Elmer Users Guide
Draft

Martti Verho Juha Ruokolainen Jouni Malinen
Petri Kallberg Jari Järvinen Harri Hakula

10th March 1999

Contents

1 Introduction **1**

1.1 About This Document 1
1.2 ELMER Development 1
1.3 Program Capabilities 2
1.3.1 Application Areas 2
1.3.2 Class of Problems 3
1.3.3 Mathematical Background 4
1.3.4 Numerical Methods 8
1.3.5 ELMER Structure 9

2 The Cad interface **11**

2.1 Introduction 11
2.2 Model data and files 12
2.3 Using Cadinterface 12
2.4 File menu 14
2.5 Edit menu 16
2.6 Display menu 20
2.7 Problem menu 21
2.8 Model menu 27
2.9 Mesh menu 32
2.10 Solver menu 33
2.11 Run menu 36

3 Elmer Mesh Generation **38**

3.1 Capabilities of the Mesh Generator 38
3.2 Mesh Generation Strategy 38
3.3 Practical Issues 39
3.4 When the System Takes Over the Control 40
3.5 Guidelines 40

4 Elmer Graph Partitioning **42**

4.1 ElmerSplit Capabilities 43
4.1.1 How Partitioning Works 43
4.1.2 Mesh coarsening 44
4.1.3 Creating partitions 46
4.1.4 Refining partitions 47
4.2 Using the ElmerSplit 48
4.2.1 Command-line syntax 48

4.2.2 General guidelines 49

5 Overview of The Solver 50

5.1 Solver Capabilities 50
5.2 Structure of a Simulation Run 51
5.3 Structure of the Equation Solvers 52
5.4 Equation Discretization at the Element Level 53
5.5 The Structure of the Base FEM Code 53
5.6 Utilities 54
5.7 Using the Solver 55

6 Elmer Post Processing 57

6.1 Elmerpost Capabilities 58
6.2 Elmerpost Element Types 59
6.3 Elmerpost Input Format 59

7 Some Walk Through Examples 60

7.1 The Backward Facing Step 60
7.1.1 The Equations to be Solved 60
7.1.2 Defining the problem 61
7.1.3 Results 64
7.2 Rayleigh-Bénard Convection 64
7.2.1 The Equations to be Solved 65
7.2.2 Defining the problem 66
7.2.3 Results 67
7.3 Radiation in Axisymmetric Enclosure 67
7.3.1 The Equations to be Solved 67
7.3.2 Defining the problem 68
7.3.3 Results 70

A A Very Short Introduction to Tensors 80

A.1 Covariant and Contravariant Tensors 80
A.2 The Metric Tensor and the Christoffel Symbols 81
A.3 Covariant to Contravariant and Back 82
A.4 Covariant Differentiation 82
A.5 The Second Fundamental Form and the Mean Curvature 83

B Discretization of the Equations 84

B.1 The Incompressible Navier-Stokes Equations 84
B.1.1 Variational Formulation 85
B.1.2 Discretization of the Variational Formulation 85
B.1.3 Stabilization 87
B.1.4 Discretization 87
B.2 Computing the Second Derivates 89
B.3 The Scalar Diffusion-Convection Equation 90
B.3.1 Variational Formulation 90
B.3.2 Discretization of the Variational Formulation 90
B.3.3 Stabilization 91
B.3.4 Discretization 92
B.4 The Compressible Navier-Stokes Equations 93

B.5 The Elastic Stress Equations 93
B.5.1 Variational Formulation 94
B.5.2 Discretization of the Variational Formulation 95
B.6 The Magnetic Induction Equation 96
B.6.1 Discretization 97

C The Diffuse Gray Radiation Boundary Condition 99
C.1 The Diffuse Gray Radiation Boundary Condition 99
C.2 Computing the View Factors in 3D 101
C.3 Normalizing the View Factors 102

D Free surfaces 104
D.1 The Free Surface Boundary Condition 104
D.2 Computing the Mean Curvature 104
D.3 The Surface Update 105
D.4 The Mesh Update 106

