Documentation of Elmer

- ElmerGUI Manual
  Manual of the new graphical user interface of Elmer software suite.
- Elmer Models Manual
  Description of the different physical models that are defined in independent solvers.
- ElmerSolver Manual
  Capabilities of the solver with an emphasis on generic library services provided by the software.

- ElmerGrid Manual with related grd-files
  Manual of ElmerGrid utility with simple meshing examples.
- Elmer Tutorial Manual with related input files
  Examples of simple Elmer cases with documentation of the solution procedures.
- Elmer Overview
  Overview over the different Elmer software with a view of the different executables, modules, manuals and strategies (meta-manual).

Manual modification of existing files

D.Sc. Peter Råback

CSC - IT Center for Science

CSC

Using ElmerGUI case as starting point

- **Only the most important solvers are supported by the GUI**
- **Minor modifications are most easily done by manual manipulation of the files**
- **The procedure**
  1. Choose your favorite text editor and open the .sif file (typically case.sif)
  2. Make the modification to the file and save
  3. Run ElmerSolver.exe
  4. Visualize the results (typically in case.ep) using your favorite visualization program
  5. …
- **Note: you cannot read in the changes made in the .sif file**
- **If you intend to use different meshes you should avoid join & divide command in ElmerGUI**
  - These will tamper with the numbering of entities

Using Elmer test case as starting point

- **There are more than 100 minimalistic test cases in Elmer**
  - See $ELMER_HOME/tests
  - Among these it is possible to find most of the implemented solvers
  - Use 'grep' in Unix or 'Search' in windows
- **To take these into use typically**
  - Increase space resolution
  - Increase number of timesteps
  - Increase output on run-time
    Max Output Level = 10
  - Activate output to .ep files (or ResultOutput)
    Post File = case.ep
  - Remove the unnecessary reference norm
- **There are various types of meshes in the test files**
  - ElmerGrid, Mesh2D, ready made
  - You may check from the local Makefile how the mesh generation is done

Exercise

- Copy a test case that you're interested in to your working directory
- Try to enhance the case so that you get run-time information and output to a file
- If possible increase the space resolution or increase number of timesteps

Derived data in Elmer

D.Sc. Peter Råback

CSC - IT Center for Science

Outline

- **This presentation deals with the computation and saving of derived data that is done separate from the primary solvers**
- **There exists a number of auxiliary solvers for computing derived fields**
  - Grad, Div, Curl, Streamlines, …
- **Solvers for dimensional reduction: 3D -> 2D**
  - Averaging over time or space dimension
- **Solvers for outputting 0D data**
  - Energy, flux, time, CPU time, number of iterations, etc…
- **Typically these solvers in advanced context are added by copy-paste**
- **Also GUI interfaces exist for some of the solvers**

Derived fields

- **Often it is desirable to compute a derived field from the solution**
- **Elmer offers several small auxiliary routines for this purpose**
  - Many solvers have internal options for computing derived fields (fluxes, heating powers,…)
  - SaveMaterials: makes a material parameter into field variable
  - Streamlines: computes the streamlines of 2D flow field
  - FluxComputation: given potential, computes the flux $q = - c \nabla \phi$
  - VorticitySolver: computes the vorticity of vector field, $w = \nabla \times v$
  - DivergenceSolver: computes the divergence of vector field, $w = \nabla \cdot v$
  - PotentialSolver: given flux, compute the potential $- c \nabla \phi = q$
  - Filtered Data: compute filtered data from time series (mean, fourier coefficients,…)
  - …
- **Usually auxiliary data need to be computed only after the iterative solution is ready or needed for saving**
  - Exec Solver = after timestep
  - Exec Solver = before saving

FluxSolver

- Computes the flux –c*grad(f) of a scalar field
- If the coefficient is not given reduces to computation of gradient
- Rather cheap default linear algebra settings (diagonally dominated equation)

```
Solver 2
  Equation = ComputeFlux
  Procedure = "FluxSolver" "FluxSolver“
  Flux Variable = String Temperature
  Flux Coefficient = String "Heat Conductivity“
End
```

VorticitySolver

- **Computes vorticity curl(v) of a vector field**

```
Solver 2
  Equation = ComputeVorticity
  Procedure = “VorticitySolver" “VorticitySolver“
  Flux Variable = String Velocity
End
```

DivergenceSolver

- Computes divergence Div(v) of a vector field
- Note: this has been in the built system only a week or so

```
Solver 2
  Equation = ComputeDivergence
  Procedure = “DivergenceSolver" “DivergenceSolver“
  Flux Variable = String Velocity
End
```

StreamlineSolver

- Computes the streamlines for 2D steady-state flows

```
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
End
```

FilterTimeSeries

- **Time averaging over given time interval**
- **Weighing with given function or sine/cosine series (Fourier transform)**
- **Ideally suited for DNS/LES simulations**

```
Solver 2
  Procedure = "FilterTimeSeries" "FilterTimeSeries"
  Variable 1 = “Temperature“
  Start Time 1 = Real 0.0
  Stop Time 1 = Real 1.0
  Variable 2 = “Velocity 1“
  Start Time 2 = Real 1.0
  Stop Time 2 = Real 2.0
End
```

ProjectToPlane

- **May be used for dimensional reduction**
  - 3D -> 2D
  - 3D -> axi symmetric
- **The solver must be active on a reduced dimensional part of the geometry**

```
Solver 2
  Procedure = “ProjectToPlane" “ProjectToPlane"
  Convert From Equation Name = String Navier-Stokes
  Convert From Variable = String Velocity 1
  Variable = MeanVelo
End

Boundary Condition i
  Body Id = Integer
  …
End
```

