hat{x}_i$ may be interpolated using the cell with indexes $m_i = \text{ceiling}((\hat{x}_i - b_i)/a_i)$ and $m_i + 1$ using linear interpolation with weighing factor $q_i = m_i - (\hat{x}_i - b_i)/a_i$.

$$\hat{f} = q_i f_{m_i} + (1 - q_i) f_{m_i+1}. \tag{70.2}$$

This generalizes into multiple dimensions using recursion. For example, in 2D the bilinear interpolation reads

$$\hat{f}(x_1, x_2) = q_1 q_2 f_{11} + (1 - q_1) q_2 f_{21} + q_1 (1 - q_2) f_{12} + (1 - q_1)(1 - q_2) f_{22}. \tag{70.3}$$

The current implementation provides bilinear and trilinear interpolation in space, and linear interpolation in time.

The algorithmic complexity of the reader and interpolation routine is linear in time. However, the current implementation assumes that each partition do their own interpolation routines which means that there may be a large number of files open in parallel runs introducing potential bottle-necks.

70.3 Keywords

`Solver` `solver id`

`Equation` `String [GridDataReader]`
The name of the equation.

`Procedure` `File "GridDataReader" "GridDataReader"`
The name of the procedure.

`Filename` `File`
Full name of the target file (with the suffix).

`X Name` `String`
Name of the 1st coordinate.

`Y Name` `String`
Name of the 2nd coordinate.

`Z Name` `String`
Name of the 3rd coordinate.

`Time Name` `String`
Name of the time coordinate.

`X Epsilon` `A`
ccuracy of the 1st coordinate assumed in interpolation. Default is machine epsilon.

`Y Epsilon` `A`
ccuracy of the 2nd coordinate assumed in interpolation. Default is `X Epsilon`.

`Z Epsilon` `A`
ccuracy of the 3rd coordinate assumed in interpolation. Default is `X Epsilon`.

`Time Epsilon` `A`
ccuracy of time assumed in interpolation. Default is machine epsilon.

`Time Point` `Real`
Value of time used in this calling. If this is not found the Elmer time will be used.

`Time Offset` `Real`
Offset used to add to the Elmer time.

`Time Multiplier` `Real`
Coefficient used to multiply the Elmer time.

`Is Time Index` `Real`
If this flag is turned on then the time instance given with the previous keywords will be understood as being the index of time in the NetCDF file (starting from 1) rather than actual time.

`Is Time Counter` `Logical`
If this flag is turned on, the time index of the NetCDF file will be increased by one each time the routine is called. This makes it ideal for looping over each timestep.

`Coordinate Transformation` `String`
Optional coordinate transformation that is applied on the Elmer coordinate prior to finding its value on the grid.

`Enable Scaling` `Logical`
If this flag is turned on the bounding box of Elmer mesh will be made compatible with that of the grid.

`Variable i` String
: Variable of the NetCDF file. There is no upper limit to the number of variables.

`Target Variable i` String
: Optional name for the target variable. If this is not given then the `Variable i` value will be used for the FE variable as well. If the target variable is not present it will be created the first time it is needed. Also, if different variables that follow each other have the same target variable these will be summed up. With the multiplication and offset features this makes it possible to derive new variables as a linear combination of any fields given in the NetCDF file.

`Interpolation Offset` Real
: A constant that will be added to the value of the interpolated field. If there are many fields with different offsets use `Interpolation Offset i` instead.

`Interpolation Multiplier` Real
: A constant that will be used to multiply the interpolated field. The default value is one. If there are many fields with different offsets use `Interpolation Multiplier i` instead.

Part XI

Experimental or Obsolete Solvers

Model 71

Lithium-Ion Battery Model

Module name: BatterySolver
Module subroutines: SolidPhaseCons, ElectrolyteCons, SolidPhasePot, ElectrolytePot, BatteryPost
Module authors: Peter Raback, Timo Uimonen
Document authors: Timo Uimonen, Joel Songok, Peter Raback
Document edited: March 2nd 2020

71.1 Introduction

This module can be used to solve a set of conservation equations that govern lithium-ion battery cycling behaviour. The macroscopic model follows multiphase porous electrode and concentrated solutions theories that describe the transfer of mass and charge between solid electrodes and liquid electrolyte [2] [3].

The current implementation provides perhaps a starting point for continued work. There is still some challenges in the robustness of the solver. It has been verified against literature but the number of coupled system iterations needed seems often excessive.

If you're interested in continuing the work, please contact the authors for more details. In the partial differential equations we have followed closely the references while the numerical solution is very different, probably both in good and bad.

71.2 Theory

For the equations we mainly follow the Master's Thesis of A. Borakhadikar [1]. For other relevant references see, for example, [3].

The model for the battery includes two coupled equations for both electrostatic potential and concentration. These are closed by chemical kinetics described by the Butler-Volmer equation. The special feature of the equations is that the solid phase concentration is to be solved within each solid phase particle. The multiscale nature is captured by solving the 1D diffusion equation in spherical coordinates in each node of the distributed system. The other equations follow pretty much standard procedures.

Conservation of species in solid phase is written as

$$\frac{\partial c_s}{\partial t} - \frac{D_s}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial c_s}{\partial r}\right) = 0, \tag{71.1}$$

where $c_s(\vec{x}, r, t)$ is the concentration, D_s is solid phase Li diffusion coefficient. At the boundary of each sphere the flux condition is given by

$$D_s \frac{\partial c_s}{\partial r}\Big|_{r=r_0} = -\frac{1}{a_s F} J_{Li}, \tag{71.2}$$

where $a_s = 3\varepsilon_s/R_s$ is the active interfacial surface area, F is Faraday constant and R_s is the radius of the solid particle. J_{Li} is the lithium ion flux between the solid and electrolyte phases. The coupling to other fields takes place only at the surface where we define $C(\vec{x},t) = c_s(\vec{x},r_0,t)$.

Conservation of species in electrolyte phase leads to a diffusion equation

$$\frac{\partial}{\partial t}(\varepsilon_e C_e) - \nabla \cdot D_e^{eff}\nabla C_e = \frac{1-t_+}{F} J_{Li}, \tag{71.3}$$

where t_+ is transference number of lithium ions and D_e^{eff} is effective electrolyte phase Li^+ diffusion coefficient. The effective diffusivity depends on the electrode porosity which is described by Bruggeman relation $D_e^{eff} = D_e\varepsilon_e^p$ where ε_e is electrolyte phase volume fraction. At the current collectors, no-flux boundary conditions are imposed,

$$\frac{\partial C_e}{\partial x}\Big|_{x=0} = \frac{\partial C_e}{\partial x}\Big|_{x=L} = 0. \tag{71.4}$$

Conservation of charge in solid phase defines the solid phase potential,

$$-\nabla \cdot \sigma^{eff}\nabla\varphi_s = -J_{Li}, \tag{71.5}$$

where the σ^{eff} refers to effective conductivity of an electrode. It is defined as $\sigma^{eff} = \sigma\varepsilon_s$ where ε_s is active particle volume fraction. The boundary condition for solid phase potential at the current collectors reads as

$$-\sigma_-^{eff}\frac{\partial\varphi_s}{\partial x}\Big|_{x=0} = \sigma_+^{eff}\frac{\partial\varphi_s}{\partial x}\Big|_{x=L} = \frac{I}{A}, \tag{71.6}$$

where $I(t)$ is applied current and A is the plate area of an electrode. The boundary conditions at the electrode-electrolyte interface are defined as follows,

$$\frac{\partial\varphi_s}{\partial x}\Big|_{x=L_{Neg}} = \frac{\partial\varphi_s}{\partial x}\Big|_{x=L_{Neg}+L_{Sep}} = 0. \tag{71.7}$$

Conservation of charge in electrolyte phase defines the potential in the electrolyte phase,

$$-\nabla \cdot \kappa^{eff}\nabla\varphi_e - \nabla \cdot \kappa_d^{eff}\nabla(\log C_e) = J_{Li}, \tag{71.8}$$

where κ^{eff} is the effective ionic conductivity, which is calculated by using Bruggeman's relation $\kappa^{eff} = \kappa\varepsilon_e^p$. The electrolyte phase ionic conductivity κ can respectively be defined as

$$\kappa = 15.8e^{-4}C_e\exp(0.85(\frac{C_e}{1000})^{1.4}). \tag{71.9}$$

The second term of the equation gives rise to numerical challenges since it may be difficult to evaluate accurately. Here κ_d^{eff} is effective diffusional conductivity which is evaluated as

$$\kappa_d^{eff} = \frac{2RT\kappa^{eff}}{F}(t_+ - 1), \tag{71.10}$$

where R is gas constant and T is ambient temperature. At the current collectors, the boundary conditions are defined as

$$\frac{\partial\varphi_e}{\partial x}\Big|_{x=0} = \frac{\partial\varphi_e}{\partial x}\Big|_{x=L} = 0. \tag{71.11}$$

The system of equation is closed by Butler-Volmer kinetic equation

$$J_{Li} = a_s i_0\left[\exp\left(\frac{\alpha_a F}{RT}\eta\right) - \exp\left(\frac{\alpha_c F}{RT}\eta\right)\right]. \tag{71.12}$$

In the equation, i_0 is the exchange current density given by

$$i_0 = k_0 C_e^{\alpha_a}(C_{s,max} - C_s)^{\alpha_a}C_s^{\alpha_c}, \tag{71.13}$$

where k_0 is kinetic rate constant, $C_{s,max}$ is the maximum solid phase concentration of an electrode while the α_a and α_c are anodic and cathodic charge coefficients. Overpotential η may be written as

$$\eta = \varphi_s - \varphi_e - U, \tag{71.14}$$

where empirical expressions are used for the equilibrium potential U. The equilibrium potential is a material property and for the particular material used in this module, the potential for negative electrode it given by

$$\begin{aligned} U_n(x) =& 8.0029 + 5.0647\,x - 12.578\,x^{\frac{1}{2}} - 8.6322e\text{-}4\,\frac{1}{x} + 2.1765e\text{-}5\,x^{\frac{3}{2}} \\ & - 0.46016\,\exp(15(0.06 - x)) - 0.55364\,\exp(-2.4326(x - 0.92)) \end{aligned} \tag{71.15}$$

and for positive, correspondingly, by

$$\begin{aligned} U_p(x) =& 85.681\,x^6 - 357.7\,x^5 + 613.89\,x^4 - 555.65\,x^3 + 281.06\,x^2 \\ & - 76.648\,x - 0.30987\,\exp(5.657\,x^{1.15}) + 13.1983, \end{aligned} \tag{71.16}$$

where $x = C_s/C_{s,max}$.

