Elmer non-GUI Tutorials

CSC – IT Center for Science

August 3, 2020

Elmer non-GUI Tutorials

About this document

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

These tuturials were coined before ElmerGUI was written (i.e. before year 2008) and they don't take use of it. See the Elmer GUI Tutorials for information how to set up the cases using ElmerGUI. These cases are not really that well fitted for tutorials but they may be usefull for power users who want to understand more deeply how models in Elmer are set up.

The present manual corresponds to Elmer software version 8.4. Latest documentations and program versions of Elmer are available (or links are provided) at `http://www.csc.fi/elmer`.

Copyright information

The original copyright of this document belongs to CSC – IT Center for Science, Finland, 1995–2019. This document is licensed under the Creative Commons Attribution-No Derivative Works 3.0 License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nd/3.0/`.

Elmer program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Elmer software is distributed in the hope that it will be useful, but without any warranty. See the GNU General Public License for more details.

Elmer includes a number of libraries licensed also under free licensing schemes compatible with the GPL license. For their details see the copyright notices in the source files.

All information and specifications given in this document have been carefully prepared by the best efforts of CSC, and are believed to be true and accurate as of time writing. CSC assumes no responsibility or liability on any errors or inaccuracies in Elmer software or documentation. CSC reserves the right to modify Elmer software and documentation without notice.

Contents

Table of Contents 2

1 Radiation heat transfer 3

2 Eigenvalue analysis of an elastic beam 7

3 Elastic linear plate 12

4 Compressible flow passing a step 16

5 Flow through a hole – determining the acoustic impedance 21

6 Electrostatics 28

7 Lossless acoustic waves 32

8 Induction heating of a graphite crucible 34

9 Thermal actuator driven with electrostatic currents 38

10 Axisymmetric coating process 43

11 Blood ejection from a ventricle into aorta 48

12 Temperature distribution with BEM 53

13 Adding user defined equation solver 56

14 Volume flow boundary condition 60

15 Streamlines 64

16 Electroosmotic flow and advected species 68

17 Active and passive elements 74

Index 78

Tutorial 1

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
```


BY-ND

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
```


CC BY-ND

```
  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. 1.1.

CC BY-ND

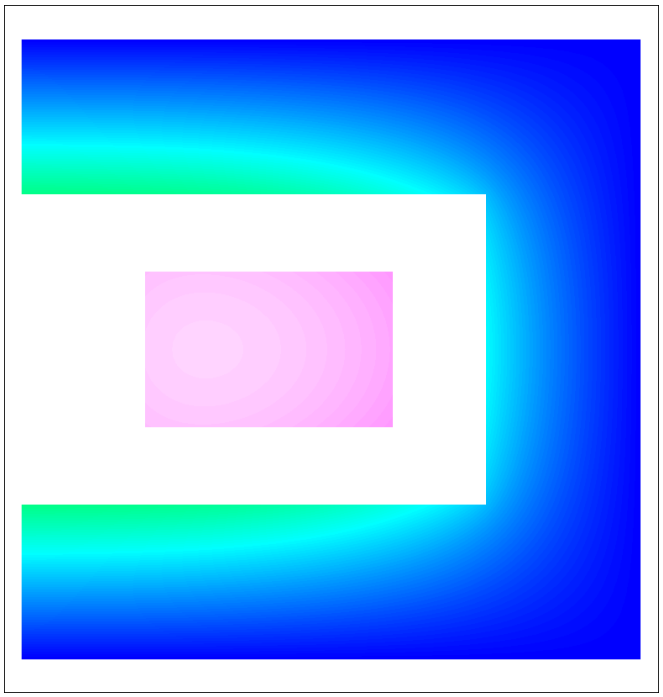

Figure 1.1: Temperature distribution in the radiation heat transfer problem

CC BY-ND

Tutorial 2

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.

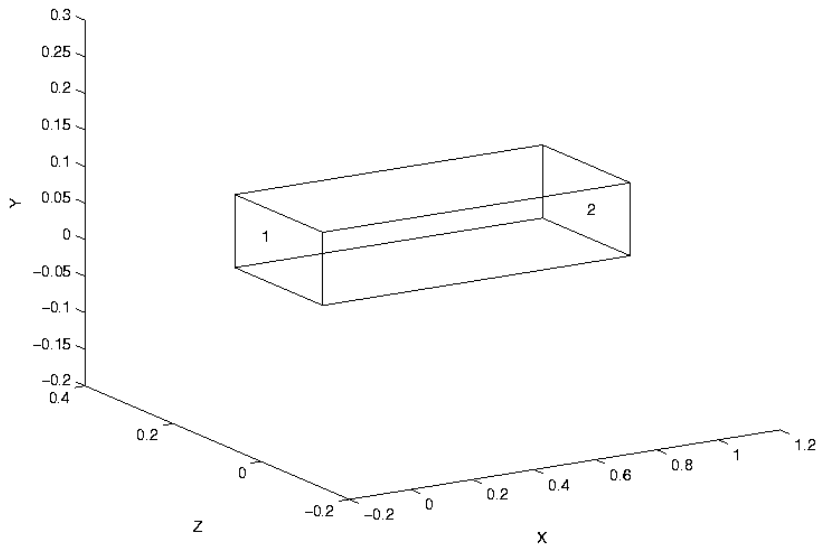

Figure 2.1: Beam.

BY-ND

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"
```