Derived nodal data

- **By default Elmer operates on distributed fields but sometimes nodal values are of interest**
  - Multiphysics coupling may also be performed alternatively using nodal values for computing and setting loads
- **Elmer computes the nodal loads from *Ax-b* where *A*, and *b* are saved before boundary conditions are applied**
- **This is the most consistant way of obtaining boundary loads**
- **Note: the nodal data is really pointwise**
  - expressed in units N, C, W etc. (rather than N/m^2, C/m^2, W/m^2 etc.)
  - For comparison with distributed data divided by the ~size of the elements

CSC

Derived lower dimensional data

- **Derived boundary data**
  - SaveLine: Computes fluxes on-the-fly
- **Derived lumped (or 0D) data**
  - SaveScalars: Computes a large number of different quantities on-the-fly
  - FluidicForce: compute the fluidic force acting on a surface
  - ElectricForce: compute the electrostatic froce using the Maxwell stress tensor
  - Many solvers compute lumped quantities internally for later use (Capacitance, Lumped spring,…)

Saving 1D data: SaveLine…

```
Solver n
 Equation = "SaveLine"
 Procedure = File "SaveData" "SaveLine"
 Filename =  "g.dat"
 File Append = Logical True
 Polyline Coordinates(2,2) = Real 0.25 -1 0.25 2.0
End

Boundary Condition m
  Save Line = Logical True
End
```

Saving 1D data: SaveLine

- **Lines of interest may be defined on-the-fly**
- **Flux computation using integration points on the boundary**
- **By default saves all existing field variables**

Saving 1D data: SaveLine…

```
Solver n
  Exec Solver = after timestep
  Equation = String SaveLine
  Procedure = File "SaveData" "SaveLine"
  Filename = File "line.dat"
  Polyline Coordinates(2,3) = 0.0 0.0 0.0 1.0 1.0 1.0
End

Boundary Condition m
  Save Line = Logical True
End
```

Saving 0D data: SaveScalars

Operators on bodies

- **Statistical operators**
  - Min, max, min abs, max abs, mean, variance, deviation, dofs
- **Integral operators (quadratures on bodies)**
  - volume, int mean, int variance
  - Diffusive energy, convective energy, potential energy

Operators on boundaries

- **Statistical operators**
  - Boundary min, boundary max, boundary min abs, max abs, mean, boundary variance, boundary deviation, boundary sum
  - Min, max, minabs, maxabs, mean
- **Integral operators (quadratures on boundary)**
  - area
  - Diffusive flux, convective flux

Other operators

- Nonlin converged, steady converged, nonlinear change, steady state change, time, timestep size, CPU time, partitions,…

Saving 0D data: SaveScalars…

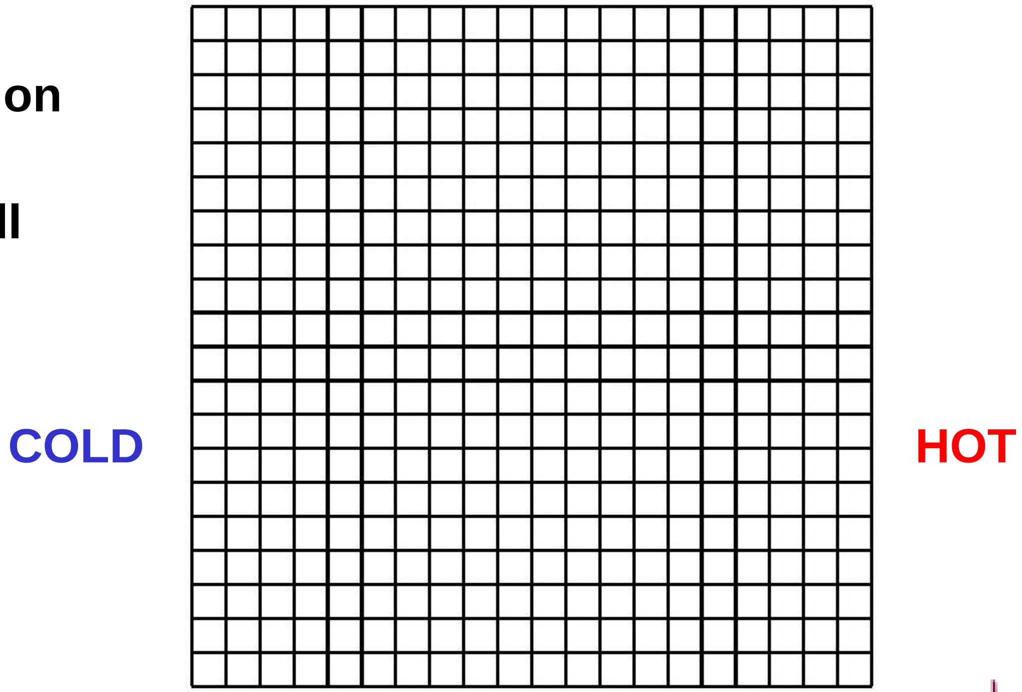
```
Solver n
  Exec Solver = after timestep
  Equation = String SaveScalars
  Procedure = File "SaveData" "SaveScalars"
  Filename = File "f.dat"
  Variable 1 = String Temperature
  Operator 1 = String max
  Variable 2 = String Temperature
  Operator 2 = String min
  Variable 3 = String Temperature
  Operator 3 = String mean
End

Boundary Condition m
  Save Scalars = Logical True
End
```

