Elmer Tutorials

CSC – IT Center for Science

February 1, 2013

Elmer Tutorials

About this document

The Elmer Tutorials is part of the documentation of Elmer finite element software. Elmer Tutorials gives examples on the use of Elmer in different field of continuum physics. Also coupled problems are included.

The tutorials starts with problems which require the use of ElmerGUI, the graphical user interface. However, also problems which assume only the use of an text editor are given. There are also obsolite problems that utilize the old graphical user interface, ElmerFront. These are provided only for backward compability but should rather not be studied by new users.

The present manual corresponds to Elmer software version 6.2. Latest documentations and program versions of Elmer are available (or links are provided) at `http://www.csc.fi/elmer`.

Copyright information

Contents

Instructions for the GUI tutorials

Here are some instructions for following the GUI tutorials:

- All the needed input files should be available among the `ElmerGUI/samples` directory that came with the installation. Look under a subdirectory named after the suffix of the sample file.
- The instructions written in `verbatim` refer to operations with the GUI. Intendation means step in the menu hierarchy. The instructions should not be mixed with those in the command file.
- The menu structure for the default set of equations is located in directory `edf`, there are a few additional ones in directory `edf-extra`. These may be copied to the directory `edf` permanently, or be appended to the menus while running the ElmerGUI.
- After having once defined the case you may go to the working directory and launch ElmerSolver from command-line. There you may edit the .sif file to alter the parameters.
- Manual alteration to the sif file will not be communicated to the ElmerGUI project. All editions will be overrun by the GUI when saving the project.
- The cases have been run a number of times but errors are still possible. Reporting them to elmeradm@csc.fi, for example, is greatly appreciated.

Part I

ElmerGUI Problems

Tutorial 1

Heat equation – Temperature field of a solid object

Directory: TemperatureGenericGUI
Solvers: HeatSolve
Tools: ElmerGUI,netgen,OpenCascade
Dimensions: 3D, Steady-state

Problem description

This tutorial tried to demonstrate how to solve the heat equation for a generic 3D object. The solid object (see figure 1.1) is heated internally by a heat source. At some part of the boundary the temperature is fixed. Mathemetically the problem is described by the Poisson equation

$$\begin{cases} -\kappa \Delta T &= \rho f \quad \text{in} \quad \Omega \\ T &= 0 \quad \text{on} \quad \Gamma \end{cases} \tag{1.1}$$

where κ is the heat conductivity, T is the temperature and f is the heat source. It is assumed that density and heat conductivity are constants.

To determine the problem we assume that the part of the boundary is fixed at $T_0 = 293$ K, the internal heat generation is, $h = 0.01$ W/kg, and use the material properties of aluminium.

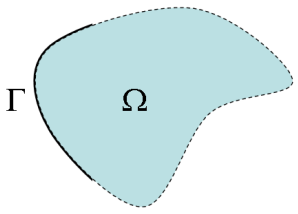

Figure 1.1: Generic object being heated

Solution procedure

Start `ElmerGUI` from command line or by clicking the icon in your desktop. Here we describe the essential steps in the ElmerGUI by writing out the clicking procedure. Tabulation generally means that the selections are done within the window chosen at the higher level.

The geometry is given in step format in file `pump_carter_sup.stp` in the `samples/step` directory of ElmerGUI, This file is kindly provided at the AIM@SHAPE Shape Repository by INRIA. The heat equation is ideally suited for the finite element method and the solution may be found even at meshes that for some other problems would not be feasible. Therefore you may easily experiment solving the same problem with different meshes. If you lack OpenCascade you might try to solve a similar problem with the `grd` files `angle3d.grd, angles3d.grd, bench.grd`, or `cooler.grd`, for example.

The CAD geometry defined by the step file is transformed on-the-fly by OpenCascade library into a stl file for which nglib creates tetrahedral volume discretization. You may also use the tetlib library (tetgen) if you have installed it as a plug-in.

Load the input file:

```
File
  Open -> pump_carter_sup.stp
```

The meshing will take a minute or two. You should obtain your mesh and may check in the number of element in the `Model summary`. With netgen the default setting generates 8371 nodes and 36820 tetrahedral elements. Visual inspection rewiels that the mesh is not quite satisfactory in geometric accuracy. We choose to modify the mesh by altering the settings in the following way.

```
View -> Cad model...
  Model -> Preferences...
    Restrict mesh size on surfaces by STL density = on
    Apply
Mesh -> Remesh
```

The meshing a take a minute or two. The modified mesh should include 16159 nodes and 65689 tetrahderal elements and be more appieling to the eye. In order to affect the mesh density study the command-line options of the netgen manual. Here we continue with the default mesh.

We want to set the temperature at the inside of the holes and in that aim you may join the three boundaries (see figure 1.2). For that aim we may choose the six pieces that constitute the boundaries as shown in the picture by pressing the `Ctrl`-key down.

```
Mesh
  Unify Surface
```

After we have the mesh we start to go through the Model menu from the top to bottom. In the `Setup` we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 3-dimensional cartesian coordinates and in steady-state. Only one steady-state iteration is needed as the case is linear.

```
Model
  Setup
    Simulation Type = Steady state
    Steady state max. iter = 1
```

Choose `Apply` to close the window.

In the equation section we choose the relevant equations and parameters related to their solution. In this case we'll have one set only one equation – the heat equation.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and therefore its easier to assign the Equation and Material to it directly, whereas the active boundary is chosen graphically.

For the linear system solvers we are happy to use the defaults. One may however, try out different preconditioners (ILU1,...), for example.

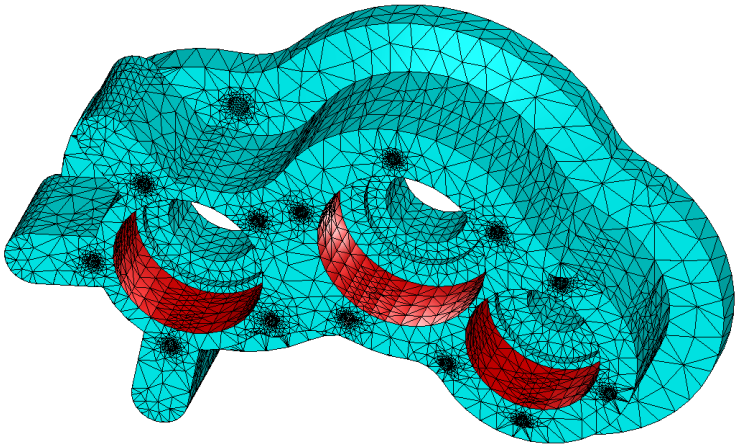

Figure 1.2: The computational mesh showing the three joined boundaries

```
Model
  Equation
    Add
      Name = Heat Equation
      Apply to bodies = Body 1
      Heat Equation
        Active = on
      Add
      OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the mass. Other properties assume a physical law, such heat conductivity. We choose Aluminium from the Material library which automatically sets for the needed material properties.

```
Model
  Material
    Add
      Material library
        Aluminium
      Apply to bodies = Body 1
      Add
      OK
```

A Body Force represents the right-hand-side of a equation that in this case represents the heat source.

```
Model
  Body Force
    Add
      Name = Heating
      Heat Source = 0.01
      Apply to bodies = Body 1
      Add
      OK
```

No initial conditions are required in steady state case.

In this case we have only one boundary and set it to room temperature. First we create the boundary condition

```
Model
  BoundaryCondition
    Add
      Heat Equation
        Temperature = 293.0
      Name = RoomTemp
      Add
      OK
```

Then we set the boundary properties

```
Model
  Set boundary properties
```

Choose the defined group of three boundaries by clicking with the mouse and apply the condition for this boundary.

```
Boundary condition
  RoomTemp
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. In saving the project all the necessary files for restarting the case will be saved to the destination directory.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. As the case is linear only one iteration was required for the solution and the second one just is needed to check the convergence. The norm of the solution should be around 432.4 K (with the default tetgen mesh 389.8 K, respectively).

Note: if you face problems in the solution phase and need to edit the setting, always remember to regenerate the sif file and save the project before execution.

Postprocessing

To view the results we may use the ElmerPost postprocessor or start the the internal VTK widget as is done here,

```
Run
  Postprocessor (VTK)
```

The default configuration shows just the object. To color the surface with the temperature choose

```
Surfaces
  Surface: Temperature
  Apply
```

The maximum temperature should be about 586.5 K. You may turn on opasity in order to see through the object, 10-20% is a good value. This way you'll able to see some isosurfaces that you might want to define. Some examples of the visualizations may be seen in figure 1.3.

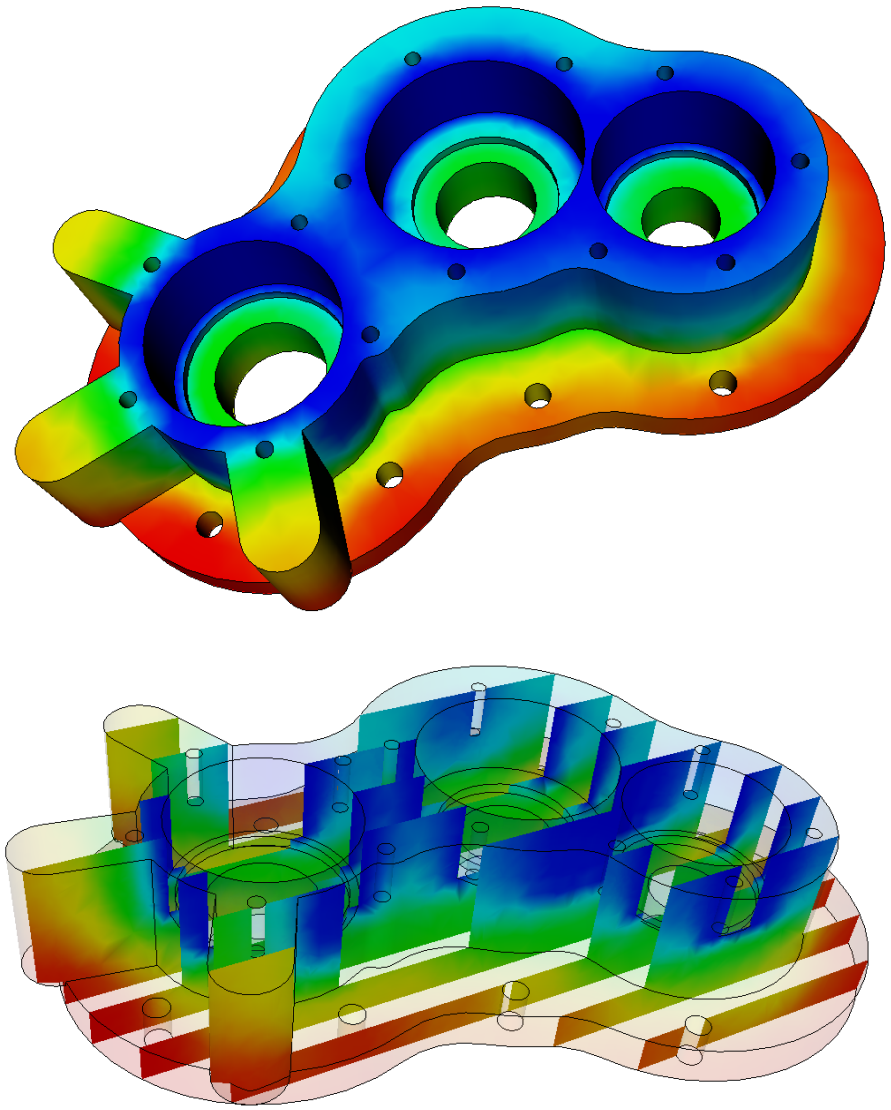

Figure 1.3: The temperature distribution of the solid object domain as visualized using the VTK-based postprocessor

Tutorial 2

Linear elasticity equation – Loaded elastic beam

Directory: ElasticBeam3D
Solvers: StressSolve
Tools: ElmerGUI
Dimensions: 3D, Steady-state

Case definition

Assume a homogenous, elastic beam being rigidly supported on one end. On the other end it is subjected with a load of 2000 N resulting from an attached object in the gravitational field. The gravity affects also the beam itself. The length of the beam is 1 m and the thickness is 0.05 m, and the width 0.1 m. Material properties of the beam are those of dry pine timber: Poisson ratio 0.37, Young's modulus $10 \cdot 10^9 \mathrm{N/m}^2$, and density 550 kg/m^3. The problem is to solve the displacement and stress field of the beam. Here the `StressSolve` routine based on the linear theory of elasticity is applied.

Solution procedure

The mesh is given in ElmerGrid format in file `beam3d.grd`, load this file.

```
File
  Open -> beam3d.grd
```

You should obtain your mesh and may check that it consists of 6073 nodes and of 1200 quadratic hexahedral elements. The second order elements give improved accuracy compared to the first order elements as they avoid the phenomenom known as locking.

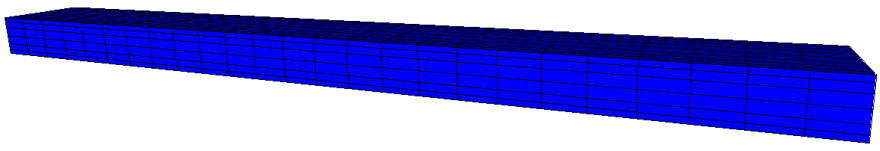

Figure 2.1: The mesh used in the computations

After we have the mesh we start to go through the Model menu from the top to bottom. In the Setup we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried in steady-state in 3-dimensional cartesian coordinates.

```
Model
  Setup
    Simulation Type = Steady state
    Steady state max. iter = 1
```

In the Equation section we choose the relevant equations which in this case only includes the `Linear elasticity` equation which solves the problem according to linear elastic theory. We also want to compute the stresses as a post-processing step. For the linear system solvers we change the default settings in order to obtain a better convergence in this case. As the equation is fully linear we also eliminate the nonlinear iteration loop.

```
Model
  Equation
    Name = Elasticity
    Apply to Bodies = Body 1
    Linear elasticity
      Active = on
      Calculate Stresses = on
    Edit Solver Setting
      Linear System
        Method = Iterative / GCR
        Preconditioning = ILU1
      Nonlinear system
        Max. iterations = 1
      Apply
    Add
    OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the mass. Other properties assume a physical law, such as Young's modulus and Poisson ratio.

```
Model
  Material
    Name = Pine
    General
      Density = 550
    Linear Elasticity
      Youngs Modulus = 10.0e9
      Poisson ratio = 0.37
    Apply to Bodies = Body 1
    Add
    OK
```

In this case there is a body force i.e. the gravity acting on the beam. We assume that the gravity points to the negative y direction.

```
Model
  BodyForce
    Name = Gravity
    Linear Elasticity
      Force 2 = $ -9.81 * 550
    Apply to Bodies = Body 1
```

```
    Add
    OK
```

Here we use a MATC expression for computing the volume force. This expression is constant and is computed when the command file is interpreted.

Convergence should be obtained with the default initial condition i.e. zero for all fields, hence no initial condition is applied.

The first boundary condition fixes the beam rigidly at the wall. The second boundary condition distributes the load of 2000 N uniformly on the area of 5.0e-3 m^2.

```
Model
  BoundaryCondition
    Name = Wall
    Linear elasticity
      Displacement 1 = 0.0
      Displacement 2 = 0.0
      Displacement 3 = 0.0
    Add
    New

    Name = Mass
    Linear elasticity
      Force 2 = -4.0e5
    Add
```

The conditions may also be assigned to boundaries in the Boundary condition menu, or by clicking with the mouse. Here we use the latter approach as that spares us of the need to know the indexes of each boundary.

```
Model
  Set boundary properties
    Choose the wall end of the beam -> set boundary condition Wall
    Choose the other end of the beam -> set boundary condition Mass
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

The simulation may take a minute or so depending on the speed of the processor. This time the convergence monitor does not have a meaningfull output since the of the different steps only one is related to the actual solution and the six other ones to the computation of stresses with the Galerkin method.

Results

When there are some results to view we may start the postprocessor, this time we use ElmerPost.

```
Run
  Start postprocessor
```

As a result the absolute value of maximum displacement is shown. The maximum displacement is 6.36 cm To visualize the displacement in the geometry using ElmerPost can be done with the following command in the `Elmer-Post` command line.

```
  math n0=nodes
  math nodes=n0+Displacement
```

To redraw the picture with new settings use the rightenmost icon on the top row. The resulting picture is shown in Fig 2.2 Note that the displacement are so large that the assumption of linearity may be severely

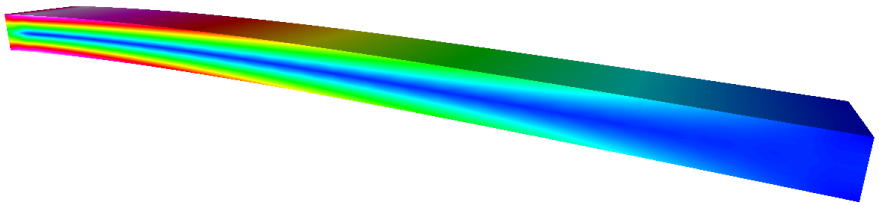

Figure 2.2: The displaced shape of the elastic beam colored with the von Mises stresses

questioned. When further increasing the loading one should resort to a solver that is able to catch the geometric nonlinearities.

Extra task: Gravity in x direction

The beam should be more rigid if the beam is oriented differently. For that aim, change the direction of gravity to orient in the negitive x. Change the body force

```
Model
  BodyForce
    Linear Elasticity
      Force 1 = $ -9.81*550
    Update
    OK
```

and the boundary condition

```
Model
  BoundaryCondition
    Linear elasticity
      Force 1 = -4.0e5
    Update
    OK
```

The rigidity should scale as dh^3 and hence the maximum displacement should be reduced roughly to one quarter of the original.

Tutorial 3

Smitc solver – Eigenmodes of an elastic plate

Directory: ElasticPlateEigenmodesGUI
Solvers: SmitcSolver
Tools: ElmerGUI
Dimensions: 2D, Eigenmode

Problem description

For thin elastic structures it is often advicable to use dimensionally reduced models i.e. study plates or shells. In this tutorial we compute the few lowest eigenmodes of an elastic plate. Our geometry is a simple pentagon which (compared to a square) eliminates some of the trivial symmetries. The pentagon is rigidly fixed at all boundaries.

For more details on the solver we refer to the documentation of Smitc solver in the Elmer Models Manual.

Solution procedure

Start `ElmerGUI` from command line or by clicking the icon in your desktop. Here we describe the essential steps in the ElmerGUI by writing out the clicking procedure. Tabulation generally means that the selections are done within the window chosen at the higher level.

Before we can start the set-up we shoud make sure that the menus for Smitc solver are present. If not, they may be found in file

```
$ELMERHOME/bin/edf-extra/elasticplate.hml
```

To load these definitions do the following

```
File
  Definitions
    Append -> choose the file
```

To see what kind of new menu structures you got you may play around with viewer collapsing and opening. Note that if you want to load an existing project you should load the xml-definitions that were used in creating the project. Therefore it may be best to place all actively used menu definitions in directory

```
$ELMERHOME/bin/edf
```

When the menu structures for plate solver are there we are ready to continue. The mesh is given in 2d netgen format in file `pentagon.grd` in the samples directory of ElmerGUI, load this file.

```
File
  Open -> pentagon.in2d
```

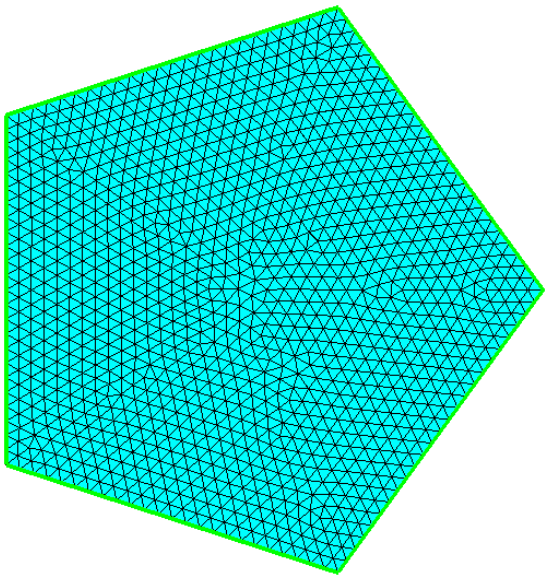

Figure 3.1: The finite element mesh in ElmerGUI

You should obtain a pentagon consisting of 5 triangles. To increase the number of elements change the parameters passed on to the nglib library by going to

```
Mesh
  Configure
    nglib / Max H: 0.05
```

You may check in the `Model summary` window that it consists of 1199 nodes and 2276 linear triangles. If the mesh was successfully imported your window should look something in figure 3.1.

After we have the mesh we start to go through the Model menu from the top to bottom. In the `Setup` we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 2-dimensional cartesian coordinates and in steady-state (also used for eigenmodes). Only one steady-state iteration is needed as the case is linear.

```
Model
  Setup
    Simulation Type = Steady state
    Steady state max. iter = 1
    Apply
```

In the equation section we choose the relevant equations and parameters related to their solution. When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and therefore its easier to assign the Equation and Material to it directly.

For the solver setting we need to activate the eigen mode computation. We also choose the direct umfpack solver which for small 2D problems often performes great.

```
Model
  Equation
    Add
      Name = Plate Equation
      Apply to bodies = 1
```

```
    Elastic Plates
      Active = on
      Edit Solver Settings
        Solver Specific Options
          Eigen Analysis = on
          Eigen System Values = 10
        Linear System
          Direct = on
            Umfpack
  Add
  OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the mass. Other properties assume a physical law, such heat Youngs modulus. As our problem is academic in nature we choose some simple ideal parameters but data from material database could also be used instead.

```
Model
  Material
    Add
      Name = Ideal
      Apply to bodies = 1
      General
        Density = 1000.0
      Elastic Plates
        Youngs Modulus = 1e9
        Poisson ratio = 0.3
        Thickness = 0.001
        Tension = 0.0
      Add
      OK
```

A Body Force represents the right-hand-side of a equation i.e. external forces. In eigenmode analysis no body forces are used. Nor are any Initial conditions required.

In this case all the boundaries are rigidly fixed we set all the components of the solution field to be zero. The 1st component is the displacement in the normal direction while the 2nd and 3rd components are its derivaties in x and y directions.

```
Model
  BoundaryCondition
    Add
      Elastic Plates
        Deflection 1 = 0.0
        Deflection 2 = 0.0
        Deflection 3 = 0.0
      Name = Fixed
      Apply to boundaries = 1 2 3 4 5
      Add
      OK
```

For the execution ElmerSolver needs the mesh files and the command file. We have now basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. In saving the project all the necessary files for restarting the case will be saved to the destination directory.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. In this case there is just one iteration and thus no curve appears.

Results

The resulting eigenvalues are shown in table 3.1. Note that some eigenmodes are degenerated but as the finite element mesh is not perfectly symmetric there will be minor differencies in the eigenvalues.

Table 3.1: Ten lowest eigenvalues for the pentagon plate

No	ω^2
1	18.9
2,3	81.3
4,5	214.5
6	281.1
7, 8	472.5
9, 10	621.0

Note: if you face problems in the solution phase and need to edit the setting, always remember to save the project before execution.

To view the results we may start the ElmerPost or use the internal VTK widget, as is done here

```
Run
  Postprocessor (VTK)
```

To show the 1st component

```
  Surfaces
    Control / Surface: Deflection.1
    Apply
    OK
```

The default configuration shows only the 1st eigenmode. To get all the eigenmodes do the following:

```
  File
    Read input file
      Timesteps / End: 10
      Apply
      OK
```

To go through all eigenmodes (treated here as timesteps)

```
  Edit
    Time step control
      Loop
```

Here you may also save the pictures to files frame*.png by activating the checkbox. In figure 3.2 the lowest eigenmodes are depicted.

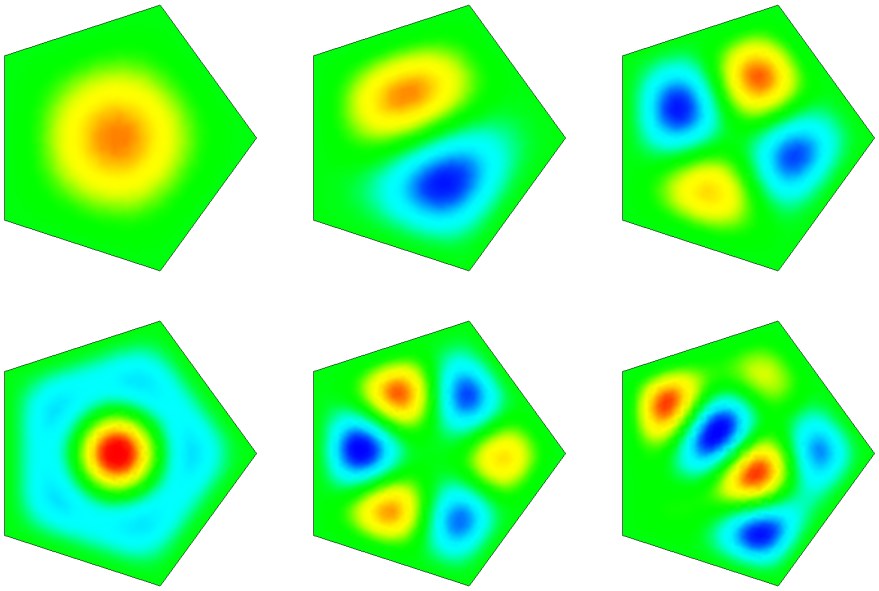

Figure 3.2: The 1st, 2nd, 4th, 6th, 7th and 9th eigenmode of the plate

Extra task

You may test the effect of pre-stressing by altering the Tension material parameter.

There are other similar geometries that you could use i.e. `hexagon.in2d`, `heptagon.in2d`, `octagon.in2d`. When the number of vertices is increased the eigenvalues should slightly decrease.

Tutorial 4

Electrostatic equation – Capacitance of two balls

Directory: CapacitanceOfTwoBalls
Solvers: StatElecSolver
Tools: netgen,ElmerGUI
Dimensions: 3D, Steady-state

Case definition

This case presents the solution of the capacitance of perfectly conducting balls in free space. A voltage difference between the balls results to electric charge being introduced to the system. The balls have also self-capacitance that comes from the voltage difference with the far field. Therefore a symmetric capacitance matrix with of size 2×2 needs to be solved. The capacitances may be computed from two different voltage configurations. For both the electrostatic equation is solved automatically.

The problem does not have an analytical solution in a closed form. However, the cross-capacitance between the balls may be approximated from the series solution [**?**, Ch. A.3]:

$$C_{12} = 4\pi\varepsilon \frac{a^2}{d}\left(1 + \frac{a^2}{d^2 - 2a^2} + \frac{a^4}{d^4 - 4d^2a^2 + 3a^4} + \dots\right) \tag{4.1}$$

and the self-capacitance from

$$C_{10} = C_{20} = 4\pi\varepsilon a\left(1 - \frac{a}{d} + \frac{a^2}{d^2 - a^2} + \frac{a^3}{d^3 - 2da^2} + \dots\right) \tag{4.2}$$

Let's mark $\tilde{C} = C/\varepsilon$. In this case $\tilde{C}_{12} \approx 1.191$ and $\tilde{C}_{10} \approx 5.019$. Unfortunately the error bounds are not given.

In this particular case the balls are assumed to have a radius of $a = 0.5$ and they are placed at distance $d = 2$ apart from each other (measured from the ball origins).

Meshing

In this case meshing is performed with the graphical user interface of netgen. Netgen creates tetrahedral quality meshes and provides a native output for Elmer. At the time of writing this tutorial the quadratic elements had some problems with numbering but these should not affect the linear elements.

The file is given as netgen geometry format in file `TwoBallsInBall.geo`. The geometry definition includes the two smaller balls inside a bigger ball. Ultimaterly the bigger ball would be infinitely large. As this is impossible here we choose a modest radius of 5. The larger this value, the better the far-field approximation of the electrostatic solution is.

The content of the file is given below:

```
#
# a large ball with two smaller balls cut off
#
algebraic3d
solid smallballs = sphere (-1.0, 0.0, 0.0; 0.5)
           or sphere (1.0, 0.0, 0.0; 0.5);
solid bigball = sphere (0.0, 0.0, 0.0; 5.0);
solid rest = bigball and not smallballs;
tlo rest -col=[0,0,1] -transparent;
```

Open the file and apply the default meshing. In this example two consecutive uniform refinements were performed (choose `Refine Uniform` under `Refinemenent`) so that the final mesh consisted of 41 693 nodes and 238 976 linear tetrahedrons.

To save the mesh first choose under `File` the `Export Filetype` to be `Elmer`. Then choose `Export Mesh` and save the mesh into a suitable directory to be opended by ElmerGUI.

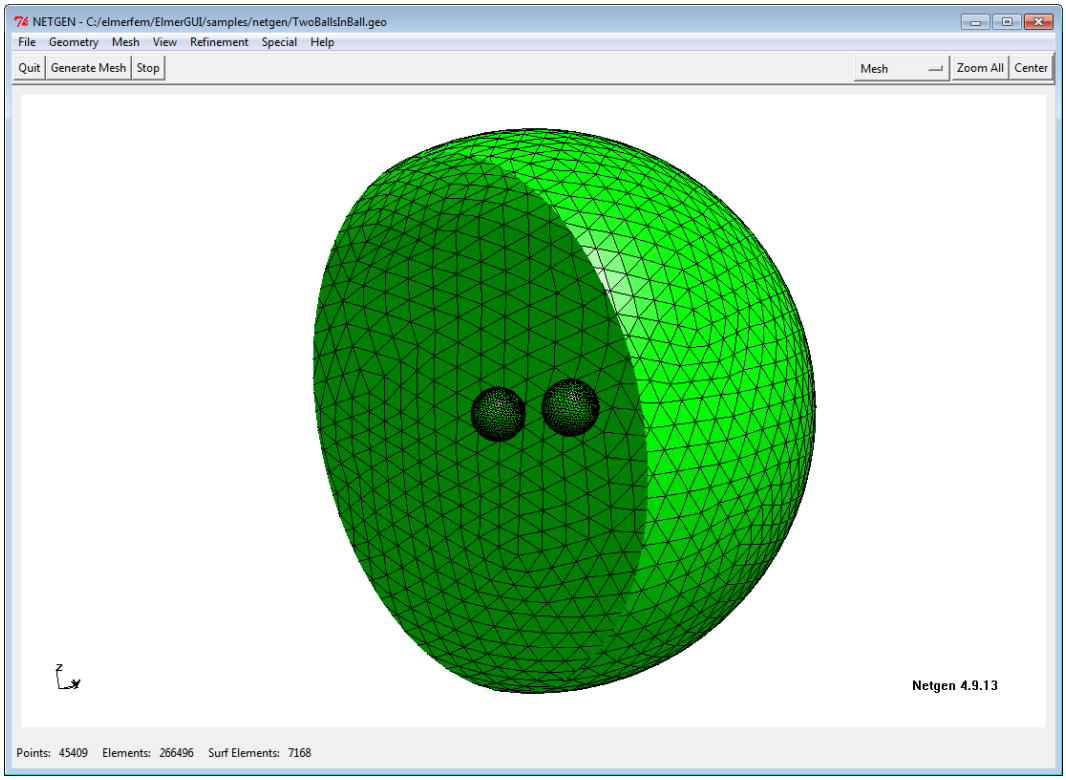

Figure 4.1: Surface mesh for the two inner balls as seen in Netgen

The order of the mesh using nodal elements may be increased by `ElmerGrid`. Assuming the mesh would recide in directory `meshlin` a mesh consisting of quadratic elements may be performed with the following command:

```
ElmerGrid 2 2 meshlin -increase -out meshquad
```

This will maintain the number of elements but the number of nodes will, in this case, increase to 359 009.

Solution procedure

The definitions for the electrostatic equation are not loaded into ElmerGUI by default. Hence, one needs to load these before starting the simulations.

