# Elmer Models Manual

CSC – IT Center for Science

September 22, 2009

# Copyrights

# About Elmer Models Manual

The Elmer Models Manual is part of the documentation of Elmer finite element software. Elmer Models Manual is a selection of independent chapters describing different modules a.k.a. solvers of the ElmerSolver.

The modular structure of the manual reflects the modular architecture of the software where new models may be written without any changes in the main program. Each solver has a separate section for theory and keywords, and often also some additional information is given, for example on 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 6.0. Latest documentations and program versions of Elmer are available (or links are provided) at `http://www.csc.fi/elmer`.

# Contents

# Model 1

# Heat Equation

**Module name**: included in solver
**Module subroutines**: HeatSolve
**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen, Ville Savolainen
**Document edited**: July 29th 2002

## 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.

## 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 $\rho$ 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 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

$$H(T) = \int_0^T \left( \rho c_p + \rho L \frac{\partial f}{\partial \lambda} \right) d\lambda, \tag{1.4}$$

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:

$$c_{p,\mathrm{eff}} = \frac{\partial H}{\partial T}, \tag{1.5}$$

and

$$c_{p,\mathrm{eff}} = \left( \frac{\nabla H \cdot \nabla H}{\nabla T \cdot \nabla T} \right)^{1/2}. \tag{1.6}$$

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

$$c_{p,\mathrm{eff}} = \frac{\partial H / \partial t}{\partial T / \partial t}. \tag{1.7}$$

### 1.2.3 Additional Heat Sources

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

$$h_f = 2\mu \bar{\bar{\varepsilon}} : \bar{\bar{\varepsilon}}. \tag{1.8}$$

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

$$h_m = \frac{1}{\sigma} \vec{J} \cdot \vec{J}. \tag{1.9}$$

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

$$\vec{J} = \sigma(\vec{E} + \vec{u} \times \vec{B}). \tag{1.10}$$

### 1.2.4 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

$$T = T_b. \tag{1.11}$$

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

Heat flux depending on heat transfer coefficient $\alpha$ and external temperature $T_{\mathrm{ext}}$ may be written as

$$-k \frac{\partial T}{\partial n} = \alpha(T - T_{ext}). \tag{1.12}$$

Both variables $\alpha$ and $T_{\mathrm{ext}}$ can be constant or functions of time, position or other variables. If the heat transfer coefficient $\alpha$ 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

$$-k\frac{\partial T}{\partial n} = \sigma\varepsilon(T^4 - T_{\text{ext}}^4). \tag{1.13}$$

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

If the surface $k$ is receiving radiation from other surfaces in the system, then the heat flux reads as

$$-k_k\frac{\partial T_k}{\partial n_k} = \sigma\varepsilon_k(T_k^4 - \frac{1}{A_k\varepsilon_k}\sum_{i=1}^{N} G_{ik}\varepsilon_i T_i^4 A_i), \tag{1.14}$$

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 Gebhardt factors, and $N$ represents the total number of radiating surfaces present in the system. Emissivities are assumed to be constant on each surface.

The heat equation is nonlinear when radiation is modelled. The nonlinear term in the boundary condition (1.13) can be linearized as

$$T^4 - T_{\text{ext}}^4 \approx (\mathcal{T}^3 + T_{\text{ext}}\mathcal{T}^2 + T_{\text{ext}}^2\mathcal{T} + T_{\text{ext}}^3)(T - T_{\text{ext}}), \tag{1.15}$$

where $\mathcal{T}$ is the temperature from the previous iteration.

One may also give an additional heat flux term as

$$-k\frac{\partial T}{\partial n} = q. \tag{1.16}$$

## 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.

`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 $\epsilon$ 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 $\epsilon$ 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 $\lambda$ 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 $\epsilon$ 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.

In some cases the geometry or the emissivities of the radiation boundaries change. This may require the recomputation of the view factors and Gebhardt factors. For that purpose also dynamic computation of the factors is enabled and it is controlled by the keywords below. The radiation factors are also automatically computed if no files for the factors are given allthough radiation boundaries exist.

`Update View Factors` `Logical`
: The recomputation of the view factors is activated by setting the value of this flag to `True`. `False` is the default.

`Update Gebhardt Factors` `Logical`
: If the emissivities depend on the solution the Gebhardt factors may need to be recomputed. This is activated by setting giving this flag value `True`. `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 Gabhardt factors less dense. This consequently will enable more efficient sparse matrix strategies in solving the Gebhardt factors. The value for this parameter might be of the order 10e-8.

`Minimum Gebhardt Factor` `Real`
: The Gebhardt factors make part of matrix dense. By neglecting the smallest Gebhardt 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 Gebhardt 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 Gebhardt factors are treated in an implicit manner whereas the smallest are treated explicitely. The value should lie in between zero (fully explicit) and one (fully implicit).

`Matrix Topology Fixed` `Logical`
: If the Gebhardt 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.

`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.

`Gebhardt 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.

`Gebhardt Factors Fixed Tolerance` `Real`
: This keywords defines the coupled system tolerance for the heat equation after which the recomputation of Gebhardt 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 Gebhardt factors as well.

`Gebhardt Factors Solver Full` `Logical`
: If the view factor matrix is relatively sparse it will make sense to use a sparse matrix equation for solving the Gebhardt factors. This flag may be used if a full matrix should be desired.

`Gebhardt Factors Solver Iterative` `Logical`
: If the Gebhardt factors are solved from a sparse matrix equation also the type of solver may be selected. The default is direct `umfpack` solver. Sometimes the memory usage may be a problem or the direct strategy simply not efficient enough. Then an iterative `cgs` solver may be used instead.

`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.

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

`Heat Equation` `String`
: 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`
: An additional heat source $h$ for the heat equation may be given with this keyword.

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

`Joule Heat` `Logical`
: If set `True`, triggers use of the electromagnetic heating. This keywords accouns 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 Heater Control` `Real`
: This keyword activates a normaliazation of the `Heat Source` so that the integral heating power is the desired objective.

`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.

`Compressiblity Model Real`
This setting may be used to set the compressibilty 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.

`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 $\alpha$ 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`. Note that, when using the diffuse gray radiation model, the file containing the Gebhardt factors must be given in the simulation section.

`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.

`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 parant elements.

`Radiation Target Body` `Integer`
This flag may be used to set the direction of the outward pointing normal. This is used when computing viewfactors. A body identification number must be given. The default is that the normal points to less dense material or outward on outer boundaries.

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

# Model 2

# Navier-Stokes Equation

**Module name**: included in solver
**Module subroutines**: FlowSolve
**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen, Peter Råback
**Document edited**: 10.8.2004

## 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 $\mu$ 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 $\gamma$ 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 $\gamma$ depends solely on the internal molecular properties of the gas.

An imcompressibe flow is characterized by the condition $\rho$=constant, from which it follows that

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

Enforcing the constraint (2.8) in (2.27), (2.2) and (2.3), the equations reduce to the Navier-Stokes equations

$$\rho \left( \frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} \right) - \nabla \cdot (2\mu\bar{\bar{\varepsilon}}) + \nabla p = \rho\vec{f}, \tag{2.9}$$

$$\nabla \cdot \vec{u} = 0. \tag{2.10}$$

Compressible flows are modelled by the equations (2.27)-(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) desrcibe 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 $\beta$ 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 $u^\theta$ is

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

where $\omega$ 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 $\gamma$ 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 $\nabla_s$ is the surface gradient operator.

The coefficient $\gamma$ 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 $\vartheta$ 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 $\gamma$ can be any user defined function in Elmer. One may also give the force vector on a boundary directly as in

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

### 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:

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

and

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

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 succesfully 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 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**

$$\eta = \begin{cases} \eta_0 \dot{\gamma}^{n-1} & \text{if } \dot{\gamma} > \dot{\gamma}_0, \\ \eta_0 \dot{\gamma}_0^{n-1} & \text{if } \dot{\gamma} \leq \dot{\gamma}_0. \end{cases} \tag{2.23}$$

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

**Carreau-Yasuda**

$$\eta = \eta_\infty + \Delta\eta \left(1 + (c\dot{\gamma})^y\right)^{\frac{n-1}{y}}, \tag{2.24}$$

where $\eta_\infty$ 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 $y = 2$.

The model can be made temperature dependent. One choice is to multiply $\Delta\eta$ and $c$ by factor $\exp(d(1/T - 1/T_0))$, where $d$ and $T_0$ are model parameters.

**Cross**

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

where again $\eta_\infty$ is the high shearrate viscosity.

**Powell-Eyring**

$$\eta = \eta_\infty + \Delta\eta \frac{\text{asinh}(c\dot{\gamma})}{c\dot{\gamma}}. \tag{2.26}$$

### 2.2.4 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 proportinal to the velocity and thus the modified momentum equation reads

$$\rho\left(\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u}\right) - \nabla \cdot \overline{\overline{\sigma}} + r\vec{u} = \rho\vec{f}, \tag{2.27}$$

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.

### 2.2.5 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

$$\vec{f_e} = -\rho_e \nabla\phi, \tag{2.28}$$

where $\rho_e$ is the charge density and $\phi$ 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.

### 2.2.6 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

$$\vec{f_m} = \vec{J} \times \vec{B}, \tag{2.29}$$

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

$$\vec{J} = \sigma(\vec{E} + \vec{u} \times \vec{B}). \tag{2.30}$$

## 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.

`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 $\epsilon$ is the value given with this keyword.

`Nonlinear System Max Iterations Integer`
The maxmimum 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 $\epsilon$ 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 achive 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 $\lambda$ 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 $\epsilon$ 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 succesfully 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`.

`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 mementum equations.

`Potential Force Logical`
If this is set true the force used for the electricstatic 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.

`Initial Condition ic id`
The initial codition 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-$\varepsilon$ 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`.

`Compressiblity Model String`
This setting may be used to set the compressibilty 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 `Compressiblity Model` is set to the value `Perfect Gas`.

`Specific Heat Ratio Real`
The ratio of specfic heats (in constant pressure versus in constant volume) may be given with this keyword. This setting applies only if the `Compressiblity 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-$\varepsilon$ 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. Ablicable 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.

`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 Ref Real`
Paremeter $T_0$ in the thermal Carreau-Yasuda model.

`Viscosity Temp Exp Real`
Paremeter $d$ in the thermal Carreau-Yasuda model.

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 a 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 tangetial 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 explicitely.

`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-$\varepsilon$ 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
**Document edited**: Oct 29th 2003

## 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 $\rho$ 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 had 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 $\vec{j}_i$ perpendicular to the boundary by

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

In relative mass units, this may be written as

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

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 $\beta$ 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 $\epsilon$ 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 $\lambda$ 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 $\epsilon$ 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 succesfully 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
**Document edited**: March 3rd 2009

## 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, $\Gamma$ 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 implies 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: $\boldsymbol{A}\vec{c} = \vec{S}$, with the system matrix $\boldsymbol{A}$ and the solution vector $\vec{c}$ on the left-hand side and the force vector $\vec{S}$ 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: $\tilde{\boldsymbol{A}}\vec{c} = \vec{\tilde{S}}$
5. a residual is obtained from the un-manipulated system: $\vec{R} = \boldsymbol{A}\vec{c} - \vec{S}$
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 Reaction Equation Variable_name]`
The name of the equation, e.g., `Advection Reaction Equation Tracer`.

`Discontinuous Galerkin Logical`
needs to be set to `true`

`Variable String Variable_name`
The name of the variable, e.g., `Tracer`. As the variable is a DG variable (i.e., not renderable e.g. in ElmerPost), the user usually adds the option `-nooutput` in order to avoid output in the output files

`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 $\epsilon$ 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 $\epsilon$ is the value given with this keyword.

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

`Exported Variable 1 String`
in order to write the DG variable `Variable_name` to a for ElmerPost (non-DG mesh) readable variable, an exported variable with an arbitrary name (e.g., `Exported Variable 1 = Variable_name Nodal Result`) has to be defined. It is then used to interpolate the DG result to nodal values in order to display them.

`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, $\Gamma$

`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

# Linear Elasticity Solver

**Module name**: included in solver
**Module subroutines**: StressSolve

**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen
**Document edited**: 22.04.2007

## 5.1 Introduction

This module computes displacement field from 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.

## 5.2 Theory

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

$$\rho \frac{\partial^2 \vec{d}}{\partial t^2} - \nabla \cdot \tau = \vec{f}, \tag{5.1}$$

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

$$\tau^{ij} = C^{ijkl}\varepsilon_{kl} - \beta^{ij}(T - T_0), \tag{5.2}$$

where $\varepsilon$ 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 $\beta$ 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:

$$\varepsilon = \frac{1}{2}(\nabla \vec{d} + (\nabla \vec{d})^T). \tag{5.3}$$

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

$$\tau = 2\mu\varepsilon + \lambda \nabla \cdot \vec{d} I - \beta(T - T_0)I, \tag{5.4}$$

where $\mu$ and $\lambda$ are the first and second Lame parameters respectively, $\beta$ the heat expansion coefficient, and $I$ is the unit tensor. Lame parameters in terms of Youngs modulus and Poisson ratio read

$$\lambda = \frac{Y\kappa}{(1+\kappa)(1-2\kappa)}, \quad \mu = \frac{Y}{2(1+\kappa)} \tag{5.5}$$

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

$$\mu = \frac{Y\kappa}{(1-\kappa^2)}. \tag{5.6}$$

Quantities $Y$ and $\kappa$ are the Youngs modulus and Poisson ratio respectively.

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

$$\tau_V = E\varepsilon_V, \tag{5.7}$$

where $\tau_V$ and $\varepsilon_V$ are the stress and strain vectors respectively. The $6 \times 6$ matrix $E$ (in 3D, $4 \times 4$ in 2D) is the matrix of elastic coefficients. The stress and strain vectors are defined as

$$\tau_V = (\tau_x \;\; \tau_y \;\; \tau_z \;\; \tau_{xy} \;\; \tau_{yz} \;\; \tau_{xz})^T \tag{5.8}$$

and

$$\varepsilon_V = (\varepsilon_x \;\; \varepsilon_y \;\; \varepsilon_z \;\; 2\varepsilon_{xy} \;\; 2\varepsilon_{yz} \;\; 2\varepsilon_{xz})^T. \tag{5.9}$$

In 2D the stress vector is

$$\tau_V = (\tau_x \;\; \tau_y \;\; \tau_z \;\; \tau_{xy})^T \tag{5.10}$$

and the strain vector

$$\varepsilon_V = (\varepsilon_x \;\; \varepsilon_y \;\; \varepsilon_z \;\; 2\varepsilon_{xy})^T. \tag{5.11}$$

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 $\phi$. The matrix $E$ is given as input for the anisotropic material model of Elmer.