71.2.1 Postprocessing

Once the conservation equations have been solved, the solution can be transformed into more readable format. The cell voltage of the battery can be expressed as

$$V_{cell} = \phi_s(t, L) - \phi_s(t, 0) - \frac{R_f}{A} I(t), \tag{71.17}$$

where $\phi_s(t, L)$ and $\phi_s(t, 0)$ are solid phase potentials at cathode and anode current collectors, R_f is resistance of current collectors and A is electrode plate area.

The battery state of charge can be estimated in variety of ways. In this module, the SOC is evaluated by default as

$$SOC(t) = \frac{(\frac{C_{s,surf}}{C_{s,max}}) - x_{0\%}}{x_{100\%} - x_{0\%}}, \tag{71.18}$$

where $x_{0\%}$ and $x_{100\%}$ are stoichiometric coefficients at 0% and 100% state of charge. Alternatively, the state of charge can also be calculated by Coulomb counting which goes as follows

$$SOC(t) = SOC(0) - \frac{1}{Q_{rated}} \int_0^t I(t)dt, \tag{71.19}$$

where Q_{rated} is rated cell capacity. The cell capacity during cycling is given by

$$Q_{cell} = \int_0^t I(t)dt. \tag{71.20}$$

71.3 Keywords

The module includes five different solvers. Usually order of solvers is arbitrary. However, here a specific order is given that has been found to be most successful.

`Solver 1`

Poisson equation for the solid phase potential, φ_s. The equation is solved in both anode and cathode.

`Equation String SolidPhasePot`

The name of the equation.

`Procedure File "BatterySolver" "SolidPhasePot"`

The name of the solver file and subroutine.

`Variable` String [Phis]
: This is the default name for φ_s assumed by other solvers.

`Linearize Flux` Logical [True]
: Linearize the flux J_{Li} with respect to φ_s. This is needed since the equation does not otherwise set the potential levels. This keyword may be applied for any of the potential and concentration solvers.

`Skip Butler Volmer` Logical [False]
: The strategy when Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation. This keyword may also be applied for any of the solvers.

`Fixed Overpotential` Logical [False]
: The strategy when the overpotential in Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation overpotential. This keyword may also be applied for any of the solvers.

`Use Time Average Flux` Logical [False]
: Whether to use instantaneous flux J_{Li} or average the flux over the timestep. When activated it is applied in all the solvers.

`Use Mean Flux` Logical [False]
: Whether to use instantaneous flux or average flux over iterations. Cannot be used together with time-averaged flux.

`Use Time Average Diffusion` Logical [False]
: Whether to use instantaneous diffusion or average diffusion over the timestep. When activated it is applied in all the solvers.

The following keywords are related to the library functionality but are nevertheless explained briefly also here. For more details look at the ElmerSolver manual.

`Linear System Solver` String [Direct]
: The type of linear solver used can be specified here.

`Linear System Direct Method` String [banded]
: The type of direct linear system solver can be specified here. `Banded` is chosen by default.

`Linear System Scaling` False
: For 1D equation the direct banded solver without any scaling is the fastest strategy. For 2D and 3D geometries, other strategies should be used.

`Nonlinear System Max Iterations` Integer
: The maximum number of nonlinear iterations the solver is allowed to do.

`Nonlinear System Relaxation Factor` Real
: Giving this keyword triggers the use of relaxation in the nonlinear equation solver. The factor might speed up the convergence.

`Nonlinear System Convergence Tolerance` Real
: A set value which specifies criterion for relative change between iterations in the nonlinear solver.

`Nonlinear System Convergence Measure` Real
: There is a interplay between nonlinear and steady state convergence criteria. Usually speed is optimized by iterating mainly at the steady state level. However, here we need to iterate also on the nonlinear level to achieve convergence.

`Steady State Convergence Tolerance` Real
: This relates to the overall convergence of the system.

`Solver` 2
: Poisson equation for the electrolyte potential, φ_e. This is solved in the whole domain.

`Equation` String ElectrolytePot
: The name of the equation.

`Procedure File "BatterySolver" "ElectrolytePot"`
The name of the solver file and subroutine.

`Variable String [Phie]`
This is the default name for φ_e assumed by other solvers.

`Linearize Flux Logical True`
Linearize the flux J_{Li} with respect to φ_e.

`Quadratic Electrolyte Diffusion Logical [False]`
This enables to toggle between the two different formulations for electrolyte diffusion. The quadratic form requires higher order elements whereas the standard weak formulation is also applicable to linear elements.

`Ignore Electrolyte Diffusion Logical [False]`
This enables to toggle off the effects of Li^+ diffusion in the electrolyte during computation.

`Use Time Average Diffusion Logical [False]`
Whether to use instantaneous diffusion or average diffusion over the timestep.

`Fixed Overpotential Logical [False]`
The strategy when the overpotential in Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation overpotential. This keyword may also be applied for any of the solvers.

`Calculate Ce sensitivity Logical [False]`
If set to `True`, the solver calculates C_e sensitivity for solver 2.

`Correct Source Disbalance Logical`
This keyword toggles an option to correct the flux imbalance in the φ_e solver.

`Solver 3`
Diffusion equation for the solid phase concentration, C_s. This is always 1D equation utilizing an internal 1D mesh as a submodel in every node.

`Equation String SolidPhaseCons`
The name of the equation.

`Procedure File "BatterySolver" "SolidPhaseCons"`
The name of the solver file and subroutine.

`Variable String [Cs onedim]`
This is the default name for C_s an 1D stride assumed by other solvers.

`Solid Phase Relaxation Factor Real`
The value of relaxation factor used by solid phase solver.

`Maximum Global Change Speed Real`
Sets the value of the largest allowed change in solid phase concentration in one iteration.

`Number of Passive Visits Integer`
Specifies the number of solid phase iterations for only electrostatic solver.

`Linearize Flux Logical True`
Linearize the flux J_{Li} with respect to C_s.

`Fixed Overpotential Logical [False]`
The strategy when the overpotential in Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation overpotential. This keyword may also be applied for any of the solvers.

`Check Material Balance Logical`
A toggle to check for flux imbalance in 1D solution.

`Save Solid Phase Average Logical True`
Toggle to save average flux between solid and electrolyte phase.

`Save Solid Phase Diff` `Logical True`
: Toggle to save the difference in flux between solid and electrolyte phase.

`Visualize Node Index` `Integer`
: A toggle to visualize a given node with 1D solution.

The following keywords are used by 1D submeshes located in the nodes of the main mesh.

`1D Mesh Create` `Logical [True]`
: Creates 1D submesh if set to `True`.

`1D Element Order` `Integer`
: The order of elements in 1D submesh can be specified here.

`1D Number Of Elements` `Integer`
: The number of elements in 1D submesh can be specified here.

`1D Mesh Ratio` `Real`
: Here the ratio of 1st and last element in the 1D submesh can be specified.

`1D Mesh Length` `Real`
: This keywords is used to specify the length of the 1D submesh.

`1D Active Direction` `Integer`
: Specifies the direction of the 1D submesh. This is set to 1 by default.

`1D Body Id` `Integer`
: This keyword is used to set the body id.

`Solver` `4`
: Diffusion equation for the electrolyte concentration, C_e. It is solved in the whole domain.

`Equation` `String ElectrolyteCons`
: The name of the equation.

`Procedure` `File "BatterySolver" "ElectrolyteCons"`
: The name of the solver file and subroutine.

`Variable` `String [Ce]`
: This is the default name for C_e assumed by other solvers.

`Linearize Flux` `Logical [False]`
: Linearize the flux J_{Li} with respect to c_e.

`Number of Passive Visits` `Integer`
: Specifies the number of solid phase iterations for only electrostatic solver.

`Use Solid Phase Relaxation` `Logical [True]`
: If set to `True`, the solver inherits relaxation factor from solid phase concentration solver.

`Fixed Overpotential` `Logical [True]`
: The strategy when the overpotential in Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation overpotential. This keyword may also be applied for any of the solvers.

`Skip Butler Volmer` `Logical [True]`
: The strategy when Butler-Volmer equation is computed affects the convergence. It may sometimes be advantageous to skip the re-computation.