```
File
  Definitions
    Append -> electrostatics.xml
```

The additional definitions should recide in the directory `edf-extra` within the distribution. Moving the desired `xml` files to the `edf`-directory enables automatic loading of the definitions at start-up. By inspecting the definitions in the `Elmer Definitions File editor` one may inspect that the new definitions were really appended.

The mesh is already created, load it from the directory that was created above.

```
File
  Load Mesh -> mesh
```

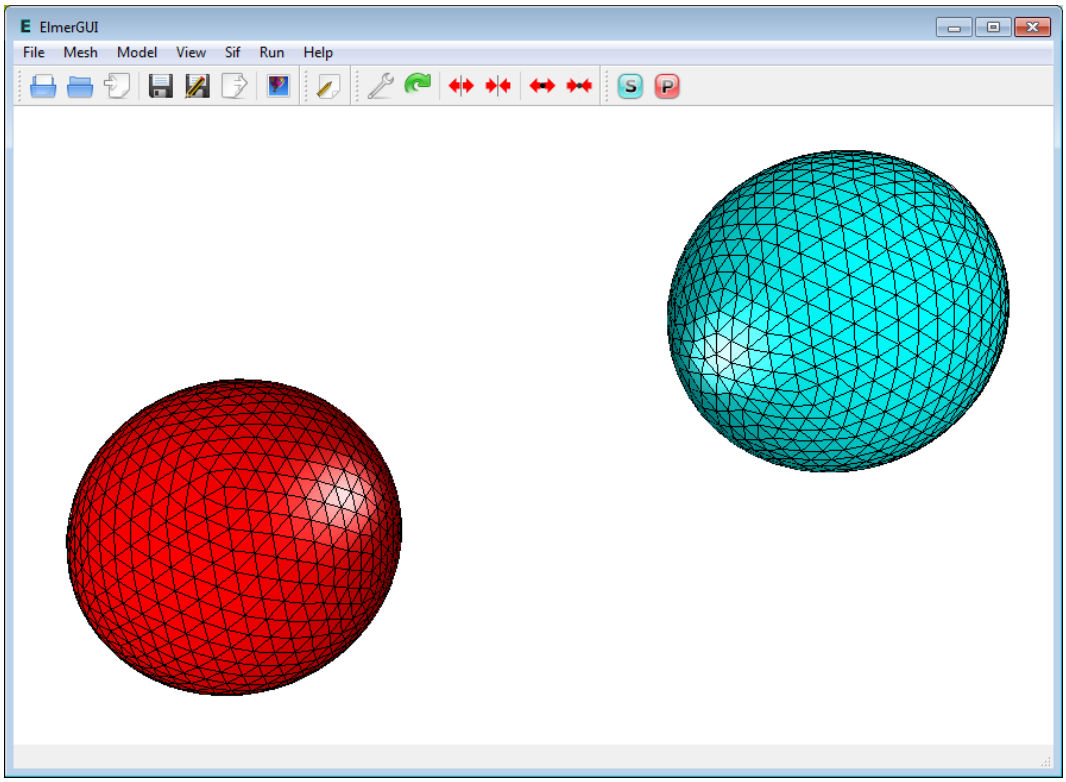

Figure 4.2: The mesh with one highlighted ball as seen in ElmerGUI

After we have the mesh we start to go through the Model menu from the top to bottom. In the Setup we choose things related to the whole simulation such as file names, time stepping, constants etc. The steady-state simulation is carried out in 3-dimensional cartesian coordinates. For convenience we also set the permittivity of vacuum ε_0 equal to one. This makes it easier to compare the results to the analytical expressions.

```
Model
  Setup
    Simulation Type = Steady state
    Vacuum Permittivity = 1.0
```

In the equation section we choose the relevant equations and parameters related to their solution. In this case we'll have only the electrostatics solver.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and therefore its easier to assign the Equation and Material to it directly.

In the solver specific options we want to activate some flags that are needed to invoke the computation of derived fields. For the linear system solvers we are happy to use the defaults. One may however, try out different preconditioners (ILU1,...) or direct Umfpack solver, for example.

```
Model
  Equation
    Name = Electrostatics
    Apply to Bodies = 1
    Electrostatics
      Active = on
      Edit Solver Settings
        Solver specific options
          Calculate Capacitance Matrix = True
          Calculate Electric Field = True
          Calculate Electric Energy = True
    Add
    OK
```

The Material section includes all the material parameters. In this case we only have the relative permittivity ε_r which we set to one.

```
Model
  Material
    Name = Ideal
    Electrostatics
      Relative Permittivity = 1.0
    Apply to Bodies = 1
    Add
    OK
```

We have two boundary conditions for the potential at the ground and at the capacitor. For other boundaries the do-nothing boundary results to zero flux over the boundary.

```
Model
  BoundaryCondition
    Name = Farfield
    Electrostatics
      Electric Infinity BC = True
    Add
    New

    Name = CapBody1
    Electrostatics
      Capacitance Body = 1
    Add
    New

    Name = CapBody2
    Electrostatics
      Capacitance Body = 2
    Add
```

The conditions may also be assigned to boundaries in the Boundary condition menu, or by clicking with the mouse. Here we use the latter approach as that spares us of the need to know the indexes of each boundary.

```
Model
```

```
  Set boundary properties
    Choose Outer sphere -> set boundary condition Farfield
    Choose one inner sphere -> set boundary condition CapBody1
    Choose the other inner sphere -> set boundary condition CapBody2
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The equation is fully linear and hence only two iterations are needed – the second one just ensures that convergence of the nonlinear level was really obtained. The norm of the solution should be?

When the solution has finished we may start the postprocessor to view some results.

```
Run
  Start postprocessor
```

Results

The essential result of this case are the values of the capacitance matrix. In this case $\tilde{C}_{12} \approx 1.691$ and $\tilde{C}_{10} \approx 5.019$. For linear elements the obtained figures are 1.6983, 5.0793 and 5.0812, for quadratic Lagrange elements 1.6641, 5.0340 and 5.0340, respectively, and finally for quadratic p-elements 1.6856, 4.9863 and 4.9884.

The values are rather satisfactory with a difference less than 2% from the series approximation.

Note that the derived fields in the StatElecSolver are computed by averaging the fields over elements – not using the Galerkin method which would provide optimal accuracy. To get optimal accuracy, use `FluxSolver`, for example

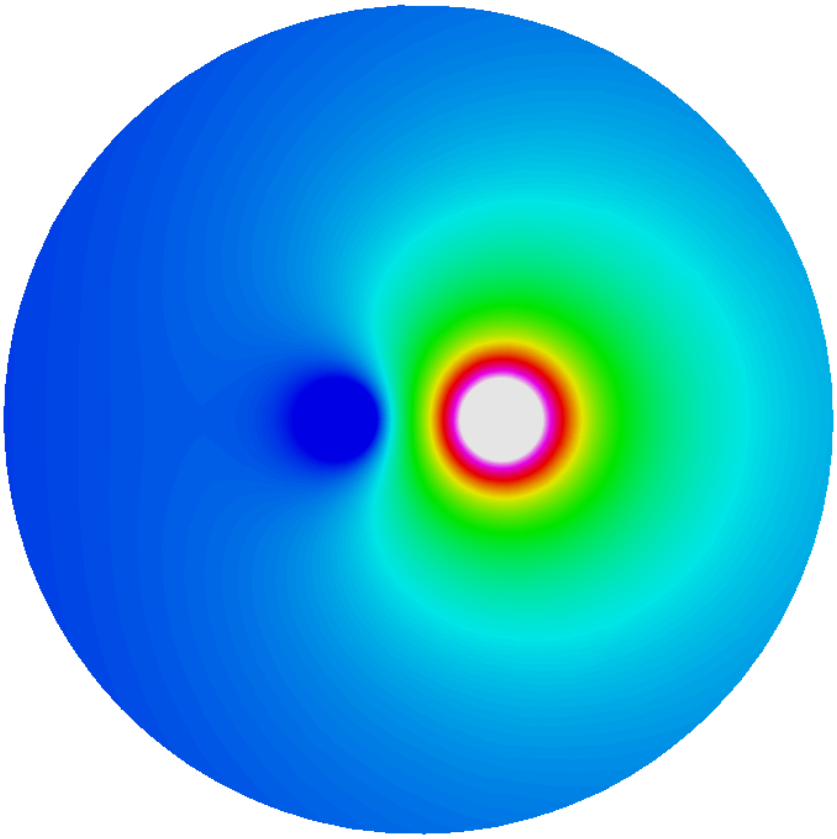

Figure 4.3: The electrostatic potential on the clipping plane. This is the latter of the two symmetric configurations where the unit voltage is applied to one ball and zero voltage to the other, respectively.

Tutorial 5

Navier-Stokes equation – Laminar incompressible flow passing a step

Directory: FlowStepGUI
Solvers: FlowSolve
Tools: ElmerGUI
Dimensions: 2D, Steady-state

Case definition

This tutorial represents the canonical step flow of viscous fluid. A fluid, flowing past a step (see figure 5.1), has the density 1 kg/m and viscosity 0.01 kg/ms. The velocity profile at the inlet is parabolic with a mean velocity $< v_x >= 1.0$ m/s and $v_y = 0.0$ m/s. At the outlet only the vertical component is defined, $v_y = 0.0$ m/s. At all other walls the no-slip boundary condition, $\vec{v} = 0$, is applied. Thus the Reynolds number for the case is around 100.

Figure 5.1: Geometry of the step flow problem

Mathematically the problem to be solved is

$$\begin{cases} -\nabla \cdot (2\mu\bar{\bar{\varepsilon}}) + \rho\vec{u} \cdot \nabla\vec{u} + \nabla p &= 0 \quad \text{in } \Omega \\ \nabla \cdot \vec{u} &= 0 \quad \text{in } \Omega \end{cases} \tag{5.1}$$

with the boundary conditions

$$\begin{cases} u_x &= 1 \quad \text{on } \Gamma_{inlet} \\ u_x &= 0 \quad \text{on } \Gamma_{no-slip} \\ u_y &= 0 \quad \text{on } \Gamma_{inlet} \cup \Gamma_{outlet} \cup \Gamma_{no-slip} \end{cases} \tag{5.2}$$

where μ is the viscosity, $\bar{\bar{\varepsilon}}$ is the strain tensor, ρ is the density, $\vec{u}$ is the velocity and p is the pressure. It is assumed that the density and viscosity are constants.

Solution procedure

The mesh is given in ElmerGrid format in file `step.grd`, load this file.

```
File
  Open -> step.grd
```

You should obtain your mesh and may check that it consists of 9696 nodes and of 9442 bilinear elements.

```
Model
  Summary...
```

After we have the mesh we start to go through the Model menu from the top to bottom. In the Setup we choose things related to the whole simulation. The steady-state simulation is carried out in 2-dimensional cartesian coordinates, which are also the defaults.

```
Model
  Setup
    Simulation Type = Steady state
    Coordinate system = Cartesian
```

In the equation section we choose the relevant equations and parameters related to their solution. In this case the only the Navier-Stokes equation is needed.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and therefore its easier to assign the Equation and Material to it directly. One could also edit the solver setting in order to try different strategies for solving the nonlinear or linear system. Initially the Navier-Stokes solver uses the more robust Picard iteration which is changed to Newton iteration after few initial steps. For the given viscosity the default values are ok, but may need tuning when going into higher Reynolds numbers.

```
Model
  Equation
    Name = Navier-Stokes
    Apply to Bodies = Body 1
    Navier-Stokes
      Active = on
      Edit Solver Setting
        Nonlinear System
          Max. iterations = 20
          Newton after iterations = 3
    Add
    OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the density. Other properties assume a physical law, such as viscosity.

```
Model
  Material
    Name = Ideal
    General
      Density = 1.0
    Navier-Stokes
      Viscosity = 0.01
    Apply to Bodies = Body 1
    Add
    OK
```

The current case does not have any body forces. Convergence should also be obtained using the default initial condition which sets all field values to zero. Hence no setting for initial condition are needed.

Only one boundary condition may be applied to each boundary and therefore all the different physical BCs for a boundary should be grouped together. In this case the Temperature and Velocity. The side walls are assumed to be adiabatic.

The parabolic inlet-profile is achieved using the MATC environment. To be able to edit the content of the inlet profile click `Enter` to open an edit box for the `Velocity 1`. The given expression will be interpreted at run-time so that $v_x = 6(y-1)(2-y)$. As $y \in [1,2]$ thereby creating a parabolic velocity profile with a mean velocity of unity.

```
Model
  BoundaryCondition
    Name = Inlet
    Navier-Stokes
      Velocity 1 = Variable Coordinate 2; Real MATC "6*(tx-1)*(2-tx)"
      Velocity 2 = 0.0
    Add
    New

    Name = Outlet
    Navier-Stokes
      Velocity 2 = 0.0
    Add
    New

    Name = Walls
    Navier-Stokes
      Noslip wall BC = on
    Add
    OK
```

The conditions may also be assigned to boundaries in the Boundary condition menu, or by clicking with the mouse. Here we use the latter approach as that spares us of the need to know the indexes of each boundary.

```
Model
  Set boundary properties
    Choose Inlet -> set boundary condition Inlet
    Choose Outlet -> set boundary condition Outlet
    Choose Walls -> set boundary condition Walls
```

For the execution ElmerSolver needs the mesh files and the command file. We have now basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case. Create a suitable directory for the case if needed.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The problem should converge in about ten iterations to a norm of 0.4347 visible on the output.

When there are some results to view we may start the postprocessor also

```
Run
  Start postprocessor
```

Results

The results may be viewed using the postprocessor as shown in Figure 5.2 and 5.3. One may also register specific values, for example the pressure difference is 0.388 Pa, the minimum and maximum lateral velocities are -0.1666 m/s and 1.5 m/s, respectively. One special result of interest is the point, on the x-axis, at which the direction of the flow changes. In this case its position is about 5.0 m after the step.

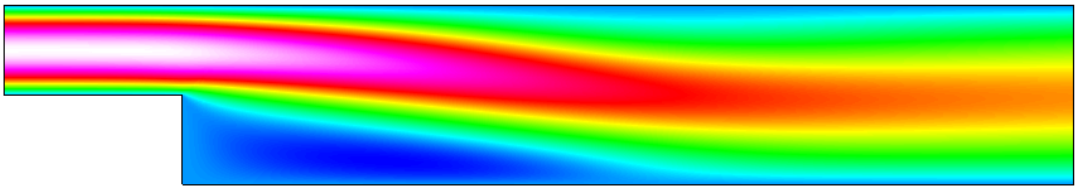

Figure 5.2: Absolute value of the velocity field

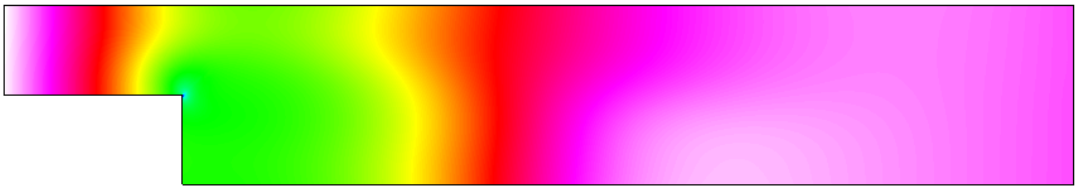

Figure 5.3: Pressure field

Extra task: Decreasing the viscosity

Try what happens if the viscosity is further decaresed by a factor 10. Convergence may be difficult to obtain. Some tricks that may be tested include

- Introducting a relaxation factor (typically in the range 0.5–0.7)
- Increasing number of nonlinear iterations
- Favoring Picard iteration over Newton
- Increasing mesh density (and length of domain)

Don't be worried if you fail to find convergence. This task will mainly act as a motivator in using turbulence models for higher Reynolds numbers.

Remember to re-perform the following phases in order to get the updated results

```
Sif
  Generate
File
  Save Project
Run
  Start solver
```

You may just reload the results in the postprocessor rather than closing and opening the program.

Tutorial 6

Vortex shedding – von Karman instability

Directory: VonKarmanGUI
Solvers: FlowSolve
Tools: ElmerGUI
Dimensions: 2D, Transient

Case definition

This tutorial is about simulating the developing of the vortex shedding i.e. the von Karman instability. The geometry is a tube with a circular obstacle. For more details on the problem look at the benckmark case definition by by M. Schäfer and S. Turek in *"Benchmark computations of laminar flow around a cylinder"*.

Solution procedure

The mesh is given in 2d netgen format in file `circle_in_channel.in2d`, load this file.

```
File
  Open -> circle_in_channel.in2d
```

You should get a mesh consisting of 749 nodes and 1328 triangles. This is a rather sparse mesh. To increase the element number

```
Mesh
 Configure
   nglib / Max H: 0.02
Mesh
  Remesh
```

This mesh includes 3464 nodes and 6506 triangles. The mesh is presented in figure 6.1.

After we have the mesh we start to go through the Model menu from the top to bottom. In the Setup we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 2-dimensional cartesian coordinates. 2nd order bdf time-stepping method is selected with 200 steps and we want the total simulation time to be 8 seconds.

```
Model
  Setup
    Simulation Type = Transient
    Steady state max. iter = 1
    Time Stepping Method = bdf
```

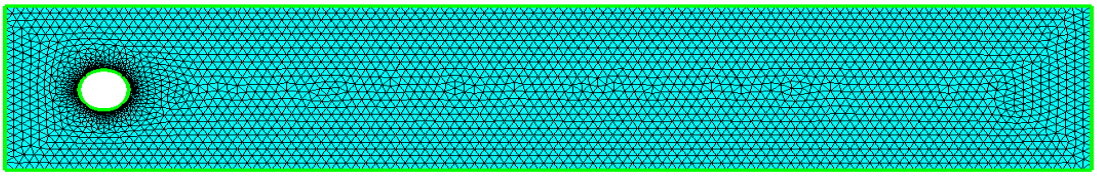

Figure 6.1: Computational mesh of the problem.

```
    BDF Order = 2
    Time Step Intervals = 200
    Time Step Sizes = $ 8/200
```

For the solver specific settings we are quite happy to use the defaults. However, we relax a little bit the convergence tolerances to get speedier simulation.

```
Model
  Equation
    Name = Navier-Stokes
    Apply to Bodies = 1
    Navier-Stokes
      Active = on
    Edit Solver Settings
      Nonlinear system
        Convergence tol. = 1.0e-4
      Linear System
        Convergence tol. = 1.0e-6
    Add
    OK
```

The Material section includes all the material parameters. Here we choose simple parameters for the academic test case

```
Model
  Material
    Name = Ideal
    General
      Density = 1
    Navier Stokes
      Viscosity = 0.001
    Apply to Bodies = 1
    Add
    OK
```

The system does not need any body forces nor initial conditions i.e. we are happy with the default guess zero.

We have three different kinds of boundaries: inlet, no-slip walls, and outlet. The inlet has a parabolic fully developed laminar profile with a maximum velocity of 1.5 m/s. Additionally for the inlet the vertical velocity component is assumed zero. The circle and the lower and upper walls are given the no-slip treatment.

For the outlet only the vertical component is set to zero since the default discretization weakly imposes a zero pressure condition if the normal velocity component is not defined.

```
Model
  BoundaryCondition
    Name = Inlet
    Navier-Stokes
      Velocity 1 = Variable Coordinate 2; Real MATC "4*1.5*tx*(0.41-tx)/0.41^2"
      Velocity 2 = 0.0
    Add
    New

    Name = Walls
    Navier-Stokes
      Velocity 1 = 0.0
      Velocity 2 = 0.0
    Add
    New

    Name = Outlet
    Navier-Stokes
      Velocity 2 = 0.0
    Add
    Ok
```

The conditions may also be assigned to boundaries in the Boundary condition menu, or by clicking with the mouse. Here we use the latter approach as that spares us of the need to know the indexes of each boundary.

```
Model
  Set boundary properties
    Choose inlet -> set boundary condition Inlet
    Choose both horizontal walls and circle -> set boundary condition Walls
    Choose outlet -> set boundary condition Outlet
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The norm after the first timestep should be around 0.695, and after last 0.749, respectively.

When there are some results to view we may start the postprocessor also

```
Run
  Start postprocessor
```

Results

Due to the number of the time-steps the simulation will take a few minutes. You may inspect the results with ElmerPost as the time-steps are computed, or wait until all timesteps have been computed. When opening the result file using ElmerGUI ElmerPost only opens the first time-step. Therefore it is important to reopen the file and load the time-steps of interest. Pressing the button `All` selects all the calculated time steps. A video of the results can be viewed by selecting the option `Timestep Control` and pressing the button `Loop` under the `Edit` menu.

In Figure 6.2 the velocity field is presented for three different timesteps. The maximum velocity in the system should be about 2.1724 m/s.

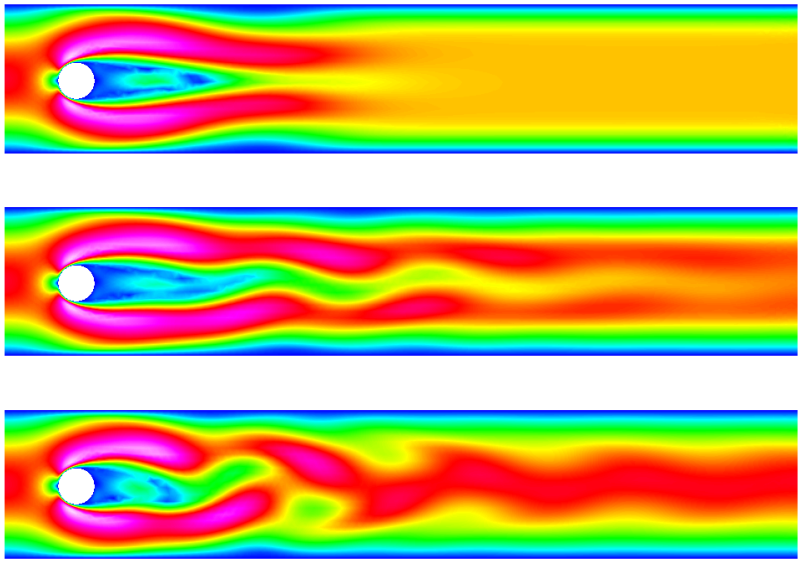

Figure 6.2: Temperature distribution at steps 20, 100 and 200

Effect of Reynolds number

The Reynolds number in this case is around 100 resulting to unsteady flow. The critical Reynolds number is around 90 and reducing the flow velocity so that Reynolds number becomes, say 20, makes the system to possess a steady-state solution. On the other hand, increasing the velocity will make the von Karman vortecis even more pronounced until they break into fully chaotic motion. This finite element mesh will allow only minor increase in Reynolds number to be able to capture the phenomena.

Tutorial 7

Thermal flow in curved pipe

Directory: CurvedPipeGUI
Solvers: HeatSolve, FlowSolve
Tools: ElmerGUI
Dimensions: 3D, Steady-state

Case definition

This tutorial demonstrates how to set up a coupled case of thermal flow in curved pipe with a finite thickness. Within the pipe both the flow and heat transfer equations need to be solved while on the solid section only heat transfer needs to be considered.

The inner diameter of the pipe is 0.01 m and the outer 0.012 m, respectively. It is bend to a 135 degree angle with a radius of 0.02 m. Beyond the bend 0.03 m of the direct part is also accounted for. The fluid flowing in the pipe is water with and original temperature of 350 K. The outer temperature of the iron pipe is 300 K making the water gradually to cool.

The water is injected with a parabolic velocity profile with a maximum of 0.01 m/s. In reality the laminar analytic profile is described by the Bessel's function. Here the flow is treated as laminar and steady-state even though at these Reynolds number 100 the unsteady nature of the flow should probably considered. This would enhance the heat transfer. The steady-state case, however, will achieve the educational goals of the tutorial.

Solution procedure

The mesh is defined in ElmerGrid format in file `curved_pipe.grd`, load this file.

```
File
  Open -> curved_pipe.grd
```

You should obtain your mesh and may check that it consists of 23670 trilinear bricks and 25245 nodes. The density of the mesh may be varied by altering the `Reference Density` in the file. For further information on the mesh definition look at the `ElmerGrid` manual. Often it is desirable to use some professional mesh generation tool in CFD and translate it into Elmer format. For educational purposes we are quite happy to use this simple geometry.

After we have the mesh we start to go through the `Model` menu from the top to bottom. In the `Setup` we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 3-dimensional cartesian coordinates in steady-state. There is nothing really to be done here, but you may verify that the defaults are correct.

```
Model
  Setup
    Coordinate system = Cartesian
```

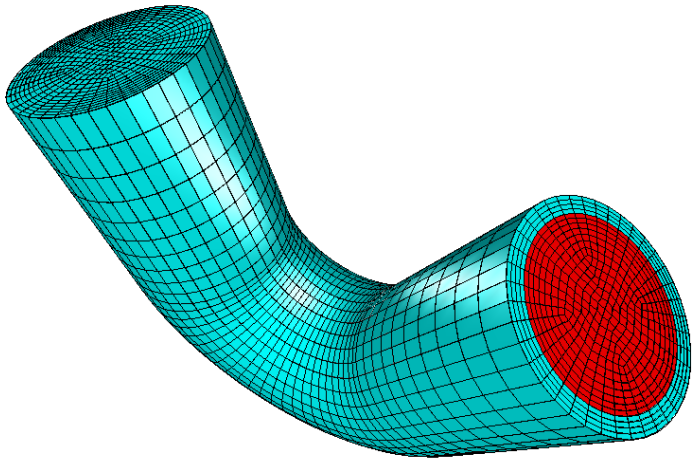

Figure 7.1: The mesh of the curved pipe as seen in ElmerGUI

```
Simulation type = Steady state
Steady state max. iter = 1
...
```

In the Equation section we choose the relevant equations and parameters related to their solution. In this case we'll have two different sets of solvers (called as Equation in Elmer slang). The other consists of heat and flow solvers, while the other includes just the heat solver. We'll name them appropriately.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we know that the fluid body has the index 1 and the solid body has the index 2. Therefore it is easy to assign the Equation and Material to the bodies directly.

Here we neglect the effect of natural convection. Therefore there is just one-directional coupling from the flow to heat transfer. In order to follow the direction of causality we address the flow solver with a higher priority than the heat solver (default is zero).

Here we are quite happy with the default solver settings of the individual equations. However, for the flow solver we change the default proconditioner `ILU0` to `ILU1` to enhance convergence (with increased memory consumption). For 3D cases the direct solvers are usually not feasible so it is better to stick with the iterative `BiCGstab` linear solver.
The equation for the fluid

```
Model
  Equation
    Add
    Name = Heat and Flow
    Apply to Bodies = 1
    Heat Equation
      Active = on
      Convection = Computed
    Navier-Stokes
      Active = on
      Priority = 1
      Edit Solver Setting
        Linear System
          Preconditioning = ILU1
    OK
```

and then for the solid

```
Model
  Equation
    Add
    Name = Just Heat
    Apply to Bodies = 2
    Heat Equation
      Active = on
      Convection = None
    OK
```

The Material section includes all the material parameters. They are divided into generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the density. Other properties assume a physical law, such as conductivities and viscosity.

Here we choose water and iron from the material library. You may click trough the material parameters of the various solvers to ensure that the properties are indeed as they should be. Any consistant set of units may be used in Elmer. The natural choice is of course to perform the computations in SI units.

```
Model
  Material
    Add
    Material library
      Water (room temperature)
    Apply to Bodies = 1
    OK

    Add
    Material library
      Iron (generic)
    Apply to Bodies = 2
    OK
```

The `Body force` section usually represents the right-hand-side of an equation. It could be used to account for the natural convection, for example. In this case, however, we do not apply any body forces.

Also an `Initial condition` could be given in steady-state case to enhance convergence. However, in this case convergence is pretty robust with the default guess of zero.

We have four different boundary conditions: thermal inflow, internal no-slip, outflow, and external fixed temperature. Otherwise natural BCs are assumed. As it is tedious to know the indexes by heart we first define the different BCs and only afterwards apply them to the existing boundaries with the mouse.

```
Model
  BoundaryCondition
    Name = HotInflow
    Heat Equation
      Temperature = 350.0
    Navier-Stokes
      Velocity 1 = 0.0
      Velocity 2 = 0.0
      Velocity 3 = Variable Coordinate
        Real MATC "100.0*(1.0e-4-tx(0)^2-tx(1)^2)"
    Add
    New
```

The condition for `Velocity 3` above may easiest be typed by pressing `Enter`-key in the edit box which will open a larger window for editing.

```
    Name = Outflow
```

```
  Navier-Stokes
    Use normal-tangential coordinate system = on
    Velocity 2 = 0.0
    Velocity 3 = 0.0
  Add
  New

  Name = NoSlip
  Navier-Stokes
    NoSlip Wall BC = on
  Add
  New

  Name = Troom
  Heat Equation
    Temperature = 300.0
  Add
```

When choosing the boundaries it is important that you choose the right inlet. For that purpose you may activate the compass,

```
View
  Compass = on
```

Now the inlet is the one with normal pointing at the z-direction. Now we are ready to choose the boundaries

```
Model
  Set boundary properties
    Choose inlet face -> set boundary condition HotInflow
    Choose outlet face -> set boundary condition Outflow
    Choose outer side -> set boundary condition Troom
```

Unfortunatly we cannot see the internal boundary. For that purpose click on the outer boundary and choose

```
View
  Hide/show selected
```

The outer boundary should vanish and you can proceed with the last BC,

```
Model
  Set boundary properties
    Choose internal side -> set boundary condition Noslip
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case. It's a good idea to give the project an illuminating name. Avoid paths which includes empty spaces since they may cause problems later on.

```
File
  Save Project
    Make New Folder -> curved_pipe
    OK
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The simulation may take a few minutes depending on your platform. After the simulation has terminated we may study the results.

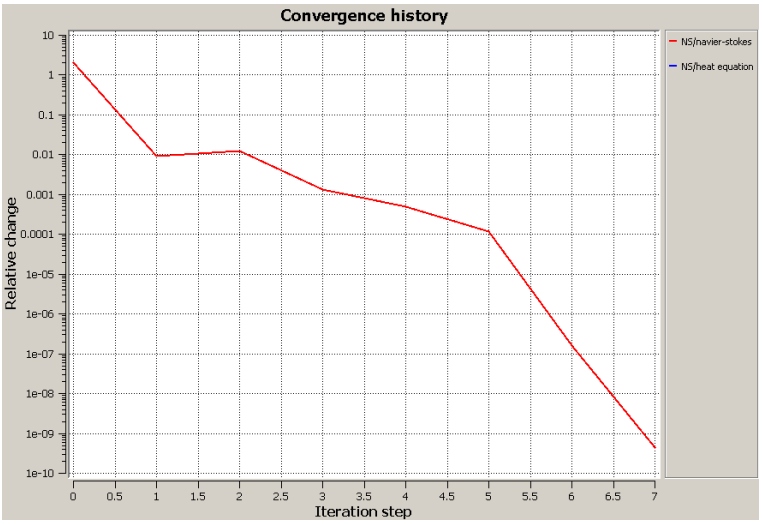

Figure 7.2: Convergence history of the case

Results

The computed norms should be 3.255 for the Navier-Stokes equation and 324.66 for the heat equation. If there is some discrepancy the setup of the system was probably different from the one in the tutorial.

To visualize the results open the postprocessor, in this case `ElmerPost` After the simulation has terminated we may open the postprocessor to view the results.

```
Run
  Start postprocessor
```

A standard way of visualizing is to choose `ColorMesh` and there choose `Surface` and the desired field variable, for example `Velocity_abs` or `Temperature`. In this case only the outflow crosssection contains any information. It may be seen in Figure 7.3 that the symmetry around pipe origin is lost in the bent.

Alternatively we may visualize the crosssection at $y = 0$. To that aim choose `Isocontours` and there set the `Number Of Isosurfaces` to 1, choose `Surface`, set `Contour Variable` to `nodes_y`, and `Color Variable` to `Temperature` etc. Now you may nicely see how the velocity profile affects the temperature distribution in the pipe.

In Figures 7.5 and 7.4 the obtained velocity and temperature distributions are presented.