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

$$\rho\omega^2\vec{\phi} = \nabla \cdot \tau(\vec{\phi}), \tag{5.12}$$

or

$$\omega^2 \int_\Omega \rho\phi_k\psi_k \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega, \tag{5.13}$$

where $\omega$ 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 $\sigma_{ij}$, and solve the variational equation

$$\omega^2 \int_\Omega \rho\phi_k\psi_k \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega + \int_\Omega \sigma_{ij} \frac{\partial\phi_k}{\partial x_i}\frac{\partial\psi_k}{\partial x_j} \, d\Omega. \tag{5.14}$$

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

$$-\lambda \int_\Omega \sigma_{ij} \frac{\partial\phi_k}{\partial x_i}\frac{\partial\psi_k}{\partial x_j} \, d\Omega = \int_\Omega \tau_{ij}(\vec{\phi})\epsilon_{ij}(\vec{\psi}) \, d\Omega, \tag{5.15}$$

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

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

### 5.2.1 Boundary Conditions

For each boundary either a Dirichlet boundary condition

$$d_i = d_i^b \tag{5.16}$$

or a force boundary condition

$$\tau \cdot \vec{n} = \vec{g} \tag{5.17}$$

must be given.

### 5.2.2 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,

$$\vec{F} = \boldsymbol{K}\vec{D} \tag{5.18}$$

where $\vec{F} = (F_x\, F_y\, F_z\, M_x\, M_y\, M_z)^T$ and $\vec{X} = (D_x\, D_y\, D_z\, \phi_x\, \phi_y\, \phi_z)^T$. 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,

$$F_i = \int_A f_i \, dA. \tag{5.19}$$

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

$$D_i = \frac{1}{A} \int_A d_i \, dA. \tag{5.20}$$

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 $\boldsymbol{K}^{-1}$. 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 as in the model lumping by pure forces which may be a reason anonther reason to favour this procedure.

## 5.3 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 [Stress Analysis]`
: The name of the equation.

`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.

`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 Stresses` `Logical`
If set to `true` the stress tensor will be computed and written to output in addition to Von Mises stress.

`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 Displacements` `Logical`
This keyword defined if the displacements or forces are set and thereby chooces the model lumping aklgorhitm.

`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 maintaied 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.

`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 situtation $\tau_{zz} = 0$. 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 1,2,3` `Real`
May be used to give volume force.

`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 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 $6 \times 6$ (3D) or $4 \times 4$ (2D and axisymmetric) matrix for the anisotropic case. Although the matrices are symmetric, all entries must be given.

`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 $3 \times 3$ tensor for 3D cases, and $2 \times 2$ 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 keyord to `True` enables the user to input Youngs Modulus matrix with respect to the principal directions of anisotropy. Otherwise Youngs 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.

`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`$= 1, 2, 3$.

`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.

# Model 6

# Mesh Adaptation Solver

**Module name**: included in solver
**Module subroutines**: MeshSolver
**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen
**Document edited**: April 5th 2002

## 6.1 Introduction

Moving boundaries are often encourtered in different types of computations, i.e. 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 contranst to regenerate the whole mesh.

For time dependent simulations the mesh deformation velocity is also computed. The name of this variable is `Mesh Velocity`.

## 6.2 Theory

The equation for elastic deformation of the mesh, given displacement of the boundaries, may be written as

$$-\nabla \cdot \tau = 0, \tag{6.1}$$

where, $\vec{d}$ is the mesh displacement field and $\tau$ the stress tensor.

The stress tensor given in terms of Lame parameters is:

$$\tau = 2\mu\varepsilon + \lambda\nabla \cdot \vec{d}I \tag{6.2}$$

where $\mu$ and $\lambda$ are the first and second Lame parameters respectively, and $I$ is the unit tensor. The linearized strains are given as:

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

Lame parameters in terms of Youngs modulus and Poisson ratio read

$$\mu = \frac{Y\kappa}{(1-\kappa)(1-2\kappa)}, \quad \lambda = \frac{Y}{2(1+\kappa)} \tag{6.4}$$

Quantities $Y$ and $\kappa$ are the Youngs modulus and Poisson ratio respectively. 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.

### 6.2.1 Boundary Conditions

For each boundary a Dirichlet boundary condition

$$d_i = d_i^b \tag{6.5}$$

may be given. Usually this the displacement is given a priori or computed by, for example, the elasticity solvers.

## 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 [Mesh Update]`
The name of the equation.

`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 in Navier equations.

`Poisson Ratio Real`
For isotropic materials 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. The one related to Navier equations are

`Mesh Update i Real`
Dirichlet boundary condition for each displacement component `i`$= 1, 2, 3$. The boundary displacement may be computed some other solver. The computed displacment field then may be used in the setting in the following way:

`Mesh Update i Equals Displacement i` with `i=1,2,3`. Including such lines in the boundary condition setting will give the mesh update on the boundary directly from the displacement solver.

## 6.4 Examples

### 6.4.1 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.

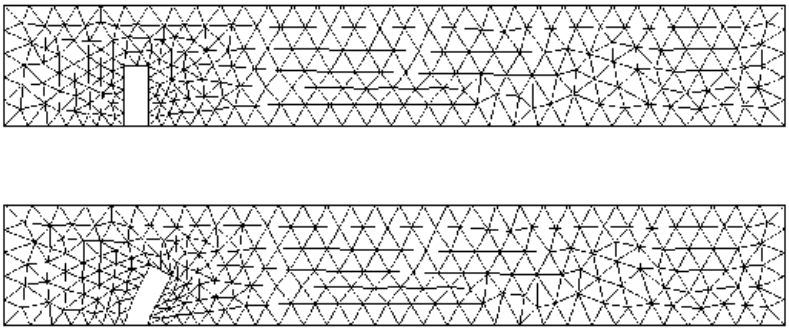

Figure 6.1: The original computational mesh (up), and the mesh of the converged solution (down) of a FSI computation.

# Model 7

# Elastic Linear Plate Solver

**Module name**: Smitc
**Module subroutines**: SmitcSolver
**Module authors**: Mikko Lyly, Jani Paavilainen
**Document authors**: Mikko Lyly, Peter Råback
**Document created**: August 26th 2002

## 7.1 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.

### 7.1.1 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 $\Omega$ is the midsurface and $t$ the thickness) is obtained from the kinematic equations

$$u_x(x,y,z) = -\theta_x(x,y) \cdot z \tag{7.1}$$
$$u_y(x,y,z) = -\theta_y(x,y) \cdot z \tag{7.2}$$
$$u_z(x,y,z) = w(x,y) \tag{7.3}$$

where $\theta_x$ and $\theta_y$ 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

$$\frac{1}{2}\int_\Omega \underline{\underline{\kappa}} : \underline{\underline{m}}\, d\Omega + \int_\Omega \underline{\gamma} \cdot \underline{q}\, d\Omega - \int_\Omega pw\, d\Omega \tag{7.4}$$

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, $\underline{\underline{m}} = \underline{\underline{\mathcal{E}}} : \underline{\underline{\kappa}}$ is the bending moment, and $\underline{q} = \underline{\underline{\mathcal{G}}} \cdot \underline{\gamma}$ 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 $\underline{\underline{\mathcal{G}}} \cdot \underline{\gamma} = Gt\underline{\gamma}$ and

$$\underline{\underline{\mathcal{E}}} : \underline{\underline{\kappa}} = K[\underline{\underline{\kappa}} + \frac{\nu}{1-\nu}(tr\underline{\underline{\kappa}})\underline{\underline{I}}] \tag{7.5}$$

where $K = Et^3/[12(1-\nu^2)]$ is the bending stiffness, $E$ is Young's modulus, $G$ shear modulus, and $\nu$ Poisson ratio. The design of the tensors $\mathcal{E}$ and $\mathcal{G}$ for orthoropic and perforated materials is discussed in section 7.3.

The minimizer of the energy satisfies the equilibrium equations

$$\underline{\nabla} \cdot \underline{\underline{m}} + \underline{q} = 0 \tag{7.6}$$
$$-\nabla \cdot \underline{q} = p \tag{7.7}$$

### 7.1.2 Surface tension

When surface tension is present, the following term is added to the energy:

$$\frac{1}{2} \int_\Omega \underline{\nabla} w \cdot \mathcal{T} \cdot \underline{\nabla} w \, d\Omega \tag{7.8}$$

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 (7.7) is then rewritten as

$$-\nabla \cdot (\underline{q} + \mathcal{T} \cdot \nabla w) = p \tag{7.9}$$

### 7.1.3 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-homogenous 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-homogenous case is trated by adding the following terms in the energy:

$$\int_{\Gamma_{free}} q_n w \, d\Gamma + \int_{\Gamma_{free}} \underline{m}_n \cdot \underline{\theta} \, d\Gamma \tag{7.10}$$

where $q_n = \underline{q} \cdot \underline{n}$ and $\underline{m}_n = \underline{\underline{m}} \cdot \underline{n}$ are prescribed functions.

### 7.1.4 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 (7.1)-(7.9) by enforcing the constraint $\underline{\gamma} = \underline{0}$. The governing equations are then reduced to

$$K \Delta \Delta w - T \Delta w = p \tag{7.11}$$

### 7.1.5 Transient and natural mode analysis

A transient plate model is obtained by adding the interia term $\rho t \ddot{w}$ on the left hand-side of (7.7), (7.9), and (7.11). Here $\rho$ 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.

## 7.2 Finite element implementation

The direct minimization of (7.4) 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 (7.4) by the following numerical modification:

$$\int_\Omega \underline{\gamma}_h \cdot \underline{q}_h \, d\Omega \tag{7.12}$$

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

$$\underline{\gamma}_{h|K} = (a_K - b_K y, a_K + c_K x) \tag{7.13}$$

for any element $K$. The parameters $a_K, b_K$, and $c_K$, are determined from the conditions

$$\int_E (\underline{\gamma} - \underline{\gamma}_h) \cdot \underline{t} \, ds = 0 \tag{7.14}$$

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 interpolateed MIN3 element of Tessler and Hughes [7], and the TRIA3 element of MacNeal [5]. We refer to [3] for a more detailed discussion.

## 7.3 Elastic parameters for perforated plates

In microelectromechanical systems the plate stuctures 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 ortotropic 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 7.1. 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 ortoropic 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],

$$C_{11} = C_{22} = \frac{E}{b^2}\left\{\frac{b(b-2a)}{1-\nu^2} + \frac{a(b-2a)^2}{b}\right\} \tag{7.15}$$

$$C_{12} = C_{21} = \frac{\nu E(b-2a)}{b(1-\nu^2)} \tag{7.16}$$

$$C_{44} = \frac{E}{4b^2(1+\nu)}\left\{2b(b-2a) + \frac{12Ka(b-2a)}{bh^3}\right\}. \tag{7.17}$$

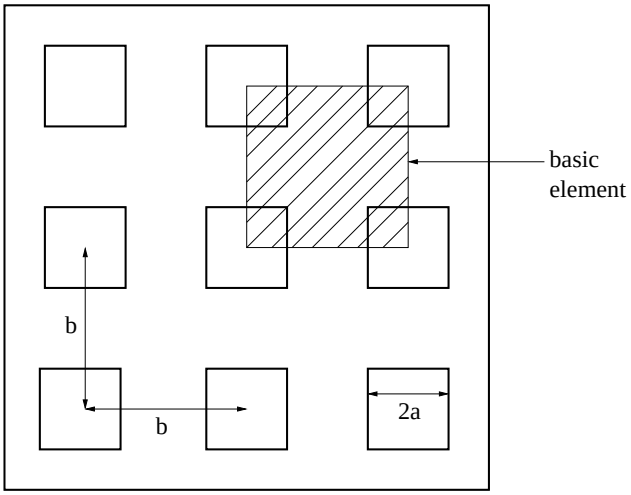


Figure 7.1: The basic element of the perforated plate consisting of five rectangular beams

where $K$ is a constant[1], defined as

$$K = \begin{cases} \frac{1}{3}\left(1 - 0.63\frac{b-2a}{h}\right)(b-2a)^3 h, & \text{jos } h > b-2a \\ \frac{1}{3}\left(1 - 0.63\frac{h}{b-2a}\right)(b-2a)h^3, & \text{jos } h < b-2a. \end{cases} \tag{7.18}$$

The midplane tension of the perfomarated plate may be reduced to lateral stresses of the ortotropic plate by a simple scaling,

$$T = \sqrt{(1 - 4a^2/b^2)}\, T_0, \tag{7.19}$$

where is the tension $T_0$ of the perforated plate. Using this reduced tension and the modified material parameters of equations (7.15), (7.16) and (7.17) the ortoropic 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.

## 7.4 Keywords

`Solver` `solver id`

`Equation` `String SmitcSolver`

`Procedure` `File "Smitc" "SmitcSolver"`
The procedure which inludes 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 displacment 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`.

`Procedure` `File "Smict" "SmitcSolver"`

`Material` `mat id`

`Density` `Real`
: Density of the plate.

`Poisson ratio` `Real`

`Youngs modulus` `Real`
: The elastic parameters are given with 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 multiplyed by the displacement.

`Damping` `Real`
: The local damping which results to a local force when multiplyed 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 8

# Helmholtz Solver

**Module name**: HelmholtzSolve
**Module subroutines**: HelmholtzSolver
**Module authors**: Juha Ruokolainen, Mikko Lyly
**Document authors**: Juha Ruokolainen
**Document edited**: March 30th 2006

## 8.1 Introduction

This module solves the Helmholtz equation, which is the Fourier transform of the wave equation.

## 8.2 Theory

For example, sound propagation in air is fairly well described by the wave equation:

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} - \nabla^2 p = 0. \tag{8.1}$$

When linear the equation may be written in frequency space as

$$k^2 P + \nabla^2 P = 0, \tag{8.2}$$

where $k = \omega/c$. This is the Helmholtz equation. The instantaneous pressure may be computed from the given field $P$:

$$p(t) = \Re(Pe^{i\omega t}) = \Re(P)\cos(\omega t) - \Im(P)\sin(\omega t), \tag{8.3}$$

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

$$(k^2 - ikD)P + \nabla^2 P = 0, \tag{8.4}$$

where $D$ is the damping factor.

### 8.2.1 Boundary Conditions

The usual boundary condition for the Helmholtz equation is to give the flux on the boundary:

$$\nabla P \cdot \vec{n} = g, \tag{8.5}$$

also Dirichlet boundary conditions may be set. The Sommerfeldt or far field boundary condition is as follows

$$\nabla P \cdot \vec{n} + \frac{i\omega}{Z} P = 0, \tag{8.6}$$

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.

## 8.3 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.

`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.

`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 use to give the value of the speed of sound.

`Sound Damping Real`
This keyword is use to give the value of the damping factor $D$ in equation 8.4.

`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 (8.6). Here the values `i`$= 1, 2$ correspond to the real and imaginary parts of $Z$.

# Model 9

# Electrostatics

**Module name**: StatElecSolve
**Module subroutines**: StatElecSolver
**Module authors**: Leila Puska, Antti Pursula, Peter Råback
**Document authors**: Peter Råback, Antti Pursula
**Document edited**: June 29th 2006

## 9.1 Introduction

The macroscopic electromagnetic theory is governed by the 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 10. Based on the potential, various field variables as well as physical parameters, such as capacitance, can be calculated.

## 9.2 Theory

When $\varepsilon\mu c^2 << 1$ we may assume that the Maxwell's equations are:

$$\nabla \cdot \vec{D} = \rho \tag{9.1}$$
$$\nabla \cdot \vec{B} = 0 \tag{9.2}$$
$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{9.3}$$
$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{9.4}$$

For linear materials the fields and fluxes are simply related, $\vec{B} = \mu\vec{H}$ăand $\vec{D} = \varepsilon\vec{E}$, where the permittivity $\varepsilon = \varepsilon_0\varepsilon_r$ is defined through the permittivity of vacuum $\varepsilon_0$ and the relative permittivity of the material $\varepsilon_r$.