`Use Effective Diffusion` `Logical [False]`
: It can sometimes be advantageous to use non-effective diffusion where the electrolyte volume fraction is not taken into account. This is due to numerical difficulties risen from the second term in solver 2.

`Calculate Cs sensitivity` `Logical [False]`
: If set to `True`, the solver calculates C_s sensitivity for solver 4.

`Use Time Average Diffusion` `Logical [False]`
: Whether to use instantaneous diffusion or average diffusion over the timestep.

`Correct Butler Volmer Fluxes` `Logical`
: Sometimes there might exist an imbalance of fluxes between the electrodes. This toggle is used to add a correction coefficient to adjust the imbalance.

`Solver` `5`
: Solver for performing postprocessing information after convergence is obtained.

`Equation` `String BatteryPost`
: The name of the postprocessing equation.

`Procedure` `File "BatterySolver" "BatteryPost"`
: The name of the solver file and subroutine.

`Study Jli Balance` `Logical`
: This keyword is used to toggle on an option to save error values in the C_s solution to study $J_L i$ balance.

`Soc Model` `String`
: State of charge model to be used. The `Default` model evaluates SOC in terms of the solid phase concentration while taking the stoichiometric coefficients into account. Selecting `Simplified`, on the other hand, makes the model to ignore them. Alternatively, selecting `Coulomb` evaluates SOC using Coulomb counting method.

`SOC on surface` `Logical`
: If set to `True`, the solver uses the surface values for SOC instead of true average.

`Initial SOC` `Real`
: The initial SOC can be specified here if the simulation starts in conditions other than fully charged or discharged.

`Cutoff Voltage` `Real`
: This keyword is used to set a lower-limit for the cell voltage at which the battery cell is considered fully discharged.

`Stoichiometric Limit` `Real`
: This keyword is used to set a lower-limit for solid phase concentration after which a warning is issued.

`Applied Current` `Real`
: This keyword is used to define the value of applied current that is only used in postprocessing. Here, negative sign refers to charge current and vice versa.

`Cell Capacity` `Real`
: This keyword is used to define value of rated capacity of the battery cell, which is used in evaluation of state of charge by Coulomb counting method.

`Use Average Cell Voltage` `Logical`
: We may either use extremum potential to compute the cell voltage or take average over anode and cathode. The first one is the default.

`Calculate Charges` `Logical`
: Toggle to calculate addition information about the charges and potentials of anode, cathode and electrolyte.

`Constants`

`Gas Constant` `Real`
: The universal gas constant R. In SI units the value is 8.314 J/K mol.

`Faraday Constant` `Real`
: Specific charge in mole, F. In SI units the value is 96485.3 C/mol.

`Transference Number` Real
: This keyword is used to define the Li^+ transference number, t_+.

`Electrode Plate Area` Real
: This keyword is used to define the electrode plate area, A.

`Ambient Temperature` Real
: This keyword is used to define the ambient temperature, T.

`Current Collector Resistance` Real
: This keyword is used to define the contact resistance of a current collector, R_f.

`Material` mat id
: Materials parameters for anode and cathode.

`Anode` Logical
: If material is anode set `True`, otherwise `False`.

`Cathode` Logical
: If material is cathode set `True`, otherwise `False` (default).

`Particle Radius` Real
: This keyword is used to define the radius of the solid phase particles, R_s.

`Active Particle Volume Fraction` Real
: This keyword is used to define the volume fraction of particles in an electrode, ε_s.

`Maximum solid phase concentration` Real
: This keyword is used to define the value of maximum lithium concentration in an electrode, $C_{s,max}$.

`Kinetic Constant` Real
: This keyword is used to define the value of kinetic rate constant, k_0.

`Anodic charge transfer coefficient` Real
: This keyword is used to define the anodic charge transfer coefficient, α_a.

`Cathodic charge transfer coefficient` Real
: This keyword is used to define the cathodic charge transfer coefficient, α_c.

`Solid Phase Diffusion Coefficient` Real
: The value of lithium diffusivity in solid phase, D_s.

`Solid Phase Electrical Conductivity` Real
: This keyword is used to define the electrical conductivity of an electrode in solid phase, σ.

`Stoichiometry at Full Charge` Real
: Stoichiometry when the battery cell is considered fully charged, $x_{100\%}$.

`Stoichiometry at Nill Charge` Real
: Stoichiometry when the battery cell is considered fully discharged, $x_{0\%}$.

Material parameters for electrolyte.

`Electrolyte Volume Fraction` Real
: This keyword is used to define the volume fraction of electrolyte, ε_e.

`Electrolyte Diffusion Coefficient` Real
: The value of Li^+ diffusivity in electrolyte phase, D_e.

`Initial Condition` ic id
: Initial conditions are used to set the state of the battery at start of the simulation. We may initialize C_s, C_e, φ_s, φ_e and J_{Li}.

`Cs` Real

`Ce` Real

`Phis` `Real`

`Phie` `Real`

`Jli` `Real`

`Boundary Condition` `bc id`

`Current Density` `Real`
: Neumann boundary condition for the current density. During discharge, current density has a positive sign at anode current collector and negative sign at cathode current collector.

Bibliography

[1] Ashwin S. Borakhadikar. One dimensional computer modeling of a lithium-ion battery. Master's thesis, Wright State University, 2017.

[2] M Doyle, T F Fuller, and J Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society; (United States)*, 140:6, 6 1993.

[3] K. A. Smith, C. D. Rahn, and C.-Y. Wang. Control oriented 1D electrochemical model of lithium ion battery. *Energy Conversion and Management*, 48:2565–2578, 2007.

Model 72

Richards equation for variably saturated porous flow

Module name: RichardsSolver
Module subroutines: RichardsSolver, RichardsFlux
Module authors: Peter Råback
Document authors: Serge-Étienne Parent and Peter Råback

72.1 Introduction

Richards equation is a non-linear partial differential equation that represents the movement of fluids through porous media.

The current implementation of the Richards equation uses normal Lagrange elements and therefore the conservation of flux cannot be guaranteed. Dense meshes are required if the variations in the permeability are high.

This version should not yet be considered a production version. However, it provides a suitable starting case for more serious testing and further development.

72.2 Theory

The transient, incompressible, variably saturated, isotropic flow of water in non-swelling porous media is expressed by the combination of Darcy's law and the continuity equation, i.e. Richards equation. The modern form of Darcy's law can be written as

$$\vec{q} = -k_w \nabla H \tag{72.1}$$

where $\vec{q}$ is the unit flux, or Darcy velocity (L/T), k_w is the fluid hydraulic conductivity of water (L/T), and H is the total head (L). Since the velocity component of total head can be treated as negligible in porous media, and air pressure can be considered as constant, total head can be expressed as $H = p + z$ where p is pressure (F/L^2) (note that $p = -\psi = u_a - u_w$), ψ is matric suction (F/L^2), u_w is the water pressure in pores (F/L^2), u_a is the air pressure in pores (F/L^2), z is elevation from a datum (depth coordinate of the geometry).

The continuity equation is expressed as

$$\frac{\partial \theta}{\partial t} = -\nabla \cdot \vec{q} + S_w, \tag{72.2}$$

where θ is the volumetric water content (L/L), t is time (T), S_w is a source/sink term (L/T). Richards equation may now be written as,

$$\frac{\partial \theta}{\partial t} = -\nabla \cdot (k_w \nabla H) + S_w, \tag{72.3}$$

since $\theta = f(\psi)$ and $k_w = f(\psi)$ the latter equation can be expressed using a pressure form,

$$\frac{\partial\theta}{\partial\psi}\frac{\partial\psi}{\partial t} = -\nabla\cdot(k_w\nabla H) + S_w, \tag{72.4}$$

Or, developing total head,

$$\frac{\partial\theta}{\partial\psi}\frac{\partial\psi}{\partial t} = -\nabla\cdot(k_w\nabla(-\psi + z)) + S_w, \tag{72.5}$$

The volumetric water content and the hydraulic conductivity are non-linear functions related to pressure head. Both are commonly expressed by van Genuchten (1980)'s equations. Volumetric water content function yields

$$\theta(\psi) = \begin{cases} \theta_s + \frac{\theta_s-\theta_r}{(1+\alpha_{vG}^{n_{vG}})^{m_{vG}}}, & \text{if } \psi > 0 \\ \theta_s, & \text{if } \psi < 0. \end{cases} \tag{72.6}$$

And the hydraulic conductivity function is

$$k_w(\psi) = \begin{cases} k_{w,sat}\frac{(1-(\alpha_{vG}\psi)^{n_{vG}m_{vG}}(1+(\alpha_{vG}\psi)^{n_{vG}})^{m_{vG}})^2}{(1+\alpha_{vG}^{n_{vG}})^{-m_{vG}/2}}, & \text{if } \psi > 0 \\ k_{w,sat}, & \text{if } \psi < 0. \end{cases} \tag{72.7}$$

where θ is the volumetric water content (L/L), θ_r is the residual volumetric water content (L/L), θ_s is the saturated volumetric water content, equal to the porosity (L/L), α_{vG}, n_{vG}, m_{vG} are fitting parameters without any units.

72.3 Implementation issues

The current implementation is carried out for the total head, H. This results to a weak form where the fluxes occur naturally. The total head is intuitive since it gives directly the ground water level. Since the time derivative of the elevation is zero, we may use the following equation to solve the total head,

$$\theta_\psi\frac{\partial H}{\partial t} + \nabla\cdot(k_w\nabla H) = S_w. \tag{72.8}$$

From the total head the matrix suction will be automatically computed, $\psi = z - H$. This makes it possible to have material laws that depend on it.