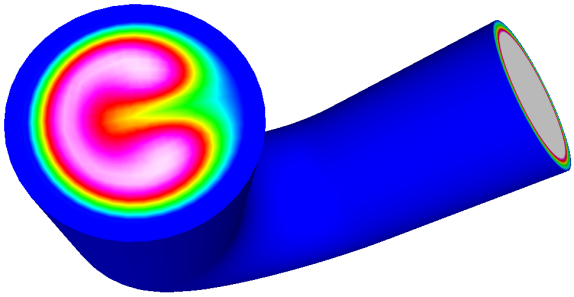

Figure 7.3: Temperature distribution at the outlet of the pipe

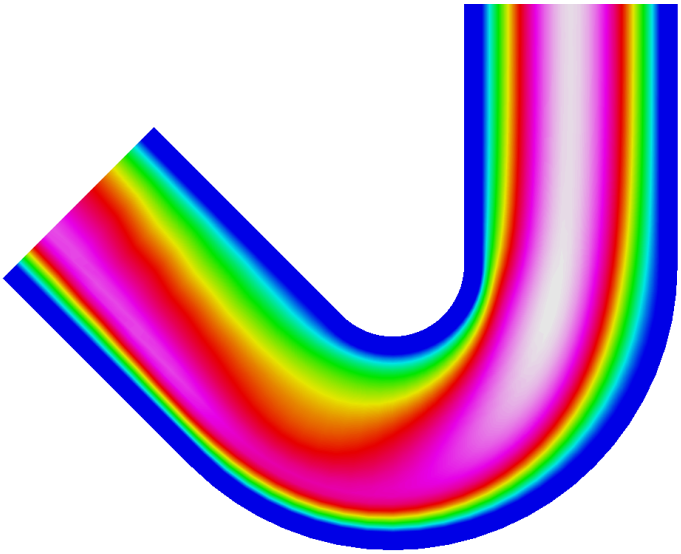

Figure 7.4: Velocity distribution at the cross section $y = 0$.

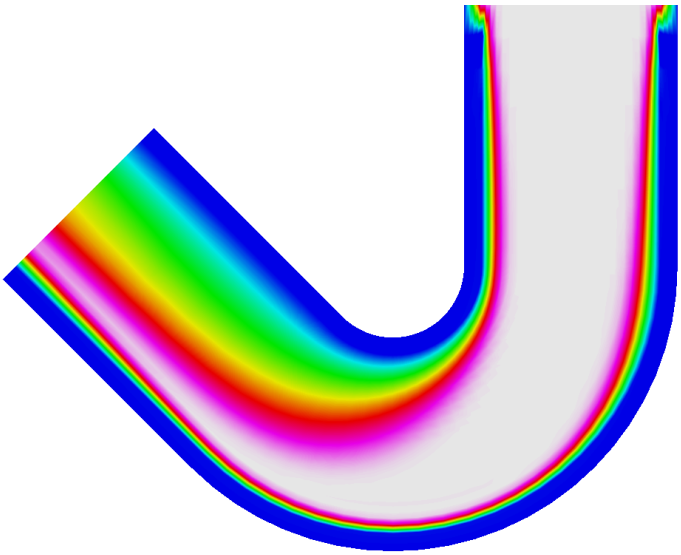

Figure 7.5: Temperature distribution at the cross section $y = 0$.

Tutorial 8

Interaction between fluid flow and elastic obstacle

Directory: FsiObstacleGUI
Solvers: FlowSolve,ElasticSolve,MeshSolve
Tools: ElmerGUI
Dimensions: 2D, Steady-state

Case definition

This tutorial demonstrates how to set up a coupled case of fluid-structure interaction. Flow initiated at one end of a channel which has an elastic obstacle that bends under the influence of fluidic forces. The resulting displacement modify the domain thereby affecting the flow of the fluid.

The channel is assumed to be 2D. The length of the is 10 m and the height is 2 m. At 2 m sits a elastic beam with a rounded top the height of which is 1.2 m and width 0.4 m. A parabolic velocity profile with maximum velocity of 1 m/s is assumed.

Material properties are assumed to be rather simple: For the structure density is 1000 kg/m^3, Youngs module is 1000 Pa, and Poisson ratio 0.3. For the fluid the density is 1 kg/m^3 and viscosity is 0.1 Pas. Additionally the fluid has elastic properties that are used to extent the displacement of the elastic beam to the fluid.

The idea of the case definition is to demonstrate simple yet strongly coupled FSI case without getting into turbulence modeling. Realistic material parameters in the given size of the domain would easily result to turbulence and just small displacements.

The case is solved using standard weak coupling with some relaxation to boost the convergence. The solution is steady-state so only the final results are will be studied.

of thermal flow in curved pipe with a finite thickness. Within the pipe both the flow and heat transfer equations need to be solved while on the solid section only heat transfer needs to be considered.

Solution procedure

The nonlinear elasticty equation is not by default actived in the menu structures ElmerGUI. Hence, the user must load these before starting the simulations.

```
File
  Definitions
    Append -> nonlinearelastity.xml
```

The additional definitions should recide in the directory `edf-extra` within the distribution. Moving the desired `xml` files to the `edf`-directory enables automatic loading of the definitions at start-up. By inspecting

the definitions in the `Elmer Definitions File editor` one may inspect that the new definitions were really appended.

The mesh is defined in .in2d format, the 2D format of netgen, in file `obstacle_in_channel.in2d`, load this file.

```
File
  Open -> obstacle_in_channel.in2d
```

The default mesh is obviously too sparse. To make the mesh more dense set

```
Mesh -> Configure -> nglib -> Max H: 0.1
```

and choose

```
Mesh -> Remesh
```

You should obtain a denser mesh and may check that it consists of around 4140 nodes and 7890 linear triangles.

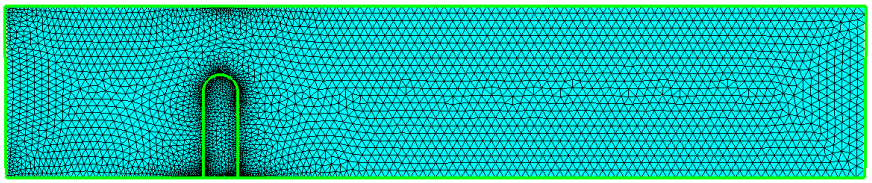

Figure 8.1: The mesh of the obstacle in channel case as seen in ElmerGUI

After we have the mesh we start to go through the `Model` menu from the top to bottom. In the `Setup` we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 2-dimensional cartesian coordinates in steady-state. There is not much to do here, just increase the number of iterations needed for the convergence of the coupled system. We also set the output interval to zero which means that results are written only at the end of the case.

```
Model
  Setup
    Coordinate system = Cartesian
    Simulation type = Steady state
    Steady state max. iter = 100
    Output Intervals = 0
    ...
```

In the Equation section we choose the relevant equations and parameters related to their solution. In this case we'll have two different sets of solvers (called as Equation in Elmer slang). The fluid domain consists of flow and mesh deformation solvers, while the elastic domain just includes the nonlinear elasticity solver. We'll name them appropriately.

To enhance the convergence and efficient use of resources we set relaxation factors of the primary solvers to 0.5 and the number of nonlinear iterations to 1. The mesh deformation solver just extents the displacements of the elasticity solver to the fluid domain, so no relaxation is needed here. For the linear systems we are quite happy with the defaults.

To honor the causality the flow solver should be solved first, then the elasticity solver and as last the mesh deformation solver. We set the priorities accordingly.
The equation for the fluid flow + mesh deformation

```
Model
  Equation
```

```
    Add
    Name = Flow and mesh deform
    Apply to Bodies = 1
    Navier-Stokes
      Active = on
      Priority = 2
      Edit Solver Setting
        Nonlinear System
          Max.iterations = 1
          Nonlinear System Relaxation Factor = 0.5
    Mesh Update
      Active = on
      Priority = 0
    OK
```

and then for the solid

```
Model
  Equation
    Add
    Name = Elasticty
    Apply to Bodies = 2
    Nonlinear Elasticty
      Active = on
      Priority = 1
      Edit Solver Setting
        Nonlinear System
          Max.iterations = 1
          Nonlinear System Relaxation Factor = 0.5
    OK
```

Next we set our rather simple material parameters. The Material section includes all the material parameters. They are divided into generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the density. Other properties assume a physical law, such as conductivities and viscosity.

```
Model
  Material
    Add
    Name = Ideal fluid
    General
      Density = 1.0
    Navier-Stokes
      Viscosity = 0.1
    Mesh Update
      Elastic Modulus = 1.0
      Poisson Ratio = 0.3
    Apply to Bodies = 1
    OK

    Add
    Name = Ideal structure
    General
      Density = 1.0e3
    Nonlinear Elasticity
      Youngs Modulus = 1.0e3
```

```
    Poisson Ratio = 0.3
  Apply to Bodies = 2
  OK
```

The `Body force` section usually represents the right-hand-side of an equation. In this case we do not need any body forces.

Also an `Initial condition` could be given in steady-state case to enhance convergence. However, in this case convergence is pretty robust with the default guess of zero.

We have five different boundary conditions: inflow, outflow, lateral walls with no-slip conditions, fsi conditions, and the beam base. As it is tedious to know the indexes by heart we first define the different BCs and only afterwards apply them to the existing boundaries with the mouse.

```
Model
  BoundaryCondition
    Name = Inflow
    Navier-Stokes
      Velocity 1 = Variable Coordinate 2
        Real MATC "tx*(2-tx)"
      Velocity 2 = 0.0
      Mesh Update 1 = 0.0
    Add
    New
```

The condition for `Velocity 1` above may easiest be typed by pressing `Enter`-key in the edit box which will open a larger window for editing.

```
    Name = Outflow
    Navier-Stokes
      Velocity 2 = 0.0
    Mesh Update
      Mesh Update 1 = 0.0
    Add
    New

    Name = Walls
    Navier-Stokes
      NoSlip Wall BC = on
    Mesh Update
      Mesh Update 1 = 0.0
      Mesh Update 2 = 0.0
    Add
    New

    Name = Base
    Nonlinear Elasticity
      Displacement 1 = 0.0
      Displacement 2 = 0.0
    Add
    New
```

The essense of fluid-structure interaction is in the following boundary condition. When the `Fsi BC` is active the fluidic forces are automatically within ElasticSolver. The backcoupling to Navier-Stokes is achieved through the change in fluid geometry which is enforced by the condiotions for the `MeshSolver`.

```
    Name = FSI
    Nonlinear Elasticity
```

```
    FSI BC = on
  Navier-Stokes
    NoSlip Wall BC = on
  Mesh Update 1 = Equals Displacement 1
  Mesh Update 2 = Equals Displacement 2
```

Now we are ready to choose the boundaries

```
Model
  Set boundary properties
    Choose inlet side -> set boundary condition Inflow
    Choose outlet side -> set boundary condition Outflow
    Choose upper and lower sides (three pieces) -> set boundary condition Walls
    Choose obstacle base -> set boundary condition Base
    Choose interface between fluid and solid (two pieces) -> set boundary condition FS
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case. It's a good idea to give the project an illuminating name. Avoid paths which includes empty spaces since they may cause problems later on.

```
File
  Save Project
    Make New Folder -> fsi_obstacle
    OK
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The simulation may take around 10 seconds depending on your platform.

The computed norms should be around 0.514 for the Navier-Stokes solver, 0.108 for the elasticity solver, and 0.0548 for the mesh update solver. These are reached after 18 iterations using the rather strict default settings.

If there is some discrepancy the setup of the system was probably different from the one in the tutorial. If the results are in agreement we are ready to look at the results.

Results

To visualize the results open the postprocessor, in this case `ElmerPost` After the simulation has terminated we may open the postprocessor to view the results.

```
Run
  Start postprocessor
```

A standard way of visualizing is to choose `ColorMesh` and there choose `Surface` or `Both` and the desired field variable, for example `Velocity_abs` or `Pressure`. The mesh deformation is not active in all output formats. To activate the deformation in ElmerPost you may enter the following sequence of command to the command line at the bottom of ElmerPost window.

```
math n0=nodes
math nodes=n0+Displacement
```

The maximum speed in the system is around 2.328 and the maximum displacement 0.2636. Note that for the saved results the displacement and mesh update fields have been merged into one. In Figures 8.2, 8.3, and 8.4 the obtained velocity, pressure and displacement distributions are presented in the deformed mesh.

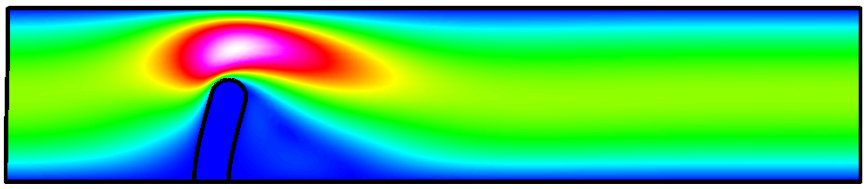

Figure 8.2: Velocity distribution of the obstacle in channel case.

Figure 8.3: Pressure distribution of the obstacle in channel case.

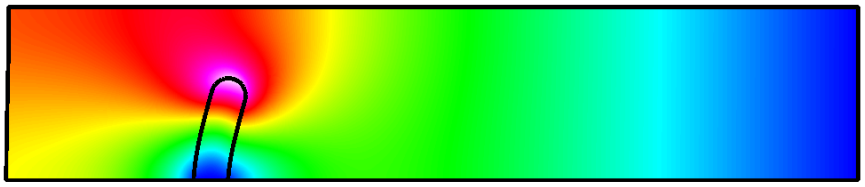

Figure 8.4: Displacement distribution of the obstacle in channel case.

Tutorial 9

Transient flow and heat equations – Rayleigh-Benard instability

Directory: RayleighBenardGUI
Solvers: HeatSolve, FlowSolve
Tools: ElmerGUI
Dimensions: 2D, Transient

Case definition

This tutorial is about simulating the developing of the Rayleigh-Benard instability in a rectangular domain (Figure 9.1) of dimensions 0.01 m height and 0.06 m length. The simulation is performed with water and the material parameters of water required by the Elmer model are presented in Table 9.1. The temperature difference between the upper and lower boundary is set to 0.5 so that lower one has the temperature of 293.5 K and the upper one has the temperature of 293 K.

The density of water is inversely proportional to its temperature. Thus, heated water starts to flow upwards, and colder downwards due to gravity. In this case we assume that the Boussinesq approximation is valid for thermal incompressible fluid flow. In other words, the density of the term $\rho\vec{f}$ in the incompressible Navier-Stokes equation can be redefined by the Boussinesq approximation

$$\rho = \rho_0(1 - \beta(T - T_0))$$

where β is the heat expansion coefficient and the subscript 0 refers to a reference sate.

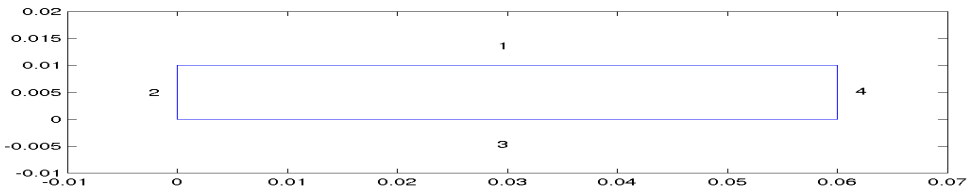

Figure 9.1: Domain.

Table 9.1: Material parameters for water

parameter	value
density	998.3 kg/m^3
viscosity	1040e-6 Ns/m^2
heat capacity	4183 J/(kg·K)
heat conductivity	0.58 W/(m·K)
heat expansion coefficient	2.07e-4 K^{-1}
reference temperature	293 K

Solution procedure

The mesh is given in ElmerGrid format in file `rectangle.grd`, load this file.

```
File
  Open -> rectangle.grd
```

You should obtain your mesh and may check that it consists of 3036 bilinear elements.

There is a possibility to divide and unify edges to simplify the case definition in the future.

```
Choose (left wall + right wall (Ctrl down)) -> unify edge
```

After we have the mesh we start to go through the Model menu from the top to bottom. In the Setup we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 2-dimensional cartesian coordinates. 2nd order bdf time-stepping method is selected with 200 steps and with step size of two seconds.

```
Model
  Setup
    Simulation Type = Transient
    Steady state max. iter = 20
    Time Stepping Method = bdf
    BDF Order = 2
    Time Step Intervals = 200
    Time Step Sizes = 2.0
    Gravity = ...
```

In the equation section we choose the relevant equations and parameters related to their solution. In this case we'll have one set of equations (named "Natural Convection") which consists of the heat equation and of the Navier-Stokes equation.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and therefore its easier to assign the Equation and Material to it directly. It is important to select the convection to be computed since that couples the velocity field to the heat equation.

The system may include nonlinear iterations of each equation and steady state iterations to obtain convergence of the coupled system. It is often a good idea to keep the number of nonlinear iterations in a coupled case low. Here we select just one nonlinear iteration for both equations. For the linear system solvers we are happy to use the defaults. One may however, try out different preconditioners (ILU1,...) or direct Umfpack solver, for example.

```
Model
  Equation
    Name = Natural Convection
    Apply to Bodies = 1
    Heat Equation
```

```
      Active = on
      Convection = Computed
      Edit Solver Setting
        Nonlinear System
          Max. iterations = 1
    Navier-Stokes
      Active = on
      Edit Solver Setting
        Nonlinear System
          Max. iterations = 1
    Add
    OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the mass. Other properties assume a physical law, such as conductivities and viscosity.

Here we choose water at room temperature from the material library. You may click trough the material parameters of the various solvers to ensure that the properties are indeed as they should be. Any consistant set of units may be used in Elmer. The natural choice is of course to perform the computations in SI units.

Apart from the properties from the material database, we reference temperature for the Boussinesq approximation.

```
Model
  Material
    Material library
      Water (room temperature)
    General
      Reference Temperature = 293
    Apply to Bodies = 1
    Add
    OK
```

A Body Force represents the right-hand-side of a equation. It is generally not a required field for a body. In this case, however, we apply the buoyancy resulting from heat expansion as a body force to the Navier-Stokes equation.

```
Model
  Body Force
    Name = Buoyancy
    Apply to Bodies = 1
    Navier-Stokes
      Boussinesq = on
    Add
    OK
```

Initial conditions should be given to transient cases. In this case we choose a constant Temperature field and an small initial velocity that initializes the symmetry break.

```
Model
  Initial Condition
    Name = Initial Guess
    Heat Equation
      Temperature = 293
    Navier-Stokes
      Velocity 1 = 1.0e-9
      Velocity 2 = 0.0
```

Only one boundary condition may be applied to each boundary and therefore all the different physical BCs for a boundary should be grouped together. In this case the Temperature and Velocity. The side walls are assumed to be adiabatic.

```
Model
  BoundaryCondition
    Name = Bottom
    Heat Equation
      Temperature = 293.5
    Navier-Stokes
      Velocity 1 = 0.0
      Velocity 2 = 0.0
    Add
    New

    Name = Top
    Heat Equation
      Temperature = 293
    Navier-Stokes
      Velocity 1 = 0.0
      Velocity 2 = 0.0
    Add
    New

    Name = Sides
    Navier-Stokes
      Velocity 1 = 0.0
      Velocity 2 = 0.0
    Add
```

The conditions may also be assigned to boundaries in the Boundary condition menu, or by clicking with the mouse. Here we use the latter approach as that spares us of the need to know the indexes of each boundary.

```
Model
  Set boundary properties
    Choose Bottom -> set boundary condition Bottom
    Choose Top -> set boundary condition Top
    Choose Sides -> set boundary condition Sides
```

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. The project includes all the files needed to restart the case.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration.

When there are some results to view we may start the postprocessor also

```
Run
  Start postprocessor
```

Results

Due to the number of the time-steps the simulation may take around ten minutes. You may inspect the results with ElmerPost as the time-steps are computed, or wait until all timesteps have been computed. When opening the result file using ElmerGUI ElmerPost only opens the first time-step. Therefore it is important to reopen the file and load the time-steps of interest. Pressing the button `All` selects all the calculated time steps. A video of the results can be viewed by selecting the option `Timestep Control` and pressing the button `Loop` under the `Edit` menu.

In Figures 9.2 and 9.3 the obtained temperature distribution and the velocity vectors are presented. The maximum velocity in the system should be about 0.516 mm/s.

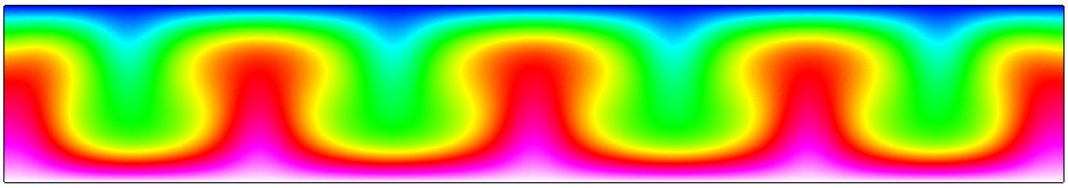

Figure 9.2: Temperature distribution at 260 s.

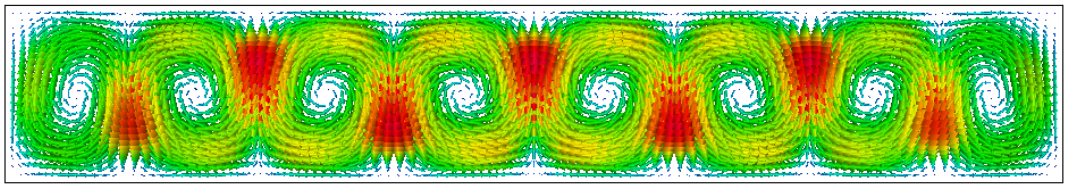

Figure 9.3: Velocity vectors at 260 s.

Extra task: Sensitivity to temperature difference

If you have time you may try to solve the case with different parameters. Changing the temperature difference is one way of affecting the instability of the system. Decreasing the tempereture differences the system eventually becomes steady state and the convection rolls vanish alltogether. Increasing the temperature difference may increase the number of convection rolls and eventually the system becomes fully chaotic. Note that changing the temperature difference also affects to the time scale of the wake.

Tutorial 10

Temperature distribution of a toy glacier

Directory: ToyGlacierTemperature
Solvers: HeatSolver
Tools: ElmerGUI,nglib
Dimensions: 2D, Steady-state

Introduction

The purpose of this simple tutorial is to be an introduction into Elmer for people dealing with computational glaceology. This tutorial shows how to apply one equation and related boundary conditions to just one domain.

Problem description

Consider a 2D toy model of a glacier with length of 7000 m and thickness of about 1000 m. There is a slight declination in the geometry which will make the glacier flow to the left. The left-hand-side is rounded to imitate a true glacier while the right-hand-side is cut off.

We solve for the temperature distribution T of the glacier. A heat flux of $q = 0.02$ W/m^2 is applied at the bottom of the glacier while the surface stays at a fixed temperature of $T_0 = -10$ C. The material properties of ice are used for the heat conductivity $\kappa(T)$. The temperature distribution in the glacier may be solved from

$$\begin{cases} -\kappa\Delta T &= 0 & \text{in } \Omega \\ T &= T_0 & \text{on } \Gamma_D \\ \kappa\frac{\partial T}{\partial n} &= q & \text{on } \Gamma_N \end{cases} \tag{10.1}$$

Starting and meshing

Start `ElmerGUI` from command line or by clicking the icon in your desktop (or in the /bin directory of you installation). Here we describe the essential steps in the ElmerGUI by writing out the clicking procedure. Tabulation generally means that the selections are done within the window chosen at the higher level.

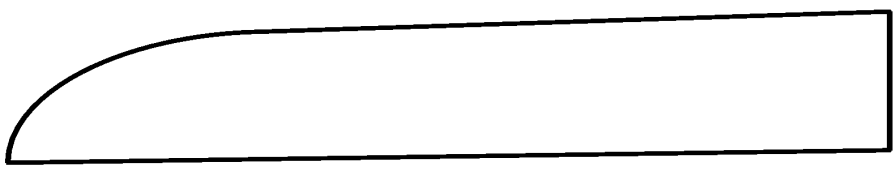

Figure 10.1: The shape of the toy glacier to be studied

The mesh is given in ElmerGrid format in file `glacier_toy.in2d` in the samples directory of ElmerGUI, load this file.

```
File
  Open -> glacier_toy.in2d
```

You should obtain a mesh consisting of just two triangular elements. The mesh is created by the `netgen` plugin and in order to increase the mesh density the in-line parameters of netgen must be defined in ElmerGUI. Here we set the maximum element size to 50.

```
Mesh
  Configure...
    nglib -> Max H: 50
```

The resulting mesh should consist of 3335 nodes and 6355 triangles as may be checked in the `Model summary` window.

Command file definition

After we have the mesh we start to go through the Model menu from the top to bottom. In the `Setup` we choose things related to the whole simulation such as file names, time stepping, constants etc. The simulation is carried out in 2-dimensional cartesian coordinates and in steady-state. Only one steady-state iteration is needed as the case is linear.

```
Model
  Setup
    Simulation Type = Steady state
    Steady state max. iter = 1
```

Choose `Accept` to close the window.

In the equation section we choose the relevant equations and parameters related to their solution. In this case we'll have one set only one equation – the heat equation.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we have just one body and one boundary and therefore its easier to assign the Equation and Material to it directly.

For the linear system solvers we are happy to use the defaults. One may however, try out different preconditioners (ILU1,...) or direct Umfpack solver, for example.

```
Model
  Equation
    Add
      Name = Heat Equation
      Apply to bodies = 1
      Heat Equation
        Active = on
  Apply
  OK
```

The Material section includes all the material parameters. They are divided to generic parameters which are direct properties of the material without making any assumptions on the physical model, such as the mass. Other properties assume a physical law, such heat conductivity. We choose ice from the Material library which automatically sets for the needed material properties.

```
Model
  Material
    Add
      Material library
        Water (frozen)
```

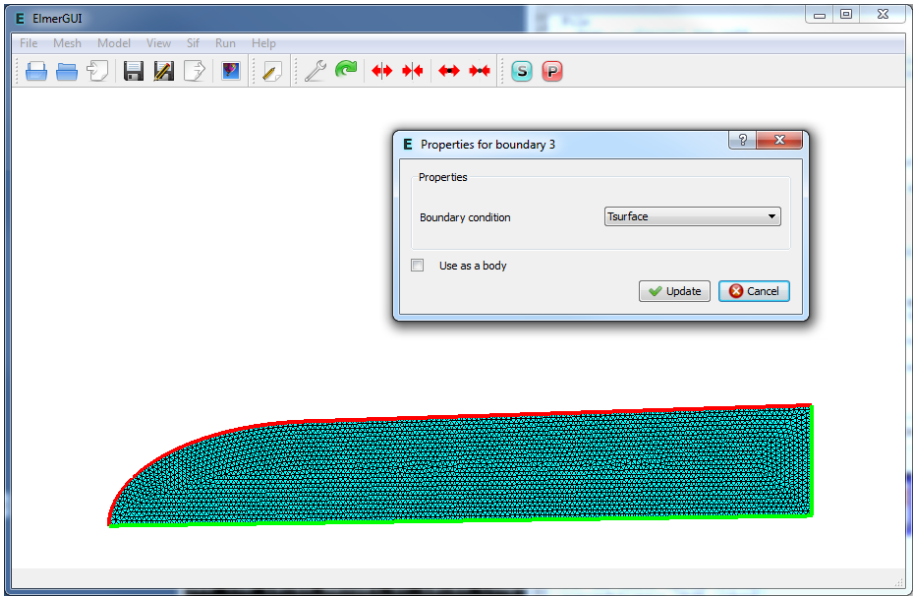

Figure 10.2: Defining boundary conditions in ElmerGUI session

```
    Apply to bodies = Body 1
    Add
    OK
```

This includes, for example, temperature dependent heat conductivity as may be seen under the HeatEquation page of the material properties. MATC language is used here to define the functional form.

A Body Force represents the right-hand-side of a equation that in this case represents the heat source. In this case there are no internal heat sources so we do not need one. Also no Initial Condition is required in steady state case.

We set three different kinds of boundary conditions. A fixed temperature, a fixed flux and natural boundary condition (zero flux). As there are several boundaries we first define the different boundary types, and thereafter assign them using the mouse. A screenshot of the case when setting the BCs is shown in figure 10.2.

```
Model
  BoundaryCondition
    Add
      Heat Equation
        Temperature = -10.0
      Name = Tsurface
      OK
    Add
      Heat Equation
        Heat Flux = 0.02
      Name = Tflux
      OK
    Add
      Name = Tnatural
      OK
```

Then we set the boundary properties

```
Model
```

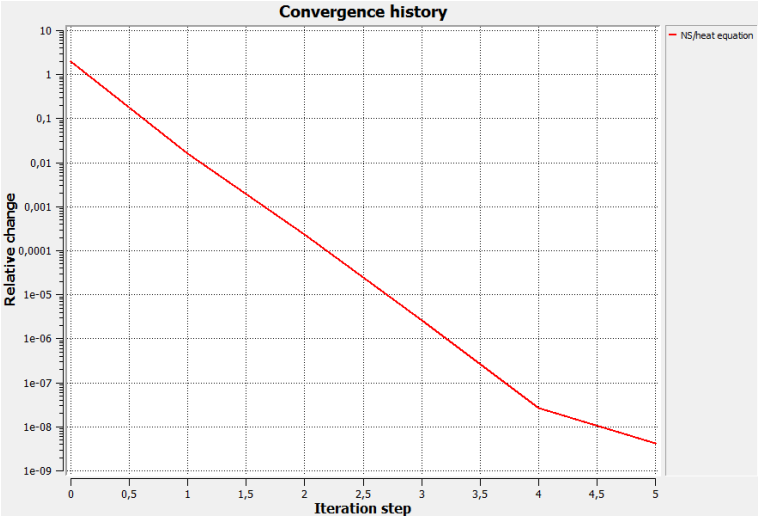

Figure 10.3: The convergence ElmerGUI

```
  Set boundary properties
```

Choose the correct boundary by clicking with the mouse and apply the condition for this boundary.

```
Boundary condition
  Click top boundary -> choose Tsurface
  Click bottom boundary -> choose Tflux
  Click r.h.s. boundary -> choose Tnatural
```

Saving and solution

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. In saving the project all the necessary files for restarting the case will be saved to the destination directory.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The heat conductivity of ice is set to be dependent on temperature and this results to a nonlinear iteration. The resulting output is shown in figure 10.3.

Note: if you face problems in the solution phase and need to edit the setting, always remember to save the project before execution.

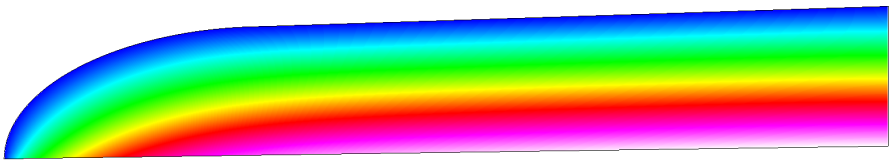

Figure 10.4: The temperature distribution of the toy glacier as visualized ElmerPost

Results

To view the results we may start the the internal VTK widget, or use ElmerPost for postprocessing as is done here

```
Run
  Start postprocessor
```

The default configuration shows the surface mesh colored with temperature as in picture 10.4. You may test the different options. For example, choosing in the `Color Mesh Edit` window the `Mesh Style` to be `Line` and `Edge Style` to be All and pressing `Apply` the wireframe of the computational mesh is seen.

The maximum temperature obtained with the above choices is 0.11166 C. With a denser mesh the result is naturally more accurate, but solving the problem takes more calculation time.

You may study the effect of mesh refinement by choosing a different value for the `Max H` parameter. under `Configure`. After choosing `Remesh` and saving the mesh the solver may be recalled with the modified mesh.

Transient simulation

We use the steady-state simulation presented above as our starting point and solve a transient version of it. Initially the glacier is assumed to be at -10 C and it is gradually heated from below.

We use 2nd order bdf time-stepping method is selected with 100 steps and with step size of 100 years - melting the ice from below with such a small flux would take quite a few years. The mathematical expression followed by "$" is evaluated when the command file is read. Here it is used to transform the time in years to those in one second.

