

Introduction to Elmer

Mikko Lyly

CSC – Scientific Computing Ltd.

<http://www.csc.fi/>

May 29, 2006, Espoo, Finland

Outline

- Elmer: Overview, core features and examples.
- Solution of multiphysical problems. Part I (basic theory).
- Solution of multiphysical problems. Part II (practice).
- Conclusions and contact information.

Overview

- Elmer is a computational tool for multiphysical problems. It has been developed in collaboration with Finnish universities, research laboratories and industry.
- Elmer includes physical models of fluid dynamics, structural mechanics, electromagnetics and heat transfer. These are described by partial differential equations which Elmer solves by the Finite Element Method (FEM).
- Elmer comprises of several different parts: The geometry, boundary conditions and physical models are defined in ElmerFront. The resulting problem definition is solved by ElmerSolver. Finally the results are visualized by ElmerPost. An

additional utility called ElmerGrid may be used for simple mesh manipulation. There are also interfaces to many well known mesh generators for being used with Elmer.

- The different parts of Elmer software may also be used independently. The strongest of the components is ElmerSolver, which is a general purpose PDE solver and finite element library. For pre- and postprocessing the users may find also other alternatives.
- ELMER has been released under GPL. More information can be found from <http://www.csc.fi/elmer>

What is “Multiphysics”? (when physics is already everything)

Quote from *P.Råback. Challenges of Multiphysical Problems. Espoo 7.5.2004:*

- Computational problems historically solved with application specific software (fluids, structures, electromagnetics, ...)
 - With the increase of computational resources also problems with more than one physical phenomenon have become feasible
- ⇒ Software capable of solving these problems are known as “multiphysics” software
(Ansys, FemLab, CFD-ACE+, Elmer, ...)
- Basically multiphysics is nothing new, most common coupled problems have been solved for decades (e.g. fluid flow and heat transport)

“Multiphysics”, our definition

In this presentation, “multiphysical problem” is used as a synonym for a PDE-system of the form

$$f_1(x_1, x_2, \dots, x_n) = 0$$

$$f_2(x_1, x_2, \dots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, x_2, \dots, x_n) = 0$$

where f_i are (possibly nonlinear) differential operators representing physical conservation laws, equilibrium conditions etc., and x_i , $i = 1, 2, \dots, n$, are functions representing the unknown physical field variables.

Model problem

To fix ideas, suppose that we want to solve the electric current and heat flux in a material where the resistance depends on temperature.

1) Model for electric conduction:

- Ohm's law: $J = \sigma E$
- Electric field is the gradient of a scalar potential: $E = \text{grad } V$
- Divergence of current density equals the rate of change of free charge density: $\text{div } J = 0$
- Electric conductivity is a function of temperature: $\sigma = \sigma(T)$

Eliminating J and E , we get the PDE

$$-div [\sigma(T) grad V] = 0$$

2) Model for heat conduction:

- Fourier's law: $Q = -k grad T$
- Divergence of heat flux equals the external heating power:
 $div Q = F$
- Heating power equals the electric power dissipation: $F = J \cdot E$

Eliminating Q , F , J and E , we get the PDE

$$-div (k grad T) - \sigma(T) |grad V|^2 = 0$$

The above PDEs form now a non-linear system

$$A(V, T) = 0$$

$$B(V, T) = 0$$

with

$$A(V, T) = -\operatorname{div} [\sigma(T) \operatorname{grad} V]$$

$$B(V, T) = -\operatorname{div} (k \operatorname{grad} T) - \sigma(T) |\operatorname{grad} V|^2$$

The system is of the standard form for $n = 2$ if we set $f_1 = A$, $f_2 = B$, $x_1 = V$, and $x_2 = T$.