E HUTIter library 107
E.1 Introduction 107
E.1.1 General 107
E.2 Using HUTI 108
E.2.1 Overall Structure 108
E.2.2 Naming Conventions 109
E.2.3 External Routines 110
E.2.4 Iteration Parameters 113
E.2.5 Header Files 115

F Format of the ELMER Cad Interface settings file 116

G Format of the Elmer Geometry File 120

H Solver Input Format 123
H.1 Array Variables 123
H.2 Parameters Depending on Field Variables or Time 123
H.3 User Defined Functions 124
H.4 The Format 125
H.4.1 The `Header` Section 125
H.4.2 The `Constants` Section 126
H.4.3 The `Simulation` Section 126
H.4.4 The `Solver` Section 128
H.4.5 The `Body` Section 130
H.4.6 The `Equation` Section 130
H.4.7 The `Body Force` Section 131
H.4.8 The `Initial Condition` Section 132
H.4.9 The `Material` Section 133
H.4.10 The `Boundary` Section 134
H.4.11 The `Boundary Condition` Section 134
H.5 An Example of Using the Input File 136

I Format of the Solver Output File **144**
I.1 The Header 144
I.2 The Output Data 145

J The Elmerpost File format **146**
J.1 The Header 146
J.2 The Mesh 147
J.3 The Solution Data 147

K Solver Utility Routines **148**
K.1 Data Structures 148
K.1.1 The Model Structure 148
K.1.2 The Element, Element Type and Node Structures 150
K.2 Retrieving Model Information Within User Routines 152

L The Solver Element Types **157**

Chapter 1

Introduction

1.1 About This Document

This document gives an introduction to the use of a computational fluid dynamics (CFD) program called ELMER. The use of the program is explained in detail by three walk-through examples, namely by backward-facing step problem, Rayleigh-Bénard convection and radiation in axisymmetric enclosure.

ELMER consists of a preprocessor, a solver and a postprocessor. The idea of this document is to explain how these parts work and how one can work with these parts rather than to describe thoroughly mathematical and numerical methods applied in ELMER.

The intended audience for this document are engineers and scientists in industry and research institutes working with CFD. It is assumed that the reader has a basic knowledge of mathematical modeling and numerical methods.

This document aims at advicing a first time user of ELMER to use the program. Experienced users can utilize this document in solving their own research problems or in supporting teaching and training at universities and research institutes.

1.2 ELMER Development

ELMER development is a part of the Finnish national CFD programme coordinated by the Technology Development Centre of Finland (TEKES) since 1995. The main objective in this project is to develop a CFD program, based on the finite element method (FEM), which can be utilized in industrial processes where heat transfer in different forms and incompressible fluid flows are present.

ELMER is under an active development. It is based on most sophisticated and commonly known numerical methods. In ELMER development aspects like program maintainance, suitability and portability to a wide range of computer architectures are emphasized.

ELMER is being developed in co-operation between the Center for Scientific Computing (CSC), Helsinki University of Technology (HUT), Okmetic Ltd., the Technical Research Centre of Finland (VTT) and the University of Jyväskylä (JYU). ELMER development is coordinated by the Coordinating board which pays close attention to the direction and quality control of research made in

Name	Organisation	Expertise
Risto Nieminen	HUT	Theoretical physics Chairman of Coordinating board
Eero-Matti Salonen	HUT	FEM
Timo Tiihonen	JYU	Mathematical modeling, FEM
Asko Vehanen	Okmetic Ltd.	Industrial applications

Table 1.1: Members of Coordinating board

Name	Organisation	Expertise
Olli Anttila	Okmetic Ltd.	Industrial applications
Harri Hakula	HUT	Mesh generation, direct and iterative solvers, parallel algorithms
Jari Järvinen	CSC	Mathematical modelling, FEM
Petri Kallberg	Digital Equipment Corp.	Graph partitioning
Jouni Malinen	CSC	Direct and iterative solvers, parallel algorithms
Juha Ruokolainen	CSC	Mathematical modelling, FEM, direct and iterative solvers, parallel algorithms, visualization
Martti Verho	VTT	Preprocessing, CAD Interface
Riikka Virkkunen	VTT	Industrial applications
Joachim Wendt	VTT	Industrial applications

Table 1.2: Members of Research group

the Research group. The members of the Coordinating board and the Research group have an unique mixture of experience in physics, mathematics, computational methods and computer science.