```
Model
  Setup
    Simulation Type = Transient
    Time Stepping Method = bdf
    BDF Order = 2
    Time Step Intervals = 100
    Time Step Sizes = $ 3600 * 24 * 365.25 * 100
    Gravity = ...
```

Initial conditions should be given to transient cases. In this case we choose a constant Temperature of -10 C. This is consistant with the boundary condition at the top wall.

```
Model
  Initial Condition
    Name = Initial Guess
    Heat Equation
      Temperature = -10.0
```

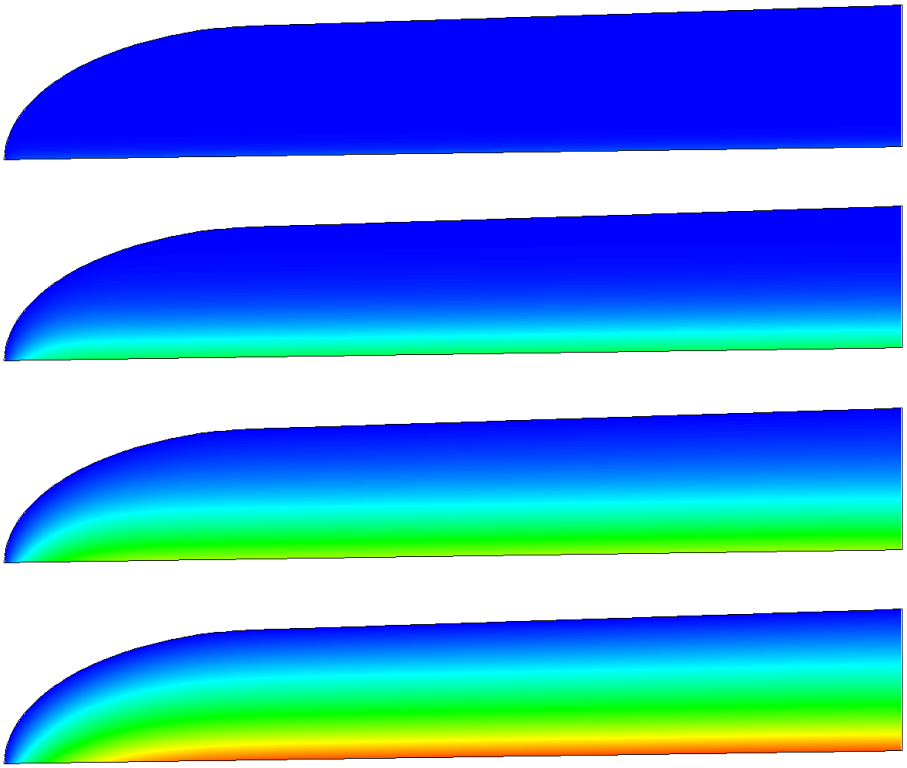

Figure 10.5: Temperature distribution after 1, 20, 50 and 100 timesteps. The temperature scale is the same that is used in the steady-state case. The maximum temperature at end should be about -3.7611 C.

Tutorial 11

Temperature and velocity distributions of a toy glacier and bedrock

Directory: ToyGlacierTemperatureAndFlow
Solvers: HeatSolver,FlowSolver
Tools: ElmerGUI,nglib
Dimensions: 2D, Steady-state

Introduction

The purpose of this simple tutorial is to be an introduction into Elmer for people dealing with computational glaceology. The tutorial is a continuation of the previous one with slightly more complex geometry. This tutorial shows how to apply two different solvers to two different bodies.

Problem description

Consider a 2D toy model of a glacier sitting an a piece of bedrock. With the bedrock present it is possible to study more accurately the temperature profiles.

Now we solve for the temperature distribution T for the combined system. A heat flux of $q = 0.02$ W/m^2 is applied at the bottom of the bedrock while the surface stays at a fixed temperature of $T_0 = -10$ C. For ice the properties from the database are assumed while for the bedrock density is assumed to be 2500 kg/m^3 and heat conductivity 3 W/mK.

We also solve for the velocity distribution $\vec{v}$ of the glacier. The velocity field is solved from the Stokes equation and is assumed to be affected only by the Gravity. As boundary conditions we apply a no-slip condition at the ice-rock interface and symmetry condition at the right-hand-side of the glacier.

Starting and meshing

Start `ElmerGUI` from command line or by clicking the icon in your desktop (or in the /bin directory of you installation). Here we describe the essential steps in the ElmerGUI by writing out the clicking procedure. Tabulation generally means that the selections are done within the window chosen at the higher level.

The mesh is given in ElmerGrid format in file `glacier_on_bedrock_toy.in2d` in the samples directory of ElmerGUI, load this file.

```
File
  Open -> glacier_on_bedrock_toy.in2d
```

When netgen is ready with the meshing. You should obtain a mesh consisting of 4329 nodes and 8406 triangular elements. Now the mesh density was predefined in the `in2d` file and therefore no command-line

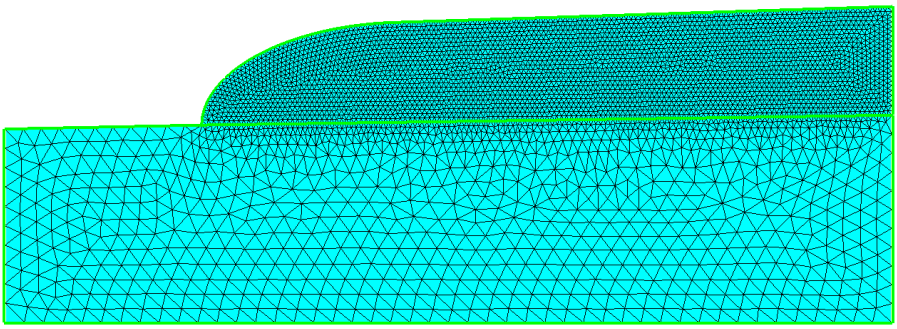

Figure 11.1: The finite element mesh in ElmerGUI

arguments are needed to refine the mesh. If you got more elements check the value of `Max H` in the netgen parameter window.

If the mesh was successfully imported your window should look something in figure 11.1.

Command file definition

After we have the mesh we start to go through the Model menu from the top to bottom. Again we are happy with the definitions in the `Setup` window.

In the Equation section we choose the relevant equations and parameters related to their solution. In this case we'll have two different sets of solvers (called as Equation in Elmer slang). The first consists of heat and flow solvers, while the other includes just the heat solver. We'll name them appropriately.

When defining Equations and Materials it is possible to assign the to bodies immediately, or to use mouse selection to assign them later. In this case we know that the fluid body has the index 1 and the solid body has the index 2. Therefore it is easy to assign the Equation and Material to the bodies directly.

Here we neglect the effect of convection to the temperature distribution. Therefore there is no coupling from velocity field to energy equation. However, the temperature is affecting the viscosity of ice and therefore it needs to be solved first. This is ensured by giving higher priority to the heat solver (default is zero). In order to obtain the Stokes equation convection of momentum is omitted from the Navier-Stokes equations.

Here we are quite happy with the default solver settings of the individual equations of the linear systems. However, the user may play around with different linear system settings to study their effect on convergence and computation time.
The equation for the ice

```
Model
  Equation
    Add
    Name = Heat and Flow
    Apply to Bodies = 1
    Heat Equation
      Active = on
      Priority = 1
      Convection = None
    Navier-Stokes
      Active = on
      Convect = off
    OK
```

and then for the solid

```
Model
```

```
  Equation
    Add
    Name = Just Heat
    Apply to Bodies = 2
    Heat Equation
      Active = on
      Convection = None
    OK
```

The Material section includes the material parameters. We choose ice from the Material library which automatically sets for the needed material properties. For the bedrock we define the two parameters that are required.

```
Model
  Material
    Add
      Material library
        Water (frozen)
      Apply to bodies = Body 1
      Add
      OK
    Add
      Name = Bedrock
      Apply to bodies = Body 2
      General
        Density = 2500.0
      Heat Equation
        Heat Conductivity = 3.0
      Add
      OK
```

A Body Force represents the right-hand-side of a equations. For the heat equation there are no source terms. For the Stokes equation we apply gravity which points to the negative y direction.

```
Model
  BodyForce
    Name = Gravity
    Navier-Stokes
      Force 2 = -9.81
    Apply to Bodies = Body 1
    Add
    OK
```

We do not need any Initial Condition since the zero temperature (in Celcius) is a good initial guess for the heat equation. For the Stokes equation a better solution could be used since if convergence problems would arise since the non-newtonian material laws do actually depend on the initial velocity.

We set four different kinds of boundary conditions. A fixed temperature, a fixed flux, no-slip condition and symmetry condition. As there are several boundaries we first define the different boundary types, and thereafter assign them using the mouse.

```
Model
  BoundaryCondition
    Add
      Heat Equation
        Temperature = -10.0
      Name = Tsurface
      OK
```

```
  Add
    Heat Equation
      Heat Flux = 0.02
    Name = Tflux
    OK
  Add
    Navier-Stokes
      No-slip Wall BC = on
    Name = NoSlip
    OK
  Add
    Navier-Stokes
      Velocity 1 = 0.0
    Name = Symmetry
    OK
```

Then we set the boundary properties

```
Model
  Set boundary properties
```

Choose the correct boundary by clicking with the mouse and apply the condition for this boundary.

```
Boundary condition
  Click top boundary of ice -> choose Tsurface
  Click the bare part of bedrock -> choose Tsurface
  Click bottom boundary of bedrock -> choose Tflux
  Click bottom boundary of ice -> choose NoSlip
  Click r.h.s. boundary of ice -> choose Symmetry
```

Saving and solution

For the execution ElmerSolver needs the mesh files and the command file. We have know basically defined all the information for ElmerGUI to write the command file. After writing it we may also visually inspect the command file.

```
Sif
  Generate
  Edit -> look how your command file came out
```

Before we can execute the solver we should save the files in a directory. In saving the project all the necessary files for restarting the case will be saved to the destination directory.

```
File
  Save Project
```

After we have successfully saved the files we may start the solver

```
Run
  Start solver
```

A convergence view automatically pops up showing relative changes of each iteration. The heat conductivity of ice is set to be dependent on temperature and this results to a nonlinear iteration. The resulting output is shown in figure 11.2.

Note: if you face problems in the solution phase and need to edit the setting, always remember to save the project before execution.

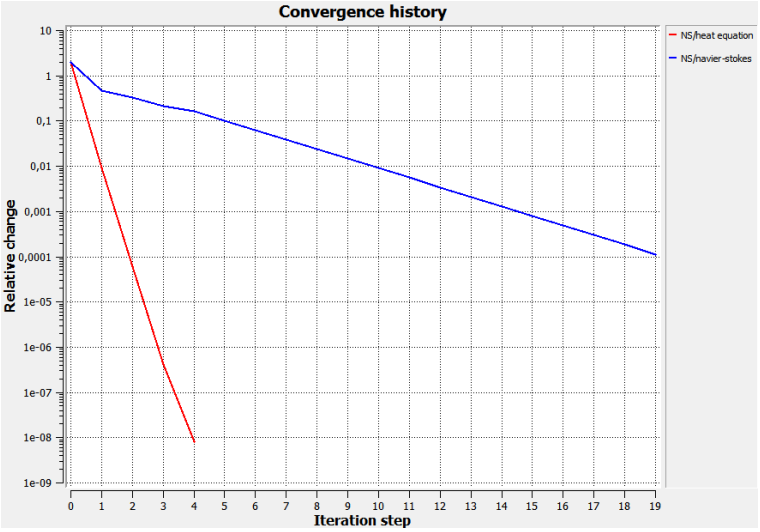

Figure 11.2: The convergence ElmerGUI

Results

To view the results we may start the the internal VTK widget, or use ElmerPost for postprocessing as is done here

```
Run
  Start postprocessor
```

The resulting temperature and velocity distributions are shown in figure 11.3.

The maximum temperature obtained with the above choices is 12.955 C. For the velocity the maximum absolute value is 6.3337 mm/s which is actually unreasonably high since it corresponds to yearly movement of around 200 km. This just shows that the shape of the toy glacier under study is very unrealistic.

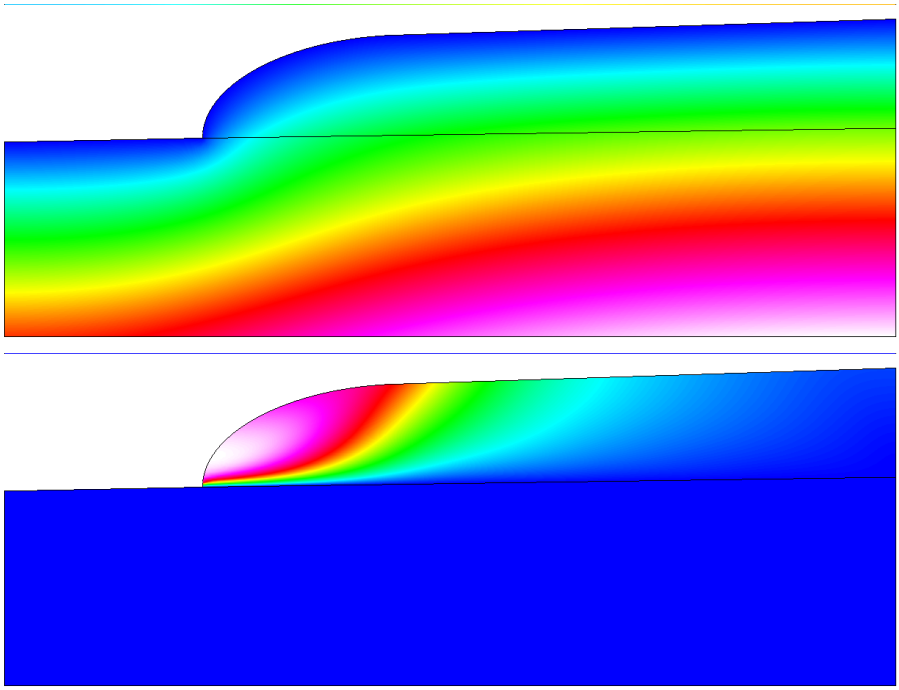

Figure 11.3: Temperature (upper figure) and velocity (lower figure) distributions of the toy glacier sitting on a bedrock as visualized ElmerPost

Part II

non-GUI Problems

Tutorial 12

Radiation heat transfer

Directory: TemperatureRadiation
Solvers: HeatSolve
Tools: ElmerGrid, editor
Dimensions: 2D, Axi-Symmetric

Case definition

At high temperature the radiation heat transfer between walls is often the dominating heat transfer mechanism. In this tutorial we look how radiation heat transfer between cocentric cylinders is modeled.

Solution Procedure

The problem is a pure heat transfer problem that may be solved with `HeatSolve`. The view and Gebharht factors associated with the radiation are solved as a first step in solving the equations. Thereafter the nonlinear heat equation is solved until convergence is reached.

The computatonal mesh is done with `ElmerGrid` in directory `radiation` with the command

```
ElmerGrid 1 2 radiation
```

The directory is given in the header of the command file

```
Header
  Mesh DB "." "radiation"
End
```

The only constant required is the Stefan-Boltzmann constant that gives the relationship between temperature and radiation power

```
Constants
  Stefan Boltzmann = 5.67e-8
End
```

The geometry is axisymmetric and the case is solved in steady state. As there is only one equation only 1 iteration for the system is required.

```
Simulation
  Coordinate System = Axi Symmetric
  Simulation Type = Steady State
  Steady State Max Iterations = 1
  Output Intervals = 1
  Output File = "radiation.result"
  Post File = "radiation.ep"
End
```

There are two bodies with the same equation but different properties.

```
Body 1
  Equation = 1
  Body Force = 1
  Material = 1
  Initial Condition = 1
End

Body 2
  Equation = 1
  Material = 2
  Initial Condition = 1
End
```

The nonlinear equation requires realistic initial conditions. Otherwise convergence may not be obtained.

```
Initial Condition 1
  Temperature = 250.0
End
```

The body force is the heating power in units W/kg.

```
Body Force 1
  Heat Source = 10000
End
```

The material properties differ only in heat conductivity. Heat capacity is not actually needed since the case is not transient.ă

```
Material 1
   Density = 1.0
   Heat Conductivity = 10.0
   Heat Capacity = 1.0
End

Material 2
   Density = 1.0
   Heat Conductivity = 1.0
   Heat Capacity = 1.0
End
```

The heat equation is solved with an itrative procedure that requires some relaxation for better convergence. There are two different ways to discretize the radiation. There are two keywords defining when to switch to the true Newtonian iteration which should give better convergence.

```
Solver 1
  Equation = Heat Equation
  Stabilize = True
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Convergence Tolerance = 1.0e-12
  Linear System Max Iterations = 500
  Linear System Preconditioning = ILU
  Nonlinear System Newton After Iterations = 1
  Nonlinear System Newton After Tolerance = 1.0e-4
  Nonlinear System Max Iterations = 50
  NonLinear System Convergence Tolerance = 1.0e-8
```

```
  Steady State Convergence Tolerance = 1.0e-8
  Nonlinear System Relaxation Factor = 0.7
End
```

The only solver is the heat equation.

```
Equation 1
  Active Solvers = 1
End
```

The radiation boundary conditions are set for two different boundaries. The first one is for the internal object and the second one for the insulation. The normal direction of the surfaces is important since a wrong direction may result to badly set problem for the view factor computation. Internal and external surfaces are very different. The normal direction may be switched with the keyword `Radiation Target Body`. A good sign of properly set case is that the view factors add up to about one.

```
Boundary Condition 1
   Target Boundaries = 1
   Heat Flux BC = True
   Radiation = Diffuse Gray
   Radiation Target Body = -1
   Emissivity = 0.6
End

Boundary Condition 2
   Target Boundaries = 2
   Heat Flux BC = True
   Radiation = Diffuse Gray
   Radiation Target Body = -1
   Emissivity = 0.1
End
```

The third boundary condition is the Dirichtlet condition for the extranal boundary. Dirichtlet conditions boost up the convergence even though the heat equation is basically well defined also with external radiation conditions.

```
Boundary Condition 3
   Target Boundaries = 3
   Temperature = 100.0
End
```

Results

With the given computational mesh the problem is solved in around 30 seconds. With 1 231 second order 9-noded rectangular elemenets the maximum temperature is 565.7 K. The corresponding results are shown in Fig. 12.1.

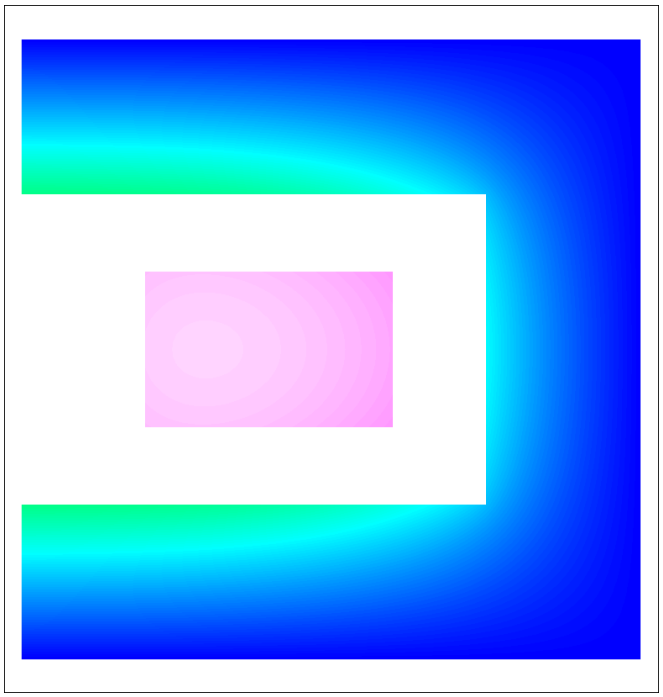

Figure 12.1: Temperature distribution in the radiation heat transfer problem

Tutorial 13

Eigenvalue analysis of an elastic beam

Directory: ElasticEigenValues
Solvers: StressSolve, EigenSolve
Tools: ElmerGrid,Editor
Dimensions: 3D, Steady-state

Case definition

A homogenous, elastic silicon beam of dimensions 1 m length, 0.1 m height and 0.2 m width is supported on its both ends (boundaries 1 and 2). A beam has the density 2330 kg/m^3, Poisson ratio 0.3 and Young's modulus 10^{11} N/m^2. The problem is to calculate the eigenvalues of the beam. Mathematically the equation to be solved is

$$-\rho\omega^2\phi = \nabla \cdot \tau(\phi)$$

where ρ is the density, ω^2 is the eigenvalue, ω is the angular frequency, ϕ is the corresponding vibration mode and τ is the stress tensor.

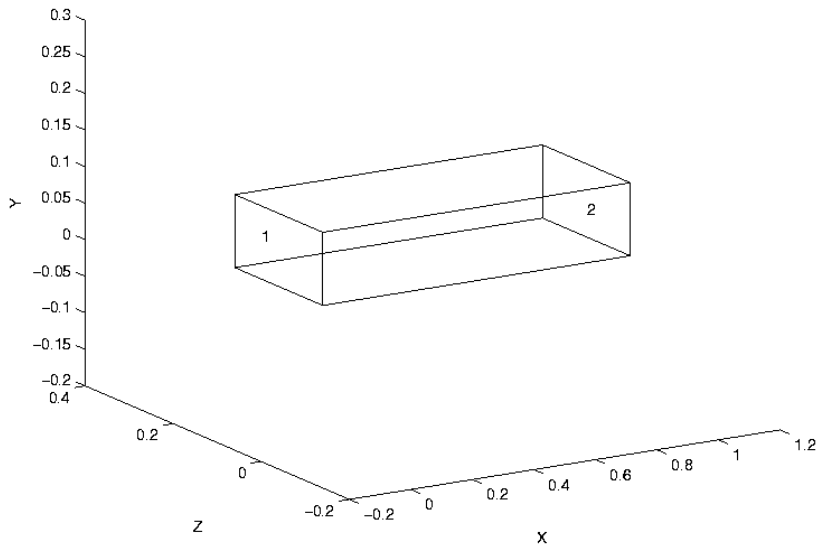

Figure 13.1: Beam.

Solution procedure

The mesh has been created by using Gambit software and it consists of 2500 elements. The mesh can be converted to Elmer format with ElmerGrid with the command

```
ElmerGrid 7 2 mesh.FDNEUT
```

This command creates the directory which contains the Elmer mesh files.

```
Header
  Mesh DB "." "mesh"
  Include Path ""
  Results Directory ""
End
```

A steady-state three-dimensional analysis is defined in the simulation section.

```
Simulation
  Coordinate System = "Cartesian 3D"
  Coordinate Mapping(3) = 1 2 3
  Simulation Type = "Steady State"
  Steady State Max Iterations = 1
  Solver Input File = "eigen_values.sif"
  Output File = "eigen_values.dat"
  Post File = "eigen_values.ep"
End
```

The geometry of the problem is simple and it includes only one body and material.

```
Body 1
  Equation = 1
  Material = 1
End

Material 1
  Youngs Modulus = 100e9
  Poisson Ratio = 0.3
  Density = 2330
End
```

The problem is solved according to linear elastic theory and due to that stress analysis is set to true.

```
Equation 1
  Stress Analysis = True
End
```

In the solver section `Stress Analysis` is selected. In addition, the value of the keyword `Eigen Analysis` have to set to true. The keyword `Eigen System Values` defines the number of the computed eigenvalues. The problem also is possible to solve with iterative solver but we have used direct solver in this time.

```
Solver 1
  Equation = "Stress Analysis"
  Eigen Analysis = Logical True
  Eigen System Values = Integer 5
  Linear System Solver = "direct"
```

```
  Variable = "Displacement"
  Variable Dofs = 3
  Linear System Iterative Method = "BiCGStab"
  Linear System Max Iterations = 1000
  Linear System Convergence Tolerance = 1.0e-08
  Linear System Abort Not Converged = True
  Linear System Preconditioning = "ILU0"
  Linear System Residual Output = 1
  Steady State Convergence Tolerance = 1.0e-05
  Nonlinear System Convergence Tolerance = 1.0e-05
  Nonlinear System Max Iterations = 1
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-02
  Nonlinear System Relaxation Factor = 1
  Linear System Precondition Recompute = 1
End
```

The beam is supported on its both ends and therefore displacements is set to zero in all the directions.

```
Boundary Condition 1
  Target Boundaries(1) = 1
  Displacement 1 = 0
  Displacement 2 = 0
  Displacement 3 = 0
End

Boundary Condition 2
  Target Boundaries(1) = 2
  Displacement 1 = 0
  Displacement 2 = 0
  Displacement 3 = 0
End
```

After that, the problem is ready to solve.

An anisotropic model

The same problem can also be solved as an anisotropic problem which causes a couple of changes in the sif-file. First, it is reasonable to rename the files in the simulation section

```
Solver Input File = "eigen_values_aniso.sif"
Output File = "eigen_values_aniso.dat"
Post File = "eigen_values_aniso.ep"
```

For anisotropic material Young's modulus have to redefine as a matrix. In this case the matrix is defined as follows

```
Youngs Modulus
Size 6 6
    Real  200e9  60e9   60e9   0     0     0
          60e9   200e9  200e9  0     0     0
          60e9   60e9   200e9  0     0     0
          0      0      0      80e9  0     0
          0      0      0      0     80e9  0
          0      0      0      0     0     80e9
    End
```

No more changes are needed in the sif-file.

Results

Both the eigenvalues of the isotropic and the eigenvalues of the anisotropic model are shown below in Elmer outputs. Figure 13.2 presents the computed eigenvectors of the beam with the isotropic model. The formula $\omega = 2\pi f$ have been used in calculating frequencies (f) (Table 13.1). According to the results the anisotropic model yielded greater eigenvalues with these values of Young's modulus.

```
EigenSolve: Computed Eigen Values:
EigenSolve: -------------------------------
EigenSolve:            1        (16737546.4275755,0.00000000000000D+000)
EigenSolve:            2        (48175589.4544061,0.00000000000000D+000)
EigenSolve:            3        (99674749.0526558,0.00000000000000D+000)
EigenSolve:            4        (110392974.959463,0.00000000000000D+000)
EigenSolve:            5        (253947166.278411,0.00000000000000D+000)
```

Isotropic model.

```
EigenSolve: Computed Eigen Values:
EigenSolve: -------------------------------
EigenSolve:            1        (29608629.8775828,0.00000000000000D+000)
EigenSolve:            2        (88782964.0905879,0.00000000000000D+000)
EigenSolve:            3        (198583949.415515,0.00000000000000D+000)
EigenSolve:            4        (205085884.544046,0.00000000000000D+000)
EigenSolve:            5        (480903841.387323,0.00000000000000D+000)
```

Anisotropic model.

Table 13.1: Computed frequencies.

step	isotropic	anisotropic
1	651.127 Hz	866.023 Hz
2	1104.673 Hz	1499.633 Hz
3	1588.959 Hz	2242.809 Hz
4	1672.210 Hz	2279.229 Hz
5	2536.249 Hz	3490.191 Hz

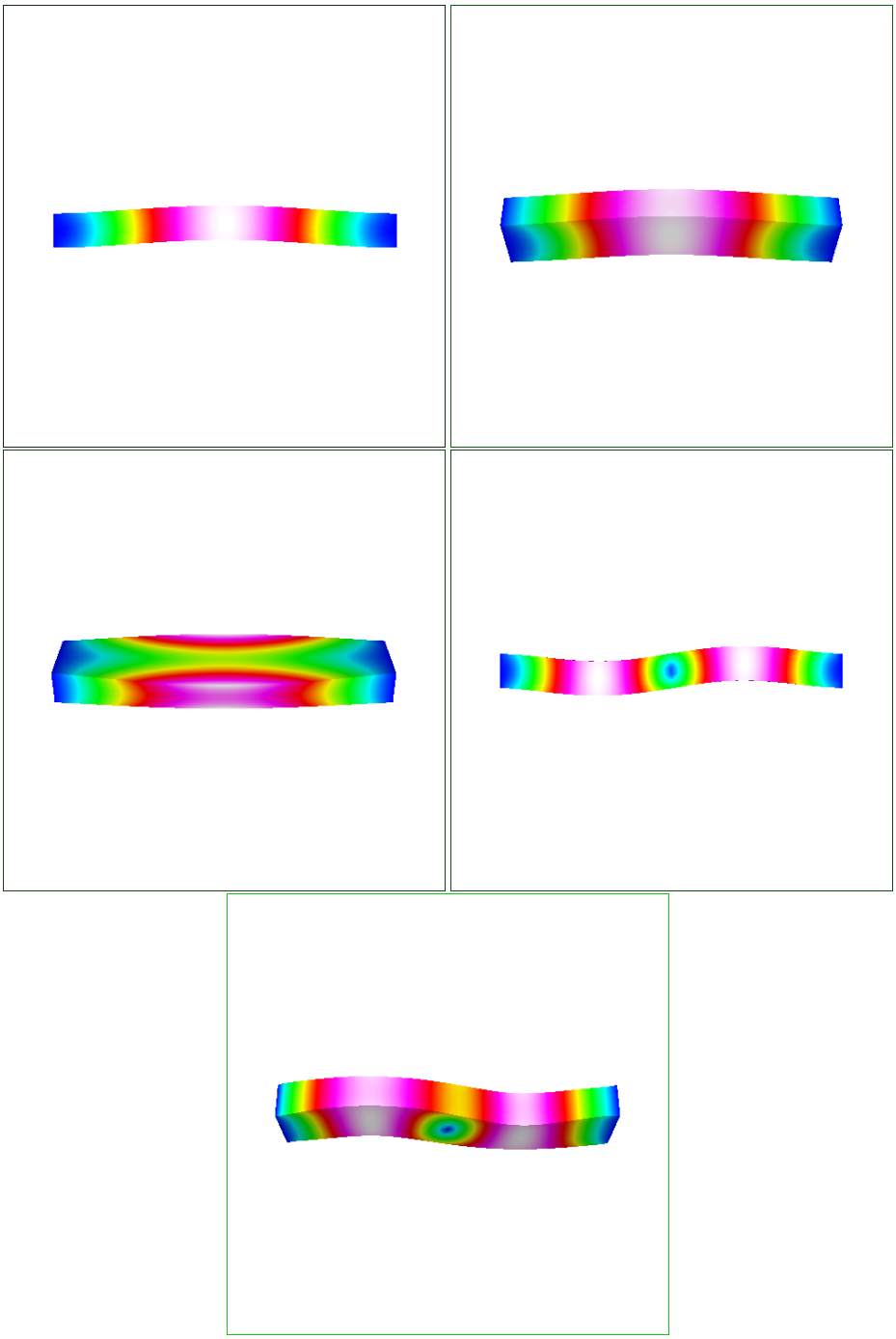

Figure 13.2: Eigenvectors

Tutorial 14

Elastic linear plate

Directory: ElasticPlateLinear
Solvers: SmitcSolver
Tools: ElmerGrid, editor
Dimensions: 2D

Case definition

This tutorial demonstrates how to use the Smitc solver to solve small deflections of plates. The Smitc solver is for elastic linear plates and uses the theory of Reissner and Mindlin.

The case under investigation is a L-shaped steel plate under pressure. The plate is shown in figure 14.1 The longer sides have the length of $2\,m$ and the shorter $1\,m$. So the area of the plate is $3\,m^2$. The plate has a thickness of $1\,cm$. We assume that on the plate there is about $15300\,kg$ of sand. The sand is uniformly distributed on the plate and the sand stays uniformly distributed even if the plate undergoes small deflection. The sand exerts to the plate a pressure of $50000\,Pa$. The plate is clamped from all sides meaning that both deflection and rotation are zero on all edges.

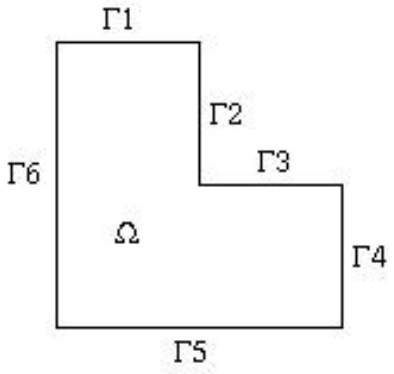

Figure 14.1: The geometry of plate and the numbering of edges.

Solution Procedure

The first thing to do is create a mesh with ElmerGrid. The definition of mesh is in the file `simple_plate.grd`. The mesh is about uniform and consist of 1000 linear square elements. The mesh is created with command

```
ElmerGrid 1 2 simple_plate
```

One thousand element should easily do the trick in this case but if more elements is needed you can edit the file `simple_plate.grd`. More specifically the line

```
Surface Elements = 1000
```

The solver input file `simple_plate.sif` starts with turning on the warnings and the definition of the proper mesh directory.