In steady-state case the electric field may be expressed with a help of an electric scalar potential $\phi$,

$$\vec{E} = -\nabla\phi. \tag{9.5}$$

Assuming linear material law and using equation 9.1 gives

$$-\nabla \cdot \varepsilon\nabla\phi = \rho. \tag{9.6}$$

This is the electrostatic equation for non-conducting media.

The energy density of the field is

$$e = \frac{1}{2}\vec{E} \cdot \vec{D} = \frac{1}{2}\varepsilon(\nabla\phi)^2. \tag{9.7}$$

Thus the total energy of the field may be computed from

$$E = \frac{1}{2}\int_{\Omega} \varepsilon(\nabla\phi)^2 d\Omega. \tag{9.8}$$

If there is only one potential difference $\Phi$ present then the capacitance $C$ may be computed from

$$C = \frac{2E}{\Phi^2}. \tag{9.9}$$

### 9.2.1 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

$$-\varepsilon\nabla\phi\cdot\vec{n} = g. \tag{9.10}$$

The flux may be defined *e.g.* by the surface dharge density: $g = \sigma$.

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

$$\varepsilon\nabla\phi\cdot\vec{n} = \frac{\varepsilon_h}{h}\phi - \frac{1}{2}\rho h - \frac{\varepsilon_h}{h}\Phi_0 \qquad \text{on the boundary,} \tag{9.11}$$

where $\varepsilon_h$ and $h$ are the permittivity and the thickness of the oxidation layer respectively, $\rho$ is the static charge density of the layer, and $\Phi_0$ 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.

### 9.2.2 Capacitance matrix

There is a possibility to compute the capacitance matrix. The algorithm takes use of the original matrix $A$ before to the initial conditions are set. Now the point charges are given by

$$q = A\phi. \tag{9.12}$$

The induced charges on a body may be computed by summing up the point charges.

If there fre $n$ different bodies the boundary conditions are permutated $n$ times so that body $i$ gets a potential unity while others are set to zero potential.

$$C_{ij} = \sum_{\Gamma_j} q. \tag{9.13}$$

The symmetry of the matrix is ensured afterwords by setting

$$C = \frac{1}{2}(C + C^T). \tag{9.14}$$

## 9.3 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 choises 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.

## 9.4 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 capcitance 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`

`Electric Flux BC` `Logical`
Must be set to `True` if electric flux BC or oxidation layer BC is used.

`Electric Flux` `Real`
Neumann boundary condition.

`Surface Charge Density` `Real`
Another way to define flux condition. Identical to the previous keyword.

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.

`Nominal Potential Difference` `Real`
: The potential difference of the system.

`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 permutated in a very specific way.

# Model 10

# Static Current Conduction

**Module name**: StatCurrentSolve
**Module subroutines**: StatCurrentSolver
**Module authors**: Leila Puska, Antti Pursula, Peter Råback
**Document authors**: Antti Pursula
**Document edited**: August 2nd 2002

## 10.1 Introduction

The macroscopic electromagnetic theory is governed by the Maxwell's equations. This module solves the electrostatic potential in conducting medium allowing volume currents and electric power loss (Joule heating) to be derived.

## 10.2 Theory

In quasi-static approximation, when $\varepsilon\mu c^2 << 1$, the first and fourth Maxwell equation can be written as follows

$$\nabla \cdot \vec{D} = \rho \tag{10.1}$$

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{10.2}$$

The continuity equation for electric charges is easily derived from the above Maxwell Eqs. 10.1 and 10.2

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{J} = 0 \tag{10.3}$$

The Ohm's law for conducting material gives the relationship between current density and electric field,

$$\vec{J} = \sigma \vec{E} \tag{10.4}$$

In steady-state case the electric field may be expressed with a help of an electric scalar potential $\phi$,

$$\vec{E} = -\nabla\phi. \tag{10.5}$$

Starting from the continuity equation 10.3 and using Eqs. 10.4 and 10.5 we get

$$\nabla \cdot \sigma \nabla \phi = \frac{\partial \rho}{\partial t}. \tag{10.6}$$

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 volume current density is now calculated by

$$\vec{J} = -\sigma \nabla \phi, \tag{10.7}$$

and electric power loss density which is turned into heat by

$$h = \nabla \phi \cdot \sigma \nabla \phi. \tag{10.8}$$

The latter is often called the Joule heating. The total heating power is found by integrating the above equation over the conducting volume.

### 10.2.1 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 current $J_b$ on specified boundaries

$$J_b = \sigma \nabla \phi \cdot \vec{n}. \tag{10.9}$$

### 10.2.2 Power and current control

Sometimes the desired power or corrent of the system is known a priori. The control may be applied to the system. When the electric potential has been computed the heating power may be estimated from

$$P = \int_\Omega \nabla \phi \cdot \sigma \nabla \phi \, d\Omega. \tag{10.10}$$

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 field by a suitable variable. For power control the coefficient is

$$C_P = \sqrt{P_0/P}, \tag{10.11}$$

where $P_0$ is the desired power. For current control the coefficient is

$$C_I = I_0/I, \tag{10.12}$$

where $I_0$ is the desired total current.

## 10.3 Note on output control

The user can control which derived quantities (*i.e.* volume current and Joule heating) are calculated and additionally specify if he/she wants to output also the electric conductivity. The latter is useful when the conductivity depends for example on temperature. This feature is available only for isotropic (scalar) conductivities.

There are also available two choises 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.