For transient problems the first term requires special attention. In the current version the sensitivity of θ to ψ is computed from

$$\theta_\psi = \begin{cases} \frac{\theta(\psi(t_i))-\theta(\psi(t_{i-1}))}{\psi(t_i)-\psi(t_{i-1})} & \text{if } |\psi(t_i) - \psi(t_{i-1})| > \epsilon \\ \frac{\theta(\psi(t_i))-\theta(\psi(t_i)-\epsilon)}{\epsilon} & \text{otherwise.} \end{cases} \tag{72.9}$$

This way the effective sensitivity is smeared over the whole timestep, $dt = t_i - t_{i-1}$.

The values of the material parameters in the Richards equation vary a great deal depending on the saturation level and type of medium. Therefore it is important to evaluate the water content and hydraulic conductivity at the Gaussian integration points using the relevant formulas, rather than computing them at nodal points and thereafter evaluating the values at the Gaussian integration points using a weighted sum over the nodal values.

72.4 Keywords

The module includes two different solvers. `RichardsSolver` solves the primary differential equation while `RichardsFlux` solves the resulting flux from the computed solution. The second solver only makes sense when the pressure field has already been computed with the first one. The second solver uses the same material parameters as the first one.

Keywords for RichardsSolver

`Solver` solver id

`Equation` String RichardsSolver
: A describing name for the solver. This can be changes as long as it is used consistently.

`Procedure` File "RichardsSolver" "RichardsSolver"
: Name of the solver subroutine.

`Variable` String TotalHead
: The name of the variable may be freely chosen as far as it is used consistently also elsewhere.

`Variable DOFs` Integer 1
: Degrees of freedom for the pressure. This should be 1 which is also the default value.

`Saturated Initial Guess` Logical
: Use saturated material parameters when computing the first equation for the total head.

`Active Coordinate` Integer
: The coordinate corresponding to the depth z in the Richards equation. By default the last coordinate is the active one.

`Calculate Matrix Suction` Logical
: Whether to compute the matrix suction from the total head.

`Bubbles` Logical
: Use stabilization by residual free bubbles.

`Nonlinear System Convergence Tolerance` Real
: The Richards equation is always nonlinear and hence keywords related to the nonlinear system control are needed. The iteration of the nonlinear system is continued till the relative change in the norm falls under the value given by this keyword.

`Nonlinear System Max Iterations` Integer
: This parameter gives the maximum number of nonlinear iterations required in the solution. This may be set higher than the typical number of iterations required as the iteration procedure should rather be controlled by the convergence tolerance.

`Nonlinear System Relaxation Factor` Real
: Keyword related to the relaxation of the nonlinear system.

`Material` mat id

`Porosity Model` String
: Currently the choices are `van Genuchten` and `Default`. The latter does not estimate the functional forms on gaussian points and hence may have inferior accuracy. Also, currently the computation of water content derivative is not supported for it limiting its usability to steady state problems.

`Saturated Hydraulic Conductivity` Real

`Saturated Water Content` Real

`Residual Water Content` Real

`van Genuchten Alpha` Real

`van Genuchten N` Real

`van Genuchten M` Real
: The parameters above are the material parameters of the van Genuchten material law that are used to compute the hydraulic conductivity and water content.

`Hydraulic Conductivity` Real

`Water Content` Real
: In case the porosity model is `constant` then the hydraulic conductivity and water content are given with this keyword.

`Body Force` `bf id`

`Richards Source` `Real`
: The source term, S_w, of the equation.

`Boundary Condition` `bc id`

`Richards Flux` `Real`
: The given flux at the boundary.

Keywords for RichardsPostprocess

This solver uses largely the same keywords that are already defined above. Only the Solver section has its own keyword settings. This solvers should be active in the same bodies than the `RichardsSolver`.

`Solver` `solver id`

`Equation` `String RichardsPostprocess`
: A describing name for the solver. This can be changes as long as it is used consistently.

`Procedure` `File "RichardsSolver" "RichardsPostprocess"`
: Name of the solver subroutine.

`Target Variable` `String`
: The name of the total head field solved by the Richards equation. The default name is `Total head`.

Model 73

Outlet Boundary Condition for Arterial Flow Simulations

Module name: ArteryOutlet
Module subroutines: OutletCompute, OutletInit, OutletPres, OutletdX, OutletdY
Module authors: Esko Järvinen, Mikko Lyly, Peter Råback
Document authors: Esko Järvinen

73.1 Introduction

Arterial elasticity is a fundamental determinant of blood flow dynamics in arteries, such as the aorta and its daughter vessel, that face the largest displacements and which takes care of the cushioning of the stroke volume. Simulation of such a phenomenon requires simultaneous solving of the equations governing both the fluid flow and wall elasticity. To be able to perform accurate fluid-structure interaction (FSI) simulations, only a segment of the circulatory system can be studied at a time. For these artificially truncated segments, which are naturally unbounded domains and in interaction with the rest of the circulation domain, one should construct in the numerical models boundary conditions which do not exhibit any unphysical behaviour, which operates transparently, and are also capable to transport a sufficient amount of information over the boundary.

A natural boundary condition at the outlet of a numerical FSI flow model of an artery is not a proper choice because it does not exhibit enough correct physiological behavior of the flow, and from the point of view of numerical approximation, it causes non-physiological reflections of the wave at the boundary. If measured data of both the pressure (or velocity) and the wall displacement at the outlet boundary are not available, the only way to get the outlet boundary of a higher order, 2D or 3D model sufficiently specified is to combine the model with some lower order model, such as a 1D or lumped model.

In order to get the outlet of the arterial FSI model to behave transparently in such cases when only forward travelling waves are considered, a simple characteristic equation of the of the one dimensional FSI model can be combined with the higher order model.

73.2 Theory

The conservation equations for a flow in an elastic artery in one dimension may be expressed as

$$\begin{cases} \frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \\ \\ \frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}(\frac{Q^2}{A}) + \frac{A}{\rho}\frac{\partial p}{\partial x} = 0, \end{cases} \tag{73.1}$$

where Q is the volume flow, A the cross section area of the artery, p is the pressure and x is the axial coordinate [3]. In order to get the system (73.1) close, an equation relating the area A to the pressure $p = p(A)$ is derived applying the theory of thin shell structures. Assuming a cylindrical shell, and neglecting the rotation on the shell cut plane, and the movements of the structure in the axial and circumferential directions, as well as applying the Kirchhoff-Love assumption, the energy balance equations is reduced to

$$\frac{E\,h^3}{12(1-\nu^2)}\,{d_R}^{(4)} + \frac{E\,h}{(1-\nu^2)}\,\frac{1}{{R_m}^2}\,d_R = p, \tag{73.2}$$

where R_m is the radius to the midplane of the wall, E, ν and d_R are the Young modulus, the Poisson ratio and the radial displacement of wall, respectively. Assuming that the first term on the left side in the equation (73.2) is much smaller than the second term, we can give the pressure-area relation in the form

$$p = p_{ext} + \beta(\sqrt{A} - \sqrt{A_0}), \qquad \beta = \frac{\sqrt{\pi} h E}{(1-\nu^2)A_0}. \tag{73.3}$$

The pressure is scaled to be equal to external pressure p_{ext} with corresponding reference artery cross sections area A_0.

The equations (73.1) and (73.3) form a closed system for the simulations of flow in an elastic tubes. The equations may be written in conservative form which is strictly hyperbolic with two distinct real eigenvalues $\lambda_{1,2} = \bar{u} \pm c$, where $\bar{u} = Q/A$ is the average axial velocity, $c = \sqrt{(A/\rho_f)(\partial p/\partial A)} = \sqrt{\beta\sqrt{A}/(2\rho_f)}$ is the speed of sound, and ρ_f is the density of blood. The system can be further decomposed into a set of the equations for the characteristic variables W_i, which are the components of the vector $W = T^{-1}U$ ($\frac{\partial W}{\partial U} = T^{-1}$) , $U = [A, Q]^T$ [2]. These equations are

$$\frac{\partial W_i}{\partial t} + \lambda_i \frac{\partial W_i}{\partial x} = 0, \tag{73.4}$$

and the characteristic variables are

$$W_{1,2} = \frac{Q}{A} \pm 2\sqrt{\frac{2}{\rho_f} + \beta\sqrt{A}}.$$

When considering a pulse propagation in a straight, infinitely long homogeneous conduit, without any bifurcations or other objects which might cause reflections of the pulse, i.e. any backward travelling waves does not exists, the computations can be done using only the first of equations in (73.4), i.e.

$$\frac{\partial W_1(U)}{\partial t} + \lambda_1(U)\frac{\partial W_1(U)}{\partial x} = 0.$$

This equation is solved in this Elmer outlet boundary condition for arterial flow simulations solver. The connection of the one dimensional model to the test models at the their outlets is done applying the following coupling [1]

$$\begin{cases} dR^- = dR^+ \\ \sigma^- = p^+ \\ W_1 = g_1(A^-, Q^-, p^-), \end{cases}$$

where dR and σ are radial displacement of the artery wall and fluid traction, respectively. The superscript '–' denotes the values in the higher order models, and superscript '+' to the values in the 1D model.

73.3 Keywords

Keywords of FlowSolve

`Initial Condition ic id`

For making the initial guess for the characteristic variable W_1

`Wnodal` `Variable Coordinate`
`Real Procedure "ArteryOutlet" "OutletInit"`

`Material` `mat id`
Material properties for the one dimensional section.