CC BY-ND

```
  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.

CC BY-ND

Results

Both the eigenvalues of the isotropic and the eigenvalues of the anisotropic model are shown below in Elmer outputs. Figure 2.2 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 2.1). 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 2.1: 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

CC BY-ND

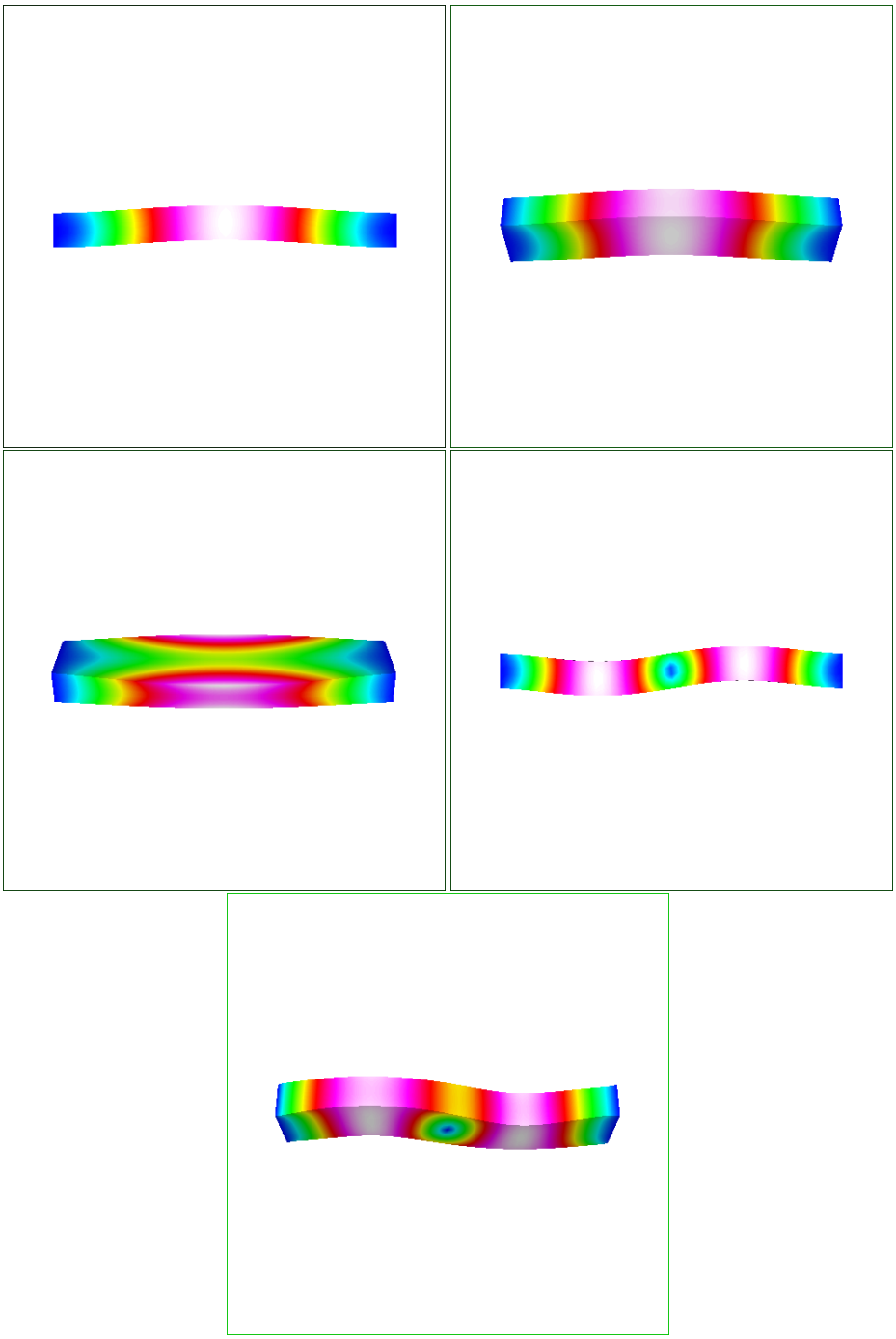

Figure 2.2: Eigenvectors

BY-ND

Tutorial 3

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 3.1 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.

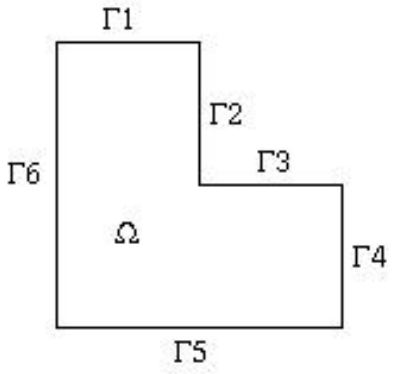

Figure 3.1: The geometry of plate and the numbering of edges.

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

CC BY-ND

```
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
```


CC BY-ND

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 3.1. 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 3.2.

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

CC BY-ND

```
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 3.2.

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`.