```
check keywords warn

Header
  Mesh DB "." "simple_plate"
End
```

The simulation uses 2D cartesian geometry. The simulation is not time dependent i.e. Steady State. There is no coupled solvers so only one iteration is needed. The output interval is one meaning all intervals (now there is only one interval). Numerical results are written to file `simple_plate.result` and ElmerPost file is `simple_plate.ep`.

```
Simulation
  Coordinate System = Cartesian 2D
  Simulation Type = Steady State
  Steady State Max Iterations = 1
  Output Intervals = 1
  Output File = "simple_plate.result"
  Post File = "simple_plate.ep"
End
```

There is just one body, the plate, and it uses Equation and Body Force 1 and is of Material 1.

```
Body 1
  Equation = 1
  Body Force = 1
  Material = 1
End
```

The equation block is now more than easy. It only states that we use Solver 1 to solve the equation.

```
Equation 1
  Active Solvers(1) = 1
End
```

In Body Force block we give the equations right hand side. It is the sands pressure and it is the same constant in every point.

```
Body Force 1
  Pressure = 5.0e4
End
```

In Material block we define the plates properties i.e. Poisson ratio, Young's modulus and density. We also give the plates thickness and possible pretension. Now there is no pretension.

```
Material 1
  Poisson ratio = 0.3
  Youngs modulus = 209e9
  Density = 7800.0

  Thickness = 1.0e-2
  Tension = 0.0
End
```

Next the Solver block.

- First we define that we use SmitcSolver and give the name of the subroutine file `Smitc` and subroutine name `SmitcSolver`.
- We name the variable Deflection and state that it has 3 degrees of freedom. First degree is the deflection and the remaining two are actually the components of rotation vector.
- We don't need eigen analysis nor is there any holes in the plate.
- We solve the matrix equation iteratively with stabilized biconjugate gradient method. We precondition the iteration with incomplete LU-factorization.
- Tolerance for the matrix system is $1 \cdot 10^{-8}$ and the tolerance should be achieved in less than 300 iteration.

```
Solver 1
  Equation = "SmitcSolver"
  Procedure = "Smitc" "SmitcSolver"

  Variable = Deflection
  Variable DOFs = 3

  Eigen Analysis = False
  Hole Correction = False

  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Preconditioning = ILU2
  Linear System Convergence Tolerance = 1.0e-8
  Linear System Max Iterations = 300
End
```

Finally we give the boundary conditions. The plate has 6 edges and the edge numbering is in figure 14.1. All the edges are clamped i.e. no deflection (Deflection 1) and no rotation (Deflection 2 and 3).

```
Boundary Condition 1
  Target Boundaries(6) = 1 2 3 4 5 6
  Deflection 1 = 0.0
  Deflection 2 = 0.0
  Deflection 3 = 0.0
End
```

Results

The problem is solved in few seconds and the results are viewed with ElmerPost. It it possible to make ElmerPost to show deflection in 3D. First we determine the number of nodes. Give commands

```
math tmp = size(Deflection.1)
math n = tmp(1)
```

to ElmerPost. Next we put the values of deflection to nodal z-values. Deflection is rather small so the values are scaled by 50.

```
math nodes(2,0:n-1) = 50*Deflection.1
```

Result is shown in figure 14.2.

Deflection.2 and Deflection.3 are the x- and y-components of rotation vector. Values are transformed to vector Rotation with commands

```
math Rotation = 0
math Rotation(0,0:n-1) = Deflection.2
math Rotation(1,0:n-1) = Deflection.3
math Rotation(2,0:n-1) = Deflection.2*0
```

The length of vector is calculated with

```
math AbsRotation = sqrt( vdot(Rotation,Rotation) )
```

Result is shown in figure 14.2.

It is rather cumbersome to write all the commands every time you solve the problem. It is possible to write the commands to file. The file, let us name it `Draw`, would be

```
math tmp = size(Deflection.1);
math n = tmp(1);

math nodes(2,0:n-1) = 50*Deflection.1;

math Rotation=0;
math Rotation(0,0:n-1) = Deflection.2;
math Rotation(1,0:n-1) = Deflection.3;
math Rotation(2,0:n-1) = Deflection.2*0;

math AbsRotation = sqrt( vdot(Rotation,Rotation) );

display;
```

The file is executed in ElmerPost with command `source Draw`.

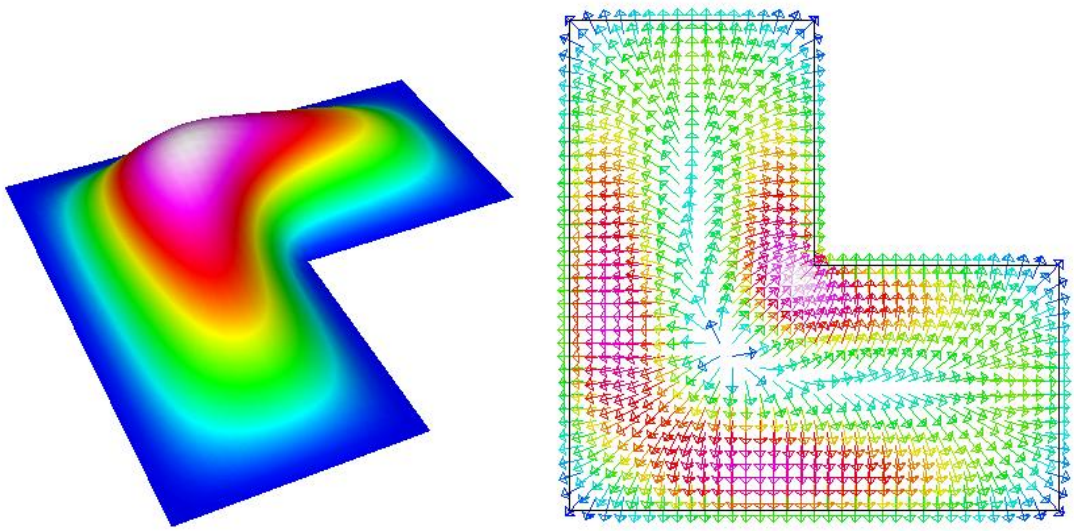

Figure 14.2: The deflection of the plate and the corresponding rotation.

Tutorial 15

Compressible flow passing a step

Directory: FlowStepCompressible
Solvers: FlowSolve, HeatSolve
Tools: ElmerGrid, Editor
Dimensions: 2D, Steady-state

Case definition

This tutorial demonstrates how to simulate the compressible air flow passing a step. The whole step has length of 1.4 m and the height of 0.2 m and the first part of it has length of 0.4 m and the height of 0.1 m (Figure 15.1). The needed material parameters of air are shown in Table 15.1. The model has three sets of boundary conditions. The air flows into the step from the inlet region and withdraws from the outlet region. The other edges of the step compose the third boundary. The flowing air is considered as an ideal gas in this case, and its density ρ depends on the pressure p and temperature T through the equation of state

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

where R is the gas constant.

Table 15.1: Material parameters.

parameter	value
viscosity	16.7e-6 Ns/m^2
heat conductivity	0.026 W/(m·K)
heat capacity	1.01e3 J/(kg·K)
specific heat ratio	1.4
reference pressure	1e5 Pa

Solution procedure

The mesh consists of 500 rectangular elements and it is constructed using ElmerGrid with the following command

```
ElmerGrid 1 2 mesh.grd
```

This command creates the subdirectory `mesh` which contains the Elmer mesh files.

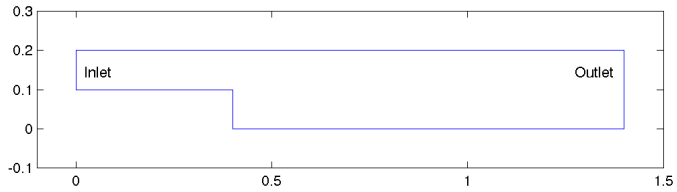

Figure 15.1: Step.

```
Header
  Mesh DB "." "mesh"
  Include Path ""
  Results Directory ""
End
```

The simulation uses 2D cartesian geometry and the problem is solved in steady state using no more than twenty steady state iterations.

```
Simulation
  Coordinate System =  Cartesian 2D
  Coordinate Mapping(3) = 1 2 3
  Simulation Type = Steady
  Steady State Max Iterations = 20
  Solver Input File = "compress_step.sif"
  Post File = "compress_step.ep"
  Output File = "compress_step.dat"
End
```

The solvers are coupled and therefore the convection is computed.

```
Equation 1
  Navier-Stokes = True
  Heat Equation = True
  Convection = "Computed"
End
```

Due to the simplicity of the model only one body is needed.

```
Body 1
  Equation = 1
  Material = 1
  Initial Condition = 1
End
```

Our intention is to model compressible flow and that is why we have to set the value ”Perfect Gas” for the keyword `Compressibility Model`. Furthermore, because perfect gas model has been chosen the settings `Reference Pressure` and `Specific Heat Ratio` must also be given. The Navier-Stokes equation also needs the value of viscosity and the heat equation needs the values of heat capacity and heat conductivity.

```
Material 1
  Compressibility Model = String "Perfect Gas"
  Reference Pressure = 1e5
  Specific Heat Ratio = 1.4
  Viscosity = 16.7e-6
  Heat Conductivity = 0.026
  Heat Capacity = 1.01e3
End
```

For the initial value of temperature we have chosen 300 K.

```
Initial Condition 1
  Temperature = 300
End
```

The Navier-Stokes equation is solved first. Here we give the linear system solver and convergence criterions for linear, nonlinear and steady state solution of the Navier-stokes equation. Note that we are solving for the compressible Navier-stokes equation and that is why a bubble function formulation is used for stabilization of the equation.

```
Solver 1
  Equation = "Navier-Stokes"
  Linear System Solver = "Iterative"
  Linear System Iterative Method = "BiCGStab"
  Linear System Max Iterations = 500
  Linear System Convergence Tolerance = 1.0e-08
  Linear System Abort Not Converged = True
  Linear System Preconditioning = "ILU2"
  Linear System Residual Output = 1
  Steady State Convergence Tolerance = 1.0e-05
  Bubbles = Logical True
  Nonlinear System Convergence Tolerance = 1.0e-05
  Nonlinear System Max Iterations = 1
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-02
  Nonlinear System Relaxation Factor = 1
End
```

The corresponding parameters for the solver of the heat equation are defined in the following solver section.

```
Solver 2
  Equation = "Heat Equation"
  Variable = "Temperature"
  Linear System Solver = "Iterative"
  Linear System Iterative Method = "BiCGStab"
  Linear System Max Iterations = 350
  Linear System Convergence Tolerance = 1.0e-08
  Linear System Preconditioning = "ILU0"
  Linear System Residual Output = 1
  Steady State Convergence Tolerance = 1.0e-05
```

```
  Bubbles = Logical True
  Nonlinear System Convergence Tolerance = 1.0e-05
  Nonlinear System Max Iterations = 1
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-02
  Nonlinear System Relaxation Factor = 1
End
```

Finally, the boundary conditions are specified. There are three sets of boundary conditions, so three `Boundary Condition` sections are needed. The first one is used to prescribe the boundary conditions in the inlet region. Note that we have defined the x-velocity and temperature as a variable of y-coordinate. This is done by setting different values for the x-velocity and temperature (the numerical values of the second column between the words `Real` and `End`) in the different y-points (the numerical values of the first column between words `Real` and `End`) of the inlet region. This kind of procedure prevents occuring singularities in the corner points of the inlet region. In addition, this kind of definition is more realistic than a condition, inwhich the values of the x-velocity and temperature remain the same in the whole inlet region.

```
Boundary Condition 1
  Target Boundaries = 1
  Velocity 1 = Variable Coordinate 2
    Real
      0.1    0
      0.15   0.02
      0.2    0
    End

  Velocity 2 = 0
  Temperature = Variable Coordinate 2
    Real
      0.1    300
      0.15   350
      0.2    300
    End
End
```

After the rest boundary conditions have been defined the problem is ready to solve.

```
Boundary Condition 2
  Target Boundaries = 2
  Velocity 2 = 0
End

Boundary Condition 3
  Target Boundaries = 3
  Velocity 1 = 0
  Velocity 2 = 0
  Temperature = 300
End
```

Results

Figure 15.2 presents the temperature distribution of the step in steady state. The maximum and minimum values of x- and y-velocities are also given as a result and they are shown in Table 15.2.

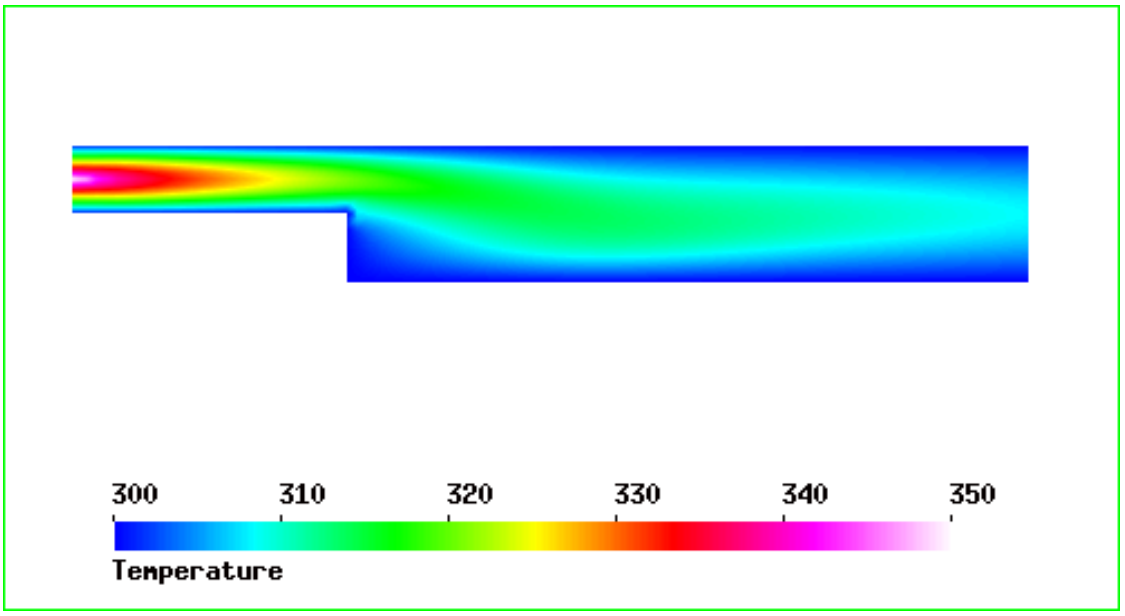

Figure 15.2: Step.

Table 15.2: Computed velocities.

velocity	value
min x-velocity	-0.0014 m/s
min y-velocity	-0.0016 m/s
max y-velocity	0.0008 m/s

Tutorial 16

Flow through a hole – determining the acoustic impedance

Directory: FlowResistance
Solvers: FlowSolve
Tools: ElmerGrid, editor
Dimensions: 3D, Steady-state

Note: This test case is available as consistency tests `FlowResNoslip` and `FlowResSlip`. This may be outdated in parts. For example, it is not necessary to use any special unit system, and also the computation of forces is now more accurate.

Case definition

The problem at hand consists of finding the resistance that a fluid faces as it is forced to flow through a hole. The flow resistance is stated by the ratio of pressure drop over the hole and the input velocity. In microsystem modeling, the hole resistance is often needed to analyse the gas damping effect in perforated structures. Here, the contribution of the holes is homogenised over the perforated structure based on a single hole resistance. For homogenisation in Elmer, the specific acoustic impedance is used to represent the flow resistance. Specific acoustic impedance z_h is defined as

$$z_h = \frac{p}{v} = \frac{F}{vA_h}, \tag{16.1}$$

where F is the net force due to gas on the moving surface, v is the velocity of the gas on the same surface, and A_h is the area of the moving surface. The calculation is best performed in a unit cell of the geometry.

In order to the homogenisation to be possible, the dependence of input velocity and the net force should be linear. Further, there should not be a phase change between these two quantities. These conditions are satisfied when the flow is incompressible. In a linear case, the fluid flow can be treated with the linear form of Navier-Stokes equations called the Stokes equation

$$\rho\frac{\partial\vec{u}}{\partial t} - \nabla\cdot(2\eta\bar{\bar{\varepsilon}}) + \nabla p = \rho\vec{f}, \tag{16.2}$$

where $\vec{u}$ is the unknown velocity field, p is the pressure, η is the viscosity of the fluid, $\rho\vec{f}$ is a body force and $\bar{\bar{\varepsilon}}$ is the linearised strain tensor. Note, that the stationary version of the above equation can be used in homogenisation calculations.

The condition for Stokes equation to apply is that the Reynolds number Re of the problem should be small

$$Re = \frac{\rho UL}{\eta}, \tag{16.3}$$

where ρ is density of the fluid and U and L are, respectively, the velocity and length scales of the problem.

The issue of compressibility is more difficult to answer. A classical condition for the compressibility is that the Mach number Ma of the problem should be small

$$Ma = \frac{U}{a} < 0.3, \tag{16.4}$$

where a is the speed of sound in the gas in operating conditions and the value 0.3 is often stated limit for a small Mach number (actually, the condition is that Ma^2 has to be small). Also the frequency and amplitude of the oscillations of the system have an effect on the validity of the linearity and incompressibility assumptions, since they affect the velocity scale of the problem.

However, also other factors have an effect on the compressibility of the gas. In microsystems, the viscous effects on pressure, or even temperature changes, can affect the density of the gas. A condition for viscous pressure changes is that Ma^2/Re has to be small, and for temperature, in addition, that the Prandtl number Pr may not be too large

$$Pr = \frac{\eta c_p}{k}, \tag{16.5}$$

where c_p is the heat capacity (*ie.* specific heat) in constant pressure and k is the thermal conductivity.

The conditions listed here for the flow to be approximately incompressible are only an overview and the validity of incompressibility assumption should be considered in each case separately. In microsystems, refer for example to the article M. Gad-el-Hak, J. Fluids Eng., 121, 5–33, 1999. Additionally, it is advisable to perform numerical checks on the issue.

One final point on the applicability of the Stokes (or Navier-Stokes) equations is the effect of gas rarefication. If the dimensions of the problem are very small the continuity assumption may not be valid anymore. The importance of the gas rarefication effects are given by the Knudsen number Kn

$$Kn = \frac{\mathcal{L}}{L}, \tag{16.6}$$

where $\mathcal{L}$ is the mean free path of the gas molecules. The mean free path depends inversely on ambient pressure, which has to take into account in stating the Knudsen number. For Knudsen numbers close to and less than 1, slip boundary conditions should be used.

To summarise, the motivation of this tutorial is to perform a linear incompressible simulation of fluid flowing through a hole. The wake for the flow is a constant velocity boundary condition for a boundary before the hole. On the same boundary, the force caused by the fluid is recorded. These two quantities can then be used to determine the specific acoustic impedance of a single hole. The constant velocity boundary condition may be interpreted as describing a moving wall with small displacement. In this particular tutorial, a symmetrical quadrant of a square-shaped hole is used.

Solution procedure

The solution for the problem is found by solving iteratively the Stokes equation. Nonlinear iterations are not needed, since the problem is linear.

The computational mesh should include enough free space after the hole so that any artificial effects due to the boundaries of the mesh are avoided. In this tutorial, the geometry is created and meshed using the ElmerGrid program by the command `elmergrid 1 2 hole.grd`. The default mesh consists of about 12000 nodes and 10500 eight-noded hexahedrons.

The header section of solver input file includes only the location of the mesh files.

```
Header
  Mesh DB "." "hole"
End
```

In the simulation section, a steady-state three-dimensional analysis is defined.

```
Simulation
  Coordinate System = Cartesian 3D
  Simulation Type = Steady State
```

```
  Steady State Max Iterations = 1
  Output File = "flow.result"
  Post File = "flow.ep"
End
```

The geometry contains only one body and no body forces or initial conditions are present. The body section reads thus as follows.

```
Body 1
  Equation = 1
  Material = 1
End
```

For solving the flow patterns the Navier-Stokes solver is used but the nonlinearity through convection is switched off in the equation block. Also, solvers for the fluidic force and saving data are enabled.

```
Equation 1
  Active Solvers(3) = Integer 1 2 3
  NS Convect = False
End
```

Just a single iteration of the Navier-Stokes solver is needed, since the equation is linear. This can be verified by switching the number of nonlinear iterations to a value more than one, and observing the change in solution between iteration steps.

```
Solver 1
   Equation = Navier-Stokes
   Variable = Flow Solution
   Variable DOFs = 3
   Linear System Solver = Iterative
   Linear System Iterative Method = BiCGStab
   Linear System Preconditioning = ILU0
   Linear System Max Iterations = 200
   Linear System Convergence Tolerance = 1.0e-08
   Stabilize = True
   Nonlinear System Convergence Tolerance = 1.0e-05
   Nonlinear System Max Iterations = 1
   Nonlinear System Newton After Iterations = 3
   Nonlinear System Newton After Tolerance = 1.0e-08
   Nonlinear System Relaxation Factor = 1.0
   Steady State Convergence Tolerance = 1.0e-05
End
```

The fluidic force solver needs to be run only once, after the flow solution is finished. With the keyword `Calculate Viscous Force` it is possible to define whether the viscous forces of the fluid are included in the force or not. If this is set to false, only the pressure integral is calculated.

```
Solver 2
  Exec Solver = After All
  Equation = Fluidic Force
  Procedure  ="FluidicForce" "ForceCompute"
  Calculate Viscous Force = True
End
```

The final solver is used to save data from the analysis. With the following definitions, the input velocity and the net force on the input boundary as well as the area of the boundary are written into a file called `flowdata.dat`.

```
Solver 3
  Exec Solver = After All
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = "flowdata.dat"
  Save Variable 1 = Velocity 3
  Save Coordinates(1,2) = 0.0 0.0
End
```

The fluid is defined to be air. Note the Elmer MEMS units used.

```
Material 1
  Name = Air
  Density = 1.293e-12
  Viscosity = 1.67e-5
End
```

Finally, the boundary conditions. BC 1 defines the input boundary, where also the fluidic force is calculated. BCs 2 and 4 are define the symmetry boundaries, BC 3 defines the no-slip conditions for the walls, and BC 5 defines an open boundary.

```
Boundary Condition 1
  Target Boundaries = 4
   Velocity 1 = 0.0
   Velocity 2 = 0.0
   Velocity 3 = 1.0e3
   Calculate Fluidic Force = True
End

Boundary Condition 2
  Target Boundaries(2) = 8 10
   Velocity 2 = 0.0
End

Boundary Condition 3
  Target Boundaries(4) = 1 2 3 7
   Velocity 1 = 0.0
   Velocity 2 = 0.0
   Velocity 3 = 0.0
End

Boundary Condition 4
  Target Boundaries(2) = 6 9
   Velocity 1 = 0.0
End

Boundary Condition 5
  Target Boundaries = 5
  Pressure = 0.0
End
```

Slip boundary conditions

The same simulation can also be performed using slip boundary conditions. These are appropriate, as stated in introduction, when the Knudsen number is between 10^{-3} and 1. The slip boundary condition implemented in Elmer is of first order

$$S \cdot \vec{u} = \overline{\overline{\sigma}} \cdot \vec{n}, \tag{16.7}$$

where S is a vector containing the slip coefficients s_i for each velocity component, μ is the viscosity, and $\overline{\overline{\sigma}}$ is the stress tensor. For Newtonian fluids and for tangential directions of the boundary this gives

$$s_i u_i = \mu \frac{\partial u_i}{\partial n}, \tag{16.8}$$

where s_i and u_i refer to the same tangential component of the slip coefficient and the flow velocity.

The value of the slip coefficient is related to the mean free path of the gas molecules λ. For example, Maxwell's first order slip boundary condition may be used (as in *e.g.* A. Beskok, *Num. Heat Transfer,* B, 40, 451–471, 2001):

$$u_i = \frac{2 - \sigma_v}{\sigma_v} \lambda \frac{\partial u_i}{\partial n}, \tag{16.9}$$

where σ_v is the tangential momentum accommodation coefficient, which models the momentum exchange of gas molecules and the surface. The accommodation coefficient is dependent on the gas and on the surface, and recent measurements give a result of $\sigma_v \simeq 0.80$ for various monoatomic gases such as Argon in contact with prime Silicon crystal.

The slip coefficient of Elmer can finally be written as

$$s_i = \frac{\mu}{\lambda} \frac{\sigma_v}{2 - \sigma_v}. \tag{16.10}$$

The mean free path is defined as

$$\lambda = \frac{\mu}{\rho} \sqrt{\frac{\pi M}{2RT}}, \tag{16.11}$$

where ρ is density, M is the molar mass, T is the temperature, and $R = 8.3145$ J/mol K is the molar gas constant.

In the Elmer analysis, only a few changes in the sif-file are needed to make the slip conditions active. The flow force boundary conditions have to be turned on and the numerical value of the slip coefficient has to be defined on each boundary (here s =2e-4 is used for air). Further below is a list of the Boundary Condition blocks. Note that there are more BCs than in the no-slip simulation, since a separate condition is needed for surfaces oriented differently in space.

Generally, a normal-tangential orientation scheme for the boundary conditions are needed, since the surfaces are not necessarily having a normal vector pointing in one of the coordinate directions. This would be done for each such boundary by the line

```
  Normal-Tangential Velocity = True
```

after which the Velocity component 1 points to the normal direction and the other components to the tangential directions.

```
! Definitions for slip boundary conditions:
Boundary Condition 1
  Target Boundaries = 4
   Flow Force BC = True
   Slip Coefficient 1 = 2e-4
   Slip Coefficient 2 = 2e-4
   Velocity 3 = 2.0e3
   Calculate Fluidic Force = True
End

Boundary Condition 2
  Target Boundaries(2) = 8 10
   Velocity 2 = 0.0
End

Boundary Condition 3
```

```
  Target Boundaries(2) = 2 3
   Flow Force BC = True
   Velocity 3 = 0.0
   Slip Coefficient 1 = 2e-4
   Slip Coefficient 2 = 2e-4
End

Boundary Condition 4
  Target Boundaries(2) = 6 9
   Velocity 1 = 0.0
End

Boundary Condition 5
  Target Boundaries = 5
  Pressure = 0.0
End

Boundary Condition 6
  Target Boundaries = 1
   Flow Force BC = True
   Velocity 1 = 0.0
   Slip Coefficient 2 = 2e-4
   Slip Coefficient 3 = 2e-4
End

Boundary Condition 7
  Target Boundaries = 7
   Flow Force BC = True
   Velocity 2 = 0.0
   Slip Coefficient 1 = 2e-4
   Slip Coefficient 3 = 2e-4
End
```

Results

The computation takes about 200 cpu seconds on an AlphaServer with 1 GHz central processor when trilinear elements are used. The results for two different input velocities taken from the file `flowdata.dat` are summarised in Table 16.1. Also the specific acoustic impedance z_h is calculated in the table. The results of slip and no-slip simulations are also compared. Note that for the force, only the component perpendicular to the surface should be used since the other components cancel out due to symmetry. The values in the table are again given in Elmer MEMS units.

Table 16.1: Results of flow simulations for two input velocities

v	slip model	F_z	z_h
$1.0 \cdot 10^3$	no-slip	36.13	$1.45 \cdot 10^{-3}$
$2.0 \cdot 10^3$	no-slip	72.25	$1.45 \cdot 10^{-3}$
$1.0 \cdot 10^3$	slip	29.30	$1.17 \cdot 10^{-3}$
$2.0 \cdot 10^3$	slip	58.60	$1.17 \cdot 10^{-3}$

The identical values obtained for the spesific acoustic impedance in Table 16.1 prove by no means that the flow in reality is linear, since this was the assumption and the simulation performed can and should not reveal any nonlinear behavior. The results indicate, though, that allowing slip on the boundaries reduces the

resistance that the fluid faces. This example shows that in microsystems, as the dimension of the smallest flow channel is in the range of a micrometer, it is reasonable to use slip boundary conditions for the velocity.

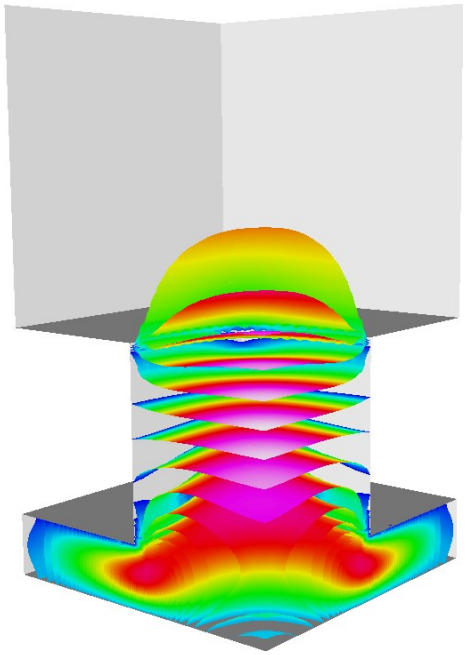

Figure 16.1: The linear flow results.

Finally, a picture of the results obtained with no-slip conditions is presented. The Fig. 16.1 shows a lot of pressure isosurfaces which are coloured using the absolute value of the velocity.

Tutorial 17

Electrostatics

Directory: Electrostatics
Solvers: StatElecSolve, ElectricForce
Tools: ElmerGrid, editor
Dimensions: 3D, Steady-state

Case definition

This case presents solving the Poisson equation for electric potential and calculating appropriate derived quantities, such as capacitance, based on the result. The geometry studied is a symmetric quadrant of a plane capacitor having a rectangular hole in another plate. A setting of this kind can be used to study the effects of geometrical features on the capacitance and on the electrostatic force, which both are meaningful quantities for coupled simulations in *e.g.* microsystems.

Solution procedure

The mesh is constructed using ElmerGrid with the following command

```
ElmerGrid 1 2 elmesh.grd
```

The mesh is extended above the hole to avoid undesired boundary effects. The geometry is presented in the Figure 17.1

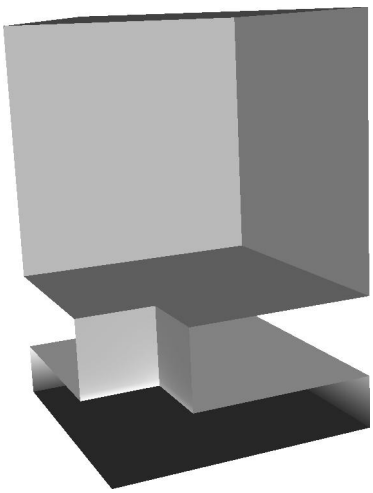

Figure 17.1: The geometry of problem.

The simulation problem includes a single body, and thus one material and one equation set, as well as three solvers. The solvers are used to compute the electric potential and related quantities, to calculate the electric force, and to save relevant data into a file. This tutorial is defined in Elmer MEMS units. The sif-file is presented below.

```
Check Keywords Warn

Header
  Mesh DB "." "elmesh"
End
```

Only a single steady state iteration is needed, since the Poisson equation is linear.

```
Simulation
  Coordinate System = Cartesian 3D
  Simulation Type = Steady State
  Steady State Max Iterations = 1
  Output File = "elstatics.result"
  Post File = "elstatics.ep"
End
```

The permittivity of vacuum has to be defined in the Constants section.

```
Constants
  Permittivity Of Vacuum = 8.8542e-12
End

Body 1
  Equation = 1
  Material = 1
End
```

Electric energy density is added into the results in Equation section. This allows energy density to be visualised in ElmerPost. Note also, that calculating electric flux (or the electric displacement field) is disabled in the Solver 1 block. Further, the potential difference used in calculating the capacitance of the system has to be defined in this section. This should be the same as the boundary conditions define for the capacitance calculation to be sensible.