## 10.4 Keywords

```
Solver  solver id

    Equation  String Stat Current 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 "StatCurrentSolve" "StatCurrentSolver"`
Following are listed three keywords with default values for output control.

`Calculate Volume Current Logical [True]`

`Calculate Joule Heating Logical [True]`

`Constant Weights Logical [True]`
Used to turn constant weighting on for the results.

`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`
Possibility for a current source, not used often though.

`Joule Heat Logical`
If this flag is active the Heat equation will automatically compute the quantity $\nabla\phi \cdot \sigma\nabla\phi$ as heat source. Then it is assumed that $\phi$ 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 11

# Magnetostatics

**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
**Document edited**: June 29th 2006

## 11.1 Introduction

The Maxwell's equations may generally be expressed with a scalar and a vector potential. The magnetic field is then the curl of the vector potential. In some cases the scalar potential vanishes and the system is fully described by the vector potential. These cases includes magnetostatics and time-harmonic induction at low frequencies.

Magnetostatics describes the time-independent magnetic fields. The magnetic field may be created by electromagnets with given current distributions or permanent ferromagnets. This solver allows the first option, with non-homogeneous and non-linear magnetic materials.

In some cases the current density varies sinusoidally with time. If the field varies slowly and there are no conductors in the system the problem is described with the stationary model. However, in conductors the magnetic field results in additional currents that make the equation complex.

## 11.2 Theory

When there are no hard ferromagnets, the magnetostatics problems may be expressed with magnetic vector potential $\vec{A}$ that satisfies $\vec{B} = \nabla \times \vec{A}$. It is obtained directly from the Ampère's law, with displacement current ignored, that

$$\nabla \times \left( \frac{1}{\mu} \nabla \times \vec{A} \right) = \vec{j}.$$

Here $\mu$ is the magnetic permeability of the material. The equation may be non-linear through the magnetic permeability curve of a ferromagnetic material. The magnetic permeability is specified in the `Material` section by the keyword `Magnetic Permeability`.

The equation above may be solved either in axial symmetry or in three dimensions. In 3D, a curl vector identity is used to transform the equation into the form

$$-\frac{1}{\mu} \nabla^2 \vec{A} = \vec{j},$$

which is valid when $\mu$ is not a function of space coordinates. Also, the vector potential $\vec{A}$ is *a priori* assumed to satisfy the Coulomb gauge ($\nabla \cdot \vec{A} = 0$).

If there are conductors in the system the electric field is obtained from

$$\vec{E} = \sigma \frac{\partial \vec{A}}{\partial t},$$

where $\sigma$ is the electric conductivity. In time-harmonic case the current density is sinusoidal $\vec{j} = \vec{j}_0 e^{i\omega t}$, where $\omega = 2\pi f$ is the angular frequency. Using a trial $\vec{A} = \vec{A}_0 e^{i\omega t}$ we obtain an equation for the amplitude

$$\nabla \times \left( \frac{1}{\mu} \nabla \times \vec{A}_0 \right) + i\omega\sigma \vec{A}_0 = \vec{j}_0.$$

If the geometry is axisymmetric, then the magnetic flux density $\vec{B}$ has only $r$- and $z$-components, and the current density $\vec{j}$ and the vector potential $\vec{A}$ only $\phi$-components, and

$$\nabla \times \left( \frac{1}{\mu} \nabla \times A_\phi \vec{e}_\phi \right) + i\omega\sigma A_\phi \vec{e}_\phi = j_\phi \vec{e}_\phi. \tag{11.1}$$

The current density is given as a body force with the keyword `Current Density`. The vector potential satisfies now automatically the Coulomb gauge.

In stady state case $A_\phi$ is real and there is 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. After solution the heat generation in the conductors may be computed from

$$h = \frac{1}{2} \sigma \omega^2 |\vec{A}_0|^2.$$

The magnetic flux density is calculated as a post-processing step from the vector potential. Both the vector potential and the magnetic flux density components are written in the result and ElmerPost files. The variable names in the result file are `magnetic vector potential` and `magnetic flux density 1`, `2` and `3`.

By definition, magnetostatics deals with steady-state problems. However, the problem may be solved nominally time-dependent. This merely means that it is solved for a set of given current densities.

### 11.2.1 Boundary Conditions

For the magnetostatics 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.

The Dirichlet boundary condition for $A_\phi$ is

$$A_\phi = A_\phi^b. \tag{11.2}$$

In practice, when Dirichlet condition is used, usually $A_\phi^b = 0$. The keyword for the Dirichlet boundary condition is `Magnetic Vector Potential`. If Dirichlet condition is not specified, natural boundary condition is used.

## 11.3 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 [Static Magnetic Field]`

The name of the equation.

`Variable` `String [Magnetic Vector Potential]`

The name of the variable.

`Procedure File ["StatMagSolve" "StatMagSolver"]`
The name of the file and subroutine.

`Calculate Magnetic Flux Logical [True]`
In large computations the automatic computation of the magnetic flux may be turned off by this keyword. The default is `True`.

`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 `True`. 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 the 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 $\epsilon$ 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, 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 achive 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 $\lambda$ 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. 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 is named $\phi$ 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:

`Magnetic Vector Potential 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.

`Magnetic Permeability Real`

The magnetic permeability $\mu$ is set with this keyword, defining the material relation $\vec{B} = \mu\vec{H}$. The magnetic permeability may be a constant (default is $4\pi 10^{-7}$) or a function of a given variable, typically given by the relation $\mu = \mu(|\vec{B}|)$. 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 $\vec{j} = \sigma\vec{E}$. 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. Dirichlet boundary condition may be set for the vector potential. The one related to the the axisymmetric magnetostatics problem is

`Magnetic Vector Potential Real`

The azimuthal component of the magnetic vector potential.

# Model 12

# Magnetic Induction Equation

**Module name**: included in solver / MagneticSolve as external procedure
**Module subroutines**: MagneticSolver
**Module authors**: Juha Ruokolainen
**Document authors**: Ville Savolainen, Antti Pursula
**Document edited**: May 24th 2005

## 12.1 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.

## 12.2 Theory

The magnetic induction equation may be derived from the Maxwell's equations, with the displacement current in Ampère's law neglected, and the Ohm's law for conducting fluids, $\vec{j} = \sigma(\vec{E} + \vec{v} \times \vec{B})$. This approximation for the behavior of electromagnetic fields in conducting, moving fluids is called magnetohydrodynamics.

The magnetic induction equation is given by

$$\frac{\partial \vec{B}}{\partial t} + \frac{1}{\sigma\mu}\nabla \times \nabla \times \vec{B} - \nabla \times (\vec{v} \times \vec{B}) = 0, \tag{12.1}$$

where $\sigma$ is the electric conductivity and $\mu$ the magnetic permeability of the material. These must be specified in the `Material` section by the keywords `Electric Conductivity` and `Magnetic Permeability`.

The force term induced by the magnetic field for the flow momentum equations is given by

$$\vec{f}_m = \vec{j} \times \vec{B}, \tag{12.2}$$

and the Joule heating in the heat equation by

$$h_m = \frac{1}{\sigma} |\vec{j}|^2 , \tag{12.3}$$

where $\vec{j}$ is the current density, calculated from the Ampère's law $\vec{j} = \nabla \times \vec{H}$. 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.

### 12.2.1 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

$$B_i = B_i^b. \tag{12.4}$$

$B_i^b$ 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 $B_\phi$ 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.

## 12.3 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 $\epsilon$ 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 achive 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 $\lambda$ 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 $\epsilon$ 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 $\mu_0$ 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 13

# Electrokinetics

**Module name**: Electrokinetics
**Module subroutines**: helmholtz_smoluchowski1, helmholtz_smoluchowski2, helmholtz_smoluchowski3, helmholtz_smoluchowski, getJouleHeat
**Module author**: Thomas Zwinger
**Document author**: Thomas Zwinger
**Document created**: April 13th 2005

## 13.1 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.

In most applications utilizing EOF, externally applied fields are sufficient small in order to justify neglecting electric heating (Joule heating) inside the fluid volume. Nevertheless, certain applications, such as High Voltage Capillary Electrophoresis (HV-CE) [2] or electrophoretic separation in polymer-based chips [3], demand the consideration of this effect .

## 13.2 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).

### 13.2.1 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]

$$\lambda_\mathrm{D} = \left( \frac{\epsilon_\mathrm{f} \epsilon_0 \, k_\mathrm{b} \, T_0}{2 \, n_0 \, z^2 \, e_0^2} \right)^{1/2} . \tag{13.1}$$

Here $e_0$ stands for the unit charge and $k_\mathrm{b}$ denotes the Boltzmann constant. The relative permittivity of the electrolyte and the permittivity of vacuum are given by $\epsilon_\mathrm{f}$ and $\epsilon_0$, respectively.

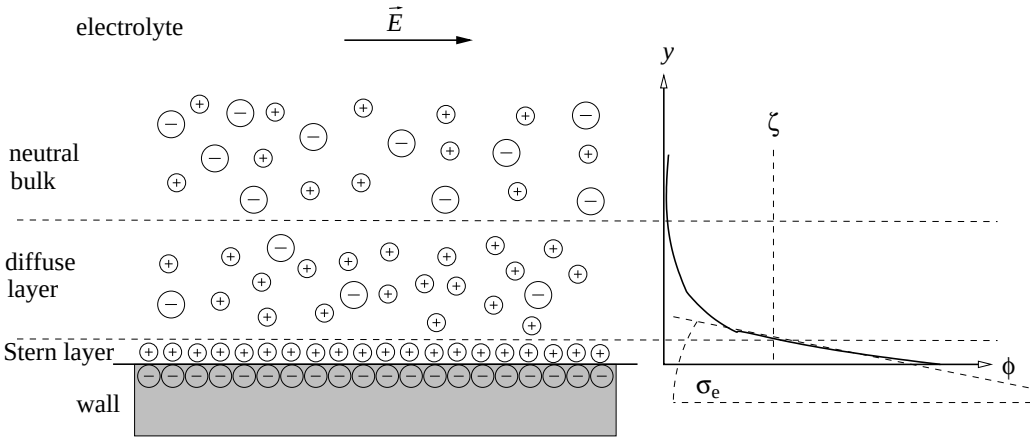


Figure 13.1: Structure of the EDL. The value of the induced potential, $\Phi$ at the Stern layer usually is referred to as the zeta-potential, $\zeta$

The potential, $\Phi$ and the volume charge density, $\rho_e$, within the EDL are tightly coupled to each other by the Poisson-Boltzmann equation (14.4) (see chapter 14). In order to exactly resolve the dynamics close to the walls, (14.4) 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 [4]. At the boundary, the tangential velocity is set to the *Helmholtz-Smoluchowski* velocity

$$\vec{u}_\mathrm{tang.} = \vec{u}_\mathrm{H-S} = \frac{\vec{E}_\mathrm{tang.}\epsilon_\mathrm{f}\epsilon_0\zeta}{\mu_\mathrm{f}}, \tag{13.2}$$

with $\mu_\mathrm{f}$ standing for the local fluid viscosity. The *zeta potential*, $\zeta$ – 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 (14.4) at the Stern layer. The tangential component, $\vec{E}_\mathrm{tang.}$, of the external electric field, $\vec{E}$, is evaluated from the outward pointing surface normal $\vec{n}$, applying the following relation

$$\vec{E}_\mathrm{tang.} = \vec{E} - \left(\vec{E}\cdot\vec{n}\right)\vec{n} \tag{13.3}$$

Alternatively, the resulting slip velocity may be related to the tangential field using the *Electroosmotic Mobility*, $\mu_\mathrm{EOF}$

$$\vec{u}_\mathrm{H-S} = \mu_\mathrm{EO}\,\vec{E}_\mathrm{tang.}. \tag{13.4}$$

A combination of (13.2) and (13.4) leads to the following identity

$$\mu_\mathrm{EO} = \frac{\epsilon_\mathrm{f}\epsilon_0\zeta}{\mu_\mathrm{f}}. \tag{13.5}$$

### 13.2.2 Joule Heating

Due to the small volume in microfluidic applications the additional heat produced by the external electric field needs to be considered. With the local electric conductivity, $\sigma$, and the local volume density, $\rho$, of the electrolyte the specific heat contribution by Joule heating from an external electric field, $\vec{E}$, is given by

$$h = \sigma\,\vec{E}\cdot\vec{E}/\rho. \tag{13.6}$$

The expression above can be added as body force to the heat transfer equation (1.1).

## 13.3 Limitations

- The Helmholtz-Smoluchowski velocity should not be applied if the non-dimensional group $\mathcal{L}$ defined in 13.2.1 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.
- The functions `helmholtz_smoluchowski{1,2,3}` cannot be applied with a number larger than the dimension of the domain.

## 13.4 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 (13.2)

`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 (13.2), 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 (13.2), then the zeta potential, $\zeta$, 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 (13.5)

`EO Mobility` `Real`
: Sets EO mobility for this boundary

### Keywords for getJouleHeat

`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`

`Electric Conductivity` `Real`
: electric conductivity of the electrolyte

`Density` `Real`
: volumetric density of the electrolyte

`Body Force` `bodyforce id`

`Heat Source` `= Variable Dummyargument Real Procedure "Electrokinetics" "getJouleHeat"`
: adds specific heat source for `HeatSolve`. The argument `Dummyargument` can be any existing variable, since it is not used to evaluate the Joule heating

# Bibliography

[1] G.E. Karniadakis and A. Beskok. *Micro flows : fundamentals and simulation*. Springer-Verlag, New York, Berlin, Heidelberg, 2001.

[2] J.H. Knox and K.A. McCormack. Temperature Effects in Capillary Electrophoresis. *Chromatographia*, 38(3/4):207–214, February 1994.

[3] T. Sikanen, T. Zwinger, S. Tuomikoski, S. Franssila, R. Lehtiniemi, C.-M. Fager, T. Kotiaho, and A. Pursula. Temperature modeling and measurement of an electrokinetic separation chip. *Microfluid Nanofluid*, 5:479–491, 2008.

[4] R.-J. Yang, L.-M. Fu, and Y.-C. Lin. Electroosmotic Flow in Microchannels. *J. Colloid and Interface Science*, 239:98–105, November 2001.

# Model 14

# Poisson-Boltzmann Equation

**Module name**: PoissonBoltzmannSolve
**Module subroutines**: PoissonBoltzmannSolve
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document edited**: 10.8.2004

## 14.1 Introduction

The macroscopic electromagnetic theory is governed by the 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.

## 14.2 Theory

The electrostatic equation for the electric potential $\phi$ yields,

$$-\nabla \cdot \varepsilon \nabla \phi = \rho, \tag{14.1}$$

where $\varepsilon$ is the permittivity of the medium and $\rho$ 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

$$\rho = \rho_0 + e(z^- n^- + z^+ n^+) \tag{14.2}$$

where $\rho_0$ 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 $\mu$ 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 $\mu_i$ is constant over the whole domain and thus the ion density obeys a Boltzmann distribution,

$$n = n_0 \mathrm{e}^{-ez\phi/k_B T} \tag{14.3}$$

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,

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 + ez^- n_0^- \mathrm{e}^{-ez^-\phi/k_B T} + ez^+ n_0^+ \mathrm{e}^{-ez^+\phi/k_B T}. \tag{14.4}$$

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

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 - 2ezn_0 \sinh(ez\phi/k_BT). \tag{14.5}$$

The Poisson-Boltzmann equation is obviously nonlinear. We will show the iterative procedure only for this case, the generic case is dealt similarly.

### 14.2.1 Iteration scheme

Defining $\alpha = 2ezn_0$ and $\beta = ez/k_BT$ the Poisson-Boltzmann equation for a symmetric electrolyte may be written as

$$-\nabla \cdot \varepsilon \nabla \phi = \rho_0 - \alpha \sinh(\beta\phi). \tag{14.6}$$

The straight-forward iterative procedure treats only the left-hand-side of the equation in an implicit manner,

$$-\nabla \cdot \varepsilon \nabla \phi^{(n+1)} = \rho_0 - \alpha \sinh(\beta\phi^{(n)}). \tag{14.7}$$

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

$$\sinh(\beta\phi^{(n+1)}) \approx \sinh(\beta\phi^{(n)}) + \beta \cosh(\beta\phi^{(n)})(\phi^{(n+1)} - \phi^{(n)}) \tag{14.8}$$

which results to the Newton iteration scheme

$$\begin{aligned} &\left[-\nabla \cdot \varepsilon \nabla + \alpha\beta \cosh(\beta\phi^{(n)})\right] \phi^{(n+1)} \\ &= \rho_0 - \alpha \sinh(\beta\phi^{(n)}) + \alpha\beta \cosh(\beta\phi^{(n)})\phi^{(n)}. \end{aligned} \tag{14.9}$$

This scheme has good convergence properties and is usually the method of choice.

### 14.2.2 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

$$\sigma = \varepsilon \nabla \phi \cdot \vec{n}, \tag{14.10}$$

where $\sigma$ is the surface charge density.

### 14.2.3 Derived quanties

When the potential has been solved the electric field may be obtained as a postprocessing step from

$$\vec{E} = -\nabla \phi. \tag{14.11}$$

Charge density may be obtained as the right-hand-side of the Poisson equation,

$$\rho = \rho_0 + ez^- n_0^- \mathrm{e}^{-ez^-\phi/k_BT} + ez^+ n_0^+ \mathrm{e}^{-ez^+\phi/k_BT}. \tag{14.12}$$

which in symmetric case yields,

$$\rho = \rho_0 - 2ezn_0 \sinh(ez\phi/k_BT). \tag{14.13}$$

The energy density of the field ay be computed from

$$e = \frac{1}{2}\vec{E} \cdot \vec{D} = \frac{1}{2}\varepsilon(\nabla\phi)^2. \tag{14.14}$$

However, in a more generic treatment also the connribution of the concentration should be included in the expression of the energy.

## 14.3 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.

## 14.4 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 iteraration 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 iteraration 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 occuring 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 unsymmtric 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 $\beta$ to the the Zeta potential.

`Poisson Boltzmann Alpha` `Real`
: This keyword gives the parameter $\alpha$

`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 15

# Reynolds Equation for Thin Film Flow

**Module name**: ReynoldsSolver
**Module subroutines**: ReynoldsSolver, ReynoldsHeatingSolver
**Module authors**: Peter Råback
**Module status**: Alpha
**Document authors**: Peter Råback
**Document created**: 24.10.2007
**Document edited**: 24.10.2007

## 15.1 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.

## 15.2 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 trough perforation holes the pressure may be solved from the equation

$$\nabla \cdot \left( \frac{\rho h^3}{12\eta} \nabla p \right) - Y \rho p = \frac{1}{2} \nabla \cdot (\rho h \vec{v}_t) + h \frac{\partial \rho}{\partial t} + \rho v_n, \tag{15.1}$$

where $\rho$ is the density, $\eta$ 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:

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

Here $\gamma = C_p/C_V$ is the specific heat ratio and $\beta$ 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,

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

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 (15.1) may be written in the following form:

$$\nabla \cdot \left( \frac{\rho h^3}{12\eta} \nabla p \right) - Y\rho p - \rho_p h \frac{\partial p}{\partial t} - \frac{1}{2}\rho_p h \vec{v}_t \cdot \nabla p = \frac{1}{2}\rho \nabla \cdot (h\vec{v}_t) + \rho v_n. \tag{15.2}$$

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

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

The normal velocity and gap height are naturally related by

$$v_n = \frac{\partial h}{\partial t}. \tag{15.3}$$

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

### 15.2.1 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,

$$Y = \frac{D^2}{32\eta b}, \tag{15.4}$$

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

$$Y = \frac{W^2}{12b\eta}. \tag{15.5}$$

### 15.2.2 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 $\eta$ depends on the Knudsen number, $K_n$, which is defined by

$$K_n = \frac{\lambda}{h}, \tag{15.6}$$

where $\lambda$ 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

$$\lambda = \frac{1}{1 + p/P_0}\lambda_0. \tag{15.7}$$

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

$$\eta = \frac{\eta_0}{1 + 9.638K_n^{1.159}}. \tag{15.8}$$

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

### 15.2.3 Boundary conditions for the Reynolds equation

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

$$\frac{\partial p}{\partial n} = 0. \tag{15.9}$$

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

$$\frac{\partial p}{\partial n} = \frac{p}{L}, \tag{15.10}$$

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

$$L = 0.8488(1.0 + 2.676K_n^{0.659})h. \tag{15.11}$$

### 15.2.4 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

$$\vec{q} = -\frac{h^3}{12\eta}\nabla p + h\vec{v}_t. \tag{15.12}$$

The total force acting on the surface is

$$\vec{F} = \int_A \left(p\vec{n} + \frac{\eta}{h}\vec{v}_t\right)\, dA, \tag{15.13}$$

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

$$h = \frac{h^3}{12\eta}|\nabla p|^2 + \frac{\eta}{h}|\vec{v}_t|^2. \tag{15.14}$$

Therefore the total heating power of the system is

$$Q = \int_A q\, dA. \tag{15.15}$$

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.

## 15.3 Keywords

The module includes two different solvers. `ReynoldsSolver` solves the differential equation (15.2) while `ReynoldsHeatingSolver` solves the equation (15.14) 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.

`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.

`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 $\beta$ 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.

`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 ReynoldsFluxSolver

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

`Reynolds Pressure Variable Name` `String`
: The name of the field that is assumed to provide the pressure field. The default is `FilmPressure`.

## Keywords for ReynoldsHeatingSolver

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

`Variable` `String FilmHeating`
: 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" "ReynoldsHeatingSolver"`
: The name of the module and procedure. These are fixed.

`Reynolds Pressure Variable Name` `String`
: The name of the field that is assumed to provide the pressure field. The default is `FilmPressure`.

# Bibliography

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

[2] P. Råback, 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. Lahdenperä, and T. Ryhänen. 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. Råback. 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.

# Model 16

# BEM Solver for Poisson Equation

**Module name**: PoissonBEM
**Module subroutines**: PoissonBEMSolver
**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen
**Document edited**: May 27th 2003

## 16.1 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.

## 16.2 Theory

The Poisson equation is mathematically described as

$$-\Delta\Phi - f = 0, \text{ in } \Omega, \tag{16.1}$$

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

$$-\int_\Omega \Delta\Phi w \, d\Omega = \int_\Omega \nabla\Phi \cdot \nabla w \, d\Omega - \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma. \tag{16.2}$$

Similarily we may write an equation reversing the roles of $\Phi$ and $w$

$$-\int_\Omega \Delta w\Phi \, d\Omega = \int_\Omega \nabla w \cdot \nabla\Phi \, d\Omega - \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma. \tag{16.3}$$

Substracting the two equations we have

$$-\int_\Omega \Delta\Phi w \, d\Omega = -\int_\Omega \Delta w\Phi \, d\Omega - \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma \tag{16.4}$$

Next we choose the weight $w$ as follows:

$$-\Delta w = \delta_r(r'), \tag{16.5}$$

so that

$$-\int_\Omega \Delta w \Phi \, d\Omega = \Phi(r), \tag{16.6}$$

The weight $w$ chosen this way is the Green's function for the Laplace operator, i.e.

$$w(r,r') = \frac{\log(r-r')}{2\pi} \text{ in 2d }, w(r,r') = \frac{1}{4\pi(r-r')} \text{ in 3d }. \tag{16.7}$$

Finally we add the source term, and we have the equation

$$\Phi(r) - \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma - \int_\Omega f w \, d\Omega = 0. \tag{16.8}$$

Only the source term is now integrated over the volume. This equation may now be discretized by standard methods.

### 16.2.1 Boundary Conditions

Boundary conditions may be set for either potential

$$\Phi = \Phi_\Gamma \text{ on } \Gamma, \tag{16.9}$$

or normal flux

$$-\frac{\partial \Phi}{\partial n} = g \text{ on } \Gamma. \tag{16.10}$$

## 16.3 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 17

# BEM Solver for Helmholtz Equation

**Module name**: HelmholtzBEM
**Module subroutines**: HelmholtzBEMSolver
**Module authors**: Juha Ruokolainen
**Document authors**: Juha Ruokolainen
**Document edited**: May 27th 2003

## 17.1 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.

## 17.2 Theory

The Helmholtz equation is mathematically described as

$$(k^2 + \Delta)\Phi = 0, \text{ in } \Omega. \tag{17.1}$$

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

$$\int_\Omega (k^2 + \Delta\Phi)w \, d\Omega = \int_\Omega k^2 w\Phi d\Omega - \int_\Omega \nabla\Phi \cdot \nabla w \, d\Omega + \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma. \tag{17.2}$$

Similarily we may write an equation reversing the roles of $\Phi$ and $w$

$$\int_\Omega (k^2 + \Delta)w\Phi \, d\Omega = \int_\Omega k^2 w\Phi d\Omega - \int_\Omega \nabla w \cdot \nabla\Phi \, d\Omega + \int_\Gamma \frac{\partial w}{\partial n}\Phi \, d\Gamma. \tag{17.3}$$

Substracting the two equations we have

$$\int_\Omega (k^2 + \Delta)\Phi w \, d\Omega = \int_\Omega (k^2 + \Delta)w\Phi \, d\Omega - \int_\Gamma \frac{\partial\Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n}\Phi \, d\Gamma \tag{17.4}$$

Next we choose the weight $w$ as follows:

$$(k^2 + \Delta)w = \delta_r(r'), \tag{17.5}$$

so that

$$\int_\Omega (k^2 + \Delta)w\Phi \, d\Omega = \Phi(r), \tag{17.6}$$

The weight $w$ chosen this way is the Green's function for the Helmholtz operator, i.e.

$$w(r,r') = \frac{1}{i4} H_0(k(r-r')) \text{ in 2d }, w(r,r') = \frac{1}{4\pi} \exp^{-ik(r-r')} \text{ in 3d }, \tag{17.7}$$

where $H_0$ is the Hankel function.

Finally we have the equation

$$\Phi(r) - \int_\Gamma \frac{\partial \Phi}{\partial n} w \, d\Gamma + \int_\Gamma \frac{\partial w}{\partial n} \Phi \, d\Gamma = 0. \tag{17.8}$$

### 17.2.1 Boundary Conditions

Boundary conditions may be set for either pressure

$$\Phi = \Phi_\Gamma \text{ on } \Gamma, \tag{17.9}$$

or normal flux

$$-\frac{\partial \Phi}{\partial n} = g \text{ on } \Gamma. \tag{17.10}$$

## 17.3 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 18

# Free Surface with Constant Flux

**Module name**: FreeSurfaceReduced
**Module subroutines**: FreeSurfaceReduced
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document edited**: August 5th 2002

## 18.1 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.

## 18.2 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

$$f(R,z) = \int_{R_0}^{R} (\vec{u}\cdot\vec{n})\, rds. \tag{18.1}$$

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

$$f(R,z_j) = f(R_1,z_1) \quad \forall j \in [1,M]. \tag{18.2}$$

Note that the factor $2\pi$ 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

$$u_n R^{(m)}\, dR^m = f(R^{(m)},z) - f(R_1,z_1) \tag{18.3}$$

or

$$R^{(m+1)} = R^{(m)} + \frac{f(R^{(m)},z) - f(R_1,z_1)}{u_n R^{(m)}}. \tag{18.4}$$

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.

## 18.3 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.

## 18.4 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 $\epsilon$ 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 19

# 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
**Document edited**: March 4th 2008

## 19.1 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.

## 19.2 Theory

The implicit equation describing the free surface is given by

$$F(\vec{x},t) = z - h(x,y,t), \tag{19.1}$$

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.,

$$\frac{\partial F}{\partial t} + \vec{u}_{\mathrm{m}} \nabla F = 0. \tag{19.2}$$

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

$$a_{\perp} = (\vec{u}_{\mathrm{m}} - \vec{u}) \cdot \vec{n}. \tag{19.3}$$

In Geophysical context (e.g., Glaciology), $a_{\perp}$ often is referred to as the net accumulation. With the surface unit normal defined as

$$\vec{n} = \frac{\nabla F}{\|\nabla F\|}, \tag{19.4}$$

this leads to

$$\frac{\partial F}{\partial t} + \vec{u} \nabla F = -\|\nabla F\| a_{\perp}. \tag{19.5}$$

Using the definition in (19.1), (19.5) can be rewritten in its explicit form

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} - w = \left[1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right]^{1/2} a_\perp, \tag{19.6}$$

with the components of fluid velocity vector at the free surface given as $\vec{u} = (u,\, v,\, w)^{\mathsf{T}}$. The variational formulation of (19.6) reads as

$$\int_\Omega \left(\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y}\right) \varphi \, dV = \int_\Omega \left\{ w + \left[1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right]^{1/2} a_\perp \right\} \varphi \, dV, \tag{19.7}$$

where the occurrence of $h$ in the right hand side is inserted from the previous time-step/non-linear iteration, hence linearizing the equation.

### 19.2.1 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

$$h_{\text{min}} \leq h \leq h_{\text{max}} \tag{19.8}$$

to (19.7) converting the variational formulation into a variational inequality. In order to obtain a with (19.8) consistent solution a method using Dirichlet constraints within the domain is applied. The exact procedure is the following:

1. construct the linear system: $\boldsymbol{A}\vec{h} = \vec{f}$, with the system matrix $\boldsymbol{A}$ and the solution vector $\vec{h}$ on the left-hand side and the force vector $\vec{f}$ on the right hand side
2. set nodes as *active* if (19.8) 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: $\tilde{\boldsymbol{A}}\vec{\tilde{h}} = \vec{\tilde{f}}$
5. a residual is obtained from the un-manipulated system: $\vec{R} = \boldsymbol{A}\vec{\tilde{h}} - \vec{f}$
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.

## 19.3 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.

## 19.4 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.

`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 $\epsilon$ 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.

`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 20

# Phase Change Solver

**Module name**: PhaseChangeSolve
**Module subroutines**: PhaseChangeSolve
**Module authors**: Peter Råback, Jussi Heikonen, Juha Ruokolainen
**Document authors**: Peter Råback
**Document created**: 22.10.2004
**Document edited**: 28.4.2006

## 20.1 Introduction

The boundary which separates a liquid and solid phase of a material is called a phase change or Stefan boundary. This subroutine defines the position of the phase change boundary. The phase change problem may occur for example in crystal growth and casting processes.

Phase change problems may be modeled using a fixed grid or alternatively distorting the grid so that the phase change boundary surface is exactly described by the computational mesh. Elmer has an internal fixed grid phase change model where the phase change is modelled by modifying the definition of heat capacity. This method is not limited by topological constraints but its accuracy may be questionable. If the phase change occurs within very sharp temperature interval the current method where the phase change surface is set exactly should be preferred,

The current methodology is limited into two dimensional cases where the phase change surface is nearly aligned with either of the coordinate axis.

## 20.2 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

$$L\rho\vec{v}\cdot\vec{n} = (\kappa_s\nabla T_s - \kappa_l\nabla T_l)\cdot\vec{n}, \tag{20.1}$$

where $\vec{n}$ is the normal of the phase change boundary, $\vec{v}$ is the velocity of the phase change boundary, $\rho$ is the density of the solid and $T_s$ and $T_l$ are the temperatures of the solid and liquid phases, and $\kappa_s$ and $\kappa_l$ are the thermal conductivities, respectively. In steady state the velocity of the phase change boundary should be equal to pull velocity, $\vec{v} = \vec{V}$ (bulk velocity of the solid phase).

### 20.2.1 Steady state algorithm

In steady state case the basic algorithm is based mainly on geometrical ideas. First the heat equation for temperature $T$ is solved by using a flux condition for the interface

$$q = L\rho V_n. \tag{20.2}$$

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

$$s_y = (y_{j,1} - y_i) + (x_i - x_{j,1})\frac{y_{j,2} - y_{i,1}}{x_{j,2} - x_{j,1}} \tag{20.3}$$

assuming that $x_i \in [x_{j,1}, x_{j,2}]$ while $s_x = 0$.

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

$$U = \int_A \vec{s} \cdot \vec{n}\, dA. \tag{20.4}$$

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,

$$c^{(m)} = c^{(m-1)}\frac{U^{(m-1)}}{U^{(m-1)} - U^{(m)}}. \tag{20.5}$$

In numerical tests this formula was found occasionally to overshoot and therefore a less aggressive version is used instead,

$$c^{(m)} = c^{(m-1)}\frac{1}{2}\frac{U^{(m-1)} + U^{(m)}}{U^{(m-1)} - U^{(m)}}. \tag{20.6}$$

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,

$$s^{(m)} = \frac{T_m - T^{(m)}}{T^{(m)} - T^{(m-1)}}s^{(m-1)}. \tag{20.7}$$

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.

### 20.2.2 Transient algorithm

In transient case the interface is set to be at the melting point when solving the heat equation. From the solution a heat flux is then obtained from

$$\vec{q} = \kappa_s \nabla T_s - \kappa_l \nabla T_l. \tag{20.8}$$

Now this heat flux is assumed to be used for the melting of the solid phase into liquid phase. Assuming again that the phase change boundary is mapped to the new position moving it only in the $y$-direction we get from equation (20.1) the velocity in the $y$-direction,

$$\rho L n_y (v_y - D_v \nabla^2 v_y) = \vec{q} \cdot \vec{n}. \tag{20.9}$$

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 fulfil 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 this is also done using the Galerkin method since the possibility of an additional diffusion factor is included. Therefore the equation is of the form,

$$\frac{\partial u_y}{\partial t} - D_u \nabla^2 u_y = v_y. \tag{20.10}$$

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 transient 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.

## 20.3 Applicable cases and limitations

The method has some limitations which are described below

- 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
- 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.

## 20.4 Keywords

`Solver` `solver id`

`Equation` `String "Phase Change"`

`Procedure` `File "PhaseChangeSolve" "PhaseChangeSolve"`
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

`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 $\lambda$ 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.

`Steady Transition Timestep` `Real`
: By default the transient algorhitm is applied in time-dependent cases. However, if the timestep given by this flag is smaller than the current timestep the steady state algorhitm is used even in the transient case.

`Averaging Transition Timestep` `Real`
: This keyword defines the timescale for the steady-state algorhitm in the transient case. If the timestep is smaller than given by this flag averaging is applied to the temperature field used to determine the isolines.

Some variables have a function only in steady state while others relate to the transient case. Below are the steady-state parameters for the `Solver` section.

`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 Newton After Iterations` `Logical`
: 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 Newton 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. Depending on the type of algorithm this may mean different things. For the steady algorithm this means that the temperature used for finding the isotherm is set to be the temperature of the triple point. Hence the isoterm will travel through the triple point. In the transient algorithm this means that the interface velocity is tuned so that the velocity at the triple point is zero.

`Use Absolute Norm for Convergence` `Logical`
: The steady-state solver returns the maximum phase change update as the norm and therefore this flag should be set `True`. The transient solver gives the norm of the finite element solution in the usual manner.

The following keywords may be defined in the transient algorithm.

`Pull Rate Control` `Logical`
: In transiet case 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 crysrallization 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. The steady state algorithm is solved by just one sweep.

`Nonlinear System Convergence Tolerance` `Real`
The tolerance for terminating the transient algorithm.

`Stabilize` `Logical`
The transient algorithm may require stabilization which decreases the oscillations of the solution.

`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.

`Heat Conductivity` `Real`
In a transient case the heat conductivities of the both materials must be given.

`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.

`Pull Velocity i` `Real`
For the transient algorithm the pull velocity of the boundary may be given with this keyword.

`Latent Heat` `Real`
The latent heat is the specific internal energy related to the phase change. The latent heat may also be a variable.

`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.

The module also includes subroutines `PullRate` and `PullPosition` that may be used to give boundary conditions for the mesh update solver. These subroutines relate to transient case. The syntax of the subroutines is as the following,