`Density` `Real`
Density of blood

`Artery Wall Youngs Modulus` `Real`
Young's modulus of the artery

`Artery Radius` `Real`
Radius of the artery to the midplane of the artery wall

`Artery Wall Thickness` `Real`
Wall thickness of the artery

`Artery Poisson Ratio` `Real`
Poisson ratio of the artery

`Solver` `solver id`
Keywords for the one dimensional solver. Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation` `String [Artery Outlet Solver]`

`Variable` `[Wnodal]`
The variable which is solved

`Variable DOFs` `[1]`

`Procedure` `File "ArteryOutlet" "OutletCompute"`
The name of the file and the subroutine

`Equation` `eq id`
The equation section is used to define a set of equations for a body or set of bodies

`Artery Outlet Solver` `Logical [True]`
If set `True`, the solver is used. The name of the solver must match with the name in the `Solver` section

`Boundary Condition` `boundary id`
The pressure of the given coordinate direction `i` at the artery outlet of the higher order model is set to correspond the value given by the 1D model.

`Pressure i` `Variable Time`
`Real Procedure "ArteryOutlet" "OutletPres"`

The diameter of the artery in the appropriate direction at the outlet of the higher order model is set to correspond the value given by the outlet boundary condition solver. The subroutines `OutletdX` and `OutletdY` are located in the module `ArteryOutlet`

`Displacement i` `Variable Time`
`Real Procedure "./ArteryOutlet" "OutletdX"`

This is the inlet boundary of the one dimensional section which is coupled with both, the fluid and the solid outlet boundary of the higher order model

`Fluid Coupling With Boundary` `Integer`

`Structure Coupling With Boundary` `Integer`

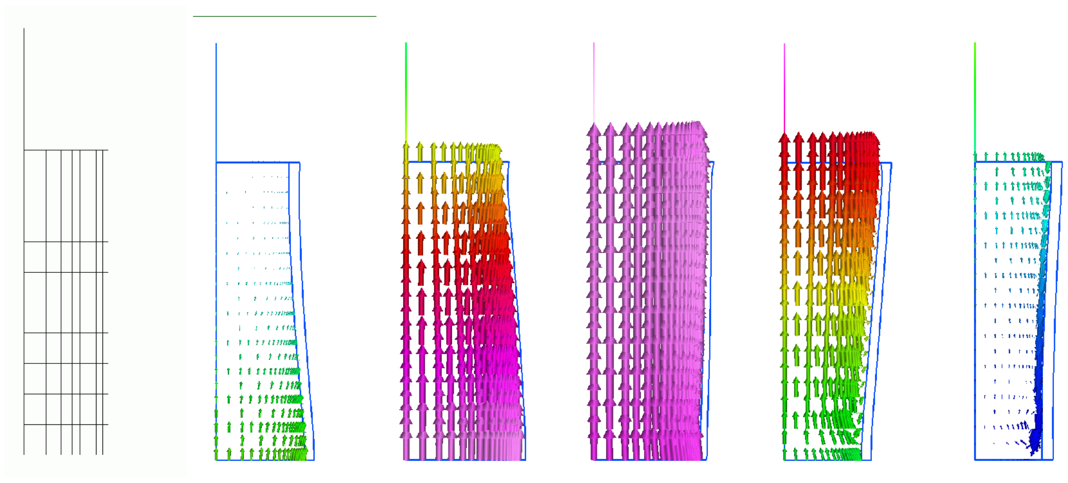

Figure 73.1: An example of the model results: pressure pulse propagation in a 2D axisymmetric model combined with an 1D model.

Bibliography

[1] L. Formaggia, J.F. Gerbeau, F. Nobile, and A. Quarteroni. Numerical treatment of defective boundary conditions for the navier-stokes equations. *SIAM J. Numer. Anal*, 40:376–401, 2002.

[2] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Concervation Laws*. Springer, 1996.

[3] T. J. R. Hughes and J. Lubliner. On the one-dimensional theory of blood flow in the large vessels. *Mathematical Biosciences*, 19:161–170, 1973.

Model 74

BEM Solver for Poisson Equation

Module name: PoissonBEM
Module subroutines: PoissonBEMSolver
Module authors: Juha Ruokolainen
Document authors: Juha Ruokolainen

74.1 Introduction

This module solves the Laplace equation by boundary element method (BEM), where the differential equation is transformed to integral equation along the boundaries. On the boundaries either potential or normal flux may be defined. A source term may be included (Poisson equation), but the source term remains a volume integral.

This is an old module which has limited use since the BEM matrix is assembled in full. This means that the memory consumption and speed of solution grow rather unfavorably with problem size. Efficient BEM solvers never create the full matrix but use iterative multilevel methods instead, such as the multipole expansion. Also, the solver only works in serial so parallel computing is not available to resolve the bottleneck.

74.2 Theory

The Poisson equation is mathematically described as

$$-\Delta\Phi - f = 0, \text{ in } \Omega, \tag{74.1}$$

where f is the given source.

In BEM we transform this equation to integral equation over boundaries. We start by multiplying the equation by a weight function and integrating over the volume, and integrating by parts

$$-\int_\Omega \Delta\Phi w \, d\Omega = \int_\Omega \nabla\Phi \cdot \nabla w \, d\Omega - \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma. \tag{74.2}$$

Similarly we may write an equation reversing the roles of Φ and w

$$-\int_\Omega \Delta w\Phi \, d\Omega = \int_\Omega \nabla w \cdot \nabla\Phi \, d\Omega - \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma. \tag{74.3}$$

Subtracting the two equations we have

$$-\int_\Omega \Delta\Phi w \, d\Omega = -\int_\Omega \Delta w\Phi \, d\Omega - \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma \tag{74.4}$$

Next we choose the weight w as follows:

$$-\Delta w = \delta_r(r'), \tag{74.5}$$

so that

$$-\int_\Omega \Delta w \Phi \, d\Omega = \Phi(r), \tag{74.6}$$

The weight w chosen this way is the Green's function for the Laplace operator, i.e.

$$w(r, r') = \frac{\log(r - r')}{2\pi} \text{ in 2d }, w(r, r') = \frac{1}{4\pi(r - r')} \text{ in 3d }. \tag{74.7}$$

Finally we add the source term, and we have the equation

$$\Phi(r) - \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma - \int_\Omega f w \, d\Omega = 0. \tag{74.8}$$

Only the source term is now integrated over the volume. This equation may now be discretized by standard methods.

74.2.1 Boundary Conditions

Boundary conditions may be set for either potential

$$\Phi = \Phi_\Gamma \text{ on } \Gamma, \tag{74.9}$$

or normal flux

$$-\frac{\partial \Phi}{\partial n} = g \text{ on } \Gamma. \tag{74.10}$$

74.3 Keywords

`Solver` `solver id`

Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere. Note also that the BEM discretization results to a full linear system in contrast to FEM discretizations and the ILU preconditioning settings are not available.

`Equation` `String [PoissonBEM]`
The name of the equation.

`Procedure` `File ["PoissonBEM" "PoissonBEMSolver"]`
This keyword is used to give the Elmer solver the place where to search for the equation solver.

`Variable` `String [Potential]`
Give a name to the field variable.

`Variable DOFs` `Integer [1]`
This keyword must be present, and *must* be set to the value 1.

`Exported Variable 1` `String Flux`
If this keyword is given, the output will include the normal flux at boundaries, the name must be exactly as given.

`Exported Variable 1 DOFs` `Integer [1]`
This keyword must be present if Flux values are to be computed, and *must* be set to the value 1.

`Equation` `eq id`

The equation section is used to define a set of equations for a body or set of bodies:

`PoissonBEM` `Logical`
if set to `True`, solve the Poisson equation, the name of this parameter must match the `Equation` setting in the `Solver` section.

If the mesh has any volume elements with a body id that corresponds to a body where to the Poisson equation is activated, the value of the potential is computed for these elements as a postprocessing step. Note that the computation of potential is not a trivial task, so large number of volume elements may result to long execution time.

`Boundary Condition` `bc id`

The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The one related to Poisson (BEM) equation are

`Body Id` `Integer`

Give body identification number for this boundary, used to reference body definitions in .sif file. This parameter must be set so that the ElmerSolver knows at which boundaries to solve the corresponding equation.

`Potential` `Real`

Known potential value at boundary.

`Flux` `Real`

Known normal flux at boundary.

`Normal Target Body` `Integer`

The direction of boundary normals are important for the success of the computation. They should point consistently outward from the boundaries. This is accomplished either if the mesh generator automatically orients the boundary elements consistently, or including in the mesh the parent (volume) elements of the boundaries and using this keyword. The value -1 of this parameter points to the side where there are no volume elements. If the parameter gets the value of the body id of the volume elements, the normal will point to that direction.

`Body Force` `bf id`

The source term for the Poisson equation may be given here. The volume integral is computed on a body with a volume mesh and the PoissonBEM equation set to true.

`Source` `Real`

The source term for the Poisson equation.

Model 75

BEM Solver for Helmholtz Equation

Module name: HelmholtzBEM
Module subroutines: HelmholtzBEMSolver
Module authors: Juha Ruokolainen
Document authors: Juha Ruokolainen

75.1 Introduction

This module solves the Helmholtz equation by boundary element method (BEM), where the differential equation is transformed to integral equation along the boundaries. On the boundaries either pressure or normal flux may be defined.