Sequential iteration

In many real-life applications the functions x_i are “weakly coupled” through the operators f_j . To be a little more specific, a weakly coupled system could be characterized by the condition

$$\left| \frac{\partial f_i}{\partial x_j} \right| \leq C_i \left| \frac{\partial f_i}{\partial x_i} \right|$$

for some $C_i \ll 1$ and for every $j \neq i$ in a neighbourhood of the solution. In this case the value of f_i depends mostly on x_i , and even a big change in x_j causes only a small change in f_i . Loosely speaking, we could say that in a weakly coupled system f_i is not sensitive to x_j , or that the system is “diagonally dominant” (more precise definitions can be found from literature).

In weakly coupled multiphysical problems Elmer utilizes the so called sequential iteration algorithm instead of solving the problem directly:

1) Set $x'_i = 0$ for $i = 1, 2, \dots, n$ (initial guess).

2) Given the functions x'_i , solve

$$f'_1(x_1) := f_1(x_1, x'_2, \dots, x'_n) = 0$$

$$f'_2(x_2) := f_2(x'_1, x_2, \dots, x'_n) = 0$$

⋮

$$f'_n(x_n) := f_n(x'_1, x'_2, \dots, x_n) = 0$$

3) If $\sum_{i=1}^n \|x_i - x'_i\| \leq \epsilon$, stop.

4) Set $x'_i = x_i$ and go to step 2.

Here $\| \cdot \|$ is some suitably chosen norm, not necessarily the same in each occurrence.

Remark 1: The algorithm is closely related to the Jacobi iteration of linear algebra.

Remark 2: Gauss-Seidel type schemes are also possible, and often more efficient than the above basic scheme.

Remark 3: The uncoupled equations can be solved in parallel.

In our model problem the algorithm reduces to the following loop:

1) Set $V' = 0$ and $T' = 0$.

2) Given V' and T' , solve

$$A'(V) := A(V, T') = 0$$

and

$$B'(T) := B(V', T) = 0$$

3) If $\|V - V'\| + \|T - T'\| \leq \epsilon$, stop.

4) Set $V' = V$, $T' = T$, and go to step 2.

The system is of the standard form if we set $f'_1 = A'$, $f'_2 = B'$, $x_1 = V$, and $x_2 = T$.

Fixed point iteration

The sequential iteration algorithm produces a sequence of (possibly non-linear) uncoupled PDEs of the form

$$f(x) = 0$$

Considering the model problem, the equations to be solved are

$$-div [\sigma(T') grad V] = 0$$

$$-div (k grad T) - \sigma(T) |grad V'|^2 = 0$$

The first equation here is linear w.r.t. V . The second equation is non-linear in T .

The non-linear PDEs are next solved numerically by fixed point iteration. The basic idea is to choose a linear operator L and a function G such that

$$f(x) = Lx - G(x) = 0$$

The choice can be almost arbitrary as long as G is contractive w.r.t. L , i.e. it holds

$$\|G(y) - G(z)\| \leq C\|Ly - Lz\|$$

for some $0 < C < 1$ and for every y and z . Usually a good choice for L and G is found easily.

The standard fixed point iteration scheme is the following:

1) Set $x^* = 0$ (initial guess).

2) Given x^* , solve

$$Lx = G(x^*)$$

3) If $\|x - x^*\| \leq \epsilon$, stop.

4) Set $x^* = x$ and goto step 2.

For the second equation in our model problem, we can choose the operators L and G for example as

$$\underbrace{-\operatorname{div} (k \operatorname{grad} T)}_{=LT} - \underbrace{\sigma(T) |\operatorname{grad} V'|^2}_{=G(T)} = 0$$

With this choice, the iteration

$$LT = G(T^*)$$

converges provided that the electric conductivity $\sigma(T)$ is continuous and decreasing.

Iterative solution of the model problem

Let us finally write down the complete solution algorithm for our model problem. Recall that the original PDE-system was

$$- \operatorname{div}[\sigma(T) \operatorname{grad} V] = 0$$

$$- \operatorname{div}(k \operatorname{grad} T) - \sigma(T) |\operatorname{grad} V|^2 = 0$$

where V is the electric potential, T is the temperature, $\sigma(T)$ is the electric conductivity and k is the heat conductivity.