Saving 0D data in parallel

- **Since last week the results of SaveScalars may be reduced by dimension automatically**
  - Parallel Reduce = Logical True
  - Otherwise you get files equal to number of partitions
- **For most operations the parallel reduction operator corresponding to the serial operator has been defined**
  - MPI_MAX, MPI_MIN, MPI_SUM
  - For a minority of operations none of these is meaningfull in parallel (e.g. mean)
- **The user may define the parallel operator so that it runs over the default setting, e.g.**
  - Parallel Operator 1 = String ”max”

Example: preliminaries

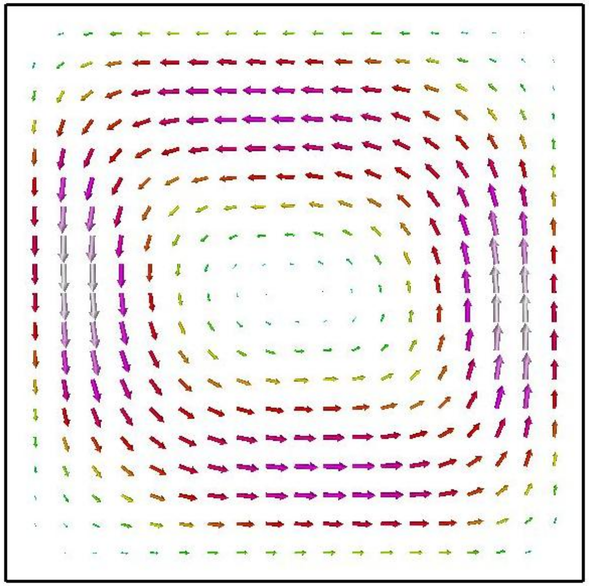
- Square with hot wall on right and cold wall on left
- Filled with viscous fluid
- Bouyancy modeled with Boussinesq approximation
- Temperature difference initiates a convection roll

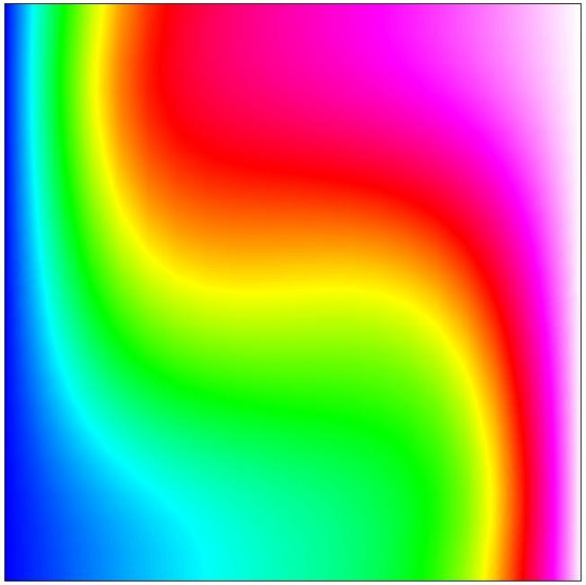


Case: eight solvers

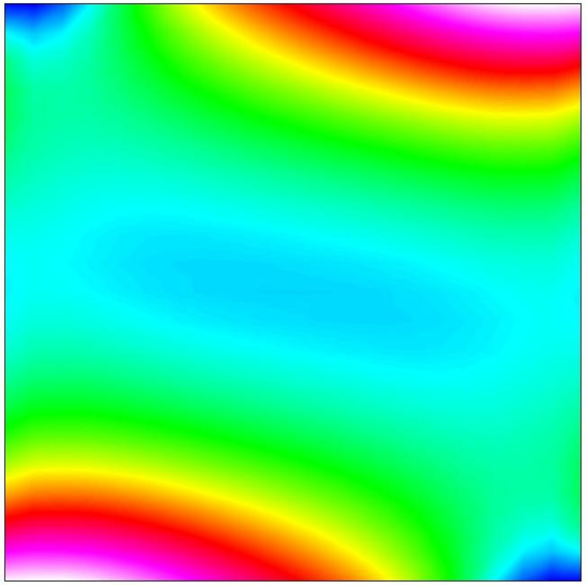
1. **Heat Equation**
2. **Navier-Stokes**
3. **FluxSolver**
4. **StreamSolver**
5. **VorticitySolver**
6. **ResultOutputSolver**
7. **SaveLine**
8. **SaveScalars**