The members of the Coordinating board and the Research group have presented in Tables 1.1 and 1.2 (situation in October, 97).

Previously Rolf Stenberg from Helsinki University of Technology and Jari Hämäläinen from the Technical Research Centre of Finland worked as active members in Coordinating board and Research group, respectively.

1.3 Program Capabilities

1.3.1 Application Areas

ELMER is applicable in multi-physical industrial processes containing various heat transfer mechanisms, i.e., conduction, convection and radiation, incompressible fluid flows, phase changes, displacements and stresses. Typical industrial applications can be

- Chemical industry - heat and mass transfer of materials, e.g., drifting of impurities
- Electronics industry - cooling of components in circuit boards and cabinets
- Semiconductor industry - Czochralski crystal growth, floating zone technique
- Metal industry - continuous casting, wire casting, heat treatment including thermal stresses
- Paper industry - heat transfer in paper machine drying section
- Material processing industry - flows of molten materials

The above list is by no means exhaustive. It gives just an idea what kind of physical phenomena can be simulated by ELMER.

1.3.2 Class of Problems

Problems to be considered can be steady or transient and they can be described either in 2D or axisymmetric geometry. Moreover, the solver (finite element routines, iterative and direct solvers) and the postprocessor support also 3D geometries. In axisymmetric geometry all field variables are independent of azimuthal direction. If there is no swirling motion, the problem is referred to as an axisymmetric problem. Otherwise the problem is referred to as a cylindrical problem.

In ELMER the following phenomena can be considered:

- Isothermal flows
- Pure heat transfer problems
- Coupled fluid flows
- Phase change problems
- Displacements and stresses

In above cases velocity vector, pressure, temperature and displacement vector form a set of primary field variables. In all cases fluid flows are assumed to be laminar. In stress analysis consideration is limited to elastic behaviour of materials.

In isothermal flows there is no temperature dependence in the momentum equation (no energy equation). On the other hand, in pure heat transfer problems there is no convection, i.e., heat is transferred by conduction or radiation only, or there is a constant convection velocity (no momentum equation). In coupled fluid flows velocity field depends on temperature distribution and vice versa. Phase change problems are purposed to model solid-liquid phase changes and they can be included either in pure heat transfer problems or in coupled fluid flows. Displacements and stresses, caused for example by temperature variations, are modeled in solid materials. In the following section the above physical mechanisms are discussed in more detail.

1.3.3 Mathematical Background

In solid and liquid materials heat transfer and incompressible fluid flow are governed by heat and Navier-Stokes equations, which can be derived from the basic principles of conservation of mass, momentum and energy. Fluid can be either Newtonian or non-Newtonian. In the latter case the consideration in ELMER is limited to purely viscous behaviour with the power-law model.

In the following we present the governing equations of fluid flow, heat transfer and stresses in elastic material applied in ELMER. Also the most usual boundary conditions applied in computations are described.

The Governing Equations

The heat equation is expressed as

$$\rho c_p \left(\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T \right) - \nabla \cdot (k \nabla T) = \rho h, \tag{1.1}$$

where ρ is the density, c_p the heat capacity at constant pressure, T the temperature, $\vec{u}$ the convection velocity, k the heat conductivity and h is source of heat.