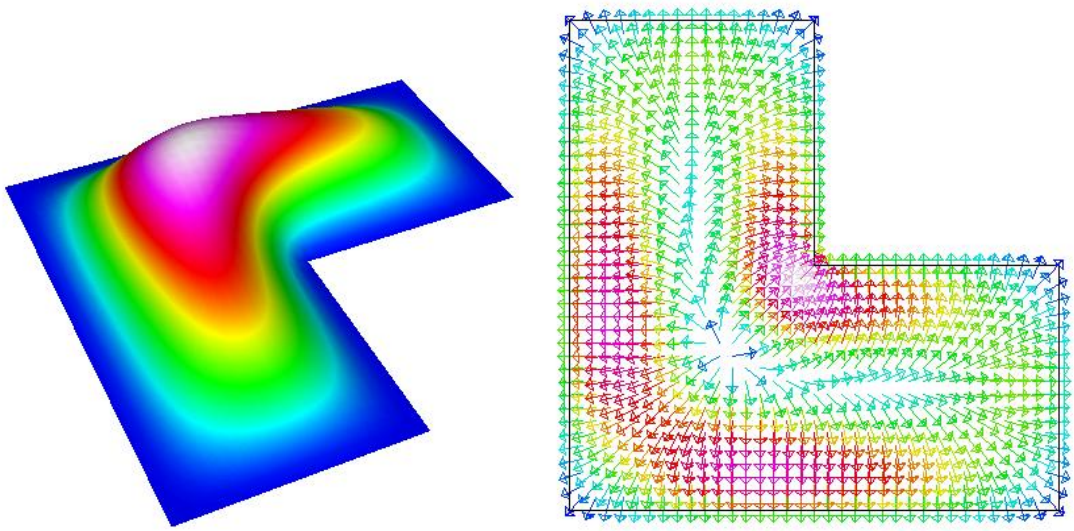

Figure 3.2: The deflection of the plate and the corresponding rotation.

BY-ND

Tutorial 4

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 4.1). The needed material parameters of air are shown in Table 4.1. 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 4.1: 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.

CC BY-ND

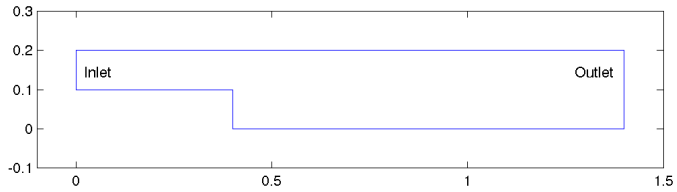

Figure 4.1: Step.

```
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
```


BY-ND

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
```


CC BY-ND

```
  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 4.2 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 4.2.

CC BY-ND

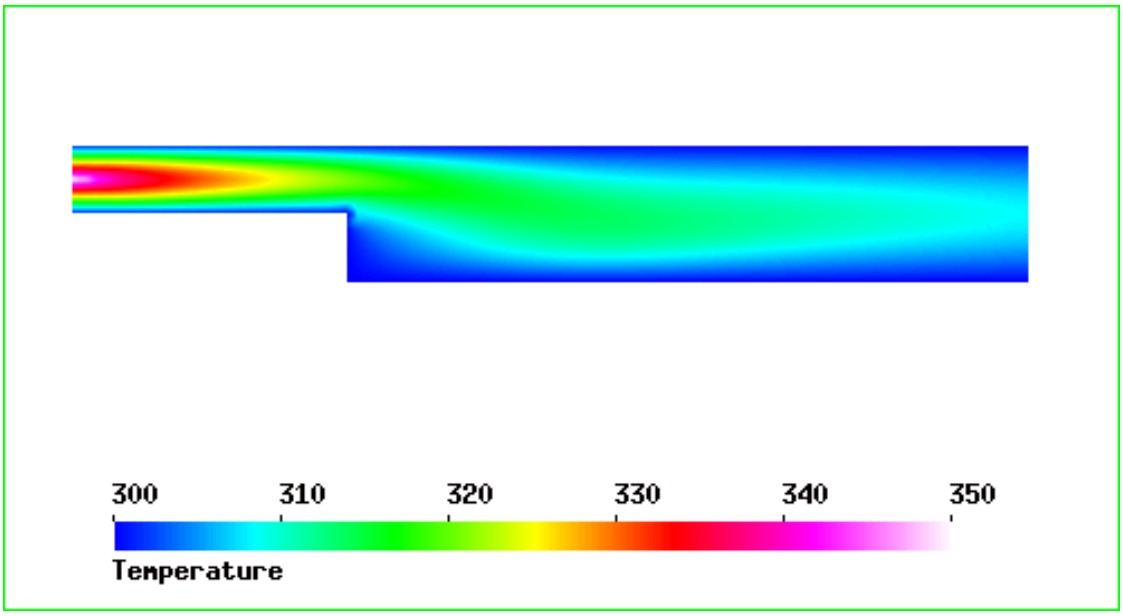

Figure 4.2: Step.

Table 4.2: 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

BY-ND

Tutorial 5

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

$$z_h = \frac{p}{v} = \frac{F}{vA_h}, \tag{5.1}$$

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

$$\rho\frac{\partial\vec{u}}{\partial t} - \nabla\cdot(2\eta\bar{\bar{\varepsilon}}) + \nabla p = \rho\vec{f}, \tag{5.2}$$

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

$$Re = \frac{\rho UL}{\eta}, \tag{5.3}$$

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

CC BY-ND

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

$$Ma = \frac{U}{a} < 0.3, \tag{5.4}$$

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

$$Pr = \frac{\eta c_p}{k}, \tag{5.5}$$

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

$$Kn = \frac{\mathcal{L}}{L}, \tag{5.6}$$

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
```


CC BY-ND

```
  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`.