Example: solution

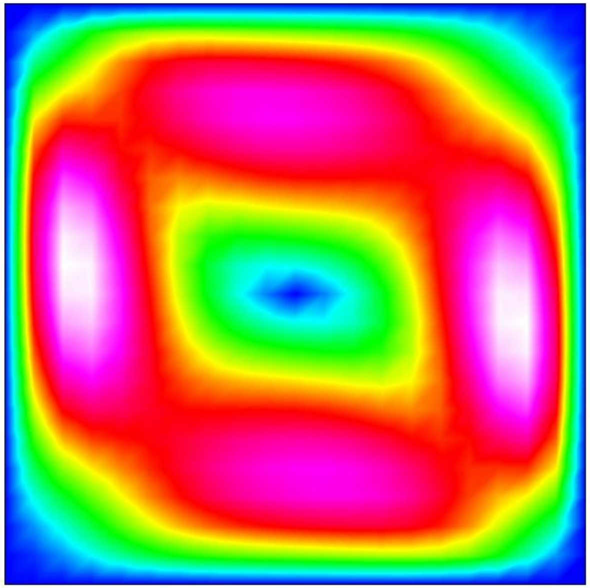




Temperature

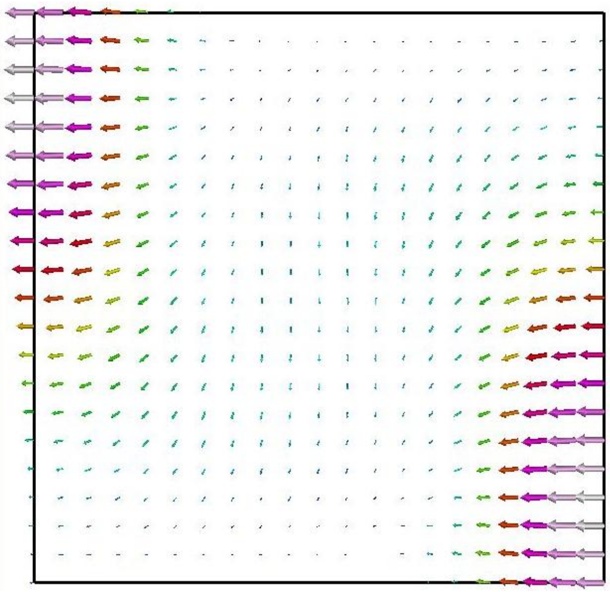


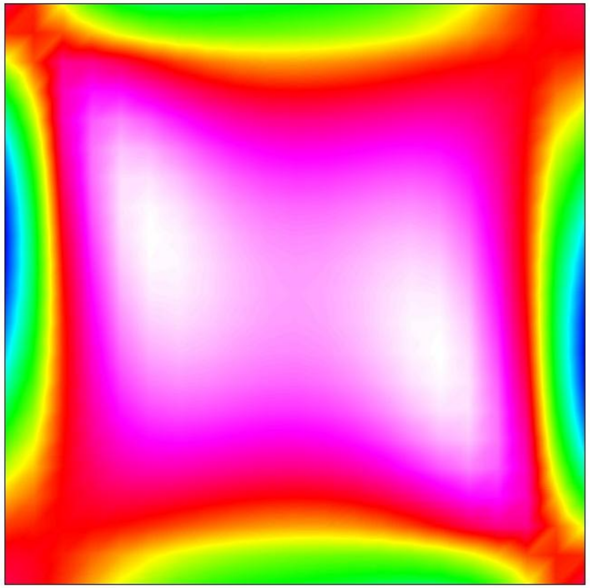
Pressure



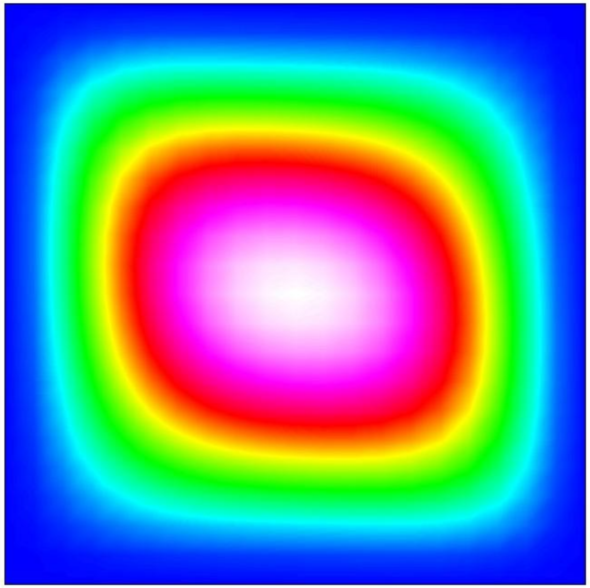
Velocity

Example: derived fields

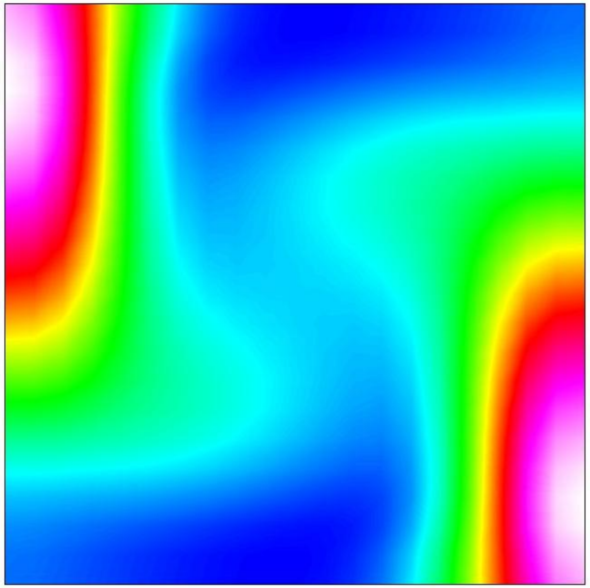