```
Mesh Update 2 = Variable Coordinate 1
  Real Procedure "PhaseChangeSolve" "PullPosition"
```

and

```
Convection Velocity 2 = Variable Coordinate 1
  Real Procedure "PhaseChangeSolve" "PullRate"
```

For the steady state case the heat equation often requires the heat flux as a boundary condition. For this there is also a precompiled subroutine that may be activated by the following lines in the command file,

```
Heat Flux BC = Logical True
Heat Flux = Variable Coordinate 1
  Real Procedure "PhaseChangeSolver" "MeltingHeat"
```

# Model 21

# Level-Set Method

**Module name**: LevelSet
**Module subroutines**: LevelSetSolver, LevelSetDistance, LevelSetIntegrate, LevelSetCurvature, LevelSetTimestep
**Module authors**: Peter Råback, Juha Ruokolainen
**Document authors**: Peter Råback
**Document created**: 5.4.2006
**Document edited**: 28.4.2006

## 21.1 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.

## 21.2 Theory

The interface is defined by a marker function $\phi$ 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

$$\frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi = a \tag{21.1}$$

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 be easily 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.

$$H_\alpha(x) = \begin{cases} 0, & x < -\alpha \\ f(\alpha/x) & |x| \leq \alpha \\ 1, & x > \alpha, \end{cases} \tag{21.2}$$

where the followin has been implemented

$$f(t) = \frac{1}{2}\left(1 + \sin\left(t\frac{\pi}{2}\right)\right) \tag{21.3}$$

while one could also use

$$f(t) = \frac{3}{4}\left(t - t^3/3\right) + \frac{1}{2}. \tag{21.4}$$

Here $\alpha$ is the interface bandwidth which equals typically the size of a few elements. Now the volume (area in 2D) is obtained by the integral

$$V = \int_\Omega H_\alpha(\phi)\, d\Omega. \tag{21.5}$$

After the same regularization the area (length in 2D) may be obtained from the integral

$$A = \int_\Omega \delta_\alpha(\phi)|\nabla\phi|\, d\Omega \tag{21.6}$$

where the delta function is

$$\delta_\alpha(x) = \begin{cases} 0, & |x| > \alpha \\ \frac{1}{2\alpha}\cos\left(\frac{x}{a}\pi\right), & |x| \leq \alpha. \end{cases} \tag{21.7}$$

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

$$d\phi = \frac{V_0 - V}{A}. \tag{21.8}$$

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,

$$\int_\Gamma \sigma\kappa \, d\Gamma = \int_\Omega \sigma\kappa\delta(\phi)\nabla\phi \, d\Omega, \tag{21.9}$$

where $\sigma$ is the surface tension coefficient and $\kappa$ the curvature of the interface given by

$$\kappa = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}. \tag{21.10}$$

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 $\kappa$,

$$\kappa - c_\kappa\nabla^2\kappa = \nabla \cdot \nabla\tilde{\phi}. \tag{21.11}$$

Here $c_\kappa$ 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.

## 21.3 Keywords

### LevelSetSolver

This subroutine uses the finite element method to solve the equation (21.1). 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 smoothening 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 $\infty$. Typically this should be larger that the level-set bandwidth $\alpha$ 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 (21.5) and (21.6). 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 (21.8). 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 avalution 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 (21.11).

```
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 levelset 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 levelset methodolohy to smear out the viscosity between inside and outside values.

`Viscosity Real`
The value of the viscosity outside the domain (negative levelset 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 22

# Density Functional Theory

**Module name**: DFTSolver
**Module subroutines**: Poisson, WaveFunctionSolver, ChargeDensitySolver, xc
**Module authors**: Olli Mali, trad (xc)
**Document authors**: Olli Mali
**Document created**: 10.12.2006
**Document edited**: 18.12.2006

## 22.1 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.

## 22.2 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

$$\begin{aligned} \Big( -\tfrac{1}{2}\Delta + V_{EXT}(\boldsymbol{r}) + V_C[\rho(\boldsymbol{r})] + V_{XC}[\rho(\boldsymbol{r})] \Big) \psi_k(\boldsymbol{r}) &= \varepsilon_k \, \psi_k(\boldsymbol{r}) \\ \rho(\boldsymbol{r}) &= \textstyle\sum_{k=1}^{N} |\psi_k(\boldsymbol{r})|^2 \quad , \end{aligned} \tag{22.1}$$

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 Approximatio 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 (22.1) 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 $\rho^j$

2. Solve new electric potential from Poisson equation,

$$-\Delta V^{j+1}(\boldsymbol{r}) = \tfrac{1}{4\pi}\rho^{j}(\boldsymbol{r}) - \sum_{i=1}^{M} Z_i\,\delta(\boldsymbol{r}-\boldsymbol{r}_i) \quad , \tag{22.2}$$

   where $\delta$ refes to Dirac's delta distribution (point load).