CC BY-ND

```
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

$$S \cdot \vec{u} = \overline{\overline{\sigma}} \cdot \vec{n}, \tag{5.7}$$

CC BY-ND

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

$$s_i u_i = \mu \frac{\partial u_i}{\partial n}, \tag{5.8}$$

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):

$$u_i = \frac{2 - \sigma_v}{\sigma_v} \lambda \frac{\partial u_i}{\partial n}, \tag{5.9}$$

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

$$s_i = \frac{\mu}{\lambda} \frac{\sigma_v}{2 - \sigma_v}. \tag{5.10}$$

The mean free path is defined as

$$\lambda = \frac{\mu}{\rho} \sqrt{\frac{\pi M}{2RT}}, \tag{5.11}$$

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
```


CC BY-ND

```
  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 5.1. 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 5.1: 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 5.1 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

CC BY-ND

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.

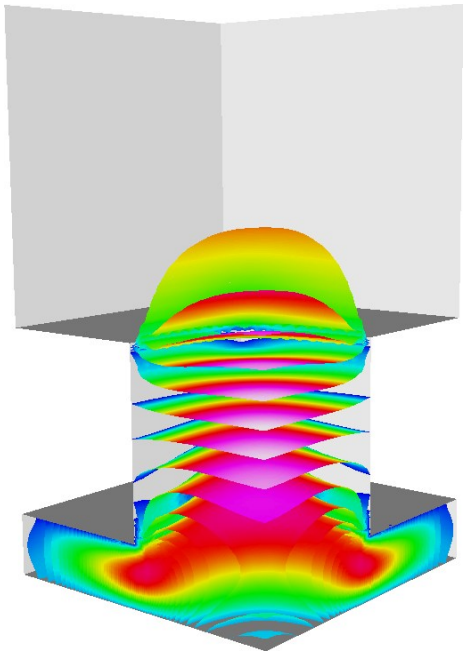

Figure 5.1: The linear flow results.

Finally, a picture of the results obtained with no-slip conditions is presented. The Fig. 5.1 shows a lot of pressure isosurfaces which are coloured using the absolute value of the velocity.

BY-ND

Tutorial 6

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 6.1

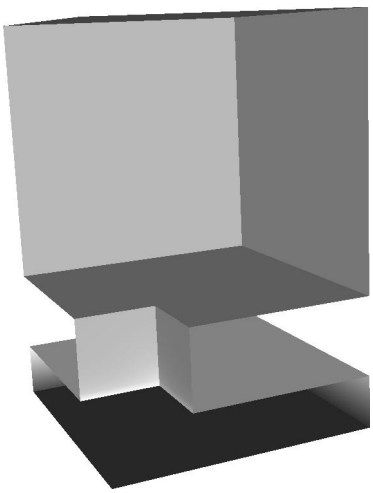

Figure 6.1: The geometry of problem.

CC BY-ND

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
```


CC BY-ND

```
  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

$$C = \varepsilon_r \varepsilon_0 \frac{A}{d}, \tag{6.1}$$

and the electrostatic force is

$$F_e = \frac{1}{2} \varepsilon_r \varepsilon_0 \frac{A}{d^2} \Phi^2, \tag{6.2}$$

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.

CC BY-ND

Table 6.1: 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 6.1 (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 6.2 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.

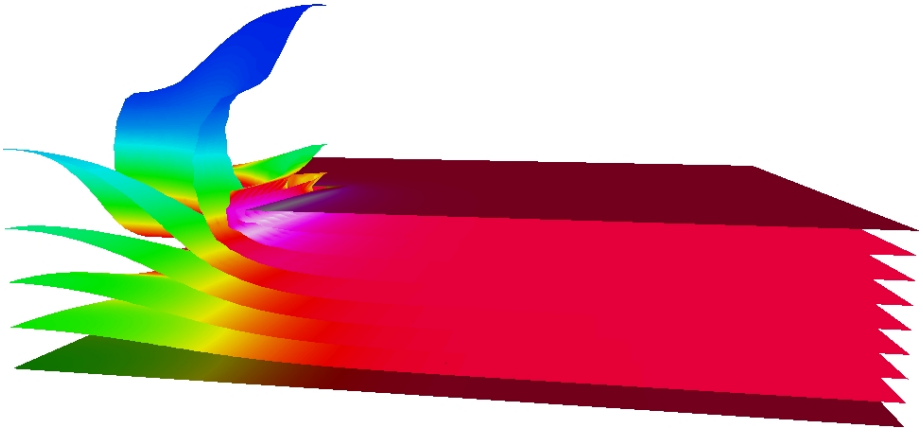

Figure 6.2: Isosurfaces of the potential coloured with electric field magnitude.

CC BY-ND

Tutorial 7

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 7.1 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.

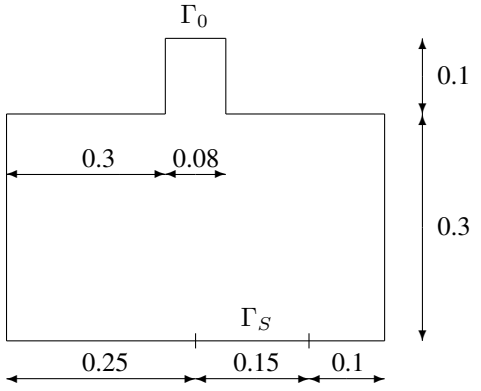

Figure 7.1: The geometry of the cavity.

BY-ND

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 $\Delta p \approx 0.205$

CC BY-ND

Tutorial 8

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 8.1. The dimensions of the induction heating crucible are given in Table 8.2. 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 8.1: 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

CC BY-ND

Table 8.2: 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
```


CC BY-ND

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. 8.1.