Vorticity



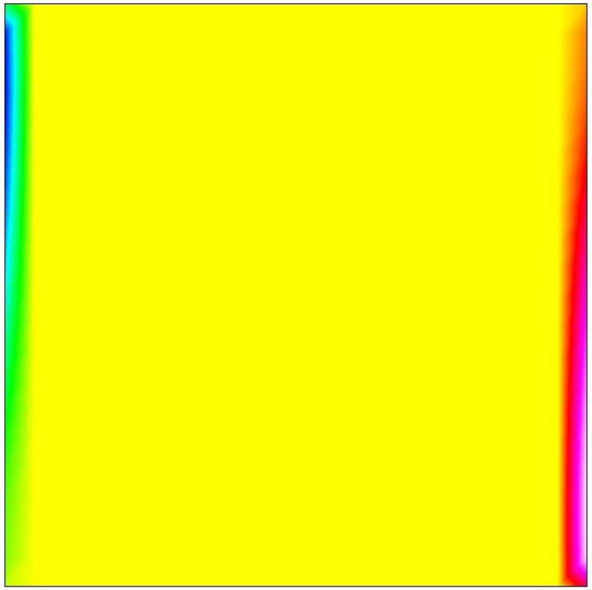
Streamlines



Diffusive flux

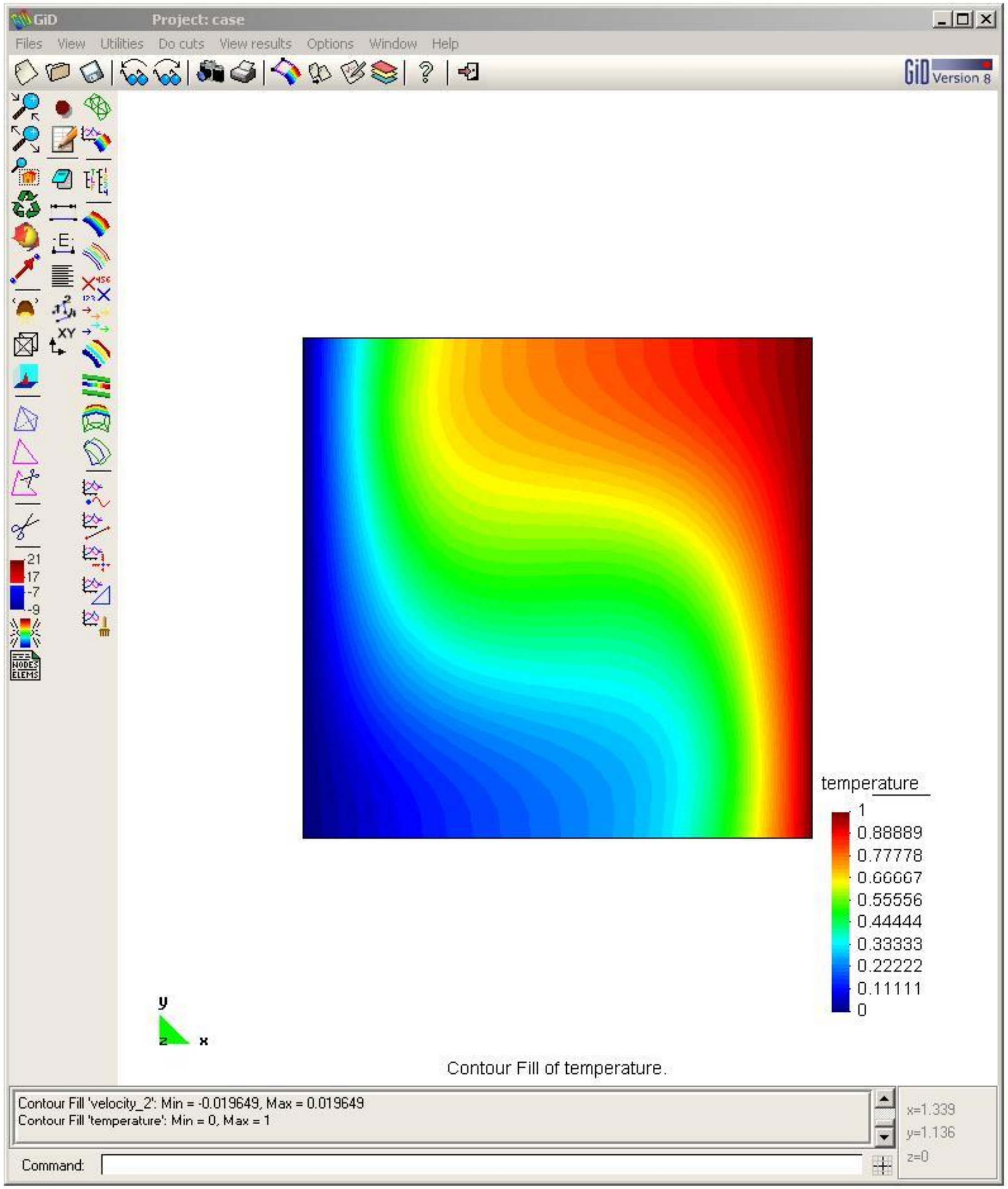
Example: nodal loads

- **If equation is solved until convergence nodal loads should only occur at boundaries**
- **Element size h=1/20 ~weight for flux**