This is an old module which has limited use since the BEM matrix is assembled in full. This means that the memory consumption and speed of solution grow rather unfavorably with problem size. Efficient BEM solvers never create the full matrix but use iterative multilevel methods instead, such as the multipole expansion. Also, the solver only works in serial so parallel computing is not available to resolve the bottleneck.

75.2 Theory

The Helmholtz equation is mathematically described as

$$(k^2 + \Delta)\Phi = 0, \text{ in } \Omega. \tag{75.1}$$

In BEM we transform this equation to integral equation over boundaries. We start by multiplying the equation by a weight function and integrating over the volume, and integrating by parts

$$\int_\Omega (k^2 + \Delta\Phi) w \, d\Omega = \int_\Omega k^2 w \Phi d\Omega - \int_\Omega \nabla\Phi \cdot \nabla w \, d\Omega + \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma. \tag{75.2}$$

Similarly we may write an equation reversing the roles of Φ and w

$$\int_\Omega (k^2 + \Delta) w\Phi \, d\Omega = \int_\Omega k^2 w \Phi d\Omega - \int_\Omega \nabla w \cdot \nabla \Phi \, d\Omega + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma. \tag{75.3}$$

Subtracting the two equations we have

$$\int_\Omega (k^2 + \Delta)\Phi w \, d\Omega = \int_\Omega (k^2 + \Delta) w \Phi \, d\Omega - \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma \tag{75.4}$$

Next we choose the weight w as follows:

$$(k^2 + \Delta) w = \delta_r(r'), \tag{75.5}$$

so that

$$\int_\Omega (k^2 + \Delta) w\Phi \, d\Omega = \Phi(r), \tag{75.6}$$

The weight w chosen this way is the Green's function for the Helmholtz operator, i.e.

$$w(r, r') = \frac{1}{i4} H_0(k(r - r')) \text{ in 2d }, w(r, r') = \frac{1}{4\pi} \exp^{-ik(r-r')} \text{ in 3d }, \tag{75.7}$$

where H_0 is the Hankel function.

Finally we have the equation

$$\Phi(r) - \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma = 0. \tag{75.8}$$

75.2.1 Boundary Conditions

Boundary conditions may be set for either pressure

$$\Phi = \Phi_\Gamma \text{ on } \Gamma, \tag{75.9}$$

or normal flux

$$-\frac{\partial \Phi}{\partial n} = g \text{ on } \Gamma. \tag{75.10}$$

75.3 Keywords

`Simulation`

`Angular Frequency Real`
Give the value of the angular frequency for the simulation.

`Solver solver id`
Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere. Note also that the BEM discretization results to a full linear system in contrast to FEM discretizations and the ILU preconditioning settings are not available.

`Equation String [HelmholtzBEM]`
The name of the equation.

`Procedure File ["HelmholtzBEM" "HelmholtzBEMSolver"]`
This keyword is used to give the Elmer solver the place where to search for the equation solver.

`Variable String [Pressure]`
Give a name to the field variable.

`Variable DOFs Integer [2]`
This keyword must be present, and *must* be set to the value 2.

`Exported Variable 1 String Flux`
If this keyword is given, the output will include the normal flux at boundaries, the name must be exactly as given.

`Exported Variable 1 DOFs Integer [2]`
This keyword must be present if Flux values are to be computed, and *must* be set to the value 2.

`Equation eq id`
The equation section is used to define a set of equations for a body or set of bodies:

`HelmholtzBEM Logical`
if set to `True`, solve the Helmholtz equation, the name of this parameter must match the `Equation` setting in the `Solver` section.

If the mesh has any volume elements with a body id that corresponds to a body where to the Helmholtz equation is activated, the value of the pressure is computed for these elements as a postprocessing step. Note that the computation of potential is not a trivial task, so large number of volume elements may result to long execution time.

`Boundary Condition` `bc id`
: The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables. The one related to Helmholtz (BEM) equation are

`Body Id` `Integer`
: Give body identification number for this boundary, used to reference body definitions in .sif file. This parameter must be set so that the ElmerSolver knows at which boundaries to solve the corresponding equation.

`Pressure 1` `Real`
: Known real part of pressure at boundary.

`Pressure 2` `Real`
: Known imaginary part of pressure at boundary.

`Flux 1` `Real`
: Known real part of normal flux at boundary.

`Flux 2` `Real`
: Known real part of normal flux at boundary.

`Normal Target Body` `Integer`
: The direction of boundary normals are important for the success of the computation. They should point consistently outward from the boundaries. This is accomplished either if the mesh generator automatically orients the boundary elements consistently, or including in the mesh the parent (volume) elements of the boundaries and using this keyword. The value -1 of this parameter points to the side where there are no volume elements. If the parameter gets the value of the body id of the volume elements, the normal will point to that direction.

Model 76

Magnetoquasistatic approximation for axial symmetry

Module name: StatMagSolve
Module subroutines: StatMagSolver
Module authors: Juha Ruokolainen, Ville Savolainen, Jussi Heikonen, Peter Råback, Antti Pursula
Document authors: Ville Savolainen, Peter Råback, Antti Pursula, Mika Malinen

76.1 Introduction

Note: This solver is obsolete!

Maxwell's equations may generally be expressed by employing a scalar potential and a vector potential. The magnetic flux density is then the curl of the vector potential. In some cases the effect of the scalar potential vanishes and the system is fully described by the vector potential. These cases include magnetostatics problems where time-independent magnetic fields may be created by electromagnets with given current distributions or permanent ferromagnets. The solver considered here allows the first option, with non-homogeneous and non-linear magnetic materials.

The scalar potential may also be ignored in two-dimensional magnetoquasistatic cases when the current density acts in a direction orthogonal to the plane considered. Then eddy current effects relating to a sinusoidal evolution of the current density may also be considered at low frequencies. If there are no conductors in the system, this approximation reduces to the equations of magnetostatics.

This solver was historically developed for the axially symmetric cases and it should only be used in those. For handling problems in orthogonal Cartesian coordinates, see the modules `MagnetoDynamics` and `MagnetoDynamics2D` for 3-D and 2-D versions, respectively.

76.2 Theory

When there are no hard ferromagnets, a magnetostatics problem may be expressed using the magnetic vector potential $\vec{A}$ that gives the magnetic flux density as $\vec{B} = \nabla \times \vec{A}$. It is obtained directly using the Ampère's law, so that

$$\nabla \times \left(\frac{1}{\mu} \nabla \times \vec{A} \right) = \vec{\jmath}. \qquad (76.1)$$

Here μ is the magnetic permeability of the material. The equation may be non-linear through the magnetic permeability curve of a ferromagnetic material. The solver discussed is intended for handling the axially symmetric version of (76.1).

The axially symmetric version of (76.1) may also employed to handle magnetoquasistatic problems where the effect of the displacement current is ignored. If there are conductors in the system, the current density is then written as $\vec{\jmath} = \sigma\vec{E} + \vec{\jmath}_0$ where σ is the electric conductivity and the electric field $\vec{E}$ is given by

$$\vec{E} = -\frac{\partial \vec{A}}{\partial t}.$$

In the time-harmonic case the source current density $\vec{\jmath}_0$ is considered to be $\vec{\jmath}_0(x,t) = \vec{\jmath}_0(x)e^{i\omega t}$, where $\omega = 2\pi f$ is the angular frequency. Using a trial $\vec{A}(x,t) = \vec{A}_0(x)e^{i\omega t}$, we then obtain an equation for the amplitude:

$$\nabla \times \left(\frac{1}{\mu}\nabla \times \vec{A}_0\right) + i\omega\sigma\vec{A}_0 = \vec{\jmath}_0.$$

In the axially symmetric case, the magnetic flux density $\vec{B}$ has only r- and z-components, while the current density $\vec{\jmath}$ and the vector potential $\vec{A}$ have only ϕ-components, so that the equation to be solved is

$$\nabla \times \left(\frac{1}{\mu}\nabla \times A_\phi\vec{e}_\phi\right) + i\omega\sigma A_\phi\vec{e}_\phi = j_\phi\vec{e}_\phi. \qquad (76.2)$$

The vector potential satisfies now automatically the Coulomb gauge. After the solution the heat generation in the conductors may be computed from

$$h = \frac{1}{2}\sigma\omega^2|\vec{A}_0|^2.$$

In contrast to the stationary case where A_ϕ is real and the equation has only one unknown, in the harmonic case the equation has two unknowns — the in-phase and the out-of-phase component of the vector potential. The magnetic flux density may generally be calculated from the vector potential as a post-processing step. Both the vector potential and the magnetic flux density components are then provided. The variable names in the result file are `magnetic vector potential` and `magnetic flux density i`, with `i=1` and `i=2`.

76.2.1 Boundary Conditions

For the magnetostatics equation one can apply either Dirichlet or homogeneous natural boundary conditions. In both cases, one must check that the computational domain is extended far enough to avoid numerical errors.

The Dirichlet boundary condition for A_ϕ is

$$A_\phi = A^b_\phi. \qquad (76.3)$$

In practice, when the Dirichlet condition is used, one usually takes $A^b_\phi = 0$. If a Dirichlet condition is not specified, the homogeneous natural boundary condition is used.

76.3 Keywords

`Simulation`

`Frequency` `Real`
: Frequency f if harmonic simulation is used.

`Angular Frequency` `Real`
: Angular frequency $\omega = 2\pi f$ if harmonic simulation is used, alternative to the previous one.