CC BY-ND

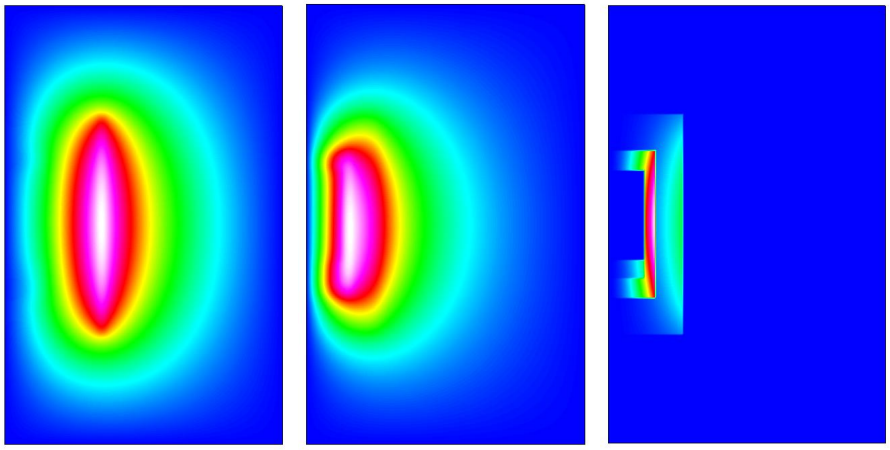

Figure 8.1: Induction heating of a simple crucible. a) in-phase component of the vector potential b) out-of-phase component of the vector potential c) Joule losses in the conductors

CC BY-ND

Tutorial 9

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. 9.1. 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.

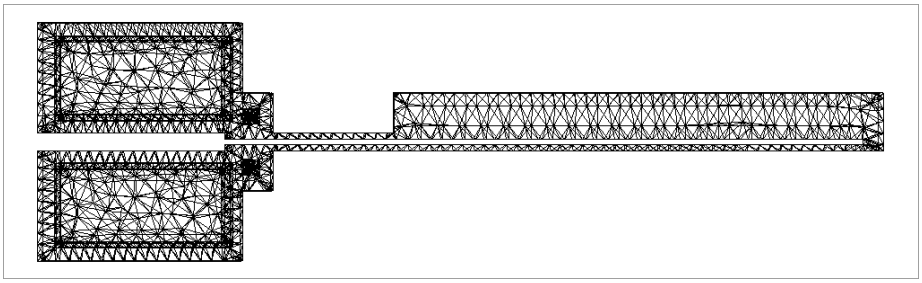

Figure 9.1: The geometry of the actuator.

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
```


CC BY-ND

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
```


CC BY-ND

```
  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
```


CC BY-ND

```
        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 9.2 and 9.3. The temperature rises unrealistically high in this example because all heat transfer mechanisms out of the structure are 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.

CC BY-ND

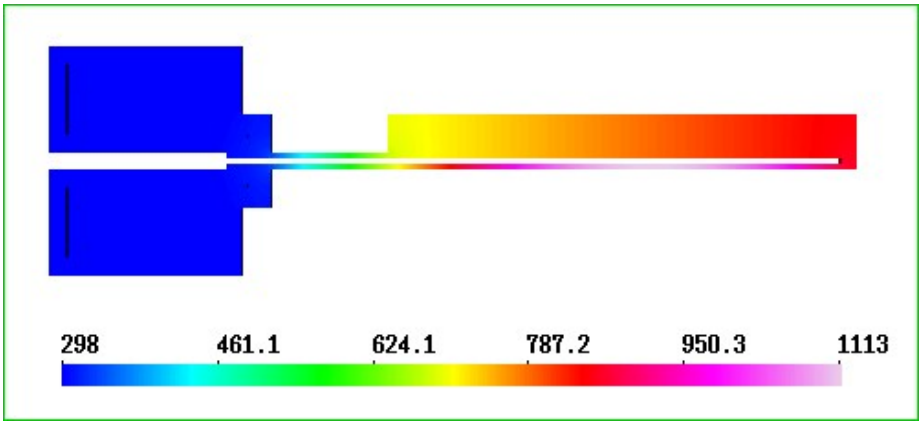

Figure 9.2: Temperature distribution.

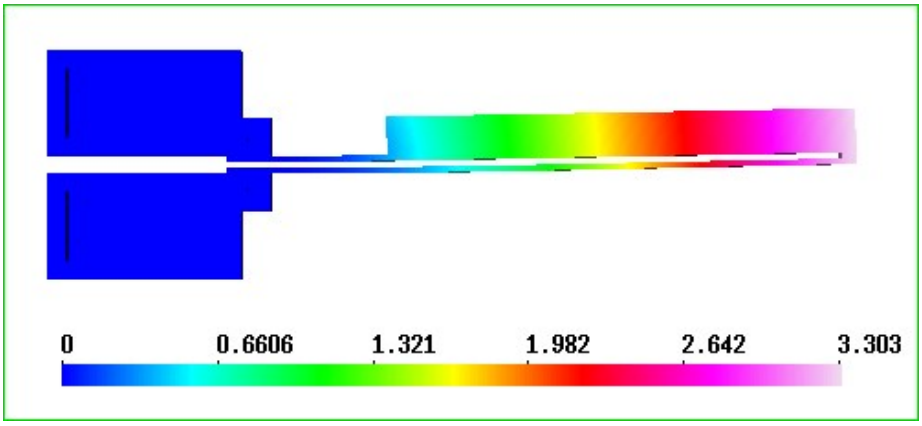

Figure 9.3: The displacement of the actuator.

BY-ND

Tutorial 10

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
```


CC BY-ND

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.