3. Solve eigenvalue problem,

$$\Big(-\tfrac{1}{2}\Delta + V^{j+1}(\boldsymbol{r}) + V_{XC}^{j+1}(\boldsymbol{r})\Big)\psi_k(\boldsymbol{r}) = \varepsilon_k\psi_k(\boldsymbol{r}) \quad , \tag{22.3}$$

   where $V_{XC}^{j+1}$ is calculated via some function $W$ from point values of charge density $\rho^j$. $V_{XC}^{j+1}(\boldsymbol{r}) = W(\rho^j(\boldsymbol{r})))$.

4. Sum new charge density,

$$\rho^{j+1}(\boldsymbol{r}) = \sum_{k=1}^{N} w_k\,\psi_k(\boldsymbol{r})^2 \quad , \tag{22.4}$$

   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 $\Omega$ is finite and we set, $V(\boldsymbol{r}) = 0$ if $\boldsymbol{r} \in \partial\Omega$. One also assumes $\Omega$ 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 (22.1), 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 slowers the convergence of the eigenvalue solver.

## 22.3 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 colums 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 choises 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 effficient 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 23

# System Reduction for Displacement Solvers

**Module name**: RigidBodyReduction
**Module subroutines**: RigidBody
**Module authors**: Antti Pursula
**Document authors**: Antti Pursula
**Document edited**: August 27th 2003

## 23.1 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 obsolite 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.

## 23.2 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 $\alpha$ 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

$$x = Pu, \tag{23.1}$$

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

$$f = Qb, \tag{23.2}$$

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

$$Ax = APu = b \tag{23.3}$$

and

$$Bu = f = Qb. \tag{23.4}$$

Combining the equations gives $Bu = QAPu$ and thus

$$B = QAP. \tag{23.5}$$

With a suitable order of the rotations one can write

$$Q = P^T \equiv C, \tag{23.6}$$

and

$$B = CAC^T. \tag{23.7}$$

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

$$x = C^T u. \tag{23.8}$$

## 23.3 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

## 23.4 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.

## 23.5 Examples

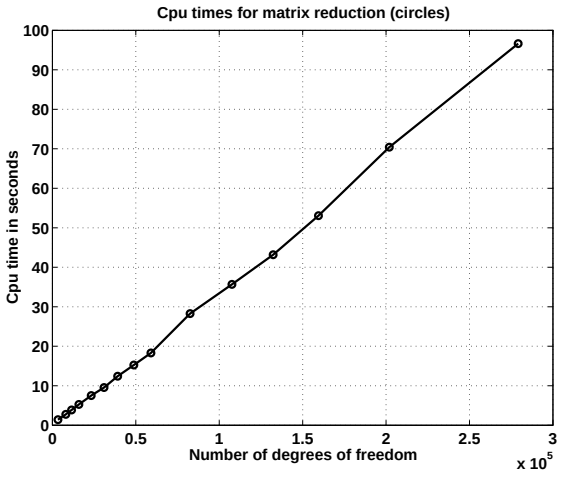


Figure 23.1: The cpu time required for the matrix reduction operations depends linearly on the degrees of freedom in the system.

# Model 24

# Electrostatics of Moving Rigid Bodies

**Module name**: MovingElstatSolver
**Module subroutines**: MovingElstatSolver
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document created**: 26.1.2006
**Document edited**: 17.5.2006

## 24.1 Introduction

This solver is tailored for solving electrostatic problems that occur in the movement of rigid bodies in respect to one another. Here the movement is assumed to be a combination of rotations and translations. We are mostly interested in lumped quantities. The most important quantity is the capacitance of the moving body at different positions. In addition the sensitivity of the capacitance and the moment of the electric force may be computed. The information is saved in a generic tabulated form and also as a lumped circuit model of Aplac.

For a more generic cases of the electrostatics and mesh adaptation the user is encouraged to use the existing separate solvers.

## 24.2 Electrostatics

Assuming a constant permittivity $\varepsilon$, absence of free charges, and non-conducting media the equation for the electrostatic potential $\phi$ yields,

$$-\varepsilon \nabla \cdot \nabla \phi = 0. \tag{24.1}$$

Obviously the constant multiplier may be dropped for convenience.

The energy of the electric field may be computed from

$$E = \frac{1}{2}\varepsilon \int_\Omega |\nabla \phi|^2 d\Omega. \tag{24.2}$$

If there is only one potential difference $\Phi$ present then the capacitance $C$ may be computed from

$$C = \frac{2E}{\Phi^2}. \tag{24.3}$$

The electric force is calculated by integrating the electrostatic Maxwell stress tensor over the specified surface. Using the stress tensor $\overline{\overline{T}}$ the total force on the surface $S$ can be expressed as

$$\vec{F} = \int_S \overline{\overline{T}} \cdot \vec{n}\, dS. \tag{24.4}$$

The components of the Maxwell stress tensor for linear medium are

$$T_{ij} = -D_i E_j + \frac{1}{2}\delta_{ij}\vec{D}\cdot\vec{E}, \tag{24.5}$$

where electric field $\vec{E}$ and electric displacement field $\vec{D}$ are obtained from

$$\vec{E} = -\nabla\phi, \tag{24.6}$$

and

$$\vec{D} = -\varepsilon\nabla\phi. \tag{24.7}$$

The moment around a given point $\vec{r}_0$ is given by,

$$\vec{F} = \int_S \overline{\overline{T}}\cdot\vec{n}\times(\vec{r}-\vec{r}_0)\,dS. \tag{24.8}$$

One may get an secondary estimation for the capacitance from the integral of the surface charges.

$$C' = \frac{1}{\Phi}\int_S \vec{D}\cdot\vec{n}\,dS. \tag{24.9}$$

This estimate of capacitance has a much bigger numerical error than the one defined by the volume integral. However the estimate may be usufull in evaluating the accuracy of the capacitance. The volume integral approaches the exact capacitance from above while the surface integral approaches it from below.

## 24.3 Mesh movement

The generic mesh movement of Elmer is based on a linear elasticity model. This is often an overkill since this makes the solution of the mesh movement computationally much more expensive than the solution of the potential equation. Therefore in the mesh movement it is assumed that the displacements of the main direction $u_i$, $i = 1, 2, 3$ are independent. Then the displacement of each directions is given by the Laplace equation

$$-\nabla\cdot\nabla u_i = 0. \tag{24.10}$$

The obvious boundary conditions for the displacements would be to fix the displacements at all walls. However, for large movement this would distort the mesh unnecessaryly. Therefore the displacements are fixed only in the direction of the surface normal. Thus, if the normal is the main direction, the other two components are free to slide. This approach is unfortunately feasible only for rectangular geometries. Also the displacements of the mesh points at the outer boundaries will be fixed. An outer boundary is assumed to be moving if it is somewhere attached to the moving body, otherwise it is assumed to be at rest.

For the moving rigid body the displacements are given by three translational $U_i$ and three rotational $\Phi_i$ degrees of freedom. Then the displacement at the moving wall yields

$$\vec{u} = \vec{U} + \boldsymbol{R}(\vec{r}-\vec{r}_0), \tag{24.11}$$

where

$$R = \begin{pmatrix} 0 & \Phi_z & -\Phi_y \\ -\Phi_z & 0 & -\Phi_x \\ \Phi_y & \Phi_x & 0 \end{pmatrix} \tag{24.12}$$

It should be noted that if the movement of the frame is pure translational then there may be no need to solve the displacements in all directions since the Laplace has a non-zero solution only if some of the boundary conditions is non-zero.

## 24.4 Implimentation issues

For simple equations, such as the Laplace equation, a large part of the computational effort goes into the assembly of the linear equation. In this special case the same matrix may be assemblied only once and used repetitively to solve the mesh adaption problem varying just the boundary conditions. Unfortunately, the potential equation must every time be reassembled as the coordinates change.

The purpose of the simulation is to get detailed information of the capacitance in respect with the rigid body movement. However, as there are six degrees of freedom making just 10 observations in each direction would result to $10^6$, that is a million, simulations. Therefore it is advisable to make the observations only to some predefined directions. For this purpose the user may give up to six basis $\vec{\eta}_i$ to define these directions. By default $\eta_{ij} = \delta_{ij}$.

For each basis $\vec{\eta}_i$ the user may also give the interval of the amplitude $[a_i, b_i]$ and the number of observation points $N_i$. Then the rigid body coordinates are

$$\vec{U} = \sum_{i=1}^{6} \left( a_i + (n_i - 1)\frac{b_i - a_i}{N_i - 1} \right) \vec{\eta}_i. \tag{24.13}$$

## 24.5 Keywords

```
Constants

    Permittivity Of Vacuum  Real [8.8542e-12]

Solver  solver id

    Equation  String MovingElstatSolver

    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 "MovingElstatSolver" "MovingElstatSolver"
        Following are listed four keywords with default values for output control.

    Moment About i  Real
        The center of coordinate system (i = 1, 2, 3) for the rigid body movement and for computing the
        moments.

    Lumping Basis j(6)  Real
        The basis η_j, j = 1, 2, ..., 6.

    Lumping Points j  Integer
        Number of observations for basis j.

    Lumping Interval j(2)  Real
        The interval of amplitude for basis j.

    Length Scale  Real
        The Aplac export assumes certain unit system. Therefore if the length unit of the mesh is not
        given in metres this value may be given to recsale the results appropriately.

    Calculate Force  Logical [True]
        Whether to calculate and save the force lumped force.

    Calculate Moment  Logical [True]
        Whether to calculate and save the force lumped moments..

    Save Displacements  Logical True
        Whether to save the displacement field in the ElmerPost format.
```

`Filename File`
All the results are saved in the file given by this keyword. Additionally a suffix `.info` is given to file that explains what is being saved. Finally, if Aplac model is created, it is given the suffix `.aplac`.

`Boundary Condition bc id`

`Potential Real`

`Moving Boundary Logical`
If this is true then displacements are fixed using the rigid body movement and potential is given value one.

`Fixed Boundary Logical`
If this is true then displacements are fixed to zero in normal direction and potential is given value zero.

`Periodic BC Potential Logical`
Periodic boundary conditions for the potential is activated by this keyword. Note that this affects only the potential solution. The displacements at the symmetric boundaries are fixed internally to zero.

`Periodic BC Integer`
The periodic counterpart of the current Boundary Conditions.

`Periodic BC Translate(3) Real`
This keyword is required for the older versions of Elmer code to give the translational vector of the periodicity.

# Model 25

# Artificial Compressibility for FSI

**Module name**: ArtificialCompressibility
**Module subroutines**: CompressibilityScale
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document created**: 16.2.2002
**Document edited**: 8.2.2006

## 25.1 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.

## 25.2 Theory

### 25.2.1 Fluid-structure interaction

The theoretical model with some results is thouroughly 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

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \tau + \vec{f} \text{ in } \Omega_s, \tag{25.1}$$

where $\rho$ 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

$$\rho \left( \frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) = \nabla \cdot \sigma + \vec{f} \text{ in } \Omega_f, \tag{25.2}$$

where $\vec{v}$ the fluid velocity and $\sigma$ the stress tensor. For Newtonian incompressible fluids the stress is

$$\sigma = 2\mu\varepsilon(\vec{v}) - pI, \tag{25.3}$$

where $\mu$ 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

$$\nabla \cdot \vec{v} = 0 \text{ in } \Omega_f. \tag{25.4}$$

For later use we, however, recall the general form of the continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \text{ in } \Omega_f. \tag{25.5}$$

The fluid-structure interface, $\Gamma_{fs}$, must meet two different boundary conditions. At the interface the fluid and structure velocity should be the same,

$$\vec{v}(\vec{r}, t) = \dot{\vec{u}}(\vec{r}, t), \quad \vec{r} \in \Gamma_{fs}. \tag{25.6}$$

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

$$\vec{g}_s(\vec{r}, t) = -\vec{g}_f(\vec{r}, t), \quad \vec{r} \in \Gamma_{fs}. \tag{25.7}$$

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 (25.6) and (25.7) 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 (25.7) 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.

### 25.2.2 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

$$\frac{dV}{V} = c\, dP. \tag{25.8}$$

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,

$$\frac{d\rho}{\rho} = c\, dP, \tag{25.9}$$

where $c$ is the artificial compressibility. This is interpreted locally and inserted to the continuity equation (25.5) while neglecting the space derivative of the density, thus

$$c\frac{dp}{dt} + \nabla \cdot \vec{v} = 0, \tag{25.10}$$

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

$$\frac{c}{\Delta t}\left(p^{(m)} - p^{(m-1)}\right) + \nabla \cdot \vec{v}^{(m)} = 0, \tag{25.11}$$

where $m$ is the current iteration step related to fluid-structure coupling. When the iteration converges $p^{(m)} \rightarrow p^{(m-1)}$ 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,

$$\int_{\Omega_f} (\nabla \cdot \vec{v}^{(m)})\varphi_p \, d\Omega + \frac{1}{\Delta t}\int_{\Omega_f} c\left(p^{(m)} - p^{(m-1)}\right)\varphi_p \, d\Omega = 0, \tag{25.12}$$

where $\varphi_p$ 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 (25.8) by applying a pressure change $dP$ to the system,

$$c = \frac{1}{V}\frac{dV}{dP}. \tag{25.13}$$

The change in volume may be calculated by comparing it to initial volume, thus

$$c = \frac{V - V_0}{V_0}\frac{1}{dP}. \tag{25.14}$$

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,

$$c = \frac{\int_{\Gamma_{fs}} (\vec{u} \cdot \vec{n}) \, dA}{\int_{\Omega_f} dV}\frac{\int_{\Gamma_{fs}} dA}{\int_{\Gamma_{fs}} dp \, dA}. \tag{25.15}$$

This way $c$ has a constant value over the domain.

### 25.2.3 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,

$$c(\vec{r}) = c_0(\vec{r}) \underbrace{\frac{\int_{\Gamma_{fs}} (\vec{u} \cdot \vec{n}) \, dA}{\int_{\Omega_f} c_0(\vec{r}) dV}\frac{\int_{\Gamma_{fs}} dA}{\int_{\Gamma_{fs}} dp \, dA}}_{\text{scaling factor}}. \tag{25.16}$$

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.

### 25.2.4 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 usefull for geometries where the elastic response changes significantly. The equation is now similar to (25.14),

$$c = \frac{V^e - V_0^e}{V_0^e} \frac{1}{dP}, \tag{25.17}$$

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 $c$.

## 25.3 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 meterial 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 dafault 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.