The Navier-Stokes equations are written as

$$\rho \frac{\partial \vec{u}}{\partial t} - \nabla \cdot (2\mu \overline{\overline{\varepsilon}}) + \rho(\vec{u} \cdot \nabla)\vec{u} + \nabla p = \rho \vec{f} \tag{1.2}$$

$$\nabla \cdot \vec{u} = 0, \tag{1.3}$$

where $\overline{\overline{\varepsilon}}$ is the linear strain rate tensor, p the pressure, and μ the viscosity of the fluid.

Most commonly the term $\rho \vec{f}$ represents a force due to gravity, in which case the vector $\vec{f}$ is the gravitational acceleration. It can also represent, for instance, the Lorenz force when magnetohydrodynamic effects are present.

In pure heat transfer problems the equation (1.1) is applied, whereas in isothermal flows only the equations presented in (1.3) are taken into account.

For coupled fluid flows we assume that the Boussinesq approximation is valid. This means that the density of the fluid is constant except in the body force term where the density depends linearly on temperature through the equation

$$\rho = \rho_0(1 - \beta(T - T_0)), \tag{1.4}$$

where β is the volume expansion coefficient and the subscript 0 refers to a reference state. Assuming that the gravitational acceleration $\vec{g}$ is the only external force, then the force $\rho_0 \vec{g}(1 - \beta(T - T_0))$ is caused in the fluid by temperature variations. This phenomenon is called Grashof convection or natural convection.

The solidification phase change model ELMER uses is an enthalpy based method. The enthalpy is defined to be

$$H(T) = \int_0^T \left(\rho c_p + \rho L \frac{\partial f}{\partial \lambda} \right) d\lambda, \tag{1.5}$$

where $f(T)$ is the fraction of solid material as a function of temperature, and L is the latent heat. The enthalpy-temperature curve is used to compute an effective heat capacity, whereupon the equations become identical to the heat equation. There are two ways of computing the effective heat capacity in ELMER:

$$c_{p,\mathrm{eff}} = \frac{\partial H}{\partial T}, \tag{1.6}$$

and

$$c_{p,\mathrm{eff}} = \left(\frac{\nabla H \cdot \nabla H}{\nabla T \cdot \nabla T}\right)^{1/2}. \tag{1.7}$$

The former method is used only if the local temperature gradient is very small, while the latter is the preferred method. In transient simulations a third method is used, given by

$$c_{p,\mathrm{eff}} = \frac{\partial H/\partial t}{\partial T/\partial t}. \tag{1.8}$$

Frictional heating may be applied by introducing a velocity dependent heat source

$$h_f = 2\mu\overline{\overline{\varepsilon}} : \overline{\overline{\varepsilon}}. \tag{1.9}$$

The magnetic induction equation describes interaction of ionized plasma and applied and flow induced magnetic fields (refer to Appendix B). The magnetic induction equation is given by

$$\frac{\partial \vec{B}}{\partial t} - \frac{1}{\sigma\mu_0}\nabla \times \nabla \times \vec{B} - \nabla \times \vec{u} \times \vec{B} = 0, \tag{1.10}$$

where σ is the electrical conductivity of the material, and μ_0 is the magnetic permeability. The magnetic field induced force term for the flow momentum equations is given by

$$\vec{f}_m = \frac{1}{\mu_0}\vec{J} \times \vec{B}, \tag{1.11}$$

and the Joule heating by

$$h_m = \frac{1}{\sigma\mu_0^2}\mathbf{J}^2, \tag{1.12}$$

where $\vec{J}$ is the current density.

ELMER has also an elastic stress analysis module, which is meant primarily for computing thermal stresses. The equation of stress for isotropic elastic media is

$$-\nabla(\lambda\nabla \cdot \vec{d}) - \nabla \cdot (2\mu\overline{\overline{\varepsilon}}) + \nabla((3\lambda + 2\mu)\beta(T - T_0)) = \vec{F}, \tag{1.13}$$

where $\vec{d}$ is the displacement vector, β the heat expansion coefficient, and $\vec{F}$ a volume force, λ and μ are the Lamé parameters

$$\lambda = \frac{E\sigma}{(1+\sigma)(1-2\sigma)}, \quad \mu = \frac{E}{2(1+\sigma)}, \tag{1.14}$$