```
Equation 1
  Active Solvers(2) = 1 2
  Calculate Electric Energy = True  ! (default False)
End

Solver 1
  Equation = Stat Elec Solver
  Variable = Potential
  Variable DOFs = 1
  Procedure = "StatElecSolve" "StatElecSolver"
  Calculate Electric Field = True  ! (default True)
  Calculate Electric Flux = False  ! (default True)
  Potential Difference = 1.0e6
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Max Iterations = 200
  Linear System Convergence Tolerance = 1.0e-07
  Linear System Preconditioning = ILU1
  Linear System ILUT Tolerance = 1.0e-03
```

```
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-4
  Nonlinear System Newton After Tolerance = 1.0e-3
  Nonlinear System Newton After Iterations = 10
  Nonlinear System Relaxation Factor = 1
  Steady State Convergence Tolerance = 1.0e-4
End
```

The static electric force solver does not need a lot of information:

```
Solver 2
  Equation = Electric Force
  Procedure = "ElectricForce" "StatElecForce"
End
```

Finally, some data is saved in file scalars.dat in working directory.

```
Solver 3
  Exec Solver = After All
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = "scalars.dat"
End
```

Only the relative permittivity of the material has to be defined.

```
Material 1
  Relative Permittivity = 1
End
```

The boundary conditions include the values of electric potential (voltage) and indication on which boundary the electric force should be calculated. On all the other boundaries a natural boundary condition is used, basically stating that the electric flux through these boundaries is zero.

```
Boundary Condition 1
  Target Boundaries = 4
  Potential = 0.0
  Calculate Electric Force = True
End

Boundary Condition 2
  Target Boundaries = 3
  Potential = 1.0e6
End
```

Results

The results obtained for capacitance and electric force are compared to those of a complete plane capacitor. For a plane capacitor, the capacitance is

$$C = \varepsilon_r \varepsilon_0 \frac{A}{d}, \tag{17.1}$$

and the electrostatic force is

$$F_e = \frac{1}{2} \varepsilon_r \varepsilon_0 \frac{A}{d^2} \Phi^2, \tag{17.2}$$

where ε_r is the relative permittivity, ε_0 is the permittivity of vacuum, A is the area of a capacitor plate, d is the separation of the capacitor plates, and Φ is the potential difference between the plates.

Table 17.1: Comparison of numerical results to analytic values

	simulation	analytic	ratio
Capacitance	$2.1361 \cdot 10^{-10}$	$2.2136 \cdot 10^{-10}$	0.965
Electric Force	$1.0406 \cdot 10^{2}$	$1.1068 \cdot 10^{2}$	0.940

The results of the simulation as well as the comparison to the complete plane capacitor values are shown in Table 17.1 (in Elmer MEMS units). Note that the fringe fields on capacitor edges are not calculated. This would require much larger mesh extending outside the capacitor boundaries.

Finally, a picture of the results is presented. The Figure 17.2 shows the isosurfaces of the electric potential with the color marking the strength of the electric field. From the picture it is clearly seen that the electric field is constant between the plates except for the proximity of the hole which causes weakening of the field magnitude. There are also strong electric fields at the edges of the hole.

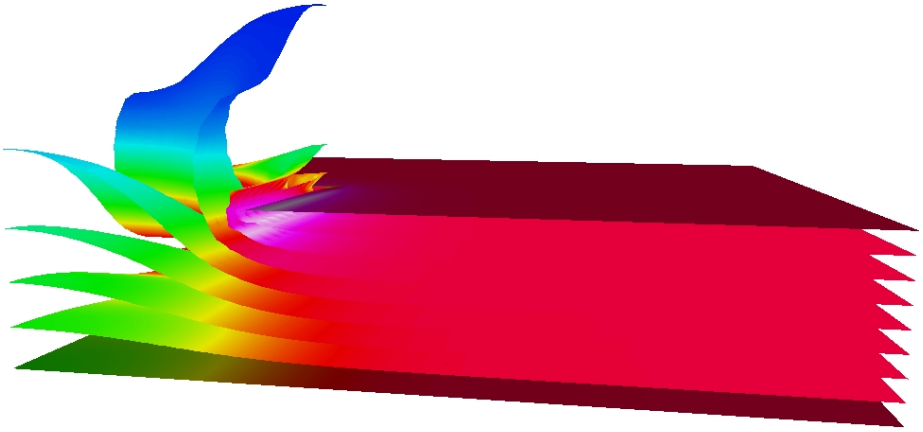

Figure 17.2: Isosurfaces of the potential coloured with electric field magnitude.

Tutorial 18

Lossless acoustic waves

Directory: AcousticWaves
Solvers: HelmholtzSolve
Tools: ElmerFront
Dimensions: 2D, Harmonic

Introduction

Elmer provides two alternative ways of conducting acoustic analyses in the frequency domain. Firstly, one may simply use the Helmholtz equation which is based on the assumption of lossless flow, i.e. the effects of viscosity and heat conduction are assumed to be negligible. More refined analyses where these effects are taken into account may be carried out by using the specific solver for the set of time-harmonic dissipative acoustic equations. The aim of this tutorial is to demonstrate the usage of the solver for the basic Helmholtz equation, which is frequently taken as the starting point in acoustic analyses.

Case description

In this problem the fluctuations of the pressure in an air-filled cavity shown in Figure 18.1 are considered. The cavity is connected with the surrounding air by an open narrow pipe. The pressure fluctuations are generated by a vibrating membrane on the boundary Γ_S with the frequency of the motion being $f = 100$ Hz. The remaining parts of the boundary are assumed to be rigid walls. In addition, the effects of gravity are assumed to be negligible.

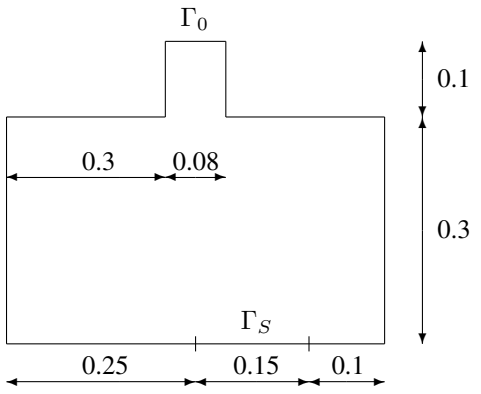

Figure 18.1: The geometry of the cavity.

Suitable boundary conditions in terms of the pressure must be given. On the rigid walls the pressure flux is prescribed to vanish which corresponds to the assumption that there is no velocity in the direction normal to the boundary. At the open end Γ_0 the impedance boundary condition suitable for forward traveling plane waves is given by setting $Z = -c$ with c being the sound speed. We assume that $c = 343$ (m/s). Finally, the wave source is given by defining a non-vanishing pressure flux on the corresponding part of the boundary. We take simply $\nabla P \cdot \vec{n} = 1$ where P is the (complex) amplitude of the pressure and $\vec{n}$ is the outward unit normal to the boundary.

Solution procedure

- Before starting Elmer copy the geometry file (`domain.egf`) to the working directory and then launch Elmer Front by giving the command

  ```
  ElmerFront
  ```

- Open the geometry file by choosing Open Cad-file in the File menu. To enable browsing with the mouse click the button on the right-hand side of the field where the file name may be written. Here the correct Cad file type is Elmer. Give also the model name (for example `helmholtz`) and write the path of the working directory in the Model directory field.

- Select the equation to be solved by selecting Equations in the Problem menu. Choose the Helmholtz equation and press `Add` button.

- Define the angular frequency for the simulation by selecting Simulation parameters in the Problem menu. Enter the value 628.3 to the field and accept the value by clicking `OK` button.

- Define the sound speed for the medium by selecting Materials in the Model menu. Enter the value 343 for the sound speed and press `Add` button.

- Prescribe the boundary conditions by selecting Boundary conditions in the Model menu. Select (with the mouse) Boundary1 and give the value for the boundary flux:

  ```
  Wave flux Re = 1
  Wave flux Re = 0
  ```

  Finally, press `Add` button. Then proceed to give the other boundary conditions in a similar manner (the value for the pressure is prescribed).

- Create a finite element mesh by selecting Define mesh in the Mesh menu. To begin with give a name for the mesh. Define the number of element edges on each boundary and then create the mesh by pressing `Generate Mesh` button.

- The problem may now be solved by selecting Solver in the Run menu.

- After the solution is done, view the results by selecting the Postprocessor from the Run menu.

  ```
  Run -> Postprocessor
  ```

- To save the created model, select Save model file from the File menu.

  ```
  File -> Save model file
  ```

Results

Using a mesh consisting of 3900 (quadratic) elements with 7601 nodes the difference of the maximum and the minimum value of the pressure is found to be $\Delta p \approx 0.205$

Tutorial 19

Induction heating of a graphite crucible

Directory: InductionHeating
Solvers: StatMagSolve
Tools: ElmerGrid, editor
Dimensions: 2D, Axi-Symmetric

Case definition

At high temperatures the most practical method to heat up the crucible is by electromagnetic induction. The induction coil generates an alternating current that flows through the crucible. The Ohmic resistance encountered by this current dissipates energy, thereby directly heating the crucible via internal heat generation.

The tutorial case is a simple axi-symmetric crucible that could be used, for example, to grow silicon carbide (SiC) by the sublimation method. The crucible is made of dense graphite and isolated by porous graphite. At the bottom of the crucible there is some SiC powder. The physical properties of the material are given in Table 19.1. The dimensions of the induction heating crucible are given in Table 19.2. Additionally, the powder thickness is 1.0 cm and there are 10 spirals in the coil. The frequency of induction heating f is 50 kHz and the current I is 10 A. The permeability of the space is $4\pi 10^{-7}$ if the other variables are in SI-units.

Solution Procedure

At low frequencies the free charges may be neglected and the induction heating problem may be solved in terms of an magnatic vector potential. The proper solver to do this is `StatMagSolver`. However, the induction heating problem can only be modeled if the helicity of the coil is neglected and an avarage current density is assumed. This current density may be computed easily when the area of the coil is known $j_0 = nI/A$, where A is the coil area.

The mesh for this problem may easily be created by ElmerGrid. The provided mesh is quite sufficient for this case but for higher frequencies the mesh should be tuned to solve the thin boundary layers. The computational mesh is created from file `crucible.grd` by the command

```
ElmerGrid 1 2 crucible
```

Table 19.1: Material parameters of the crucible

material	ε	κ [W/mk]	σ (1/Ωm)
graphite	0.7	10.0	2.0E4
insulation	0.9	1.0	2.0E3
powder	0.5	25.0	1.0E4

Table 19.2: Dimensions of the crucible

body part	r_{inner}	r_{outer}	h_{inner}	h_{outer}
graphite	2.0	2.5	6.0	8.0
insulation	2.5	4.0	8.0	12.0
coil	5.0	5.5		8.0

The mesh consists of 5 different bodies which need 4 different materials sets. Only on set of boundary conditions are required for the external boundary. Thus the header information of the command file is as follows

```
Header
  Mesh DB "." "crucible"
  Include Path ""
  Results Directory ""
End
```

In the `Simulation` section the coordinate system and time dependendy is set, among other things. Also we know that the equation is linear and therefore only one steady state iteration is requited. If the electric properties depend on the magnitude of the field several iterations are required.

```
Simulation
  Coordinate System = "Axi Symmetric"
  Simulation Type = Steady State
  Steady State Max Iterations = 1
  Output File = "crucible.result"
  Post File = "crucible.ep"
End
```

In the `Constants` section the permittivity of vacuum must be given.

```
Constants
  Permittivity Of Vacuum = 8.8542e-12
End
```

In the differential equation for the magnetic vector potential the source the is the current density. Thus, it is given in the `Body Force` section.

```
Body Force 1
  Current Density = 2.5e5
End
```

In the `Body` section the different bodies are assigned with correct equation sets and material parameters, for example

```
Body 3
  Name = "Insulation"
  Equation = 1
  Material = 2
End
```

In the `Equation` block all the relavant solvers are set to active.

```
Equation
  Name = "Vector Potential Equation"
  Active Solvers = 1
End
```

The only solver in this simple tutorial is the solver for the magnetic vector potential. Look for the relevant model manual for information about the options. Here the equation is solved iteratively and the local Joule heating and magnetic flux are computed as a postprocessing step. The Joule heating is scaled so that the total heating power is 3.0 kW. This option may be used when the total heating efficiency is known. The nonlinear solver parameters are not really needed as the material parameters are constant. Sometimes the parameters may depend on the magnetic field and thus the nonlinear problem must be solved iteratively.

```
Solver 1
  Equation = Potential Solver
  Variable = Potential
  Variable DOFs = 2

  Angular Frequency = Real 50.0e3
  Calculate Joule Heating = Logical True
  Calculate Magnetic Flux = Logical True
  Desired Heating = Real 3.0e3

  Procedure = "StatMagSolve" "StatMagSolver"
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Max Iterations = 300
  Linear System Convergence Tolerance = 1.0e-10
  Linear System Preconditioning = ILU1
  Linear System ILUT Tolerance = 1.0e-03
  Linear System Residual Output = 1
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-6
  Nonlinear System Relaxation Factor = 1
  Steady State Convergence Tolerance = 1.0e-6
End
```

In the `Material` sections all the necessary material parameters are given, for example

```
Material 2
  Name = "Insulation"
  Electric Conductivity = 2.0E3
End
```

The magnetic field must vanish at infinity. Unfortunately the computational domain is bounded and therefore the infinite distance becomes very finite. A proper distance may be checked by gradually increasing it until no change in the result occurs.

```
Boundary Condition 1
  Target Boundaries = 1
  Potential 1 = Real 0.0
  Potential 2 = Real 0.0
End
```

Results

With the given computational mesh the problem is solved in a few seconds. With the 20 072 bilinear elements the heating efficieny is 16.9 W. The corresponding results are shown in Fig. 19.1.

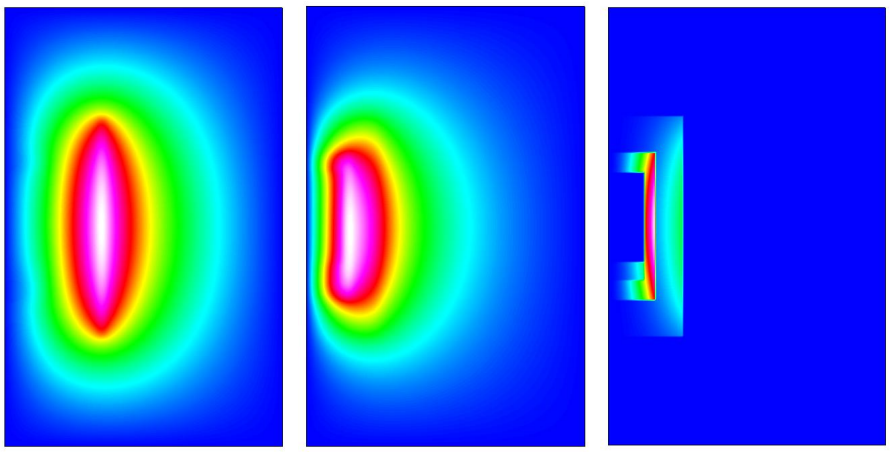

Figure 19.1: Induction heating of a simple crucible. a) in-phase component of the vector potential b) out-of-phase component of the vector potential c) Joule losses in the conductors

Tutorial 20

Thermal actuator driven with electrostatic currents

Directory: ThermalActuator
Solvers: StatCurrentSolve, HeatSolve, StressSolve
Tools: ElmerGrid, editor
Dimensions: 3D, Steady-state

Case definition

The tutorial introduces a micro mechanical thermal actuator as shown in Fig. 20.1. A static electric current is driven through the actuator. The power loss due to the resistance of the actuator is transformed into heat which in turn causes thermal stresses into the structure. The electric current thus results in deformation of the actuator. In industry, such an actuator might be used to control the position of a micromechanical component.

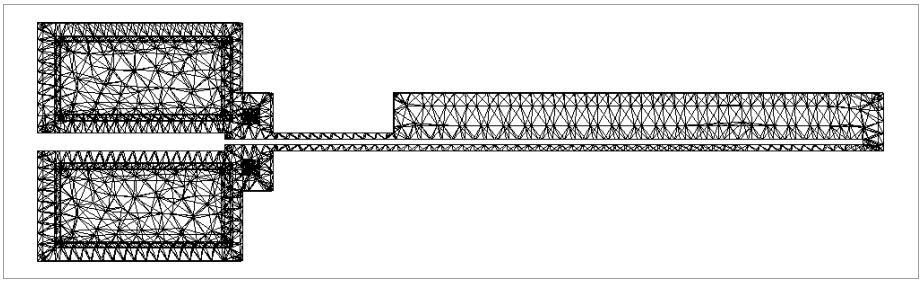

Figure 20.1: The geometry of the actuator.

Solution procedure

The problem is solved by first iterating the electrostatic current solver and heat equation until both are converged. The temperature distribution is then used as a load for stress analysis solver which calculates the actual deformation of the structure. The electric conductivity of the actuator depends on the temperature and thus the electrostatic - thermal problem is coupled in both directions.

The computational mesh for this particular tutorial is created by using Ansys software. The details of the mesh are written into files called `ExportMesh` by a certain Ansys macro and converted to Elmer format by the ElmerGrid program. The command to use is

```
ElmerGrid 4 2 ExportMesh -order 1.0 0.1 0.001 -o thermal
```

The above command reads in the Ansys mesh files, arranges the mesh nodes in a reasonable way and saves the mesh in Elmer format in a directory called `thermal`.

The geometry of the problem includes only one body and material. Boundary conditions are defined on the actuator legs, which are kept at constant electric potential, temperature and position. Thus, only Dirichlet boundary conditions are used.

The header and simulation blocks of the solver input file are

```
Header
  Mesh DB "." "thermal"
End

Simulation
  Coordinate System = Cartesian 3D
  Simulation Type = Steady State
  Steady State Max Iterations = 30
  Output Intervals = 1
  Output File = "actuator.result"
  Post File = "actuator.ep"
End
```

An initial condition for temperature is defined in order to ease the convergence of the iterative solvers. Also, a body force for the heat equation solver defining the Joule heating is needed. These both have to be declared in the body section as follows:

```
Body 1
  Equation = 1
  Material = 1
  Initial Condition = 1
  Body Force = 1
End
```

The solution procedure requires the use of three solvers: Static current solver, heat equation solver and the stress analysis solver. The equation block below defines that these solvers are used.

```
Equation 1
  Active Solvers(3) = Integer 1 2 3
  Calculate Joule Heating = True
End
```

The solver blocks define the parameters of the respecting solvers. The static current conduction problem is tackled by an iterative conjugate gradient method (CG). For heat equation, a stabilized biconjugate gradient method is used. The coupled problem of these two solvers is difficult since the static current calculated heats the structure on each step, and the rise of temperature makes the current conduction more and more difficult. To overcome this problem, a relaxation factor of 0.5 is defined for the heat equation solver.

```
Solver 1
  Equation = Stat Current Solver
  Procedure = "StatCurrentSolve" "StatCurrentSolver"
  Variable = Potential
  Variable DOFs = 1
  Calculate Volume Current = True
  Calculate Electric Conductivity = True
  Linear System Solver = Iterative
  Linear System Iterative Method = CG
  Linear System Preconditioning = ILU3
  Linear System Max Iterations = 300
```

```
  Linear System Convergence Tolerance = 1.0e-8
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0-6
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-12
  Nonlinear System Relaxation Factor = 1.0
  Steady State Convergence Tolerance = 1.0e-6
End

Solver 2
   Equation = Heat Equation
   Variable = Temperature
   Variable DOFs = 1
   Linear System Solver = Iterative
   Linear System Iterative Method = BiCGStab
   Linear System Preconditioning = ILU1
   Linear System Max Iterations = 350
   Linear System Convergence Tolerance = 1.0e-9
   Nonlinear System Max Iterations = 1
   Nonlinear System Convergence Tolerance = 1.0e-07
   Nonlinear System Newton After Iterations = 3
   Nonlinear System Newton After Tolerance = 1.0e-12
   Nonlinear System Relaxation Factor = 0.5
   Steady State Convergence Tolerance = 1.0e-07
End
```

For stress analysis, a direct solver is used instead of an iterative solver. It is often difficult for the iterative solver to find a solution for a structure that contains parts with varying stiffness properties, which is obviously the case here (try the iterative solver and see!). The stress analysis solver is called first only after the coupled iteration of two previous solvers is complete. This is possible since the deformation of the structure is so small that it does not change the current density distribution. Defining stress analysis this way saves computational time. It is possible to iterate all the three solvers until convergence by commenting the `Exec Solver` line.

```
Solver 3
  Exec Solver = After All
  Equation = Stress Analysis
  Variable = Displacement
  Variable DOFs = 3
  Linear System Solver = Direct
  Linear System Direct Method = Banded
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-6
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-12
  Nonlinear System Relaxation Factor = 1.0
  Steady State Convergence Tolerance = 1.0e-6
End
```

The material of the structure has a temperature dependent electric conductivity. This, as well as other material parameters, is defined in the material block. Note that a MEMS unit system is used.

```
Material 1
  Electric Conductivity = Variable Temperature
      Real
```

```
        298.0   4.3478e10
        498.0   1.2043e10
        698.0   5.1781e9
        898.0   2.7582e9
        1098.0  1.6684e9
        1298.0  1.0981e9
        1683.0  1.0
        2000.0  1.0
      End

  Density = 2.3e-15
  Heat Conductivity = 32.0e6
  Youngs Modulus = 169.0e3
  Poisson Ratio = 0.22
  Heat Expansion Coefficient = 2.9e-6
  Reference Temperature = 298.0
End
```

Finally, the initial condition, thermal heat load for stress analysis, and the boundary conditions are defined.

```
Initial Condition 1
   Temperature = 298.0
End

Body Force 1
  Heat Source = Equals Joule Heating
End

Boundary Condition 1
  Target Boundaries = 1
  Potential = 0
  Temperature = 298
  Displacement 1 = 0.0
  Displacement 2 = 0.0
  Displacement 3 = 0.0
End

Boundary Condition 2
  Target Boundaries = 2
  Potential = 7
  Temperature = 298
  Displacement 1 = 0.0
  Displacement 2 = 0.0
  Displacement 3 = 0.0
End
```

Results

The problem converges after 27 steady state iterations on the tolerance limits defined above. The calculation takes about 180 cpu seconds of which 40 cpus is spent in solving the stress analysis equation. The calculations were performed on a Compaq Alpha Server with a 1 GHz central processor.

Result for temperature distribution and the displacement are shown in Figs 20.2 and 20.3. The temperature rises unrealistically high in this example because all heat transfer mechanisms out of the structure are

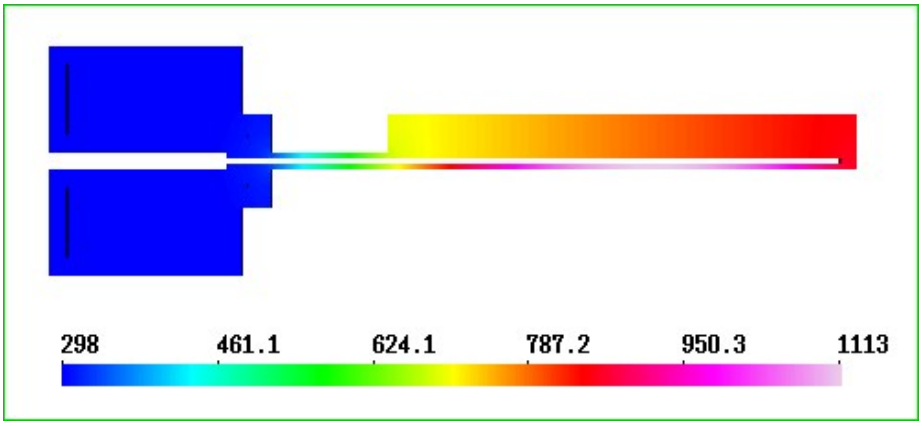

Figure 20.2: Temperature distribution.

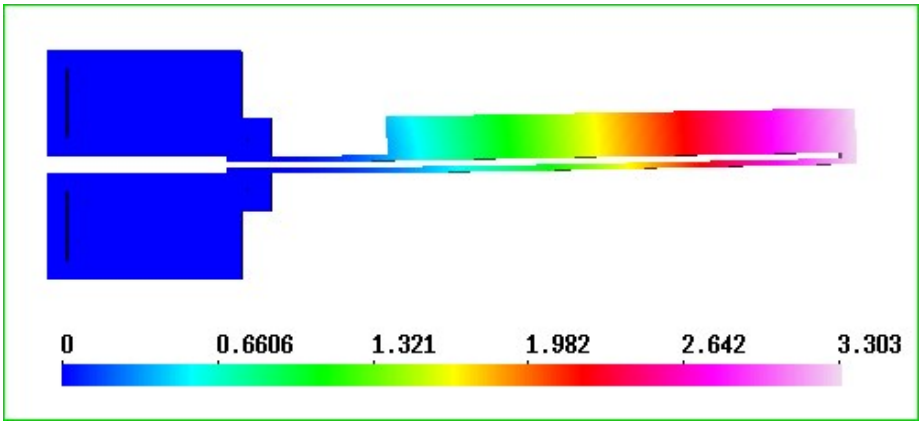

Figure 20.3: The displacement of the actuator.

neglected. Presumambly at least the heat radiation is of major importance in this case. For displacement, the results show a movement of about 3.3 micrometers for the actuator tip.

Tutorial 21

Axisymmetric coating process

Solvers: FlowSolve, FreeSurfaceReduced
Tools: ElmerGrid, editor
Dimensions: 2D, Steady-state

Case definition

The optical fibers are quite fragile and must therefore be coated with a layer of polymer before they are stored. This means that the coating process must be done with the same speed as the drawing of optical fibers. When the diameter of the fiber is only 125 μm this sets high demands for the coating geometry since it must provide even coating at high draw speeds. In Elmer a tailored free surface boundary condition allows an efficient solution of this particular problem.

Solution procedure

The mesh is done with ElmerGrid in the directory coat by the command

```
ElmerGrid 1 2 coat.grd
```

Therefore the header reads

```
Header
  Mesh DB "." "coat"
End
```

The geometry is axisymmetric and the problem is solved in steady state. Typically around 10 iterations is needed to solve the problem but to be on the safe side 30 is set as the maximum.

```
Simulation
  Coordinate System = Axi Symmetric
  Simulation Type = Steady State
  Steady State Max Iterations = 30
  Output Intervals = 1
  Output File = "coat.result"
  Post File = "coat.ep"
End
```

In this case there is only one body which comprises of the polymer floating between the coating cup and the optical fiber.

```
Body 1
  Equation = 1
  Material = 1
End
```

The presented solution used four different solvers. The Navier-Stokes solver is required to solve the flow field for the polymer.

```
Solver 1
  Equation = Navier-Stokes
  Stabilize = True
  Internal Move Boundary = Logical False
  Nonlinear System Max Iterations = 5
  Nonlinear System Convergence Tolerance = 1.0e-7
  Nonlinear System Newton After Iterations = 2
  Nonlinear System Newton After Tolerance = 1.0e-2
  Nonlinear System Relaxation Factor = 0.7
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Preconditioning = ILU1
  Linear System Max Iterations = 100
  Linear System Convergence Tolerance = 1.0e-10
  Steady State Convergence Tolerance = 1.0e-7
End
```

A tailored free surface solver is used to find the position of the free surface with a given flow field. The variable being solved is the displacement of the free surface. Relaxation is used to avoid over-shooting during the itaration. This solver does not solve any matrix equations. Instead it solves the radius from the mass conservation constraint for each node on the free surface separately. There is a possibility to do the mapping also within the solver using a 1D scheme but this is disabled by setting the `Perform Mapping` to be `False`.

```
Solver 2
  Equation = "Free Surface Reduced"
  Procedure = "FreeSurfaceReduced" "FreeSurfaceReduced"
  Variable = Dx
  Variable DOFs = 1
  Nonlinear System Relaxation Factor = 0.7
  Nonlinear System Convergence Tolerance = 1.0e-3
  Steady State Convergence Tolerance = 1.0e-3
  Perform Mapping = Logical False
End
```

The mesh update solver is required to map the computational mesh so that it corresponds to the altered geometry. Here the displacements of the free surface have already been computed and this solver solves the displacements inside the domain. Note that solvers 1, 2 and 3 are coupled and therefore the system must be solved iteratively

```
Solver 3
  Equation = Mesh Update
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGSTAB
  Linear System Preconditioning = ILU
  Linear System Convergence Tolerance = 1.0e-12
  Linear System Max Iterations = 200
  Linear System Symmetric = True
  Steady State Convergence Tolerance = 1.0e-4
End
```

In the end, an additional solver is used to compute the forces acting on the fiber. This does not affect the results.

```
Solver 4
  Equation = Fluidic Force
  Procedure = "FluidicForce" "ForceCompute"
  Calculate Viscous Force = Logical True
End
```

Addiationally there are two solvers for saving the results in a form that is more useful than plain pictures. The `SaveScalars` saves the scalar values, such as the diameter and force values, and the `SaveLine` saves the free surface.

```
Solver 5
  Equation = SaveScalars
  Procedure = "SaveData" "SaveScalars"
  Filename = "scalars.dat"
End

Solver 6
  Equation = SaveLine
  Procedure = "SaveData" "SaveLine"
  Filename = "kurvi.dat"
End
```

The equation includes only the solvers that need a permutation vector pointing out the active nodes. Therefore the save utilities do not need to belong to the set of active solvers.

```
Equation 1
  Active Solvers(4) = 1 2 3 4
End
```

The material parameters are those of the polymer. Additionally elasticity parameters are needed because the solver that updates the mesh is actually a linear elasticity solver.

```
Material 1
  Density = 1.0
  Viscosity = 1.0
  Poisson Ratio = 0.3
  Youngs Modulus = 1.0
End
```

Five different boundary conditions are needed. The origin is a symmetry axis and thefore the radial velocity is set to zero. The axial velovity is the draw velocity.

```
Boundary Condition 1
  Name = "Symmetry"
  Target Boundaries = 1
  Velocity 2 = -10.0     ! The draw velocity
  Velocity 1 = 0.0
  Compute Fluidic Force = Logical True
  Mesh Update 1 = 0.0
End
```

The free surface has a condition stating that the reduced order free surface solver should be solved for that. Additionally the free surface is a boundary condition for the mesh update, and a line to be saved.

```
Boundary Condition 2
  Name = "Free"
  Target Boundaries = 2
  Mesh Update 1 = Equals Dx
```

```
  Mesh Update 2 = 0.0
  Free Surface Reduced = Logical True
  Save Line = Logical True
End
```

At the outlet the radial velocity should vanish and the axial coordinate should be fixed.

```
Boundary Condition 3
  Name = "Outlet"
  Target Boundaries = 3
  Velocity 1 = 0.0
  Mesh Update 2 = 0.0
End
```

At the inlet it is assumed that there is no radial velocity and that the pressure acting on the surface is zero.

```
Boundary Condition 4
  Name = "Inlet"
  Target Boundaries = 4
  Velocity 1 = 0.0
  Pressure = 0.0
  Mesh Update 2 = 0.0
End
```

Finally, no-slip conditions are set for the boundaries with the walls of the coater.