### 25.3.1 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.

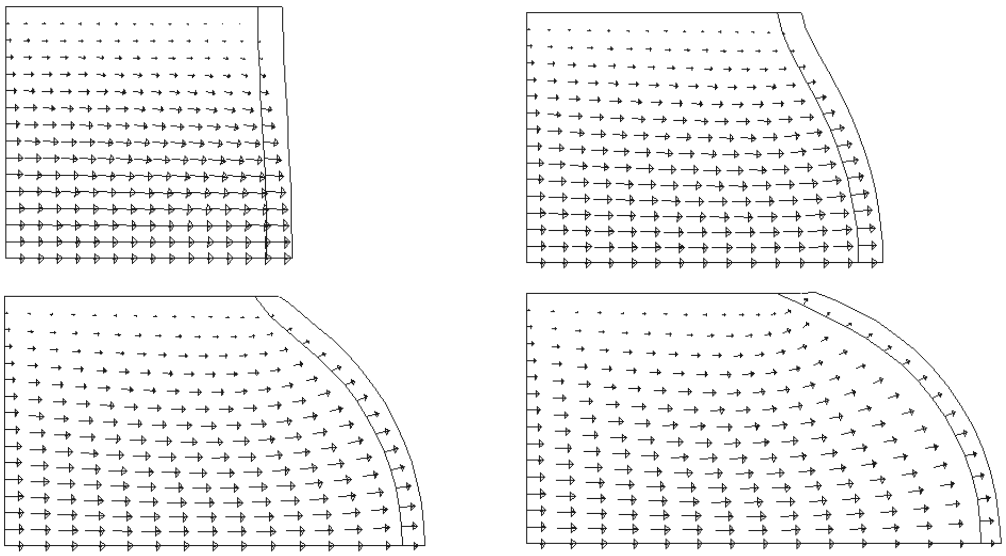

Figure 25.1: Snapshots of an elastic square being gradually filled by incompressible fluid.

# 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.

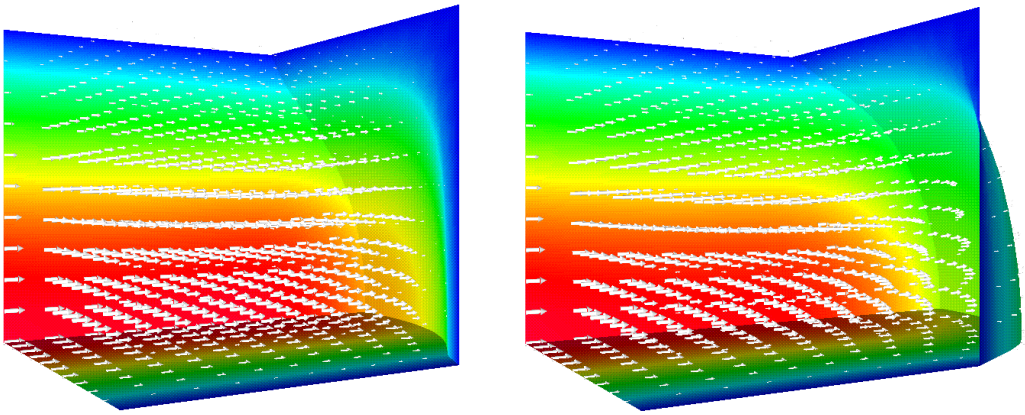

Figure 25.2: Snapshots of an elastic cube being gradually filled by incompressible fluid.

[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 26

# Rotational form of the incompressible Navier–Stokes equations

**Module name**: Stokes
**Module subroutines**: StokesSolver
**Module authors**: Mika Malinen
**Document authors**: Mika Malinen
**Document edited**: June 26th 2009

## 26.1 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.

## 26.2 Field equations

Using the vector identity

$$(\vec{u}\cdot\nabla)\vec{u} = (\nabla\times\vec{u})\times\vec{u} + \frac{1}{2}\nabla(\vec{u}\cdot\vec{u}), \tag{26.1}$$

the Navier–Stokes system for incompressible Newtonian fluid may be written as

$$\begin{aligned} \rho\left(\frac{\partial\vec{u}}{\partial t} + (\nabla\times\vec{u})\times\vec{u}\right) - 2\mu\nabla\cdot\bar{\bar{\varepsilon}}(\vec{u}) + \nabla P &= \vec{b}, \\ \nabla\cdot\vec{u} &= 0, \end{aligned} \tag{26.2}$$

where $\bar{\bar{\varepsilon}}$ is the stretching tensor (2.4) and

$$P = p + \frac{1}{2}\rho\vec{u}\cdot\vec{u} \tag{26.3}$$

is the total (Bernoulli) pressure. The stress $\bar{\bar{\sigma}}$, which may be of interest especially near boundaries, can now be expressed as

$$\bar{\bar{\sigma}} = (-P + \frac{1}{2}\rho\vec{u}\cdot\vec{u})\bar{\bar{I}} + 2\mu\bar{\bar{\varepsilon}}(\vec{u}). \tag{26.4}$$

The system (26.2) 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 (26.2) reduces to the (generalized) Stokes equations. The pressure variable then reduces to the standard pressure, i.e. $P = p$.

## 26.3 Boundary conditions

Either the normal velocity $\vec{u}\cdot\vec{n}$, with $\vec{n}$ the outward unit normal vector, or the normal surface force $\overline{\overline{\sigma}}\vec{n}\cdot\vec{n}$ 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 $\overline{\overline{\sigma}}\vec{n}\cdot\vec{t}$.

A rather common way to define an outflow boundary condition for the Navier–Stokes equations is to impose the normal surface force condition

$$\overline{\overline{\sigma}}\vec{n}\cdot\vec{n} = 0, \tag{26.5}$$

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 (26.2) can be written as

$$-P\vec{n} + 2\mu\overline{\overline{\varepsilon}}\vec{n} = \overline{\overline{\sigma}}\vec{n} - \frac{1}{2}\rho(\vec{u}\cdot\vec{u})\vec{n} = \vec{0}.$$

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

$$\overline{\overline{\sigma}}\vec{n}\cdot\vec{n} = \frac{1}{2}\rho\vec{u}\cdot\vec{u},$$

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. $\overline{\overline{\sigma}}\vec{n}\cdot\vec{t} = 0$.

## 26.4 Linearization

The linearization of the equation of motion in (26.2) 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 (26.5) is linearized as

$$-P + \frac{1}{2}\rho(2\vec{a}\cdot\vec{u} - \vec{a}\cdot\vec{a}) + 2\mu\overline{\overline{\varepsilon}}(\vec{u})\vec{n}\cdot\vec{n} = 0.$$

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

$$-P + \frac{1}{2}\rho\vec{a}\cdot\vec{u} + 2\mu\overline{\overline{\varepsilon}}(\vec{u})\vec{n}\cdot\vec{n} = 0.$$

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.

## 26.5 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 $Q_2$-$Q_1$ approximation for rectangular and brick elements.* 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 associated with mid-edge nodes.

- *A hierarchic version of $P_2$-$P_1$ approximation for triangular and tetrahedral elements.* Analogously to the previous case, 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 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.

## 26.6 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 (26.2) leads to solving linear systems of the form

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} U \\ \Pi \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}, \tag{26.6}$$

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 (26.6). Given a

previous iterate $(U_k, \Pi_k)$, the preconditioning is performed via solving approximately systems of the form

$$\begin{bmatrix} A & 0 & 0 \\ B & M & 0 \\ B & H & S \end{bmatrix} \begin{bmatrix} U_{k+1} - U_k \\ \psi_{k+1} \\ \Pi_{k+1} - \Pi_k \end{bmatrix} = \begin{bmatrix} F - AU_k - B^T \Pi_k \\ -BU_k \\ -BU_k \end{bmatrix}. \tag{26.7}$$

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 (26.6) 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.

## 26.7 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 (26.7), the time discretization must be done using BDF(1) and viscosity should be constant.

## 26.8 Keywords

`Material material-id`

`Density Real`
: This keyword is used to define the density $\rho$.

`Viscosity Real`
: This keyword is used to define the viscosity $\mu$.

`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 $\lambda$ is the value of this keyword.

`Block Preconditioning` `Logical`
: If the block preconditioning via (26.7) 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 (26.6).

`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 (26.6). 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 (26.5) 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 (26.7), 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.

# Model 27

# Linear Constraints

**Module name**: included in solver (SolverUtils)
**Module subroutines**: SolveWithLinearRestriction
**Module authors**: Mika Juntunen
**Document authors**: Mika Juntunen
**Document edited**: August 5th 2003

## 27.1 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]).

## 27.2 Theory

The problem at hand is

$$\min_x x^T A x - x^T f \tag{27.1}$$

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.

$$\min_x x^T A x - x^T f + \lambda^T (Bx - g) \tag{27.2}$$

If $A$ is symmetric, then simple variational approach leads to solving $x$ out of system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{27.3}$$

Symmetry of $A$ is not always needed, but then more powerful methods have to be used to get to the above system.

## 27.3 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 therefor 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.

## 27.4 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 28

# 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
**Document created**: April 28th 2006
**Document edited**: April 28th 2006

## 28.1 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-strucrure 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.

## 28.2 Theory

The conservation equations for a flow in an elastic artery in one dimension may be expressed as

$$\begin{cases} \frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \\ \\ \frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}(\frac{Q^2}{A}) + \frac{A}{\rho}\frac{\partial p}{\partial x} = 0, \end{cases} \tag{28.1}$$

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 (28.1) 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

$$\frac{E\,h^3}{12(1-\nu^2)}\,{d_R}^{(4)} + \frac{E\,h}{(1-\nu^2)}\,\frac{1}{{R_m}^2}\,d_R = p, \tag{28.2}$$

where $R_m$ is the radius to the midplane of the wall, $E$, $\nu$ and $d_R$ are the Youngs modulus, the Poisson ratio and the radial displacement of wall, respectively. Assuming that the first term on the left side in the equation (28.2) is much smaller than the second term, we can give the pressure-area relation in the form

$$p = p_{ext} + \beta(\sqrt{A} - \sqrt{A_0}), \qquad \beta = \frac{\sqrt{\pi}hE}{(1-\nu^2)A_0}. \tag{28.3}$$

The pressure is scaled to be equal to external pressure $p_{ext}$ with corresponding reference artery cross sections area $A_0$.

The equations (28.1) and (28.3) 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 $\rho_f$ 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

$$\frac{\partial W_i}{\partial t} + \lambda_i \frac{\partial W_i}{\partial x} = 0, \tag{28.4}$$

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 (28.4), 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 bondary 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 $\sigma$ 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.

## 28.3 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 thichness of the artery

`Artery Poisson Ratio  Real`
Poisson ration 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 codition 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
```

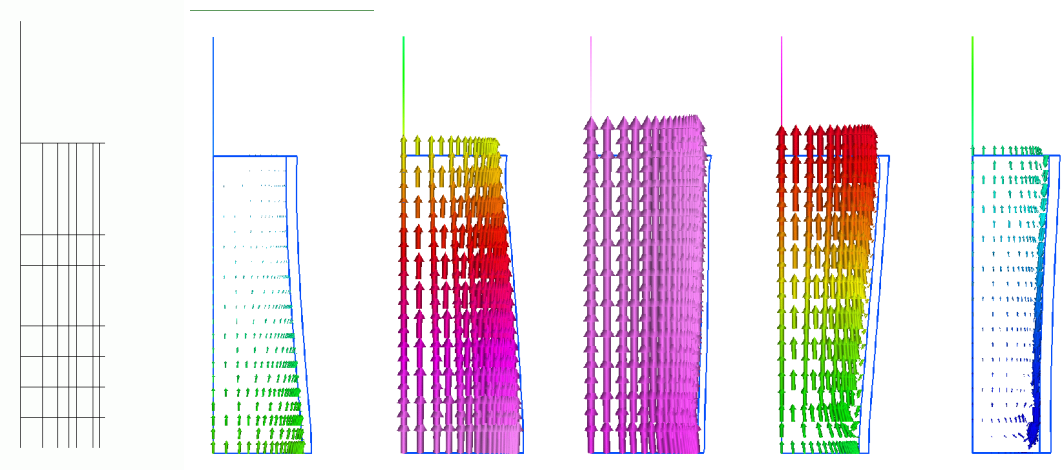

Figure 28.1: An example of the model results: pressure pulse prapagation in a 2D axisymmetric model combined with an 1D model.

# 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 29

# Streamlines

**Module name**: StreamSolver
**Module subroutines**: StreamSolver
**Module authors**: Mika Juntunen
**Document authors**: Mika Juntunen
**Document edited**: July 30th 2003

## 29.1 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 line lie all those flow elements that at some earlier instant passed through 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.

## 29.2 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 $\psi$ is defined

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{29.1}$$

Here the geometry is $(x, y)$ and the corresponding flow is $\vec{u} = (u, v)$. Let $\Omega$ be the domain of the flow and $\vec{v}$ a test function for the flow. Definition (29.1) leads to finite element approximation

$$\int_\Omega \nabla \psi \cdot \vec{v} \, \mathrm{d}\Omega = \int_\Omega \vec{u}^\perp \cdot \vec{v} \, \mathrm{d}\Omega \tag{29.2}$$

In axis symmetric geometry the mass conservation calculated in a diffenrent way. This leads to following definition for stream function.

$$u = \frac{1}{r}\frac{\partial \psi}{\partial r}, \quad v = -\frac{1}{r}\frac{\partial \psi}{\partial z} \tag{29.3}$$

where the cylinderical coordinates are $(z, r, \phi)$, 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.

$$\int_\Omega \nabla\psi \cdot \vec{v}\,\mathrm{d}\Omega = \int_\Omega \vec{u}^\perp \cdot \vec{v} r\,\mathrm{d}\Omega \tag{29.4}$$

Here the $\phi$ component of the flow is excluded.

From definitions (29.1) and (29.3) it is apparent that stream function is constant along the streamlines. So drawing the contours of stream function gives the streamlines.

## 29.3 Limitations

Some limitations of the current implementation:

- The flow field is asumed 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 symetric 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.

## 29.4 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 first set to zero. Non-positive values are set to 1 and too large values are set to largest possible i.e. 'the last node'. Default is 1.

`Stream Function Shifting` `Logical`
: Shift the smallest value to zero. Default is `True`.

`Stream Function Scaling` `Logical`
: Scale largest absolut value to 1. Default is `False`.

`Stokes Stream Function` `Logical`
: This keyword forces the stream function type regardles of the coordinate system. If the coordinate system is axis symmetric, then the default is `True`, else the default is `False`.

# Model 30

# Flux Computation

**Module name**: FluxSolver
**Module subroutines**: FluxSolver
**Module authors**: Juha Ruokolainen, Peter Råback
**Document authors**: Peter Råback
**Document edited**: 21.6.2007

## 30.1 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.

## 30.2 Theory

The flux resulting from a potential field is assumed to be proportional to the gradient of the field, $\phi$. The proportinality factor is here called conductivity, $c$. The flux may therefore be expressed as

$$q = -c\nabla\phi. \tag{30.1}$$

For heat equation the potential would this be temperature and the conductivity would be the heat conductivity.

## 30.3 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 consistant method for flux computation is, however, using the finite element method to solve the equation (32.1). The Galerkin method creates a diagonally dominated matrix equation that may be computed easily with iterative methods even with poor preconditioners.

The flux computation may be done component-wise so that for 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 a internal boundary over which the flux is not continous the calculated value does not make sense.

## 30.4 Keywords

`Solver solver id`

`Equation String Flux Solver`

`Procedure File "FluxSolver" "FluxSolver"`

`Variable String "-nooutput tempvar"`
The variable is usually only used to allocate the corresponding matrix. Therefore output is not required unless the solver is only used to compute one component.

`Flux Result Variable String f1`
This string gives the name of the variable that is known to be at disposal for saving the results. This variable is allocated with the following keyword.

`Exported Variable 1 String "fl[Heat Flux:3]"`
This command is used to allocate space for the result and at the same time the components may be renamed to be later identified as a vector in ElmerPost. If only one component of the flux is computed this keyword is obsolite.

`Flux 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 potential variable used to compute the gradient. By default the coefficient is `Heat Conductivity`. If an non-existing material parameter is given the coefficient will be assumed to be one, i.e. $c = 1$. This way the solver may be used to compute the gradient only.

`Flux Component Integer`
If only one component of the flux need to be computed it may be given by this keyword. If the keyword is not specified the solver computes all the components of the flux.

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 "BiCGStab"
Linear System Preconditioning None
Linear System Max Iterations 500
Linear System Convergence Tolerance 1.0e-10
```

# Model 31

# Vorticity Computation

**Module name**: VorticitySolver
**Module subroutines**: VorticitySolver
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document edited**: 14.2.2008

## 31.1 Introduction

This module is used to calculate the vorticity of vector fields. Vorticity may be of interest mainly in the postprocessing of flow fields or electromagnetic fields.