Our solution algorithm for this system is the following:

- 1) Initialize the sequential iteration: Set $V' = 0$ and $T' = 0$.
- 2) Given the temperature T' , compute the new potential V from

$$-div [\sigma(T') grad V] = 0$$

As this equation is linear, we do not have to apply the fixed point scheme here.

- 3) Given the potential V' , compute the new temperature T from

$$-div(k grad T) - \sigma(T)|grad V'|^2 = 0$$

As this equation is non-linear, we solve it numerically by successive linearizations:

3a) Initialize the fixed point iteration: Set $T^* = T'$.

3b) Given T^* , compute the new temperature T from

$$-\text{div} (k \text{ grad } T) = \sigma(T^*) |\text{grad } V'|^2$$

3c) If $\|T - T^*\| \leq \epsilon$, goto step 4.

3d) Set $T^* = T$ and goto step 3b)

4) If $\|V - V'\| + \|T - T'\| \leq \epsilon$, stop.

5) Set $V' = V$ and $T' = T$ and goto step 1.

In Elmer, the sequantial iteration is also referred to as “Steady State Iteration”. The fixed point scheme is called “Nonlinear System Iteration”. All this is done automatically by the program.

Variational equations and Galerkin's method

At this point, the sequential and fixed point iterations have been applied to split the original PDE-system into a sequence of linear PDEs of the form

$$Lu = f$$

Assuming that the operator L is elliptic, we can rewrite the remaining linear problems in an generic variational form as

$$a(u, v) = (f, v)$$

with

$$a(u, v) = (Lu, v) = (u, Lv)$$

Here v is an arbitrary test function and (\cdot, \cdot) is the inner product (or duality pairing) in an appropriate (problem dependent) function space.

In our model problem, the linear PDEs to be solved are Poisson equations

$$-\operatorname{div}(A \operatorname{grad} u) = f \quad \text{in } \Omega$$

where Ω denotes is the computational domain, $A > 0$ is a coefficient function and f is a source term.

Assuming that u satisfies the homogenous Dirichlet boundary condition $u = 0$ on $\partial\Omega$, we get a boundary value problem for which

the variational form is obtained by integrating by parts:

$$\int_{\Omega} A \operatorname{grad} u \cdot \operatorname{grad} v \, d\Omega = \int_{\Omega} f v \, d\Omega$$

The equation can be written as $a(u, v) = \int_{\Omega} f v$ if we set

$$a(u, v) = \int_{\Omega} A \operatorname{grad} u \cdot \operatorname{grad} v \, d\Omega$$

The reason for introducing the variational formulation is that we can then apply efficient and flexible numerical solution schemes based on Galerkin's method:

1) Choose a set of basis functions ϕ_i , $i = 1, 2, \dots, N$.

2) Expand the solution and test function in terms of the basis:

$$u \approx \sum_i \phi_i u_i \quad \text{and} \quad v \approx \sum_j \phi_j v_j$$

3) Find u such that $a(u, v) = (f, v)$ for all v , viz.

$$u_i \in R : \quad \sum_i \sum_j a(\phi_i, \phi_j) u_i v_j = \sum_j (f, \phi_j) v_j \quad \forall v_j \in R$$

It is straight forward to show that the parameters u_i satisfy

$$[a(\phi_i, \phi_j)][u_j] = [(f, \phi_i)]$$

where $[a(\phi_j, \phi_j)]$ is the “stiffness matrix” (usually big and sparse, depends on the choice of the basis) and $[(f, \phi_i)]$ is referred to as the

“force vector”. $[u_j]$ is usually called the “vector of degrees of freedom”.

The equation can be solved numerically by means of linear algebra.

Finite Element Method

In Elmer, the numerical solution of linear PDEs is based on the Finite Element Method (FEM). Roughly speaking, FEM is nothing but a clever way of choosing the basis functions ϕ_i , $i = 1, 2, \dots, N$ in Galerkin's method:

- 1) First divide the computational domain into “elements” (line segments, triangles, quadrilaterals, tetrahedra, octahedra, ...) satisfying some compatibility conditions (the intersection of any two elements must be either empty, a common vertex point, an edge, a face, or the element it self).
- 2) The basis functions are defined as (mapped) piecewise polynomial functions over the elements and parametrized in an appropriate way.