E is Young's modulus, σ is the Poisson ratio, and $\overline{\overline{\varepsilon}}$ is the linear strain tensor

$$\varepsilon_{ij} = \frac{1}{2}\left(d_{i,j} + d_{j,i}\right). \tag{1.15}$$

If one is interested in the stresses and not in the displacement field, which is the primary variable, one may compute the stress tensor as a post processing step (one can actually do this inside the ELMER Post program), using the stress-strain relations for elastic media:

$$\overline{\overline{\sigma}} = \lambda(\nabla\cdot\vec{d})\overline{\overline{I}} + 2\mu\overline{\overline{\varepsilon}} - (3\lambda + 2\mu)\beta(T - T_0)\overline{\overline{I}}, \tag{1.16}$$

where $\overline{\overline{I}}$ is the unit tensor.

In ELMER one can choose between transient and steady state analysis. In transient analysis one has to set, besides boundary conditions, also initial values for unknown variables.

The Boundary Conditions

For temperature one can apply boundary conditions and have either temperature or heat flux prescribed.

Dirichlet boundary condition (temperature is prescribed) reads as

$$T = T_0. \tag{1.17}$$

The value of T_0 can be constant or a function of time, position or other variables.

Heat flux depending on heat transfer coefficient α and external temperature T_{ext} may be written as

$$-k\frac{\partial T}{\partial n} = \alpha(T - T_{ext}). \tag{1.18}$$

Both variables α and T_{ext} can be constant or functions of time, position or other variables. If the heat transfer coefficient α is equal to zero, it means that the heat flux on a boundary is identically zero. The Neumann boundary condition $-k\partial T/\partial n = 0$ is also used in a symmetry axis in 2D, axisymmetric or cylindrical problems.

Heat flux can consist of idealized radiation whereupon

$$-k\frac{\partial T}{\partial n} = \sigma\varepsilon(T^4 - T_{\mathrm{ext}}^4). \tag{1.19}$$

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

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

$$-k_k\frac{\partial T_k}{\partial n_k} = \sigma\varepsilon_k(T_k^4 - \frac{1}{S_k\varepsilon_k}\sum_{i=1}^{N} G_{ik}\varepsilon_i T_i^4 S_i), \tag{1.20}$$

where the subscripts i and k refer to surfaces i and k, and the parameters S_i and S_k to the specific surface areas. The factors G_{ik} are Gebhardt factors, and N represents the total number of radiating surfaces present in the system. Emissivities are assumed to be constant on each surface.

One may also give an additionl heat flux term as in

$$-k\frac{\partial T}{\partial n} = q. \tag{1.21}$$

Similarly, boundary conditions for velocity components and tangential or normal stresses can be defined.

In 2D or axisymmetric cases the Dirichlet boundary condition for velocity component u_i is simply

$$u_i = u_i^0. \tag{1.22}$$

A value u_i^0 can be constant or a function of time, position or other variables. In cylindrical cases the Dirichlet boundary condition for angular velocity u_θ is

$$u_\theta = \omega, \tag{1.23}$$

where ω is the rotation rate.

In axisymmetric geometries one has to set $u_r = 0$ and $\partial u_z/\partial r = 0$ (in cylindrical problems also $u_\theta = 0$) on the symmetry axis.

If there is no flow across the surface, then

$$\vec{u}\cdot\vec{n} = 0 \tag{1.24}$$

where $\vec{n}$ is the outward unit normal.

Surface stresses can be divided into normal and tangential stresses. Normal stress is usually written in the form

$$\sigma_n = \frac{\gamma}{R} - p_a \tag{1.25}$$

where γ is the surface tension coefficient, R the mean curvature and p_a the atmospheric (or external) pressure. Tangential stress has the form

$$\sigma_\tau = \nabla_s\gamma, \tag{1.26}$$

where ∇_s is the surface gradient operator.