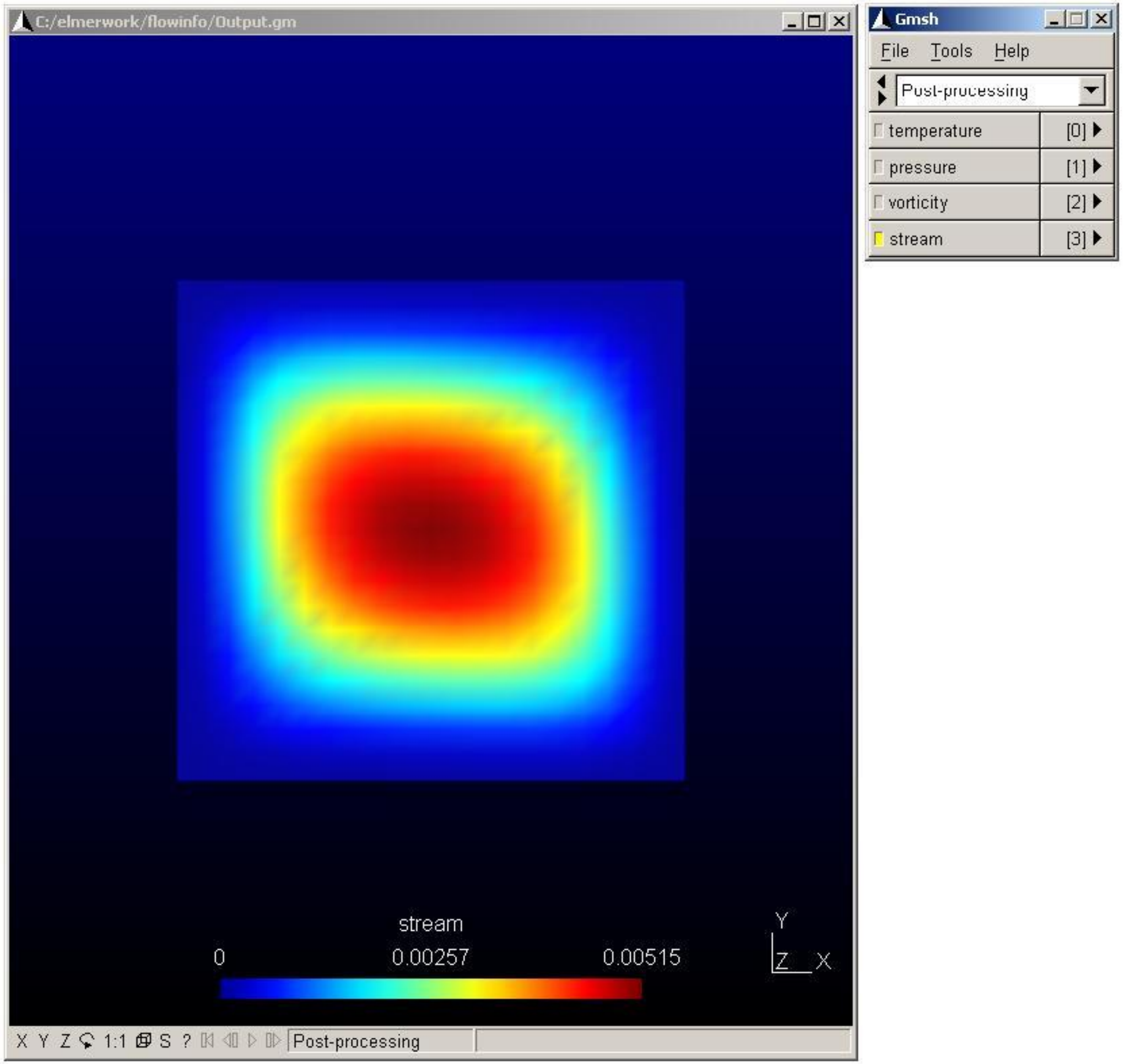


Nodal heat loads

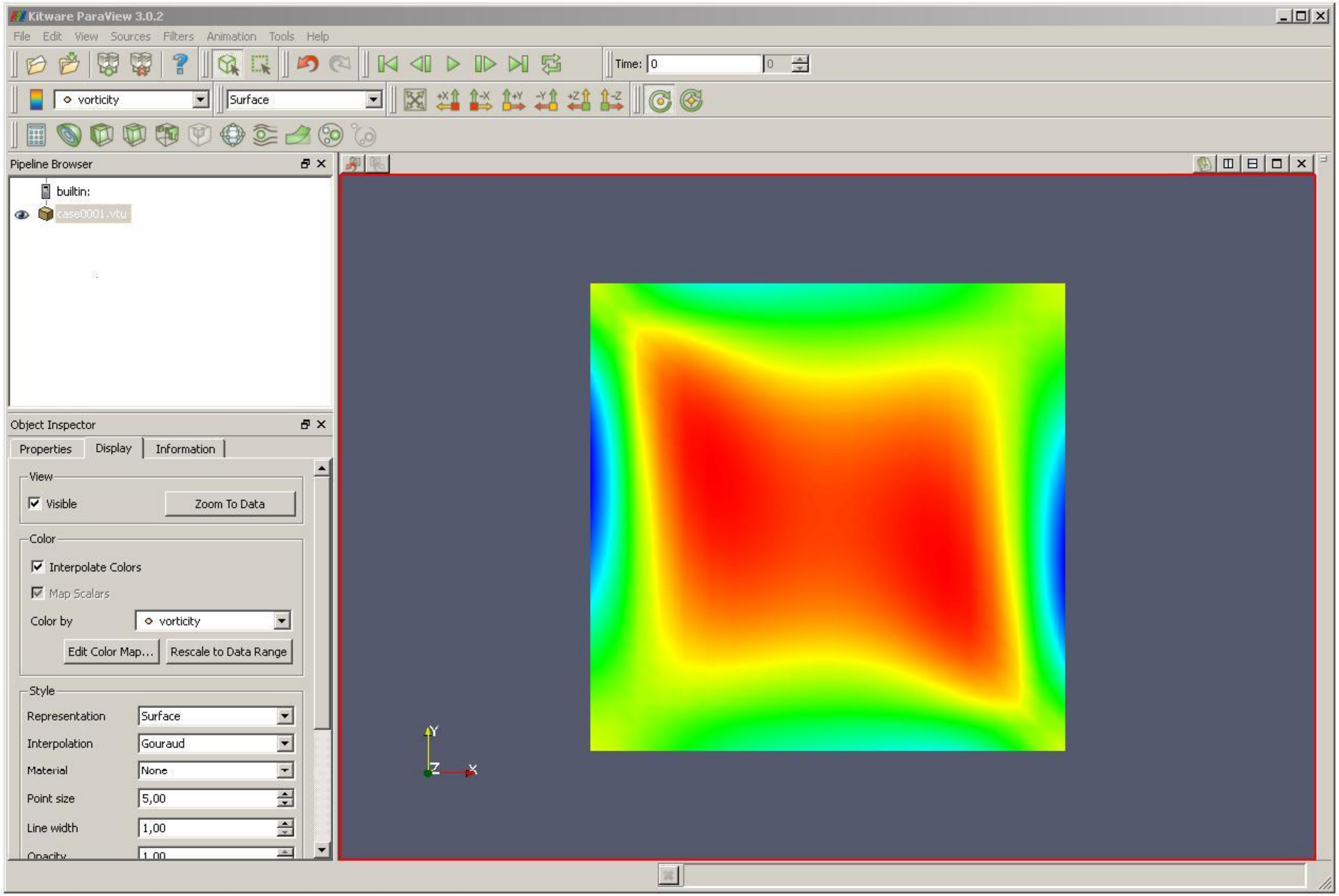
Example: view in GiD



Example:
view in Gmsh

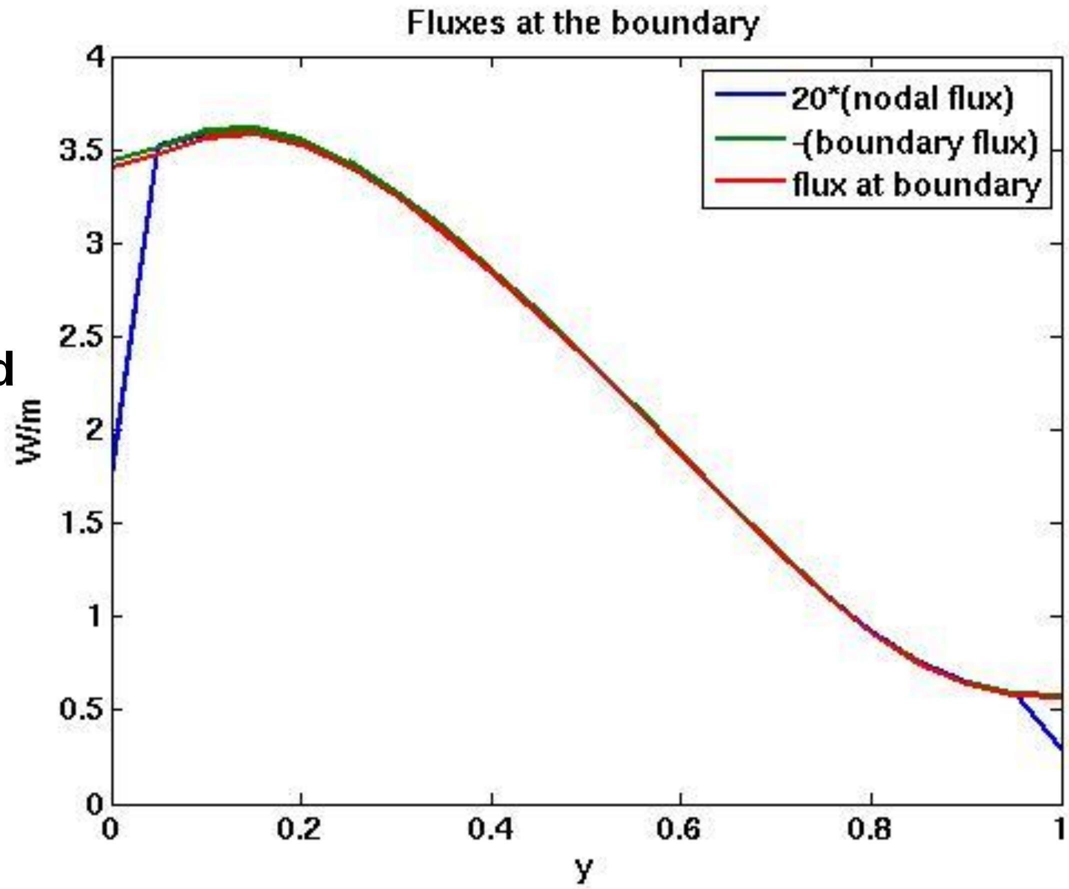


Example: view in Paraview



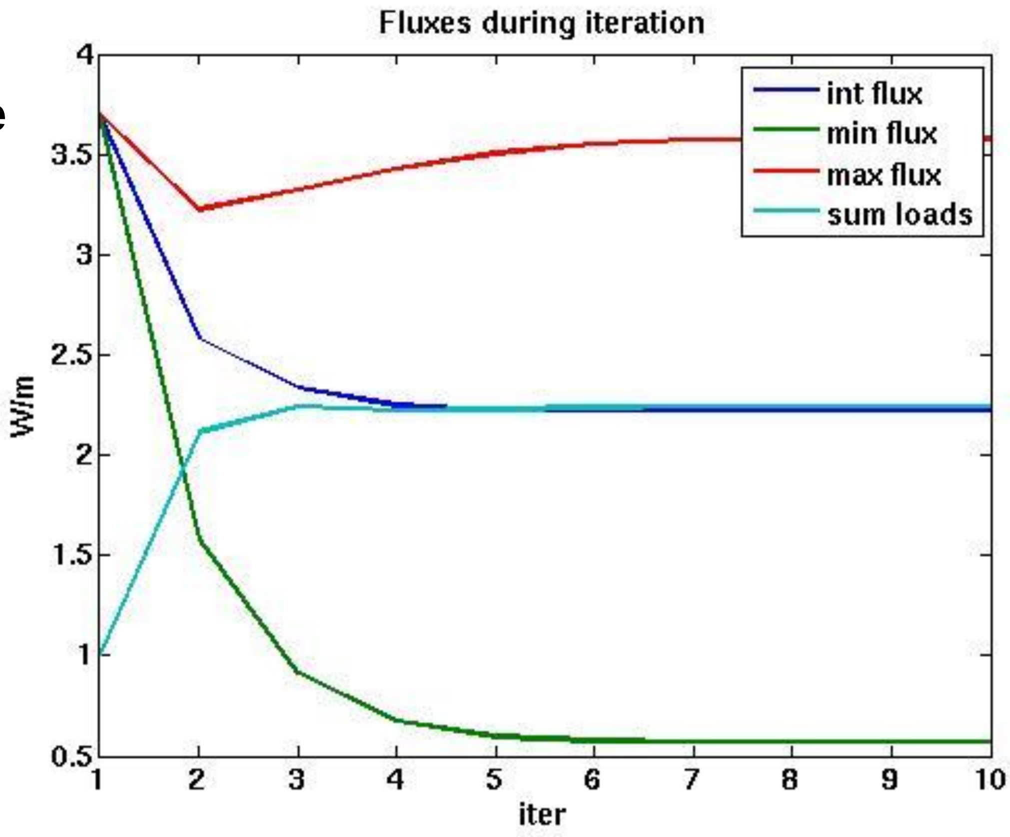
Example: boundary flux

- **Saved by SaveLine**
- **Three ways of computing the boundary flux give different approximations**
- **At the corner the nodal flux should be normalized using only *h/2***



Example: total flux

- **Saved by SaveScalars**
- **Two ways of computing the total flux give different approximations**
- **When convergence is reached the agreement is good**



Variable 1 = Temperature
Operator 1 = diffusive flux
Coefficient 1 = Heat Conductivity
Variable 2 = Temperature Loads
Operator 2 = boundary sum

Exercise

- **Add an auxiliary solver to some previously used analysis:**
  - Flux computation to scalar field
    Heat equation
  - Streamline computation to 2D flow field
    Flow passing a step
  - Vorticity computation to vector field
    Rayleigh-Bernard convection
  - Etc…

Some secrets of DefUtils

D.Sc. Peter Råback
CSC - IT Center for Science

Features that come with DefUtils

- **The default utilities have many additional features that are automatically accessible via commands in the .sif file**