As an example, suppose that we are working in $3D$. Let us divide

the computational domain into affine tetrahedra T_i , $i = 1, 2, \dots, M$, and denote by x_j , $j = 1, 2, \dots, N$, the vertex points of the elements. We can then define the basis functions such that each ϕ_i is a linear polynomial over all T_j and satisfies the condition

$$\phi_i(x_j) = \delta_{ij}$$

The parametrization has in this case the following interpretation:

$$u_i = u(x_i) = \text{value of } u \text{ in vertex } x_i$$

A finite element program is usually a set of libraries and subroutines dedicated to perform the meshing of the computational domain, generating and evaluating the basis functions ϕ_i , computing the problem dependent inner products $a(\phi_i, \phi_j)$ and (f, ϕ_i) , assembly of

the “stiffness matrix” $[a(\phi_i, \phi_j)]$ and “force vector” $[(f, \phi_i)]$, solving the algebraic equation $[a(\phi_i, \phi_j)][u_j] = [(f, \phi_i)]$, and post processing/visualizing the solution $u = \sum_i \phi_i u_i$.

This is exactly what Elmer does on this level of the solution process, after having split the original non-linear PDE-system into a sequence of uncoupled linear sub-problems. Here’s a list of some core features of the base-FEM-code of Elmer:

- Automatic control of sequential iteration.
- Problem dependent fixed point iterations easy to write and control.
- Automatic evaluation of basis functions ϕ_i and their gradients.

- Numerical quadrature available for the evaluation of the bilinear and linear forms $a(\phi_i, \phi_j)$ and $b(\phi_i)$.
- Automatic assembly of the matrix $[a(\phi_i, \phi_j)]$ and vector $[b(\phi_i)]$ in Compressed Row Storage (CRS) -format.
- Several numerical solution methods for the final algebraic equation. (LU-decomposition, Krylov-space methods, Multigrid).
- Adaptive refinement of the computational mesh.
- Visualization and post processing of the results.

Mesh generation in Elmer

An Elmer-mesh is given in four separate ascii files:

- `mesh.nodes` Coordinates of the node points
- `mesh.elements` Bulk element connectivities
- `mesh.boundary` Boundary element connectivities
- `mesh.header` Number of nodes, element types,...

The contents of these files will be explained later.

The mesh.* -files can be generated e.g. by

1) ElmerFront

2) ElmerGrid

3) Netgen

4) GiD

5) By any FE mesh-generator through appropriate filters.

ElmerFront is a pre processor and graphical user interface for the rest of the Elmer-package. ElmerFront contains a 2d Delaunay mesh generator based on advancing front methods.

ElmerGrid is a simple 2d/3d mesh generator dedicated to very simple geometries (e.g. the unit square). The program can be downloaded from Elmer's download pages at <http://www.csc.fi/elmer>

Netgen is an automatic 3d mesh generator developed by the research group of J. Schoberl in the university of Linz in Austria. Netgen accepts as input data the so called “constructive solid geometry” files, stl surface-triangulations, and finite element meshes. The mesh density can be controlled over “background meshes”, which makes the program suitable for adaptive refinement. Netgen is free from charge, and the code is available in public CVS. The program is

under LGPL (Library General Public Licence). Drawback: Limited cad features.

GiD is a “personal pre and post processor and mesh generator” that includes a powerful CAD -module, and a good 2d/3d mesh generator. The program has been developed in CIMNE, Spain. GiD accepts as input data most major CAD-formats, stl surfaces, and finite element meshes. Elmer mesh files can be exported from GiD easily. GiD also accepts background meshes for controlling the mesh density, i.e., it can be used for isotropic adaptive mesh refinement. The price of the program is reasonable.

Interfaces e.g. from Ansys and Femlab to Elmer are also available.