CC BY-ND

```
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
```


CC BY-ND

```
  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.

CC BY-ND

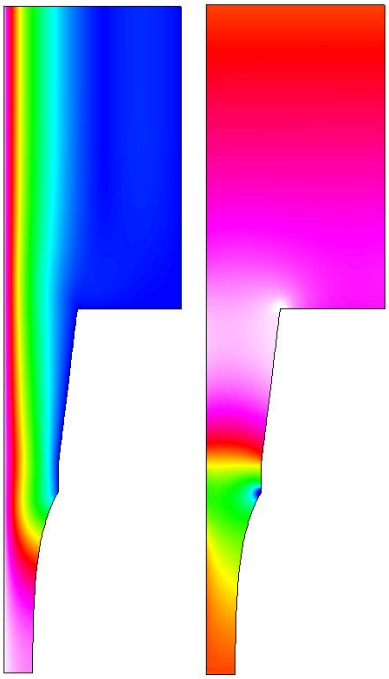

Figure 10.1: The velocity and pressure fields in a simple coating geometry. The solution utilizes the reduced dimensional free surface solver.

CC BY-ND

Tutorial 11

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)
```


CC BY-ND

```
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 11.1. 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"
```


CC BY-ND

```
  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 11.1 and the velocity fields at different time levels are presented in the figure 11.2. Postprocessing instructions are given in the file `PostProcessingInstr.txt`.

CC BY-ND

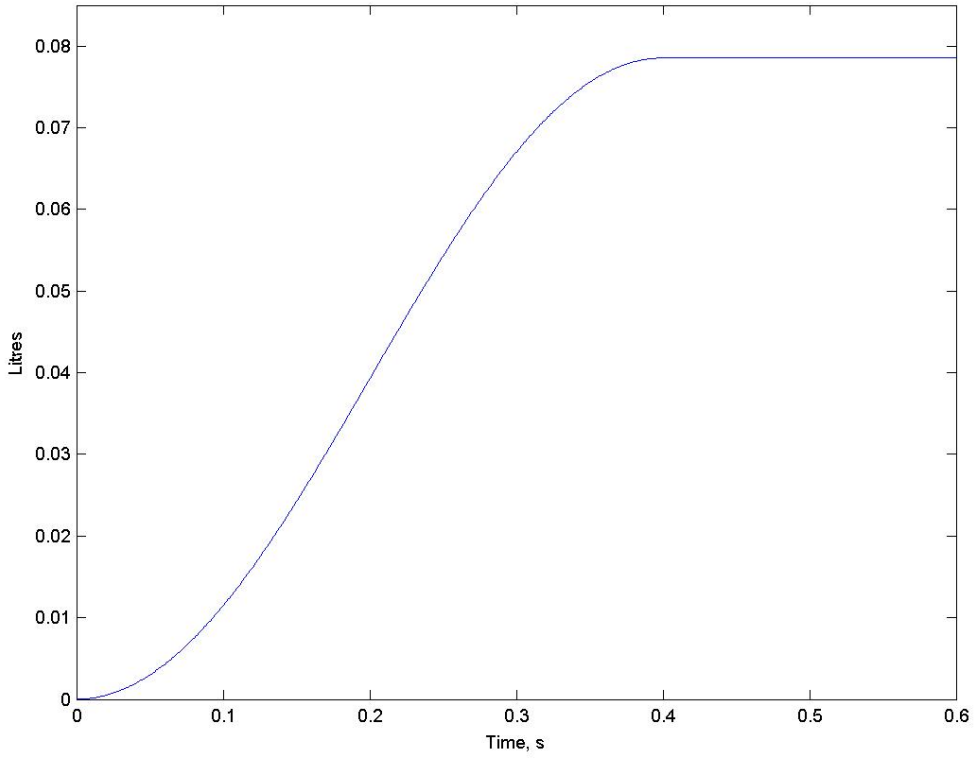

Figure 11.1: Contraction curve generated by the function `Motion`.

CC BY-ND

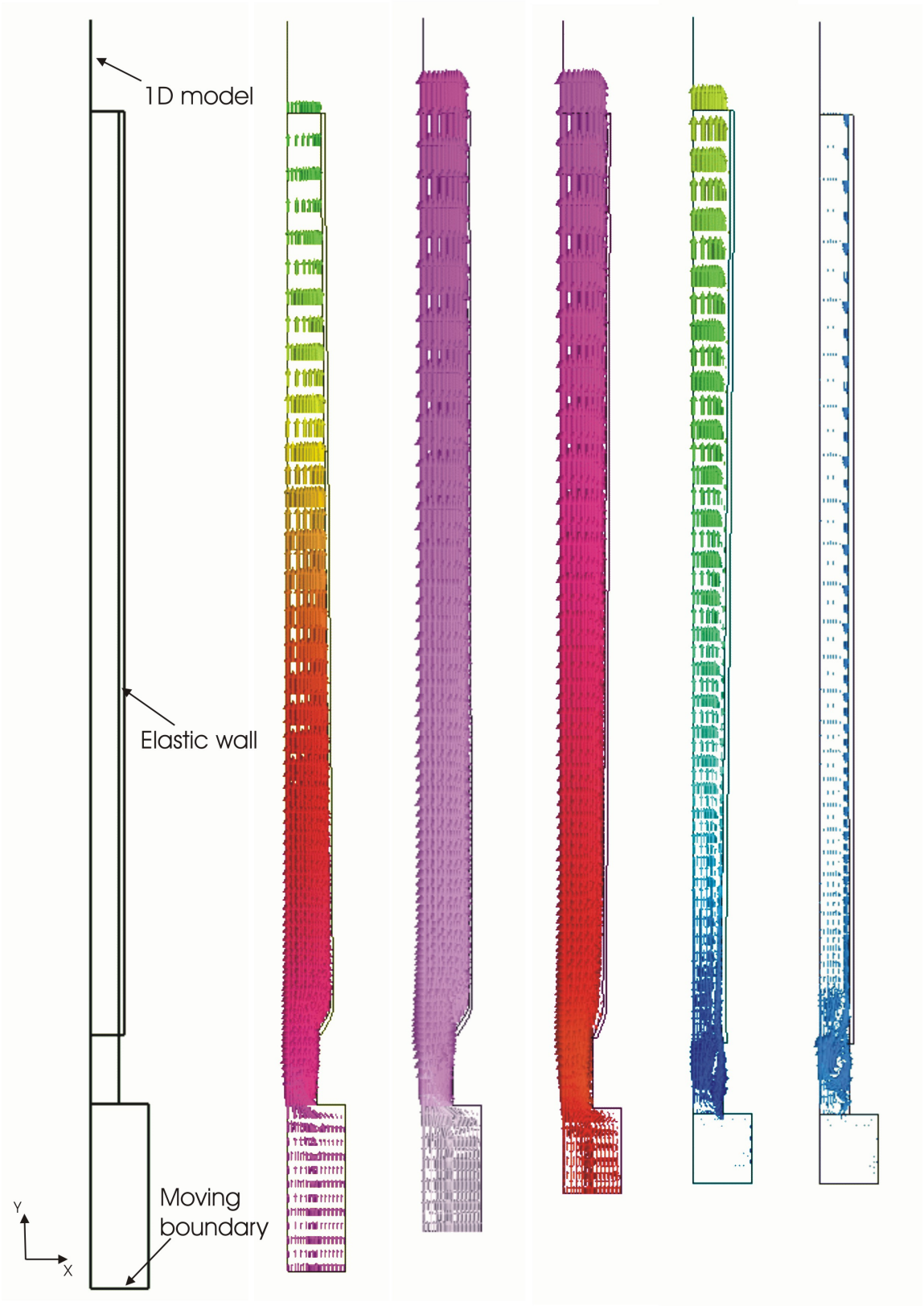

Figure 11.2: The geometry of the model and velocity fields at 5 time steps, 100, 200, 300, 400 and 500 ms. The displacements of the wall are magnified by factor of 10.

BY-ND

Tutorial 12

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 12.1. 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).

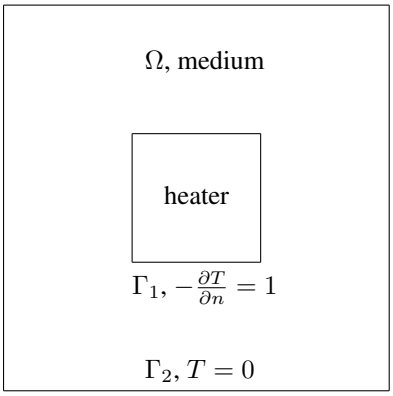

Figure 12.1: Simulation setting

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.

BY-ND