## 31.2 Theory

The vorticity $\vec{w}$ of a vector field $\vec{v}$ is obtained simply from the curl of the field,

$$\vec{w} = \nabla \times \vec{v}. \tag{31.1}$$

Component-wise the equations for the vorticity read

$$w_x = \frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \tag{31.2}$$

$$w_y = \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \tag{31.3}$$

$$w_z = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}. \tag{31.4}$$

Thus, all three components exist only in 3D while in 2D and axisymmetric cases only the $z$-component is present

The most consist ant method for computing the vorticity in conjunction with the finite elements is to solve the equations (31.4) using the Galerkin method. The resulting matrix is diagonally dominated and may be computed 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. If only one component is desired in the 3D computations then the $z$-component is computed.

## 31.3 Keywords

```
Solver  solver id
```

```
Equation String Vorticity Solver
Procedure File "VorticitySolver" "VorticitySolver"
```

`Vorticity 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 following keywords are not usually needed as they are set by the initialization procedure of the solver.

`Variable String "-nooutput tempvar"`
: The variable is usually only used to allocate the corresponding matrix. Therefore output is not required unless the solver is only used to compute one component.

`Vorticity Result Variable String f1`
: This string gives the name of the variable that is known to be at disposal for saving the results. This variable is allocated with the following keyword.

`Exported Variable 1 String "fl[Vorticity:3]"`
: This command is used to allocate space for the result and at the same time the components may be renamed to be later identified as a vector in ElmerPost. If only one component of the flux is computed this keyword is obsolete.

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 32

# Scalar Potential Resulting to a Given Flux

**Module name**: ScalarPotentialSolver
**Module subroutines**: ScalarPotentialSolver
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document edited**: 15.2.2008

## 32.1 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 an 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.

## 32.2 Theory

The flux resulting from a potential field is assumed to be proportional to the gradient of the field, $\phi$. The proportionality factor is here called conductivity, $c$. The flux may therefore be expressed as

$$q = -c\nabla\phi. \tag{32.1}$$

For heat equation the potential would this be temperature and the conductivity would be the heat conductivity.

This solver solves the equation in the reverse form, i.e. given the flux solver for the potential. In the weak formulation this is solved so that the test function is the gradient of the shape function. 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.

## 32.3 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 an non-existing material parameter is given the coefficient will be assumed to be one, 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 zero 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 33

# Fluidic Force

**Module name**: FluidicForce
**Module subroutines**: ForceCompute
**Module authors**: Juha Ruokolainen, Antti Pursula
**Document authors**: Antti Pursula
**Document edited**: Feb 28th 2005

## 33.1 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 than 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 present when calling the procedure. Also the torque with respect to a given point can be requested.

## 33.2 Theory

The force due to fluid is calculated as a product of the stress tensor and normal vector integrated over the surface

$$\vec{F} = \int_S \overline{\overline{\sigma}} \cdot \vec{n}\, dS. \tag{33.1}$$

The stress tensor is

$$\overline{\overline{\sigma}} = 2\mu\overline{\overline{\varepsilon}} - \frac{2}{3}\mu(\nabla \cdot \vec{u})\overline{\overline{I}} - p\overline{\overline{I}}, \tag{33.2}$$

where $\mu$ 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.

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{33.3}$$

The torque about a point $\vec{a}$ is given by

$$\vec{\tau} = (\vec{r} - \vec{a}) \times \vec{F}(\vec{r}), \tag{33.4}$$

where $\vec{r}$ is the position vector.

## 33.3 Additional output

There is also a feature for saving the tangential component of the surface force i.e. the shear stress elementwise 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.

## 33.4 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 apllies 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 1,2,3  Real`
Coordinates for the point on which the torque is returned.

# Model 34

# Electrostatic force

**Module name**: ElectricForce
**Module subroutines**: StatElecForce
**Module authors**: Antti Pursula
**Document authors**: Antti Pursula
**Document edited**: February 7th 2003

## 34.1 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 24 of this Manual).

## 34.2 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

$$\vec{F} = \int_S \overline{\overline{T}} \cdot \, d\vec{S}. \tag{34.1}$$

The components of the Maxwell stress tensor for linear medium are

$$T_{ij} = -D_i E_j + \frac{1}{2}\delta_{ij}\vec{D} \cdot \vec{E}, \tag{34.2}$$

where electric field $\vec{E}$ and electric displacement field $\vec{D}$ are obtained from the electric potential $\Phi$

$$\vec{E} = -\nabla\Phi, \tag{34.3}$$

and

$$\vec{D} = -\varepsilon_0\varepsilon_r\nabla\Phi, \tag{34.4}$$

where $\varepsilon_0$ is the permittivity of vacuum and $\varepsilon_r$ is the relative permittivity of the material, which can be a tensor.

## 34.3 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 35

# Save Data

**Module name**: SaveData
**Module subroutines**: SaveScalars, SaveLine, SaveMaterials, SaveBoundaryValues
**Module authors**: Peter Råback, Ville Savolainen, Thomas Zwinger
**Document authors**: Peter Råback
**Document created**: Oct 3rd 2002
**Document updated**: January 8th 2008

## 35.1 Introduction

This module does not include any physical models per se. The module includes utilities for computing derived quantities and saving scalars as well as lines in matrix format. Scalars are saved with the subroutine `SaveScalars` and lines with the subroutine `SaveLine`, correspondingly. 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. In addition there is a utility called `SaveMaterials` that 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.

## 35.2 Theory

The theoretical problem in saving data comes from the fact that often the data should be saved in points or 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. 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

$$\vec{r}_1 + a(\vec{r}_2 - \vec{r}_1) = \vec{e}_1 + b(\vec{e}_2 - \vec{e}_1) + c(\vec{e}_3 - \vec{e}_1) \tag{35.1}$$

has a solution for which $0 \leq a, b, c \leq 1$. This results to a matrix equation

$$\begin{pmatrix} r_{2x} - r_{1x} & e_{1x} - e_{2x} & e_{1x} - e_{3x} \\ r_{2y} - r_{1y} & e_{1y} - e_{2y} & e_{1y} - e_{3y} \\ r_{2z} - r_{1z} & e_{1z} - e_{2z} & e_{1z} - e_{3z} \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} e_{1x} - r_{1x} \\ e_{1y} - r_{1y} \\ e_{1z} - r_{1z} \end{pmatrix} \tag{35.2}$$

which may be easily solved with standard methods linear algebra. Because the face element is a triangle there is an additional condition that $b + c \leq 1$.

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,

$$< f >= \frac{\sum_{i=1}^{n} f_i}{n}. \tag{35.3}$$

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

$$< f >= \frac{\int f \, d\Omega}{\int d\Omega}. \tag{35.4}$$

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

$$E_{diff} = \frac{1}{2} \int_{\Omega} \nabla f \cdot c \nabla f \, d\Omega, \tag{35.5}$$

where $c$ may a tensor or a scalar. Kinetic energy related to convection is of type

$$E_{con} = \frac{1}{2} \int_{\Omega} c\vec{v} \cdot \vec{v} \, d\Omega, \tag{35.6}$$

and potential type of energy

$$E_{pot} = \int_{\Omega} cf \, d\Omega. \tag{35.7}$$

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

$$F_{con} = \int_{\Gamma} c\vec{v} \cdot \vec{n} \, d\Gamma, \tag{35.8}$$

where $\vec{n}$ is the surface normal. Diffusive fluxes may be computed from

$$F_{diff} = \int_{\Gamma} c \nabla f \cdot \vec{n} \, d\Gamma, \tag{35.9}$$

where $c$ may also be a tensor.

## 35.3 Keywords

### Keywords of solver SaveScalars

```
Solver  solver id

    Procedure  File "SaveData" "SaveScalars"

    Filename  String
        Name of the file where results are to be saved, the default is scalars.dat.

    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.

`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`.

`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 operators working on the set of numbers, `max, min, max abs, min abs, mean, variance` and `deviation`. There are also a few operators that use statistics over the volume, `int mean` and `int variance`. 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`.

  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`.

  Three different energy type of quantities may be computed by domain integral operators `diffusive energy, convective energy, potential energy`. Finally, 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.

  There may be up to 99 different operators. 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.

`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 trough 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.

`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`.

## Keywords of subroutine SaveLine

`Solver` `solver id`

`Procedure` `File "SaveData" "SaveLine"`

`Filename` `String`
: Name of the file where results are to be saved, the default is `sides.dat`.

`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.

`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 dependece, 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 $\kappa$ 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 buondaries 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.

`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.

## 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.

# Model 36

# Result Output for Other Postprocessors

**Module name**: ResultOutputSolve
**Module subroutines**: ResultOutputSolver
**Module authors**: Erik Edelmann, Mikko Lyly, Peter Råback
**Document authors**: Peter Råback
**Document created**: 11.12.2006
**Document edited**: 3.8.2007

## 36.1 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 aquinted with some other visuzalition software. Currently supported formats include GiD, Gmsh, VTK legacy, XML coded VTK file bearing the suffix VTU and Open DX.

## 36.2 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.

The following keywords related only to the GiD, Vtu and Gsmh formats. In the other formats all available degrees of freedom are saved.

`Scalar Field i String`
: The scalar fields to be saved, for example `Pressure`. Note that the fields must be numbered continously 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.

# Model 37

# Reload Existing Simulation Results

**Module name**: ReloadData
**Module subroutines**: ReloadSolution
**Module authors**: Antti Pursula
**Document authors**: Antti Pursula
**Document created**: August 9th 2007

## 37.1 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 variales 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 advicable to use `Exec Solver = Before Timestep` for this module.

## 37.2 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 $i$th 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 advicable to switch this feature off. Defining `False` will slow down reading especially from large ASCII files.

# Model 38

# Runtime Control of the Input Data

**Module name**: ReloadInput
**Module subroutines**: ReloadInput
**Module authors**: Juha Ruokolainen
**Document authors**: Peter Råback
**Document created**: Februrary 5th 2003

## 38.1 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 on additional solver that is called similarly as any other solver during the solution process.

The most likealy 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 critaria 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.

## 38.2 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.

## 38.3 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 39

# Filtering Time-Series Data

**Module name**: FilterTimeSeries
**Module subroutines**: FilterTimeSeries
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document created**: 13.2.2008
**Document updated**: 13.2.2008

## 39.1 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).

## 39.2 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,

$$< f >_T = \frac{1}{T} \int_0^T f(t)\, dt. \tag{39.1}$$

Its discrete counterpart assuming piecewise constant integration is

$$< f >_n = \frac{1}{T_n} \sum_i^n f_i\, dt_i, \tag{39.2}$$

where $T_n = \sum dt_i$. Now this may be presented inductively as

$$< f >_n = \frac{T_{n-1} < f >_{n-1} + f_n dt_n}{T_{n-1} + dt_n} \tag{39.3}$$

$$T_n = T_{n-1} + dt_n. \tag{39.4}$$

### Weighted mean

It's also possible to take a weighted mean with a user defined function $g(t)$ depending on time only. Then similarly,

$$< fg >_n = \frac{T_{n-1} < fg >_{n-1} + f_n g_n dt_n}{T_{n-1} + dt_n}. \tag{39.5}$$

### 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

$$f \approx \sum_{k=1}^{m_s} s_k \sin(2\pi kwt) + \sum_{k=1}^{m_c} c_k \cos(2\pi kwt), \tag{39.6}$$

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

$$T_{n-1} := T_{n-1} \exp(-dt_n/\tau), \tag{39.7}$$

where $\tau$ 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 $dt/\tau$.

### 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,

$$\sigma^2 = < (f - < f >)^2 > = < f^2 > - < f >^2 . \tag{39.8}$$

## 39.3 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 $\tau$ in computing continuous means.

`Decay Timestep i` `Real`
: The number of timestep needed to perform averaging. If the timestep given is $N$ then the dacay 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 40

# Rigid Mesh Transformation

**Module name**: RigidMeshMapper
**Module subroutines**: RigidMeshMapper
**Module authors**: Peter Råback
**Document authors**: Peter Råback
**Document edited**: 24.9.2009

## 40.1 Introduction

Sometimes there is a need to transform meshes without the need of generation a new mesh. The most simple case is that of rigid mesh movement were some bodies move with prescribed rotations, translations or scalings. Typically this could be a preprocessing step in a parametric study in some problem. Then this solver may be used to perform the mesh transformation.

In addition to applying rigid transformations to bodies this solver includes also a relaxation parameter which may be used to define which fraction of the mesh 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 in respect to one-another if there is a mesh between the bodies. Then the `MeshUpdate` solver should be used instead.

## 40.2 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

$$\vec{x}_1 = \mathcal{S}(\mathcal{R}(\vec{x}_0 - \vec{o}) + \vec{t}) + \vec{o}, \tag{40.1}$$

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.

## 40.3 Keywords

`Solver  solver id`

`Equation  String [RigidMeshMapper]`
The name of the equation.

`Procedure  File "RigidMeshMapper" "RigidMeshMapper"`
The name of the procedure.

`Use Original Coordinates` `Logical`
: This keyword applied only to cases where the solver is called repeatedly. With this keyword being true the mesh transformation is always applied to the original coordinates. Otherwise the mesh transformation is performed recursively.

`Body Force` `bf id`
: The mesh transformations are defined in this section.

`Mesh Translate` `Real` $[t_x t_y t_z]$
: The translational vector.

`Mesh Rotate` `Real` $[\alpha_x \alpha_y \alpha_z]$
: The rotation around main coordinate directions.

`Mesh Scale` `Real` $[s_x s_y s_z]$
: The scaling around of main directions.

`Mesh Origin` `Real` $[o_x o_y o_z]$
: The origin used in rotation and scaling.

`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.

# Model 41

# Internal cost function optimization

**Module name**: FindOptimum
**Module subroutines**: FindOptimum
**Module authors**: Peter Råback
**Module status**: Alpha
**Document authors**: Peter Råback
**Document created**: 18.3.2009
**Document edited**: 18.3.2009

## 41.1 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 an 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.

## 41.2 Theory

The optimization routines must be sligthly 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 worth mentioning.

## 41.3 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` 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`
Maxium 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 Offset 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 Opposite 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 keyords 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.

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
```

# Index