Example 1: Solution of the heat equation by ElmerFront

Let us consider a simple heat transfer problem in an L-shaped domain $\Omega = (-1, 1) \times (-1, 1) \setminus (-1, 0) \times (-1, 0)$. The boundary value problem to be solved is

$$\begin{aligned} -\Delta u &= 1 \text{ in } \Omega \\ u &= 273 \text{ on } \partial\Omega \end{aligned}$$

The geometry file for ElmerFront is in this case the following.

Header

Model Name "L"

Dimension

Integer 2

End

Vertices

Points

Size 6 3

Real

0.0 0.0 0.0

0.0 -1.0 0.0

1.0 -1.0 0.0

1.0 1.0 0.0

-1.0 1.0 0.0

-1.0 0.0 0.0

End

```
Body 1
  Polygon
    Size 6
      Integer 1 2 3 4 5 6
End
```

The problem is defined using the graphical user interface ElmerFront (demo).

As an output, ElmerFront creates the so called solver-input-file, by which the solution process of ElmerSolver is controlled. The file has the following block structure. First, we have a header block in which the databases etc. are defined:

```
Header
  CHECK KEYWORDS Warn
  Mesh DB "MESHDIR" "verkko"
```

```
Include Path ""
Results Directory ""
End
```

Next, there is a block called “Simulation”. In this block the problem type, spatial dimension, coordinate systems etc., are defined:

```
Simulation
```

```
Min Output Level = 0
Max Output Level = 31
Output Caller = True
```

```
Coordinate System = "Cartesian 2D"
Coordinate Mapping(3) = 1 2 3
```

```
Simulation Type = "Steady State"
Steady State Max Iterations = 20
Output Intervals = 1
```

```
Solver Input File = "L.sif"  
Post File = "L.ep"  
End
```

The Constants-block defines some general physical facts:

```
Constants  
Gravity(4) = 0 -1 0 9.82  
Stefan Boltzmann = 5.67e-08  
End
```

At this point, we define the PDE-system to be solved in different computational domains:

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

The PDE(-system) to be solved is in this case the heat equation:

```
Equation 1
```

```
  Heat Equation = True
```

```
End
```

The numerical solution parameters for the heat equation are given in the Solver-block:

```
Solver 1
```

```
  Exec Solver = "Always"
```

```
  Equation = "Heat Equation"
```

```
  Variable = "Temperature"
```

```
  Variable Dofs = 1
```

```
  Linear System Solver = "Iterative"
```

```
  Linear System Iterative Method = "BiCGStab"
```

```
  Linear System Max Iterations = 350
```

```
  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 model parameters and initial/boundary conditions are finally defined as follows.

```
Body Force 1  
Heat Source = 1
```

End

```
Boundary Condition 1
```

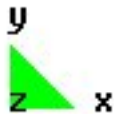
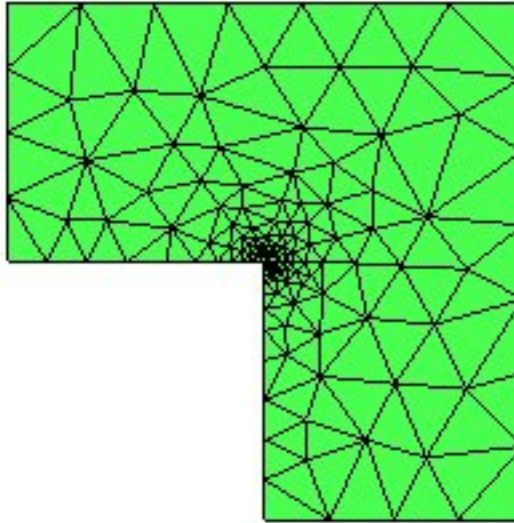
```
  Target Boundaries(6) = 1 2 3 4 5 6
```

```
  Temperature = 273
```

```
End
```

Given the above solver-input file “L.sif”, and the “mesh.*” files in directory “verkko”, the problem is finally solved by giving the command “ElmerSolver”.

Example 2: Mesh generation by GiD



Example 3: Solution of the model problem using Elmer/GiD