```
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
```


CC BY-ND

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 12.2 is the temperature distribution.

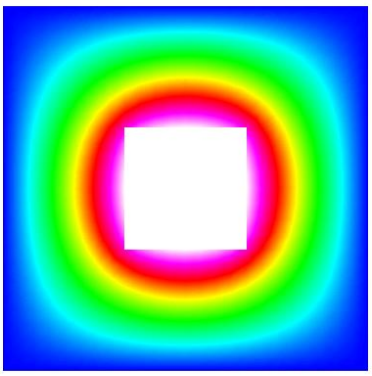

Figure 12.2: The temperature distribution.

BY-ND

Tutorial 13

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

$$\begin{cases} -\Delta\Phi &= f & \text{in } \Omega \\ \Phi &= 0 & \text{on } \Gamma \end{cases} \tag{13.1}$$

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

BY-ND

```
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.

CC BY-ND

  - 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. )
```


CC BY-ND

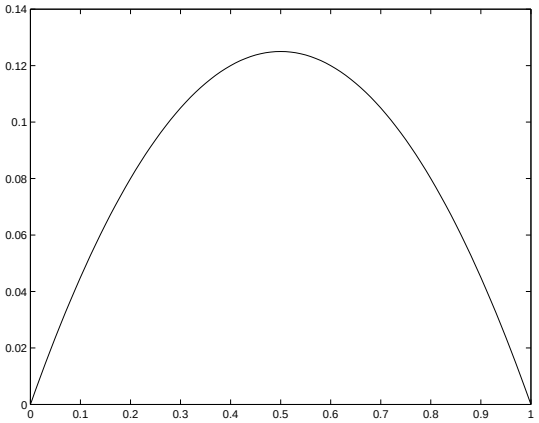

Figure 13.1: Solution of the Poisson Equation.

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 13.1

BY-ND

Tutorial 14

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.

$$\frac{1}{N}\sum_{i=1}^{N} u_i = \frac{volume}{time} \tag{14.1}$$

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

$$\sum_{i=1}^{N} u_i = 10N\sin(2\Pi t) \tag{14.2}$$

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.

CC BY-ND

```
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
```


CC BY-ND

```
  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.

CC BY-ND

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 14.1.