```
Boundary Condition 5
  Name = "No-slip"
  Target Boundaries = 5
  Velocity 1 = 0.0
  Velocity 2 = 0.0
  Mesh Update 1 = 0.0
  Mesh Update 2 = 0.0
End
```

Results

In the given case solution is obtained after 13 iterations. The solution gives the final radius, the forces, and the profile of the free surface. To visualize the true free surface you may do the following. Read in the only the last timestep and in `ElmerPost` give the following commands:

```
math nodes0 = nodes
math nodes = nodes0 + Mesh.Update
```

Note that this does not work if there is more than one set of variable values.

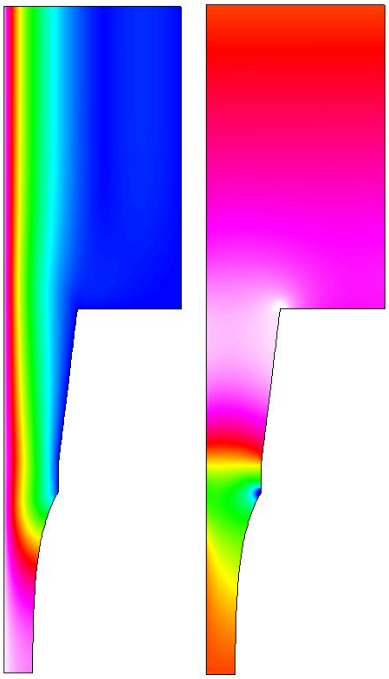

Figure 21.1: The velocity and pressure fields in a simple coating geometry. The solution utilizes the reduced dimensional free surface solver.

Tutorial 22

Blood ejection from a ventricle into aorta

Directory: ArteryFlow
Solvers: FlowSolve,ElasticSolve, OutletCompute
Tools: Editor, Fortran 90 compiler, ElmerGrid
Dimensions: 2D, Transient

Case description

This tutorial is about simulating blood ejection in to the elastic human aorta. The idea is to mimic left ventricle contration and resulting pulse propagation in an elastic conduit. In the simulation about 0.8 desiliters of blood is ejected to a 50 cm long elastic aorta during a time period of 400 ms. In order to get the outlet of the model behave physiologically more realistic way, a one dimensional model is coupled with the higher order model.

Solution procedure

First we generate the mesh of 366 eight-node quadrilaterals elements with the command

```
ElmerGrid 1 2 contra
```

Next we generate one dimensional mesh to the outlet of the 2D model. The program `AddOneDim` is posed to be run in the mesh directory `contra`. The length, the number of the elements, and the coordinate direction of the 1D section will be asked.

In the simulation block the timestep is set equal to 1 ms and total simulation time equal to 600 ms. The geometry consists of five bodies of which the first three are for the fluid volume. Body number 1 os the contracting volume. Body 2 is a short rigid channel between the body 1 and the elastic artery. Artificial compressibility method is used for the fluid volume (body 3) which is in contact with the elastic wall (body 4). One dimesional model is the body 5. Material settings for those are following:

```
! Bodies 1 and 2 (blood)
Material 1
  Density = 1000
  Viscosity = 3.5e-3
  Youngs Modulus = 1
  Poisson Ratio = 0.3
End

! Body 3 (blood)
```

```
Material 2
  Density = 1000
  Viscosity = 3.5e-3
  Youngs Modulus = 1
  Poisson Ratio = 0.3
  Compressibility Model = Artificial Compressible
  Artificial Compressibility  = 3.3E-5
End

! Body 4 (elastic wall)
Material 3
  Density = 1010
  Youngs Modulus = 3.0e5
  Poisson Ratio = 0.45
End

! One dimesional model
Material 4
   Density = 1010.0
   Artery Wall Youngs Modulus = Real 3.0e5
   Artery Radius = Real 0.0135
   Artery Wall Thickness = Real 0.002
   Artery Poisson Ratio  = Real 0.45
End
```

Notice that the radius of the one dimesional model (`Artery Radius`) is to the midplane of the wall (inner radius + half of the wall thickness). The overall FSI iteration scheme is started by one dimesional solver (`OutletCompute`, see the solver manual), after that Navier-Stokes, elasticity and mesh update solvers are run. Steady state convergence tolerance is set equalt to 1.0E-4 for each of the solvers. The nonlinearities of each of the solvers are computed within the FSI scheme loop, that is, the flag `Nonlinear System Max Iterations` is set equal to 1. Artificial compressibility coefficient is computed by the equation $c = (1 - \nu^2)[D/(E\ h)]$, where ν is the Poisson ratio of the artery wall, D, E and h are the inner diameter, Young's modulus and the thichness of the artery, respectively.

The only driving force of the system, the wall motion of the contracting fluid domain is given by the fortran function `Motion`, see the figure 22.1. The boundary condition setting is

```
! Moving boundary
Boundary Condition 1
  Target Boundaries = 1
  Velocity 1 = 0
  Velocity 2 = Equals Mesh Velocity 2
  Mesh Update 1 = Real 0
  Mesh Update 2 = Variable Time
       Real Procedure "./Motion" "Motion"
End
```

At the outlet, the pressure boundary condition is given by the function `OutletPres` and the corresponding radial displacement of the end wall of the outlet is given by the function `OutletdX`

```
! Outlet pressure of the 2D model
Boundary Condition 2
  Target Boundaries = 2
  Flux Integrate = Logical True
  Flow Force BC = True
  Pressure 2 = Variable Time
      Real Procedure "./ArteryOutlet" "OutletPres"
```

```
  Mesh Update 2 = Real 0
End

! Radial displacement of the end wall at the outlet of 2D model
Boundary Condition 9
  Target Boundaries = 9
  Displacement 1 = Variable Time
      Real Procedure "ArteryOutlet" "OutletdX"
  Displacement 2 = 0
End
```

FSI interface boundary is described as following

```
! FSI interface boundary
Boundary Condition 11
  Target Boundaries = 11
  Velocity 1 = Equals Mesh Velocity 1
  Velocity 2 = Equals Mesh Velocity 2
  Mesh Update 1 = Equals Displacement 1
  Mesh Update 2 = Equals Displacement 2
  Force BC = Logical True
End
```

Finally, the coupling of the 1D model with the 2D is done at the inlet boundary as

```
Boundary Condition 16
  Target Boundaries = 16
  Fluid Coupling With Boundary = Integer 2
  Structure Coupling With Boundary = Integer 9
End
```

Results

The contraction is curve seen in the figure 22.1 and the velocity fields at different time levels are presented in the figure 22.2. Postprocessing instructions are given in the file `PostProcessingInstr.txt`.

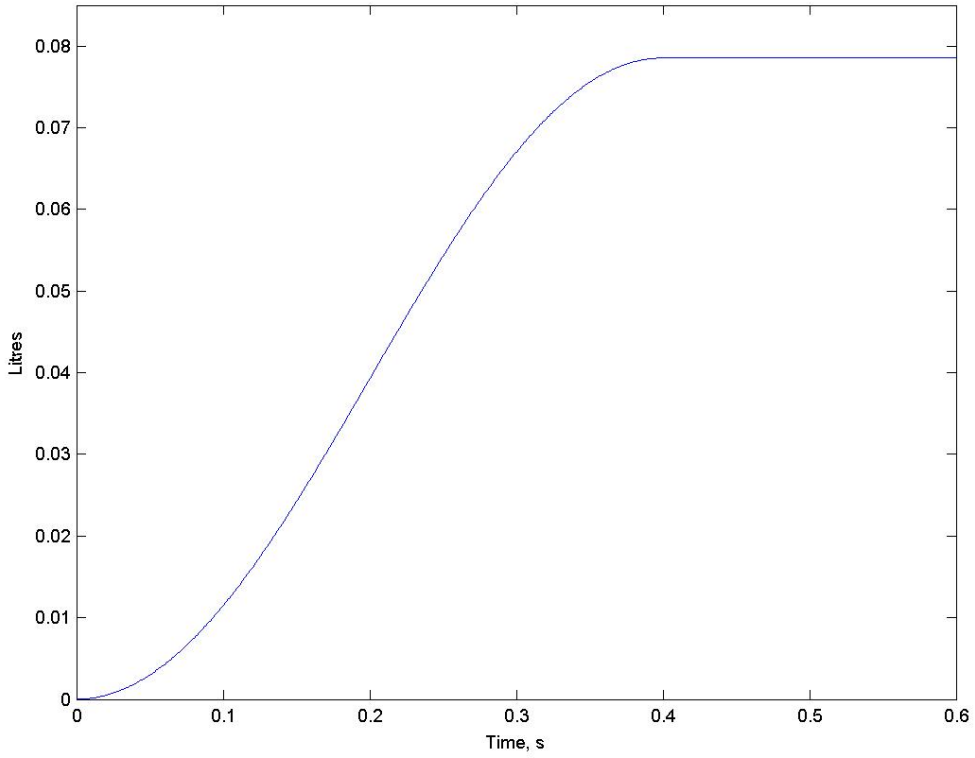

Figure 22.1: Contraction curve generated by the function `Motion`.

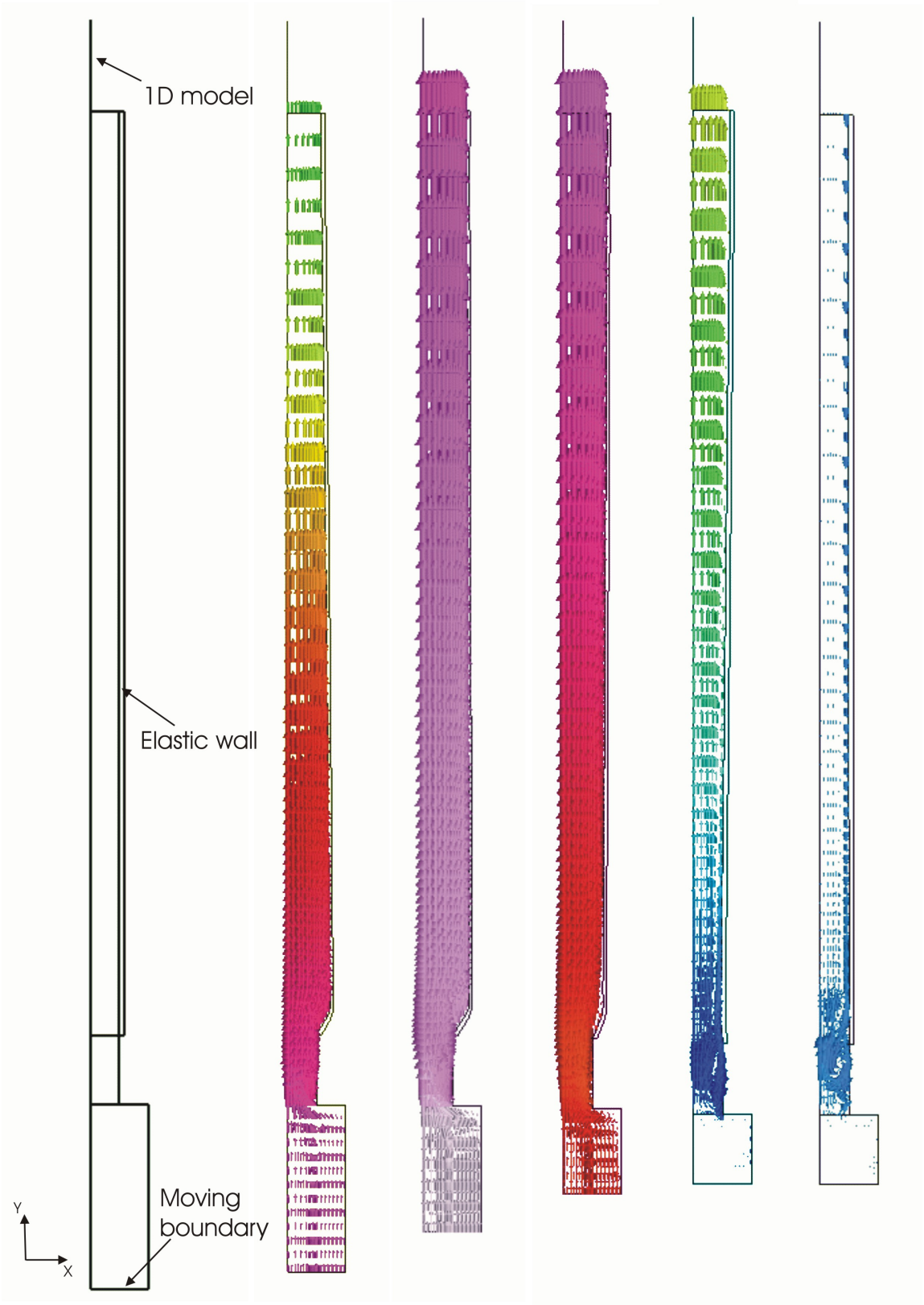

Figure 22.2: The geometry of the model and velocity fields at 5 time steps, 100, 200, 300, 400 and 500 ms. The displacements of the wall are magnified by factor of 10.

Part III

Utility Problems

Tutorial 23

Operator splitting in the evolutionary heat equation

Directory: TemperatureOperatorSplitting
Solvers: HeatSolve, TransportEquationSolver, RateOfChangeSolver
Tools: Editor
Dimensions: 2D, Transient simulation

Introduction

The drawback of the stabilized finite element formulations available in Elmer to solve the convection-diffusion equation and Navier-Stokes equations is that these methods are computationally expensive, in particular when the residual-free-bubbles formulation is used. In evolutionary problems the reduction of computational cost may be attempted by applying operator splitting techniques in which the original equation at each time step is splitted up into subproblems that are easier to solve numerically.

The aim of this chapter is to provide an illustrative example on using operator splitting capability in the solution of the time-dependent heat equation. For the theoretical background of the operator splitting scheme applied here the reader is referred to Elmer Models Manual and references therein.

Case description

The problem considered here is the solution of the time-dependent heat equation in the homogeneous L-shaped plate the geometry of which is shown in Figure **??**. The density of the material is taken to be unity, while the heat conductivity k of the material is taken to be a parameter with values ranging between 0.05 and 0.01. The plate is heated by a constant internal heat source magnitude of which equals to unity. The convection velocity field is assumed to be constant with the two Cartesian components of the velocity vector equaling to unity. The boundaries Γ_i, $i = 1, 4, 5, 6$, are kept at constant temperature 0, while the boundaries Γ_2 and Γ_3 are assumed to be insulated, i.e. the heat flux on this part of the boundary vanishes.

The problem is to solve the temperature distribution in the plate. The time interval for the simulation is taken to be [0,2] and the temperature of the plate is 0 at the time $t = 0$.

Solution procedure

Using operator splitting the solution of the heat equation may be replaced at each time step by the solution of three subproblems consisting of two time-dependent Poisson equations and the convective transport problem. The effects of diffusion and convection are decoupled by this splitting so that the diffusion and convection phenomena are taken into account by the steps involving the solution of the Poisson equation and the convective transport problem, respectively.

The time-dependent Poisson equations can be solved using the basic heat equation solver in Elmer. To avoid the use of stabilized finite element formulations in the solution of the convective transport problem this subproblem is solved by discretizating an equivalent wave-like equation formulation (second order in time equation).

When the operator splitting method is applied, specific care is needed in prescribing boundary and initial conditions for the simulation. While the boundary conditions (and initial conditions) for the steps involving the solution of the Poisson equation may be defined in the usual manner, the boundary conditions for the convective transport problem may be prescribed only on the part of inflow boundary on which the temperature is prescribed. In the example the inflow boundary is the union of the boundary segments Γ_i, $i = 1, 2, 5, 6$, so the boundary conditions for the convective transport equation should be given on Γ_i, $i = 1, 5, 6$. In addition to prescribing the boundary conditions, the rate of change of the field subject to the convection operator (here temperature) is needed as an initial condition at the beginning of pure convection step. This field can be solved using a specific solver (`RateOfChangeSolver`) prior to the solution of the convective transport problem. The boundary conditions for this solver should be prescribed on the same part of the boundary on which the boundary conditions for the convective transport problem are prescribed.

It should be noted that in connection with the operator splitting method the user should specify an even number of time steps. Although the details of implementation need not be known in order to use the operator splitting ability, it is noted that the the running of two successive time steps actually constitutes a single operator splitting scheme step as described in Elmer Models Manual. There are thus three equations (referred to as `Heat Equation`, `Rate Of Change Equation` and `Transport Equation`) which may be solved at one time step. At odd time steps all these equations are solved meaning that both the diffusion and convection steps are taken, whereas at even time steps the solution of the convective transport problem is omitted so that the diffusion step is performed only. Physically meaningful results satisfying all the essential boundary conditions may thus be written to the results file only after even time steps.

The mesh for this example was created using Femlab and tools for converting meshes from Femlab to Elmer format. The mesh in Elmer format is given in the tutorial files. This unstructured mesh consists of 8458 elements each of which having three nodes.

The Header section of the solver input file is used to declare the directory containing the Elmer mesh files:

```
Header
  Mesh DB "." "femlab-mesh"
End
```

The Simulation section of the solver input file is used to declare the coordinate system and simulation type as well as simulation parameters related to the time discretization scheme:

```
Simulation
  Coordinate System = String "Cartesian 2D"
  Simulation Type = String "Transient"
  Timestepping Method = String "Crank-Nicolson"
  Timestep Intervals(1) = Integer 200
  Timestep Sizes(1) = Real 0.01
  Output Intervals(1) = Integer 1
  Steady State Max Iterations = Integer 1
  Post File = File "os-example.ep"
End
```

Here the keyword `Timestepping Method` is used to define the time discretization scheme that is used in the solution of the time-dependent Poisson equations. Note that one should not use multi-step methods in connection with the operator splitting method. The time discretization scheme that is used in the solution of the convective transport problem is fixed and need not be specified. It should be noted also that the number of time steps should be even. Since the equations solved at one time step are not coupled and can thus be solved in sequential manner, `Steady State Max Iterations` may be taken to be 1.

The Body section of the solver input file is used to declare integer identifiers for body forces, equations, initial conditions and materials:

```
Body 1
  Body Force = Integer 1
  Equation = Integer 1
  Initial Condition = Integer 1
  Material = Integer 1
End
```

The Equation section of the solver input file in turn has the following declaration

```
Equation 1
  Active Solvers(3) = Integer 1 2 3
End
```

indicating that three equations are to be solved using solvers with the integer identifiers 1, 2 and 3. Accordingly, three Solver sections are needed. The first Solver section is used for the Poisson equation and has the following declarations:

```
Solver 1
  Equation = String "Heat Equation"
  Variable = String "Temperature"
  Variable Dofs = Integer 1
  Linear System Solver = String "Iterative"
  Linear System Iterative Method = String "BiCGStab"
  Linear System Max Iterations = Integer 350
  Linear System Convergence Tolerance = Real 1.0e-08
  Linear System Abort Not Converged = Logical True
  Linear System Preconditioning = String "ILU0"
  Steady State Convergence Tolerance = Real 1.0e-05
  Stabilize = Logical False
  Bubbles = Logical False
End
```

Note that there is no need to use stabilization, so the values of the keywords `Stabilize` and `Bubbles` may be set to be `False` to reduce the computational cost.

The remaining two Solver sections are related to the convective transport problem. The first one of these sections is used to declare the solver parameters related to the problem (`Rate Of Change Equation`) the solution of which gives the approximation to the rate of change of the temperature at the beginning of pure convection step. The contents of this solver section is

```
Solver 2
  Equation = String "Rate Of Change Equation"
  Procedure = File "RateOfChange" "RateOfChangeSolver"
  Variable = String "Udot0"
  Variable Dofs = Integer 1
  Advection = String "Constant"
  Advection Variable = String "Temperature"
  Linear System Solver = String "Iterative"
  Linear System Iterative Method = String "BiCGStab"
  Linear System Max Iterations = Integer 350
  Linear System Convergence Tolerance = Real 1.0e-08
  Linear System Abort Not Converged = Logical True
  Linear System Preconditioning = String "ILU0"
  Steady State Convergence Tolerance = Real 1.0e-05
End
```

Here the keyword `Advection Variable` is used to declare the quantity which is subject to the convection operator, while the keyword `Advection` is used to define the type of the velocity vector. The name `Udot0` is used for the solution of this problem.

Finally, the Solver section for the wave-like equation, which is equivalent to the convective transport equation, has the following declarations:

```
Solver 3
  Equation = String "Transport Equation"
  Procedure = File "TransportEquation" "TransportEquationSolver"
  Time Derivative Order = Integer 2
  Variable = String "U"
  Variable Dofs = Integer 1
  Advection = String "Constant"
  Advection Variable = String "Temperature"
  Rate Of Change Equation Variable = String "Udot0"
  Linear System Solver = String "Iterative"
  Linear System Iterative Method = String "BiCGStab"
  Linear System Max Iterations = Integer 350
  Linear System Convergence Tolerance = Real 1.0e-8
  Linear System Abort Not Converged = Logical True
  Linear System Preconditioning = String "ILU0"
  Steady State Convergence Tolerance = Real 1.0e-05
End
```

The name `U` is used for the solution of the convective transport problem. The value of the keyword `Time Derivative Order` must be 2. The use of the keywords `Advection Variable` and `Advection` is similar to that explained in connection with the second solver section.

The Material section is used to declare the material properties and, as the velocity vector in the convection operator is of constant type, also the components of the velocity vector. In the case $k = 0.01$ the contents of this section is

```
Material 1
  Density = Real 1
  Heat Capacity = Real 1
  Heat Conductivity = Real 0.01
  Advection Velocity 1 = Real 1
  Advection Velocity 2 = Real 1
End
```

Body Force section is used to declare the body force in the Poisson equations.

```
Body Force 1
  Heat Source = Real 1
End
```

Finally, the initial conditions and boundary conditions are specified. The temperature at $t = 0$ is defined by giving the declaration

```
Initial Condition 1
  Temperature = Real 0
End
```

Two Boundary Condition sections are needed. The first one is used to prescribe the boundary conditions on the part of inflow boundary where the temperature is given:

Table 23.1: The maximum temperature at $t = 2.0$. For comparison the maximum temperature according to the steady state solution is also recorded.

Method	k		
	0.05	0.025	0.01
OS	1.0235	1.0424	1.0677
S	1.0269	1.0436	1.0418
S(steady state)	1.0279	1.0437	1.0418
RFB	1.0271	1.0458	1.0853
RFB(steady state)	1.0286	1.0462	1.1050

```
Boundary Condition 1
  Target Boundaries(3) = Integer 1 2 4
  Temperature = Real 0
  Udot0 1 = Real 0
  U 1 = Real 0
End
```

The rate of change of the temperature is zero on this part of the boundary as the temperature is kept fixed. Thus the zero boundary conditions for `Udot0` are defined. The boundary value of the solution to the convective transport problem should equal to the temperature. Therefore zero boundary conditions for `U` are also defined.

The second Boundary Condition section is used to define the Dirichlet boundary conditions on the outflow boundary:

```
Boundary Condition 2
  Target Boundaries(1) = Integer 3
  Temperature = Real 0
End
```

Note that the zero heat flux condition need not be specified explicitly. Similarly, the treatment of the outflow boundary conditions for the wave-like equation are handled automatically by the code and need not be specified.

Results

From a numerical point of view the solution of the problem becomes increasingly difficult as the heat conductivity k tends to zero. In order to examine possible dependence on the heat conductivity parameter the problem was solved in three cases with k taking values 0.05, 0.025 and 0.01. For comparison the same case was solved using three alternative methods. Here the operator splitting scheme is referred to as OS, S is the stabilized finite element method and RFB is the method based on the residual-free-bubbles formulation. In the case of methods S and RFB the simulation was performed using 100 time steps which corresponds to the number of 200 time steps used in the case of operator splitting scheme.

The maximum temperature at $t = 2.0$ is recorded in Table 23.1. The CPU time used in the simulation to obtain solution for a certain value of the heat conductivity is shown in Table 23.2.

Discussion

The benefit of the wave-like equation formulation of the convective transport problem is that this formulation can be discretizated without using stabilized finite element formulations. Thus all the subproblems arising from operator splitting can be solved using standard FE techniques. On the other hand, the expense of this approach is that one is lead to handle the second order in time equation.

Table 23.2: CPU time used by the different methods for simulation ($k = 0.01$).

Method	CPU
OS	211.13
S	82.16
RFB	340.70

In the current implementation of the method the wave-like equation is discretizated in time using the Crank-Nicolson scheme which is expected to perform well if the solution to the convective transport problem is smooth. Spurious oscillations may however occur in the case of a rough solution changing rapidly in short length scales. The user of the method should be aware that the deterioration of accuracy may thus occur if the solution is not smooth and $\epsilon = k/||\vec{u}||_\infty \longrightarrow 0$, meaning that convection dominates.

In the cases considered convection dominates, the parameter ϵ ranging between $3.5 \cdot 10^{-2}$ and $7.1 \cdot 10^{-3}$, and the solution has also sharp boundary layer near the upper edge. No spurious oscillations are however detected in the solution. Nevertheless, the results recorded in Table 23.1 indicate that the differences between the methods become larger as $\epsilon \longrightarrow 0$.

In view of the results shown in Table 23.2 the accuracy of the operator splitting scheme in predicting the maximum temperature is comparable to that of the residual-free-bubbles method while the computational cost measured in CPU time is reduced by approximately 40 % when the operator splitting scheme is used.

It should be noted that the operator splitting scheme has the favourable feature that small spurious oscillations present in the solution after pure convection step may naturally be damped by the step involving diffusion phenomena. Note also that each of the subproblems arising from the operator splitting may be solved very efficiently using multigrid techniques, whereas robust multigrid solvers amenable to solving linear systems arising from direct discretization of the convection-diffusion equation are still to be found. This makes the operator splitting scheme attractive in the solution of problems having a very large number of unknowns.

Tutorial 24

Temperature distribution with BEM

Directory: PoissonBEM
Solvers: PoissonBEMSolver
Tools: ElmerGrid, editor
Dimensions: 2D

Case definition

This tutorial uses boundary element method (BEM) to solve Poisson equation. Even though Elmer is primarily a finite element software the are limited support also for BEM computation. One should however note that Elmer does not include any multilevel strategies essential for a good performance. For more details about BEM check the Elmer Models Manual. The simulation setting is described in Figure 24.1. A heater with constant heat flux is placed inside a box and the walls of the box are in fixed temperature. We are interested in the temperature distribution in the medium around the heater (Ω) and on the surface of the heater (Γ_1). We also want to know the heat flux through the walls of the box (Γ_2).

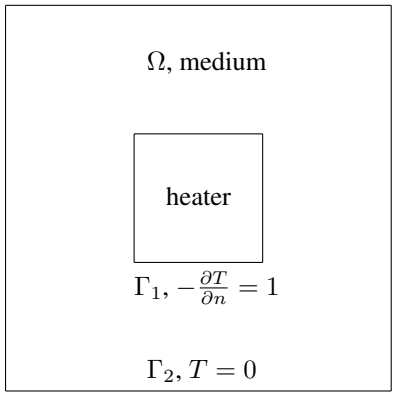

Figure 24.1: Simulation setting

Solution Procedure

First we create a mesh with ElmerGrid. The mesh is defined in `heater.grd` and it is created with command

```
ElmerGrid 1 2 heater
```

The solver input file `PoissonBEM.sif` starts with the definition of the mesh directory.

```
Header
  Mesh DB "." "heater"
End
```

The simulation uses 2D cartesian geometry, searches a steady state and since there is no coupled solvers only one iteration is needed. Numerical results are written to file `BEM_Temperature.result` and ElmerPost file is `BEM_Temperature.ep`.

```
Simulation
  Coordinate System =  Cartesian 2D
  Coordinate Mapping(3) = 1 2 3

  Simulation Type = Steady
  Steady State Max Iterations = 1

  Output Intervals = 1
  Post File = "BEM_Temperature.ep"
  Output File = "BEM_Temperature.result"
End
```

There is just one body, the medium around the heater, and it uses equation 1.

```
Body 1
  Name = "medium"
  Equation = 1
End
```

In equation block we say that we use the solver named `PoissonBEM`.

```
Equation 1
  PoissonBEM = Logical True
End
```

In solver block the `Equation` keyword must match the one in equation block. We also need to define the procedure, name the variable (`Temperature`) and tell the degrees of freedom of the variable. Keyword `Optimize Bandwidth` must be set to false with BEM solver. Since we were interested in the flux, we must now export it to the results. The lines beginning `Exported` must be exactly as below. Keywords beginning `Linear System` can be used except that the preconditioning cannot be ILU.

```
Solver 1
  Equation = PoissonBEM
  Procedure = "PoissonBEM" "PoissonBEMSolver"
  Variable = Temperature
  Variable DOFs = 1

  Optimize Bandwidth = False

  Exported Variable 1 = String Flux
  Exported Variable 1 DOFs = 1

  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Preconditioning = Jacobi
  Linear System Max Iterations = 100
  Linear System Convergence Tolerance = 1.0e-8

  Steady State Convergence Tolerance = 1.0e-6
End
```

Finally we give the boundary conditions for the heater surface and for the walls of the box. The keyword `Body Id` tells the reference body of this boundary. Here it is 1. The keyword `Normal Target Body` tells the direction of the outer normal. Value -1 means the side where there are no volume elements. We didn't mesh the inside of the heater and so we can use value -1 in both cases. The heat flux from heater to medium is 1 and the walls of the box are set to zero temperature. The keyword `Temperature` matches the name of the variable in solver block.