- **Here some of the features are listed according to the subroutine that they are related to**
- **For more details see Solver Manual**
  - There is a keyword index!

DefaultUpdate

- **Most timestepping keywords**
- **…**
- **Time Derivative Active = Logical**
- **Time Derivative Condition = Real**
- **Timestep Scale = Real**
- **Field Passive = Real**

CSC

DefaultSolve

- **All ”Linear System …” keywords**
  - Linear System Method = String
    iterative, direct, multigrid
  - Linear System Preconditoning = String
    none, ILU, Parasails, multigrid,…
  - Linear System Direct Method = String
  - Linear System Iterative Method = String
  - Linear System Convergence Tolerance = Real
  - Linear System Convergence Measure = String
  - Linear System Symmetric = Logical
  - …
  - Linear System Timing = Logical
  - Linear System Timing Cumulative = Logical

DefaultSolve

- **Many ”Nonlinear System …” keywords**
  - Nonlinear System Convergence Tolerance = Real
  - Nonlinear System Convergence Measure = String
  - Nonlinear System Relaxation Factor = Real
  - Nonlinear System Convergence Absolute = Logical
  - Nonlinear System Norm Degree = Integer
  - Nonlinear System Norm Dofs = Integer
- **Some are solver specific**
  - Nonlinear System Newton …
  - Nonlinear System Max Iterations

DefaultSolve

- **All ”Steady State …” keywords**