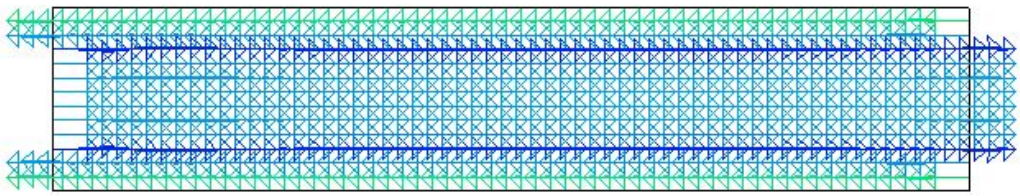

Figure 14.1: Solution of the velocity field. Note the flow to both directions.

[1] J.Physiol (1955) 127, 553-563

CC BY-ND

Tutorial 15

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

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{15.1}$$

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
```


CC BY-ND

```
  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

CC BY-ND

- 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
```


BY-ND

Results

Problem is solved with command `Solver`. The results are then viewed with ElmerPost. In figure 15.1 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.

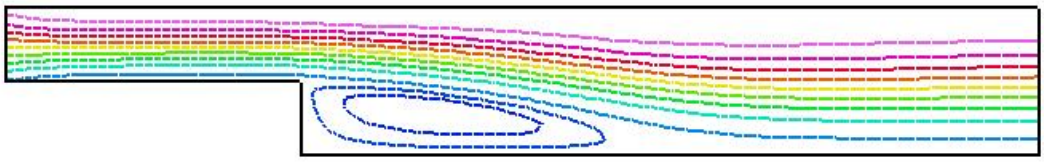

Figure 15.1: The streamlines of the flow.

BY-ND

Tutorial 16

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
```


CC BY-ND

```
  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
```


CC BY-ND

```
  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
```


CC BY-ND

```
  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
```


CC BY-ND

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. 16.1.

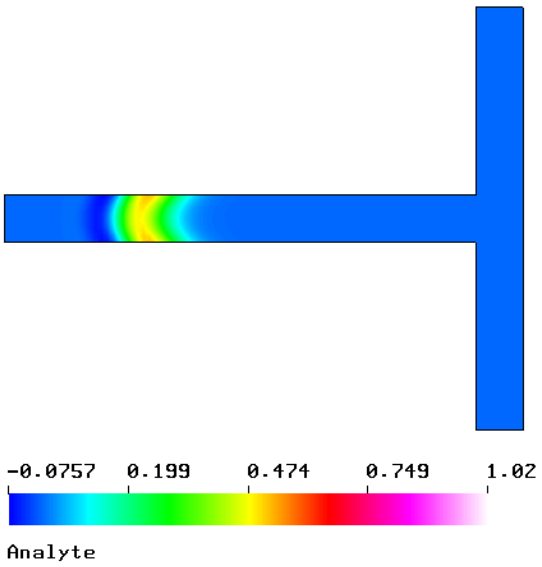

Figure 16.1: Analyte concentration on time instant 0.00025 seconds.

BY-ND

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.

CC BY-ND

Tutorial 17

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
```


CC BY-ND

```
  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.

CC BY-ND

```
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`.

CC BY-ND

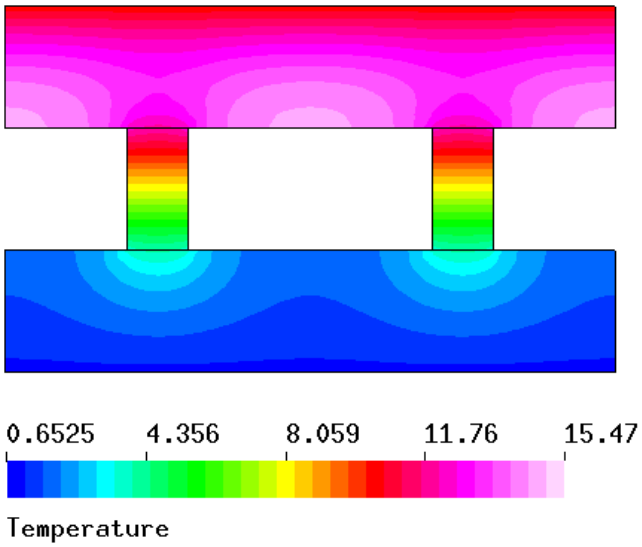

Figure 17.1: Temperature distribution of the system at the final time instant (with spectral_32 color map).

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. 17.1.

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.

BY-ND

Index

acoustic impedance, 21
acoustic waves, 32
AdvectionDiffusion, 68
Ansys, 38

BEM, 53

capacitance, 28
compressed row storage, 60

EigenSolve, 7
ElasticSolve, 48
ElectricForce, 28
Electrokinetics, 68
ElmerFront, 32
ElmerGrid, 3, 7, 12, 16, 21, 28, 34, 38, 43, 48, 53, 56, 60, 64, 68, 74

FlowSolve, 16, 21, 43, 48, 60, 64
FlowSolve , 68
Fortran 90 compiler, 48, 56, 60
FreeSurfaceReduced, 43

HeatSolve, 3, 16, 38, 74
HelmholtzSolve, 32

MEMS, 24, 29

OutletCompute, 48

passive elements, 74
PoissonBEMSolver, 53
PoissonSolver, 56

SaveData, 24, 30
SmitcSolver, 12
SolveWithLinearRestriction, 60
StatCurrentSolve, 38
StatElecSolve, 28, 68
StatMagSolve, 34
Stokes equation, 21
StreamSolver, 64
StressSolve, 7, 38

Womersley velocity profile, 63

BY-ND