`Constants`

`Permeability of Vacuum` `Real` [$4\pi 10^{-7}$]

`Solver` `solver id`
: Note that all the keywords related to linear solver (starting with `Linear System`) may be used in this solver as well. They are defined elsewhere.

`Equation` `String [Static Magnetic Field]`
: The name of the equation.

`Variable` `String [Aphi]`
: The name of the variable.

`Variable Dofs` `Integer`
: Number of dofs in the field, this should be one for the steady-state case and two for time-harmonic analysis.

`Procedure` `File ["StatMagSolve" "StatMagSolver"]`
: The name of the file and subroutine.

`Harmonic Simulation` `Logical`
: Assume time-harmonic simulation.

`Calculate Magnetic Flux` `Logical [True]`
: By this flag the computation of the magnetic flux is activated. The default is `False`.

`Calculate Magnetic Flux Abs` `Logical [True]`
: Sometimes it is useful to have the absolute magnetic flux available for nonlinear material laws. Then this flag can be turned on. The default is `False`.

`Calculate Joule Heating` `Logical [True]`
: In large computations the automatic computation of the Joule heating may be turned off by this keyword. The default is `False`. The keyword is only applicable for the harmonic case. The computation results to two additional variables. `Joule Heating` gives the absolute heating and `Joule Field` the field that gives the heating when multiplied by the electric conductivity. This may be needed if the electric conductivity is discontinuous making also the heating power discontinuous.

`Desired Heating Power` `Real`
: A constant that gives the desired total heating power in Watts. If the keyword is active, then the `Joule Heating` and `Joule Field` are multiplied by the ratio of the desired and computed heating powers.

`Nonlinear System Convergence Tolerance` `Real`
: This keyword gives a criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations k is small enough

$$||A_\phi^k - A_\phi^{k-1}|| < \epsilon ||A_\phi^k||,$$

where ϵ is the value given with this keyword.

`Nonlinear System Max Iterations` `Integer`
: The maximum number of nonlinear iterations the solver is allowed to do. If neither the material parameters nor the boundary conditions are functions of the solution, the problem is linear and this should be set to be 1.

`Nonlinear System Relaxation Factor` `Real`
: Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achieve convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:

$$A_\phi^{'} = \lambda A_\phi^k + (1-\lambda) A_\phi^{k-1},$$

where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

`Equation` `eq id`
: The equation section is used to define a set of equations for a body or set of bodies:

`Static Magnetic Field` `Logical`
: If set to `True`, solve the magnetostatics equation.

`Body Force` `bf id`
The body force section may be used to give additional force terms for the equations.

`Current Density` `Real`
Specifies the azimuthal component of the current density. It may be a positive or negative constant, or a function of a given variable.

`Current Phase Angle` `Real`
Specifies the phase angle of the current density in degrees. The default phase angle is zero. Applies only to the time-harmonic case.

`Joule Heat` `Logical`
If this flag is active, the heat equation will automatically include the computed Joule heating as a heat source. Then it is assumed that Joule heating field ϕ is named `Joule field`. If there is no heat equation, this flag has no effect.

`Initial Condition` `ic id`
The initial condition section may be used to set initial values for the field variables. The following variable is active:

`Aphi` `Real`
The azimuthal component of the magnetic vector potential.

`Material` `mat id`
The material section is used to give the material parameter values. Material parameter available for the magnetostatics equation are.

`Relative Permeability` `Real`
The relative magnetic permeability μ is set with this keyword, defining the material relation $\vec{B} = \mu_r \mu_0 \vec{H}$. By default the relative magnetic permeability is one, but it may also be set otherwise or be a function of a given variable, typically given by the relation $\mu_r = \mu_r(|\vec{B}|)$. The value of the magnetic flux density $|\vec{B}|$ is available by the variable named `Absolute Magnetic Flux`.

`Electric Conductivity` `Real`
The electric conductivity defines the relation $\vec{\jmath} = \sigma \vec{E}$. Only isotropic case is possible. The parameter is needed only in the time-harmonic case.

`Boundary Condition` `bc id`
The boundary condition section holds the parameter values for various boundary condition types. A Dirichlet boundary condition may be set for the vector potential. The one related to the the axially symmetric magnetostatics problem is

`Aphi` `Real`
The azimuthal component of the magnetic vector potential.

Model 77

Linear Constraints

Module name: included in solver (SolverUtils)
Module subroutines: SolveWithLinearRestriction
Module authors: Mika Juntunen
Document authors: Mika Juntunen

77.1 Introduction

This subroutine allows user to solve problems with linear constraints. Here constraints are forced with Lagrange multipliers. This method, however, does not always lead to a well-posed problem. Conditions that ensure a (unique) solution are excluded here, but the conditions are found in many books (check for example [1]).

77.2 Theory

The problem at hand is

$$\min_x x^T Ax - x^T f \tag{77.1}$$

Let's assume that we can solve this. Now we also want that the solution solves the system $Bx = g$. This gives constraints to our solution. The rank of B should be less or equal to the rank of A. Loosely speaking, the number of rows in B should be less or equal to the number of rows in A. The method of Lagrange multipliers fixes these two equations together and gives a new functional to minimize.

$$\min_x x^T Ax - x^T f + \lambda^T (Bx - g) \tag{77.2}$$

If A is symmetric, then simple variational approach leads to solving x out of system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{77.3}$$

Symmetry of A is not always needed, but then more powerful methods have to be used to get to the above system.

77.3 Limitations

- **General usage of the subroutine**
  This subroutine can not be used by just adding keywords to solver input file. You must somehow create the constraint matrix and then call for SolveWithLinearRestriction in your own subroutine or function. The reader is encouraged to check for details in ElmerTutorials.

- **EMatrix-field**
  The EMatrix-field of the solved system matrix is used passing constraint matrix to SolveWithLinear-Restriction. This will be a problem if some other function or subroutine tries to use the EMatrix-field. EMatrix-field of the constraint-matrix is internally used by SolveWithLinearRestriction and should therefore be left alone.
- **Exported Multipliers**
  The length of the vector that holds the multipliers is limited to be a multiply of the number of nodes in mesh. This means that the vector usually has extra entries. These entries are set to zero. This leads to problems in extracting the correct values from the result file. Also post processing with ElmerPost is at least tricky.
- Parallel solving is not yet implemented.

77.4 Keywords

`Solver` `solver-id`

`Export Lagrange Multiplier` `Logical`
: If the multiplier has some physical meaning, you can save it to result file and to post file. This feature has certain drawbacks, check subsection Limitations. Default is `False`.

`Lagrange Multiplier Name` `String`
: The name you want to call the exported multipliers. This keyword has no meaning if the previous keyword is set to `False`. Default name is `LagrangeMultiplier`.

Bibliography

[1] V. Girault and P.A. Raviart. *Finite element methods for Navier-Stokes equations*. Springer-Verlag, New York, Berlin, Heidelberg, 1986.

Model 78

Density Functional Theory

Module name: DFTSolver
Module subroutines: Poisson, WaveFunctionSolver, ChargeDensitySolver, xc
Module authors: Olli Mali, trad (xc)
Document authors: Olli Mali

78.1 Introduction

This is an instructional text for using Elmer solvers I created for DFT calculations during the year 2006 while preparing my Master's Thesis [7]. These Solvers are rather experimental and I would not recommend their use for highly complicated problems. Nevertheless they provide nice backbone for creating own DFT-solvers with finite element method.

78.2 Theory

In DFT, Kohn-Sham equations [1, 2] play central role. They are set of highly nonlinear equations which define uniquely the exact ground state charge density. From charge density the total energy of the system in ground state can be calculated, which is unfortunately not implemented in present code.

The Kohn-Sham equations have a form

$$\begin{aligned} \Big(-\tfrac{1}{2}\Delta + V_{EXT}(\boldsymbol{r}) + V_C[\rho(\boldsymbol{r})] + V_{XC}[\rho(\boldsymbol{r})]\Big)\psi_k(\boldsymbol{r}) &= \varepsilon_k\,\psi_k(\boldsymbol{r}) \\ \rho(\boldsymbol{r}) &= \textstyle\sum_{k=1}^{N} |\psi_k(\boldsymbol{r})|^2 \quad , \end{aligned} \tag{78.1}$$

where KS-orbitals $\psi_k(\boldsymbol{r})$ are normalized, $\int \psi_k(\boldsymbol{r})^2 d\boldsymbol{r} = 1$, for eack $k = 1, 2, \ldots, N$. V_{EXT} is the external potential caused by the nuclei, V_C is the non-interacting Coulomb potential and V_{XC} is the exchange correlation potential that includes all the complicated many body effects, at least approximates. Nice explanation from the widely used Local Density Approximation can be found from [3]. Nonlinearity occurs in eigenvalue problem, where the operator depends on the solution of the eigen problem.

Self-Consistent iteration

The equations (78.1) are solved with self-consistent iteration (fixed point iteration). In this iteration Coulomb and external potentials are solved from Poisson equation. The iteration steps are as follows:

1. Begin with previous or initial guess for charge density ρ^j
2. Solve new electric potential from Poisson equation,

$$-\Delta V^{j+1}(\boldsymbol{r}) = \tfrac{1}{4\pi}\rho^j(\boldsymbol{r}) - \sum_{i=1}^{M} Z_i\,\delta(\boldsymbol{r} - \boldsymbol{r}_i) \quad , \tag{78.2}$$

where δ refers to Dirac's delta distribution (point load).