```
Boundary Condition 1
  Name = "heater_surface"
  Target Boundaries = 1

  Body Id = 1
  Normal Target Body = Integer -1
  Flux = Real 1
End

Boundary Condition 2
  Name = "box_walls"
  Target Boundaries = 2

  Body Id = 1
  Normal Target Body = Integer -1
  Temperature = 0
End
```

Results

Problem is solved with command `Solver`. The results are then viewed with `ElmerPost`. In Figure 24.2 is the temperature distribution.

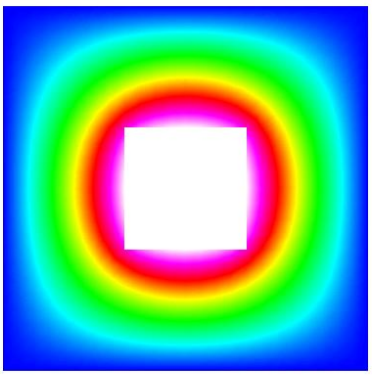

Figure 24.2: The temperature distribution.

Tutorial 25

Adding user defined equation solver

Directory: Temperature1D
Solvers: PoissonSolver
Tools: Editor, Fortran 90 compiler, ElmerGrid
Dimensions: 1D, Steady-state

Problem description

This tutorial is about creating the code for a simple poisson equation solver. The solver is applied to 1d case with internal source term and fixed boundaries.

Mathematically the problem we solve is

$$\begin{cases} -\Delta\Phi &= f & \text{in } \Omega \\ \Phi &= 0 & \text{on } \Gamma \end{cases} \tag{25.1}$$

Allthough this example is in 1d the same solver code also applies to 2D and 3D problems.

Solution procedure

Own codes solving some specific equation may be added dynamically to Elmer software. Here we create a very simple equation solver code. The final code may be found in the tutorial directory as well as the files for running the example. The solution may be attempted as follows:

- Copy all the files from tutorial directory to current directory
- Setup Elmer
- Give the following commands:

```
elmerf90 -o Poisson Poisson.f90
ElmerGrid 1 2 1dheat
ElmerSolver
ElmerPost
```

The solver code

The example Fortran code may be found in the tutorial files under the name Poisson.f90. The example run is defined in 1dheat.sif. Only a rough guidline is given here of both of the files, refer to the files themselves for more details.

All the equation solvers in Elmer have the following common interface

```
SUBROUTINE PoissonSolver( Model, Solver, dt, TransientSimulation )
  USE SolverUtils

  TYPE(Model) :: Model
  TYPE(Solver_t), POINTER :: Solver
  REAL(KIND=dp) :: dt
  LOGICAL :: TransientSimulation

    ...
END SUBROUTINE PoissonSolver
```

The argument Model contains pointers to the whole definition of the Elmer run. The argument Solver contains parameters specific to our equation solver. The argument dt and TransientSimulation are the current timestep size, and a flag if this run is steady or transient. These don't concern us this time.

When starting the ElmerSolver looks the solver input (.sif) file for a Solver section with keyword "Procedure". This should contain reference to the compiled code

```
Procedure = "Poisson" "PoissonSolver"
```

where the first string in the right hand side is the file name of the compiled code, and second argument is the name of the subroutine to look for in the given file.

In the Solver section one also gives the name of the field variable (here Poisson) and the DOFs/node (here 1).

The basic duty of the equation solver is to solve one or more field variables inside the time progressing- or steady state iteration-loop of ElmerSolver. Here we use FEM to discretize the Poisson equation and finally solve the equation by calling ElmerSolver utility SolveSystem.

The solution progresses the following way:

- Get the space for variables and temporaries from ElmerSolver and compiler. The matrix structure and space for solution and RHS vector have already been allocated for you before you enter the equation solver.

  The matrix is of type Matrix_t and may be obtained from the arguments as

  ```
  TYPE(Matrix_t), POINTER :: StiffMatrix
  StiffMatrix => Solver % Matrix
  ```

  Usually one doesn't need to know the internal storage scheme or the fields of the Matrix type, but one just passes this pointer further to ElmerSolver utility routines.

  Similarly, the force vector may be accessed as follows:

  ```
  REAL(KIND=dp), POINTER :: ForceVector(:)
  ForceVector => StiffMatrix % RHS
  ```

  The solution vector is obtainable similarily

  ```
  TYPE(Variable_t), POINTER :: Solution
  Solution => Solver % Variable
  ```

  The Variable_t structure contains the following fields

  - DOFs: the number of degrees of freedom for one node. This value is for information only and should'nt be modified.
  - Perm: an integer array that is nonzero for nodes that belong to computational volume for this equation. The entry $Perm(i)$ holds the index of the global matrix row (for 1 DOF) for nodal point i. This array should'nt be modified by the equation solver.

  - Values: Space for the solution vector values. Note that the values are ordered the same way as the matrix rows, .i.e. the value of Potential at node n is stored at

```
val = Solution % Values( Solution % Perm(n) )
```

- Initialize the global system to zero. Calling the utility routing

```
CALL InitializeToZero( StiffMatrix, ForceVector )
```

  is usually enough.

- Go trough the elements for which this equation is to be solved, get the elemental matrices and vectors and add them to the global system:

```
DO i=1,Solver % NumberOfActiveElements
   CurrentElement => Solver % Mesh % Elements( Solver % ActiveElements(i) )
      ...
   CALL LocalMatrix( ... )
   CALL UpdateGlobalEquations( ... )
END DO
CALL FinishAssembly( ... )
```

  Here the LocalMatrix is your own subroutine computing elemental matrices and vectors. In the example code LocalMatrix uses three routines from ElmerSolver utilities. The function

```
dim = CoordinateSystemDimension()
```

  returns the dimension of the current coordinate system, i.e. the return value is 1, 2 or 3 depending on the input file setting of keyword "Coordinate System". The function GaussPoints returns structure containing the integration point local coordinates and weights

```
TYPE(GaussIntegrationPoints_t) :: IntegStuff
IntegStuff = GaussPoints( Element )
```

  The fields of the type GaussIntegrationPoints_t are

```
INTEGER :: n
REAL(KIND=dp) :: u(:), v(:), w(:), s(:)
```

  the integer value n is the number of points selected. The arrays u,v and w are the local coordinates of the points, and the array s contains the weights of the points. One may call the GaussPoints-routine with second argument,

```
IntegStuff = GaussPoints( Element, n )
```

  if the default number of integration points for given element is not suitable.

  Inside the integration loop the function ElementInfo is called:

```
TYPE(Element_t), POINTER :: Element
TYPE(Nodes_t) :: Nodes
REAL(KIND=dp) :: U,V,W,detJ, Basis(n), dBasisdx(n,3), ddBasisddx(n,3,3)

stat = ElementInfo( Element, Nodes, U, V, W, detJ,  &
     Basis, dBasisdx, ddBasisddx, .FALSE. )
```

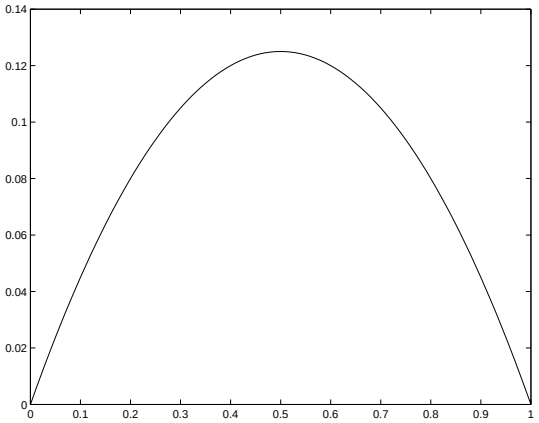

Figure 25.1: Solution of the Poisson Equation.

This routine returns determinant of the element jacobian (detJ), basis function values (Basis(n)), basis function global derivative values (dBasisdx(n,3)), basis function second derivative values (ddBasisddx(n,3,3)). The second derivatives are only computed if the next logical flag is set to true. All the values are computed at the point U,V,W inside element defined by structures Element and Nodes.

Refer to the code for more details.

- Set boundary conditions. Here only dirichlet boundary conditions are used. These may be set by using the utility routine SetDirichletBoundaries.
- Solve the system by calling utility routine SolveSystem.

Results

In the elmerpost file there is a variable called Potential which contains the solution of this simple example. See figure 25.1

Tutorial 26

Volume flow boundary condition

Directory: FlowLinearRestriction
Solvers: FlowSolve, SolveWithLinearRestriction
Tools: Editor, Fortran 90 compiler, ElmerGrid
Dimensions: 2D, Transient

Case definition

This tutorial gives an example how to use SolveWithLinearRestriction. It also describes how to execute own functions before the original system is solved. In order to understant the case reader should be familiar with compressed row storage matrixes and elmer basics. This tutorial gives only the guidelines and reader is adviced to read the files in order to get more through understanding.

We simulate the flow of incompressible fluid in a pipe. The pipe has a length of 5 and a width of 1. On the left end we want to describe a certain timedependent volume flow. In other words, we don't want to describe the velocity field here but we want the velocity field be such that it transports certain amount of volume in timeinterval. We could integrate the correct volume flow, but let's now approximate it to make the more important aspects more visible. Our approximation here is that the volume flow is proportional to average velocity on the edge i.e.

$$\frac{1}{N}\sum_{i=1}^{N} u_i = \frac{volume}{time} \tag{26.1}$$

Here u_i are the nodal velocities parallel to the pipe on the left edge and N is the number of nodes on the left edge. We want to set a nicely scaled sinusoidal volume flow on the edge, which leads to

$$\sum_{i=1}^{N} u_i = 10N\sin(2\Pi t) \tag{26.2}$$

This equation we can (easily) force with lagrange multiplier.

Solution procedure

First we make a uniform mesh of 800 four-node quadrilaterals with command

```
ElmerGrid 1 2 mflow
```

Next we construct the solver input file. Header is simply

```
Header
  Mesh DB "." "mflow"
End
```

The simulation block is also very simple. Here we need to define the timestepping method and timescale.

```
Simulation
  Coordinate System = Cartesian 2D

  Simulation Type = Transient
  Steady State Max Iterations = 1

  Timestepping Method = BDF
  BDF Order = 1

  Timestep Sizes = 0.02
  Timestep Intervals = 100

  Output Intervals = 1

  Output File = "mflow.result"
  Post File = "mflow.ep"
End
```

The body, material and equation blocks are as usual. The material parameters, of course, have affect on the solution and interested reader is encouraged to modify these values and recalculate the solution.

```
Body 1
  Material = 1
  Equation = 1
End

Material 1
  Density = 3.0
  Viscosity = 0.1
End

Equation 1
  Navier-Stokes = TRUE
  Active Solvers(1) = 1
End
```

The solver block has the usual Navier-Stokes keywords and two keywords for volume flow boundary. The `Before Linsolve` keyword defines binaryfile and function that is called before the system is solved. This function we must write and compile and we will come to it shortly. The following keyword, `Export Lagrange Multiplier`, states that we are not interested in the value of the Lagrenge multiplier and it is therefore not saved.

```
Solver 1
  Equation = Navier-Stokes
  Stabilize = True

  Before Linsolve = "./AddMassFlow" "AddMassFlow"
  Export Lagrange Multiplier = Logical FALSE

  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Preconditioning = ILU1
  Linear System Max Iterations = 500
  Linear System Scaling = False
  Linear System Convergence Tolerance = 1.0e-8
```

```
  Nonlinear System Max Iterations = 15
  Nonlinear System Convergence Tolerance = 1.0e-8
  Nonlinear System Newton After Tolerance = 1.0e-4
  Nonlinear System Newton After Iterations = 8
  Nonlinear System Relaxation Factor = 1.0

  Steady State Convergence Tolerance = 1.0e-7
End
```

In boundary conditions we state that both parallel and perpendiculer velocities are zero on the pipe sides and on both edges the perpendicular velocity is zero. Here we also define the number tags for the boundaries. The tag 2 is assigned to boundary that has number 4 in grd-file, which is the left edge of the pipe. To this tag number 2 we shall refer in our AddMassFlow-function.

```
Boundary Condition 1
  Target Boundaries(2) = 1 3
  Velocity 1 = 0.0
  Velocity 2 = 0.0
End

Boundary Condition 2
  Target Boundaries = 4
  Velocity 2 = 0.0
End

Boundary Condition 3
  Target Boundaries = 2
  Velocity 2 = 0.0
End
```

AddMassFlow function

Here we shall only give some rough guidelines of the function, for more information check the code. This function creates the constraint matrix and RHS that forces the equation mentioned above. Then it calls SolveWithLinearRestriction to solve the system. The coinstraint matrix is actually only a row-vector and the RHS is only one value.

- The function parameters are defined in Elmer so you shouldn't change them.
- First we set a pointer to EMatrix-field of the given system matrix. If the pointed matrix is not yet allocated, calculate the number of nodes on the edge we want to define the volume flow. This gives us the number of non-zeros in our constraint matrix and we can allocate the matrix.
- Set the rows, cols and diag -fields of the matrix. This sets the non-zeros on their right places in the constraint matrix.
- Set all values of the constraint matrix to unity.
- Calculate the RHS-value. The current time was checked in the beginning of the function, so this is possible.
- Call SolveWithLinearRestriction
- Return 1 which tells the ElmerSolver that the system is already solved.

The function is the compiled with command

```
elmerf90 -o AddMassFlow AddMassFlow.f90
```

Here it is assumed that the source file name is AddMassFlow.f90.

Results

Just say `ElmerSolver` and you should get the solution in few minutes. The velocity perpendicular to the pipe is practically zero and the velocity parallel to the pipe is an example of Womersley velocity profile [1]. An interesting feature of this velocity profile is that on some timesteps the fluid flows to both directions, see figure 26.1.

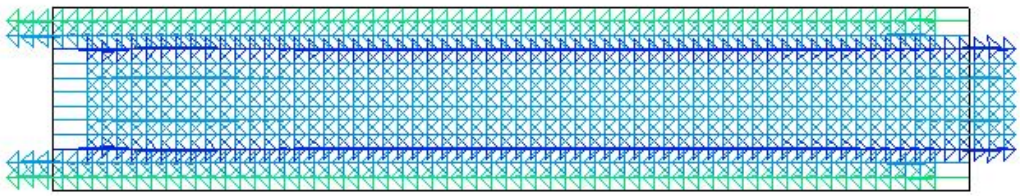

Figure 26.1: Solution of the velocity field. Note the flow to both directions.

[1] J.Physiol (1955) 127, 553-563

Tutorial 27

Streamlines

Directory: FlowStreamlines
Solvers: StreamSolver, FlowSolve
Tools: ElmerGrid, editor
Dimensions: 2D

Case definition

The case definition is the same as in the incompressible flow passing a step. The mathematical definition of the stream function ψ is

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{27.1}$$

where u, v are the velocity components in x, y geometry. For more info check Elmer Models Manual.

Solution Procedure

First we create a mesh with ElmerGrid. The mesh is defined in `step.grd` and it is created with command

```
ElmerGrid 1 2 step
```

You may need to compile the StreamSolver yourself. If the Elmer environment is succesfully setup the compilation command should look like the following lines,

```
elmerf90 -o StreamSolver StreamSolver.f90
```

The solver input file `streamlines.sif` starts with the definition of the mesh directory.

```
Header
  Mesh DB "." "step"
End
```

The simulation uses 2D cartesian geometry and searches a Steady State. There is no coupled solvers so only one iteration is needed. Numerical results are written to file `streamlines.result` and ElmerPost file is `streamlines.ep`.

```
Simulation
  Coordinate System =  Cartesian 2D
  Coordinate Mapping(3) = 1 2 3

  Simulation Type = Steady
  Steady State Max Iterations = 1
```

```
  Output Intervals = 1
  Post File = "streamlines.ep"
  Output File = "streamlines.result"
End
```

There is just one body and it uses equation 1 and is of material 1.

```
Body 1
  Equation = 1
  Material = 1
End
```

The equation block states that we use Solvers 1 and 2 to solve the problem and that we use Navier-Stokes equations.

```
Equation 1
  Active Solvers(2) = 1 2
  Navier-Stokes = True
End
```

In material block we define the density and the viscosity of the fluid.

```
Material 1
  Density = 1
  Viscosity = 0.01
End
```

Solver 1 is for the Navier-Stokes equations. Here we give the linear system solver [1] and convergence criterions for linear, nonlinear and steady state solution of the Navier-Stokes equations.

```
Solver 1
  Equation = "Navier-Stokes"
  Stabilize = True

  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Max Iterations = 500
  Linear System Convergence Tolerance = 1.0e-8
  Linear System Preconditioning = ILU1

  Nonlinear System Convergence Tolerance = 1.0e-6
  Nonlinear System Max Iterations = 15
  Nonlinear System Newton After Iterations = 8
  Nonlinear System Newton After Tolerance = 1.0e-4
  Nonlinear System Relaxation Factor = 1.0

  Steady State Convergence Tolerance = 1.0e-6
End
```

Then the solver for streamlines.

- Name of the equation. This may be what ever you like.
- Name of the binary file and the subroutine. If you compiled the StreamSolver yourself, then you may need to change this to `Procedure = "./StreamSolver" "StreamSolver"`.
- Name of the variable. This may be what ever you like.

[1] Biconjugate gradient method with incomplete LU preconditioning

- Stream function is scalar, so the degree of freedom is 1.

Next set of keywords is for the StreamSolver. More info on keywords is in the Elmer Models Manual.

- Name of the flow field variable. The name of the FlowSolves variable is FlowSolution.
- Global number of the offset node. 1 is always a safe choise.
- Shift the smallest value to zero.
- Scale the maximum value to 1.
- Use the normal stream function i.e. don't use Stokes stream function.

Then we define the linear system solver and convergence criterions.

```
Solver 2
  Equation = "StreamSolver"
  Procedure = "StreamSolver" "StreamSolver"
  Variable = "StreamFunction"
  Variable DOFs = 1

  Stream Function Velocity Variable = String "Flow Solution"
  Stream Function First Node = Integer 1
  Stream Function Shifting = Logical TRUE
  Stream Function Scaling = Logical TRUE
  Stokes Stream Function = Logical FALSE

  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Max Iterations = 500
  Linear System Convergence Tolerance = 1.0e-8
  Linear System Preconditioning = ILU1

  Steady State Convergence Tolerance = 1.0e-6
End
```

Finally we give the boundary conditions. The condition 1 is for the lower and upper side of the step (Γ_1,Γ_2,Γ_3,Γ_5 in case definition). Here both velocities are zero. The condition 2 is for the output edge (Γ_4). Here vertical velocity is zero. The condition 3 is for the input edge (Γ_6). Here horzontal velocity is 1 and vertical velocity is zero.

```
Boundary Condition 1
  Target Boundaries = 1
  Velocity 1 = 0
  Velocity 2 = 0
End

Boundary Condition 2
  Target Boundaries = 2
  Velocity 2 = 0
End

Boundary Condition 3
  Target Boundaries = 3
  Velocity 1 = 1
  Velocity 2 = 0
End
```

Results

Problem is solved with command `Solver`. The results are then viewed with ElmerPost. In figure 27.1 are some contour lines of the stream function. These are also flows streamlines. The contour values are manually selected to get a nice picture. Note the swirl after the step.

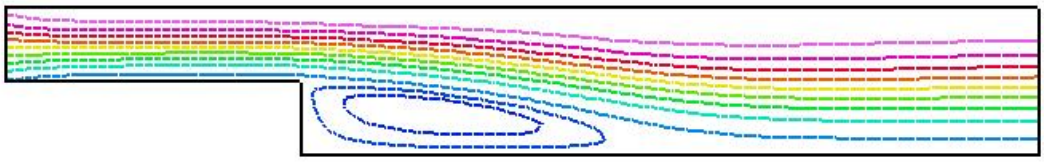

Figure 27.1: The streamlines of the flow.

Tutorial 28

Electroosmotic flow and advected species

Directory: Microfluidic
Solvers: StatElecSolve, FlowSolve , AdvectionDiffusion, Electrokinetics
Tools: ElmerGrid, editor
Dimensions: 2D

Case definition

This tutorial is an example of setting up a simulation for (microfluidic) electroosmotic flow advecting a passive scalar quantity. Diffusion of the species is also included. The geometry of the system is a simple 2D microchannel with T crossing. The flow is induced by the applied electric field and the electric double layer at the channel walls. The analyte (species) is inserted into the system from the left hand side inlet.

More details on the electrokinetic capabilities of Elmer are found on the Models Manual, chapter "Electrokinetics".

Solution Procedure

The computatonal mesh is done with `ElmerGrid` in directory `Tcross` with the command

```
ElmerGrid 1 2 Tcross -scale 1e-5 1e-5 1e-5
```

The scale option above is used to obtain appropriate dimensions from a geometry template defined in nondimensional units.

The command file may be written with a text editor. The file includes the following information.

The mesh directory is given in the header of the command file

```
Header
  Mesh DB "." "Tcross"
End
```

The simulation block of the command file defines, eg., the case to be time dependent (transient) with 10^{-5} second timesteps and altogether 120 time intervals. Results from every second timestep are saved into the files `diffusion1.*`.

```
Simulation
  Coordinate System = Cartesian 2D

  Simulation Type = "Transient"
  Steady State Max Iterations = 20
```

```
  Timestep Intervals = 120
  Timestep Sizes = 1e-5
  Output Intervals = 2

  Timestepping Method = BDF
  BDF Order = 2

  Binary Output = Logical True

  Output File = "diffusion1.res"
  Post File = "diffusion1.ep"

  Output Version Numbers = Logical True
  Max Output Level = 32
End
```

The electrostatics and electrokinetics solvers require the value of the permittivity of vacuum. This is actually not even needed here since the case deals with conducting media. Thus the value has been fixed as 1.0 to avoid warnings on missing constant definitions.

```
Constants
!  Permittivity Of Vacuum = 8.8542e-12  ! C^2/Nm^2
  Permittivity Of Vacuum = 1.0 ! manipulation for conducting material
End
```

The case includes only one body. The corresponding equation definitions are found in section `Equation 1` and material parameters from section `Material 1`.

```
Body 1
  Equation = 1
  Material = 1
End
```

All three solvers are active in this equation set. Further definitions include, first, that the convection of the species is switched on (besides diffusion). Then that for Navier-Stokes equations the convective term may be left out resulting in laminar Stokes flow, and finally that the elctric field is computed by the electrostatics rather than given by the user.

```
Equation 1
  Active Solvers(3) = 1 2 3
  Convection = Computed
  NS Convect = False
  Electric Field = String "computed"
End
```

Following are the solver definitions. Solver 1 is the electrostatics solver. The equation is linear and thus no nonlinear iterations are needed. The equation is solved using a fast direct method UMFPack.

```
Solver 1
  Equation = "Stat Elec Solver"
  Procedure = "StatElecSolve" "StatElecSolver"
  Variable = String "Potential"
  Variable DOFs = 1

  Calculate Electric Field = True
  Calculate Electric Flux = False
  Linear System Convergence Tolerance = 1.0E-10
```

```
  Linear System Solver = Direct
  Linear System Direct Method = UMFPack
  Linear System Preconditioning = ILU1
  Linear System Residual Output = 1

  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-10
  Steady State Convergence Tolerance =  1.0E-10
End
```

The next solver is for the Navier-Stokes equations. Here nonlinear iterations are required.

```
Solver 2
  Equation = "Navier-Stokes"

  Linear System Convergence Tolerance = 1.0D-08
  Linear System Solver = Iterative
  Linear System Iterative Method = "BiCGStab"
  Linear System Max Iterations = 500
  Linear System Abort Not Converged = False  ! True
  Linear System Preconditioning = ILU1
  Linear System Residual Output = 10

  Nonlinear System Convergence Tolerance = 1.0e-6
  Nonlinear System Max Iterations = 30
  Nonlinear System Newton After Iterations = 10
  Nonlinear System Newton After Tolerance =  Real 1.0D-8
  Nonlinear System Relaxation Factor = 1.0
  Steady State Convergence Tolerance =  1.0D-03

  Stabilize = True
End
```

The advection-diffusion equation does not affect neither the electrostatic field nor the flow, thus it may be solved only after a converged solution for the previous two equations is available. This is achieved with the `Exec Solver` definition below. The advected quantity is given the name Analyte. The advection-diffusion solver uses bubble stabilization method to avoid numerical problems associated with convection type equations.

```
Solver 3
  Exec Solver = After Timestep
  Equation = "Analyte transfer"
  Procedure = "AdvectionDiffusion" "AdvectionDiffusionSolver"
  Variable = String "Analyte"
  Variable DOFs = 1

  Bubbles = True

  Linear System Convergence Tolerance = 1.0E-06
  Linear System Solver = "Iterative"
  Linear System Iterative Method = "BiCGStab"
  Linear System Max Iterations = 500
  Linear System Preconditioning = ILU2
  Linear System Residual Output = 1
```

```
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-5
  Steady State Convergence Tolerance =  1.0D-06
End
```

The material parameters are given below.

```
Material 1
  Density = 1e3

  Viscosity = 1e-03

  Relative Permittivity = 1.0    !  this is actually electric conductivity

  Analyte Diffusivity = Real 1e-10
End
```

Finally the boundary conditions are defined. The first BC is given for the channel walls. Here, tangential velocity (velocity components 1 and 2) is computed by the Helmholtz-Smoluchowski slip velocity condition, which means that the velocity is computed using the computed electric field and the electroosmotic mobility as inputs.

```
Boundary Condition 1
  Name = "channel-walls"
  Target Boundaries(2) = 4 5

  EO Mobility = Real 5e-08

  Velocity 1 = Variable Pressure
      Real Procedure  "Electrokinetics" "helmholtz_smoluchowski1"
  Velocity 2 = Variable Pressure
      Real Procedure  "Electrokinetics" "helmholtz_smoluchowski2"
End
```

The next BC is the inlet condition. We give a potential of 100 Volts and define that there is no flow in y-direction. The analyte concentration at the inlet is defined as a function of time using table format. The concentration is 1.0 up until time instant $3 \cdot 10^{-5}$, is zero after $4 \cdot 10^{-5}$s and decreases linearly between these two time instants.

```
Boundary Condition 2
  Name = "el_A"
  Target Boundaries = 1

  Potential = 100.0

  Velocity 2 = 0.0

  Analyte = Variable Time
    Real
      0.0      1.0
      3.0e-5   1.0
      4.0e-5   0.0
      0.5      0.0
    End

End
```

The final two boundary conditions are for the outlets. Different potentials for these are defined as well as a condition for velocity component.

```
Boundary Condition 3
  Name = "el_B"
  Target Boundaries = 2

  Potential = 30.0

  Velocity 1 = 0.0
End

Boundary Condition 4
  Name = "el_C"
  Target Boundaries = 3

  Potential = 0.0

  Velocity 1 = 0.0
End
```

After writing the command file is finished, the problem can be solved by entering the command `ElmerSolver` on the command line. The results can be examined with `ElmerPost`.

Results

Solving the problems takes less than a minute cpu time on a PC. The maximum and minimum concentration over the whole simulation are 1.0235 and -0.075748. The solution of this problem should be between 0 and 1. This shows that some numerical discretization errors are present in the simulation. The errors would diminish when using smaller timesteps and also with denser mesh. Simulation results at the time instant of 0.00025 seconds are shown in Fig. 28.1.

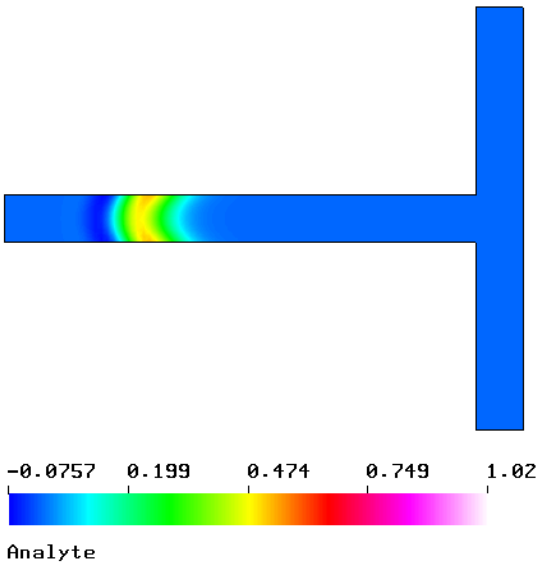

Figure 28.1: Analyte concentration on time instant 0.00025 seconds.

The maximum value of the magnitude of the velocity in the results is 0.105 m/s.

The electric field is written into the output only componentwise. The magnitude of the field may be visualised after giving the following commands on the ElmerPost command line (assuming one has read in all 61 timesteps):

```
math E(0,time(0):time(61)-1) = Electric.field.1
math E(1,time(0):time(61)-1) = Electric.field.2
math E(2,time(0):time(61)-1) = 0
math E_abs = sqrt(vdot(E,E))
```

Now visualising the variable `E_abs` reveals that the electric field magnitude is between 2453 and $2.18 \cdot 10^6$.

Notes

When checking the simulation results the user will notice that the electric potential does not change in time and that the flow reaches steady-state within a few timesteps. This is quite clear also from the problem setup: the electric field is due to invarying potential with constant material parameters. Also the flow in microchannels is usually laminar.

Thus the most efficient way to simulate the current case would be to compute first the steady-state solution for the electrokinetic flow and use the steady flow to advect the analyte. This would be done by running two separate simulations; resolve first the steady flow, and to use this flow solution as restart for the advection-diffusion equation.

Tutorial 29

Active and passive elements

Directory: PassiveElements
Solvers: HeatSolve
Tools: ElmerGrid, editor
Dimensions: 2D

Case definition

This tutorial shows an example of using passive elements in Elmer. This feature allows the activation and deactivation of parts of the geometry during the simulation. This tutorial uses the heat eqution solver to demonstrate this capability. Use with other solvers is analogous.

The geometry of the problem consists of two parts. The lower edge of the lower part is held at constant temperature of 1 degrees. The upper body is heated with a constant heating power. Between time steps 5 and 6 the two bodies are connected by two heat conductors, and the heat is conducted from the higher body to the lower one. The goal of the simulation is to model the temperature distribution in the system over time.

The problem is a pure heat transfer problem that may be solved with `HeatSolve`.

Solution Procedure

The computatonal mesh is done with `ElmerGrid` in directory `tmesh` with the command

```
ElmerGrid 1 2 tmesh
```

The command file may be written with a text editor. The file includes the following information.

The mesh directory is given in the header of the command file

```
Header
  Mesh DB "." "tmesh"
End
```

The simulation block of the command file defines, eg., the case to be time dependent (transient) with 1 second timesteps and altogether 15 time intervals.

```
Simulation
  Max Output Level = 32
  Coordinate System = Cartesian 2D
  Simulation Type = Transient
  Timestepping Method = BDF
  BDF Order = 2
  Timestep Intervals = 15
  Timestep Sizes = 1
  Output Intervals = 1
```

```
  Steady State Max Iterations = 1
  Output Version Numbers = Logical True
  Output File = heat.res
  Post File = heat.ep
End
```

The heat equation solver asks for the Stefan-Boltzmann constant that gives the relationship between temperature and radiation power, although radiation is not needed here. Let us define it anyway to avoid warnings of missing parameters.

```
Constants
  Stefan Boltzmann = 5.67e-8
End
```

There are three bodies with the same equation but different material properties. Body 3 is heated by a constant body force. Body 2 forms the connecting parts of the system. An initial condition as well as a body force is defined for this body. The body force contains the initial deactivation, and later activation, of the connecting part. Note that this part is included in the geometry all the time, but the command file is used to define when they are included into the simulation.

```
Body 1
  Equation = 1
  Material = 1
End

Body 2
  Equation = 1
  Material = 2
  Body Force = 2
  Initial Condition = 1
End

Body 3
  Equation = 1
  Material = 1
  Body Force = 1
End
```

The only solver is the heat solver (Solver 1)

```
Equation 1
  Active Solvers = 1
End
```

The initial condition for the initially passive elements is taken to be 1 degrees; the same temperature than the colder part of the system has as a boundary condition.

```
Initial Condition 1
  Temperature = 1.0
End
```

The heating power is defined to be 10 W/kg

```
Body Force 1
  Heat Source = 10
End
```

Now the passive condition for the connecting part is defined. When the parameter `Temperature Passive` has a value larger than zero, the current element is excluded from the solution, otherwise it is included as a normal element. The parameter may depend on variables, coordinates or time. Here it is defined to depend on time using a tabular format.

```
Body Force 2
  Temperature Passive = Variable Time
    Real
      0.0    1.0
      5.0    1.0
      5.2   -1.0
      8.0   -1.0
    End

End
```

The material properties of the system are artificial. The following three properties are needed for each material.

```
Material 1
  Heat Capacity = 1
  Heat Conductivity = 1
  Density = 1
End

Material 2
  Heat Capacity = 10
  Heat Conductivity = 1
  Density = 1
End
```

The heat equation is solved with an iterative method. The system is linear, thus multiple nonlinear iterations are not needed.

```
Solver 1
  Equation = heat equation
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Preconditioning = ILU0
  Linear System Max Iterations = 300
  Linear System Convergence Tolerance = 1.0e-6
  Linear System Abort Not Converged = Logical False
  Nonlinear System Max Iterations = 1
  Nonlinear System Convergence Tolerance = 1.0e-5
  Steady State Convergence Tolerance = 1.0e-5
End
```

The boundary conditions are simple. The lower boundary of the lower body is held at 1 degrees and the upper boundary of the upper body at 10 degrees.

```
Boundary Condition 1
  Target Boundaries = 1

  Temperature = 1
End

Boundary Condition 2
  Target Boundaries = 4

  Temperature = 10
End
```

After writing the command file is finished, the problem can be solved by entering the command `ElmerSolver` on the command line. The results can be examined with `ElmerPost`.

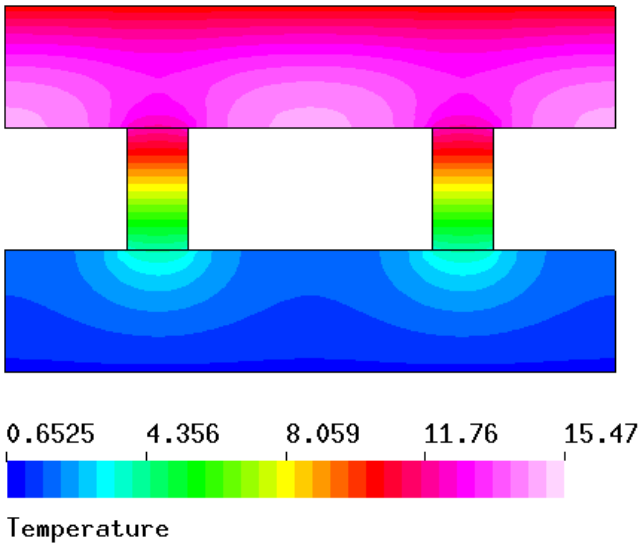

Figure 29.1: Temperature distribution of the system at the final time instant (with spectral_32 color map).

Results

With the given computational mesh the problem is solved in a few seconds. The maximum and minimum temperatures in the system over the whole simulation are 15.466 degrees and 0.6525 degrees respectively. The maximum and minimum temperature at the final time instant are 14.207 degrees and 1.000 degrees, respectively. The results at the time instant of 15 seconds are shown in Fig. 29.1.

Notes

For equations with more than one components (such as displacement for Stress Analysis solver in 2D or 3D) the passive elements feature apply to all the components. The feature is activated by defining, eg., `Displacement Passive` in the Body Force section. Note that for Navier-Stokes equations one should use `Flow Solution Passive`, and that this affects the Pressure as well as the Velocity components.

However, when using multiple solvers, one can define some of them passive and some of them active at the same time.

Index