  - Steady State Min Iterations = Integer
  - Steady State Max Iterations = Integer
  - Steady State Convergence Tolerance = Real
  - Steady State Convergence Measure = String
  - Steady State Relaxation Factor = Real
  - Steady State Convergence Absolute = Logical
  - Steady State Norm Degree = Integer
  - Steady State Norm Dofs = Integer

DefaultSolve

- **And bunch of other keywords:**
  - Exec Solver = String
  - Exec Interval = Integer
  - Minimize Bandwidth = Logical
  - Linear System Scaling = Logical
  - Update Exported Variables = Logical
  - Calculate Loads = Logical
  - Calculate Weights = Logical
  - Calculate Velocity = Logical (1st order only)
  - Harmonic Analysis = Logical
    Frequency = Real
  - Eigen Analysis = Logical
    Eigen System Values = Integer
  - Calculate Energy Norm = Logical

DefaultDirichlet

- **BC**
  Field = Real
  Periodic BC Field = Logical BC
  Anti Periodic BC Field = Logical
  Target Coordinates = Real
  Target Coordinates Eps = Real
  Target Nodes = Integer
  Field Condition = Real

- **Body Force**
  Field = Real …
  Field Condition = Real

Exercise

- **Try to add some solver section keywords applicable to your problem**
  - Computation of nodal loads
  - Conditional dirichlet conditions
  - Pointwise dirichlet conditions
  - Different convergence measures
  - Etc…