3. Solve eigenvalue problem,

$$\Big(-\tfrac{1}{2}\Delta + V^{j+1}(\boldsymbol{r}) + V_{XC}^{j+1}(\boldsymbol{r})\Big)\psi_k(\boldsymbol{r}) = \varepsilon_k\psi_k(\boldsymbol{r}) \quad , \tag{78.3}$$

   where V_{XC}^{j+1} is calculated via some function W from point values of charge density ρ^j. $V_{XC}^{j+1}(\boldsymbol{r}) = W(\rho^j(\boldsymbol{r})))$.

4. Sum new charge density,

$$\rho^{j+1}(\boldsymbol{r}) = \sum_{k=1}^{N} w_k\,\psi_k(\boldsymbol{r})^2 \quad , \tag{78.4}$$

   where the weight coefficients w_k depend on the numbers of electrons in orbitals. Extensive overview of calculation of molecular orbitals can be found from [4, 5].

The point load at the nuclei location requires, that *exactly at each nuclei there has to be a node* in the mesh. For the functionality of the solvers no other requirements exists for the mesh or domain.

Unfortunately convergence of this iteration procedure is not guaranteed. For simple atoms (Z = 1,2,3,4) code converges within any tolerance limits but for more complicated molecules or atoms usually not. Sensible tolerances were found to between 10^{-6} or 10^{-4}.

Boundary Conditions

In theory the zero level of the potential can be set arbitrarily and often in practice one uses condition $V(\boldsymbol{r}) \to 0$, when $|\boldsymbol{r}| \to \infty$. Of course in real calculations the domain Ω is finite and we set, $V(\boldsymbol{r}) = 0$ if $\boldsymbol{r} \in \partial\Omega$. One also assumes Ω to be large enough, so that charge density vanishes on the boundary, $\rho(\boldsymbol{r}) = 0$ if $\boldsymbol{r} \in \partial\Omega$, so we set $\psi_k(\boldsymbol{r}) = 0$ if $\boldsymbol{r} \in \partial\Omega$.

In Kohn-Sham -equations in order to obtain positive definite coefficient matrix on the left-hand side of eigenvalue problem (78.1), one sets $V(\boldsymbol{r}) \to C$, when $|\boldsymbol{r}| \to \infty$. The constant C has to be large enough, so the eigenvalues are shifted positive. But too large value slows the convergence of the eigenvalue solver.

78.3 Keywords

From the structure of the self-consistent iteration it was natural to divide the solution procedure for three solvers, Poisson solver, eigensolver and charge density summation. For each solver some keywords to control the solution procedure were added.

Poisson Solver

Poisson Solver demands knowledge about the locations of the nuclei and their atomic numbers. There has to be nodes in the mesh at the nuclei locations, or else error will occur. Following example demonstrates how nuclei of the water molecule with two atoms of atomic numbers $Z = 1$ (Hydrogen) and single with $Z = 8$ (Oxygen) are set to the coordinates $(0.0, 0.0, 0.0)$ (Oxygen) and $(-1.43, 1.11, 0.0)$ and $(1.43, 1.11, 0.0)$ (Hydrogens). The rows beginning with ! are comments.

```
!
! NOFnuclei is the number of nuclei in the structure.
!

NOFnuclei = Integer 3

!
! NucleiTable is an array of the form
```

```
 ! NucleiTable( NOFnuclei, 4 ) where each row
 ! includes the information of one nucleus.
 ! The columns are from left to right :
 !
 ! atomic number, x-coordinate, y-coordinate and z-coordinate.
 !

NucleiTable(3,4) =  Real 8.0  0.0  0.0  0.0 \
                         1.0 -1.43 1.11 0.0 \
                         1.0  1.43 1.11 0.0
```

The self-consistent iteration requires heavy (under) relaxation to avoid divergence. Relaxation means linear mixing of present solution with previous one(s). It is possible to use *Guaranteed Reduction Pulay - method* [6, 7] where the mixing constants are calculated every time as a solution of a minimization problem, it's sensible to begin GR Pulay after some steps of linear mixing.

In following example the exponential relaxation scheme is changed to GR Pulay after 5 steps or if the mixing parameter exceeds value 0.5 . Use of constant mixing parameter instead of increasing one can be easily done by commenting out the first four uncommented lines and removing the comment sign `!` from following two lines.

```
!
! Select the relaxation method used, possibilities are
! constant mixing parameter a(k) = A or varying parameter
! with scheme a(k) = C + 1- A * Exp( B * k )
!

Relaxation Method = String "Exponential mixing"
Relaxation Parameter A = Real 1.0
Relaxation Parameter B = Real 0.05
Relaxation Parameter C = Real 0.005

! Relaxation Method = String "Constant mixing"
! Relaxation Parameter A = Real 0.01

Start GRPulay after iterations = Integer 5
Start GRPulay if relaxation factor is more than = Real 0.5
```

Eigenproblem Solver

Eigenproblem solver demands knowledge about the type of exchange correlation approximation used. Namely the expression of W in third self-consistent iteration step. In module `xc.f90` there are several different formulae for LDA approximations. Some of them include spin directions and are to be used with different solver composition where KS-orbitals for up- and down-spins are calculated separately.

```
!
! Choose the type of the XC Potential, possible choices are:
!   "None"
!   "Perdew-Zunger"
!   "Von Barth-Hedin"
!   "Gunnarsson-Lundqvist"
!   "Perdew-Wang"
```

```
!

XC Potential type = String "Perdew-Zunger"
```

Charge Density Solver

Charge density solver demands knowledge about the number of KS-orbitals to be summed and the weights of each orbital. These are the N and w_k's in fourth self-consistent iteration step. In following example one sets $N = 5$ and $w_k = 2$, for all $k = 1, \ldots, 5$.

```
! Define the number of eigenmodes included on the
! calculation of charge density. Set weights for the
! eigen states. By default they are all 1.

Number of Eigenmodes Included = Integer 5
Weights of Eigen States(5,1) = Real 2.0 2.0 2.0 2.0 2.0
```

`Weights of the Eigen States` table has to be size $(N, 1)$. Naturally N has to be equal or less for the number of eigenstates to be solved in eigenvalue solver.

Bibliography

[1] P. Hohenberg, W Kohn *Inhomogeneous Electron Gas* Physical Review, Volume 136, Number 3B, 9 November 1964

[2] W. Kohn, L. J. Sham, *Self-Consistent Equations Including Exchange and Correlation Effects* Physical Review, Volume 140, Number 4A, 15 November 1965

[3] M. P. Das, *Density Functional Theory: Many-Body Effects Without Tears* Proceedings of the Miniworkshop on "Methods of Electronic Structure Calculations", ICTP, Trieste, Italy 10 August - 4 September 1992

[4] Peter Atkins, Ronald Frieman, *Molecular Quantum Mechanics* Oxford University Press Inc., Fourth edition 2005

[5] Andrew R. Leach, *Molecular Modelling*, Pearson Education Limited, Second edition 2001

[6] D. R. Bowler, M. J. Gillan, *An efficient and robust technique for achieving self consistency in electronic structure calculations* Chemical Physics Letters 325 sivut 473-476 (2000)

[7] O. Mali, *Kohn-Sham -yhtälöiden ratkaiseminen elementtimenetelmällä*,
Diplomityö, 19. syyskuuta 2006

Model 79

Parallel I/O using HDF5 library

Module name: XdmfWriter
Module subroutines: XdmfWriter
Module authors: Mikko Lyly
Document authors: Mikko Lyly

79.1 Introduction

This subroutine is intended for saving parallel results in Xdmf/HDF5 format. The advantages of the Xdmf/HDF5-format over the native parallel ep-format are the following:

- All results are stored in only two files (results.xmf and results.h5)
- The results are stored in a binary format with reduced storage requirements
- The results can be visualized on the fly during the solution

The result files written by XdmfWriter can be opened and visualized e.g. with Paraview.

At the moment, the module is available for the parallel version of Elmer only. Because of this, it has been isolated from the main build system. The source code for XdmfWriter can be found from the source tree in `misc/xdmf` and it should be compiled by the user as follows:

```
elmerf90 -I$HDF5/include -L$HDF5/lib -o XdmfWriter XdmfWriter.f90 -lhdf5_fortran -lhdf5
```

The environment variable `$HDF5` defines the installation directory for the HDF5-library.

79.2 Keywords

`Solver` `solver id`

`Equation` `String "Xdmf"`
The name of the equation.

`Procedure` `File "XdmfWriter" "XdmfWriter"`
The name of the file and subroutine.

`Base File Name` `String`
Specifies the base file name of the output files. The default is `results`.

`Single Precision` `Logical`
This keyword specifies the output precision (4 byte single precision floating point numbers vs. 8 byte double precision floating point numbers). The default is `false`.

The following keywords define the scalar and vector fields to be saved:

`Scalar Field i` `String`
: The scalar fields to be saved, for example `Pressure`.

`Vector Field i` `String`
: The vector fields to be saved, for example `Velocity`.

The number `i` must be in the range 1...1000.

79.3 Example

The following SIF-block saves results in Xdmf/HDF5-format for the Navier-Stokes equations:

```
Solver 1
  Equation = String "Xdmf"
  Procedure = File "XdmfWriter" "XdmfWriter"
  Base File Name = String "MyResults"
  Single Precision = Logical True
  Scalar Field 1 = String "Pressure"
  Vector Field 1 = String "Velocity"
End
```

Index