The coefficient γ is a thermophysical property depending on the temperature. Temperature differences on the surface influence the transport of momentum and heat near the surface. This phenomenon is called Marangoni convection or thermocapillary convection. The temperature dependence of the surface tension coefficient can be approximated by a linear relation:

$$\gamma = \gamma_0(1 - \vartheta(T - T_0)), \tag{1.27}$$

where ϑ is the temperature coefficient of the surface tension and the subscript 0 refers to a reference state. If a Boussinesq hypothesis is made, i.e., the surface tension coefficient is constant except in (1.26) due to (1.27), the boundary condition for tangential stress becomes

$$\sigma_\tau = -\vartheta\gamma_0\nabla_s\gamma. \tag{1.28}$$

In equation (1.25) it holds then that $\gamma = \gamma_0$. The linear temperature dependence of the surface tension coefficient is naturally only one way to present the dependence. In fact, the coefficient γ can be any user defined function in ELMER. One may also give the force vector on a boundary directly as in

$$\overline{\overline{\sigma}} \cdot \vec{n} = \vec{f}. \tag{1.29}$$

For stress analysis one may prescribe displacement on a boundary

$$d_i = d_i^0, \tag{1.30}$$

or force on a boundary as in

$$\overline{\overline{\sigma}} \cdot \vec{n} = \vec{f}. \tag{1.31}$$

1.3.4 Numerical Methods

As is well known, the convective transport term of the Navier-Stokes equations and the heat equation is a source of both physical and numerical instability. The numerical instability must be compensated somehow in order to solve the equations on a computer. For this reason so called stabilized finite element method ([1],[2]) is used in ELMER to discretize these equations. The equations in the form implemented in ELMER and the discretization are described in more detail in Appendix B. In Appendix C a brief introduction to the methods used in computing the diffuse gray radiation boundary condition is given. In Appendix D the free surface problem is presented.

The convection term of the Navier-Stokes equations is nonlinear and has to be linearized for computer solution. There are two linearizations of the convection term in ELMER:

$$\vec{u} \cdot \nabla \vec{u} \approx \vec{\mathcal{U}} \cdot \nabla \vec{\mathcal{U}} \tag{1.32}$$

and

$$\vec{u} \cdot \nabla \vec{u} \approx \vec{u} \cdot \nabla \vec{\mathcal{U}} + \vec{\mathcal{U}} \cdot \nabla \vec{u} - \vec{\mathcal{U}} \cdot \nabla \vec{\mathcal{U}}, \tag{1.33}$$

where $\vec{\mathcal{U}}$ is the velocity vector from the previous iteration. The first of the methods is called Picard iteration or the method of the fixed point, while the latter is called Newton iteration. The convergence rate of the Picard iteration is of first order, and the convergence might at times be very slow. The convergence rate of the Newton method is of second order, but to succesfully use this method, a good initial guess for velocity and pressure fields is required. The solution to this problem is to first take a couple of Picard iterations, and switch to Newton iteration after the convergence has begun.

The heat equation is also a nonlinear when radiation is modelled. The remarks made previously for the Navier-Stokes equations apply here as well.

When iterating the nonlinearity one may also use relaxation, where the current solution vector is replaced by a vector defined as

$$u_i' = \lambda u_i + (1 - \lambda) u_{i-1}, \tag{1.34}$$

where λ is the relaxation factor. A factor below unity might help to stabilize the nonlinear solver. A factor above unity might speed up the convergence.

Time integeration of the equations is done with the implicit Euler method. Mathematically this reads

$$M\frac{\partial \mathbf{u}}{\partial t} + A\mathbf{u} = F \Rightarrow \frac{1}{\Delta t}M(\mathbf{u}_i - \mathbf{u}_{i-1}) + A\mathbf{u}_i \approx F_i. \tag{1.35}$$

ELMER has both a direct and iterative solver for solving the various linear systems. The direct solver uses a band matrix solver from the LAPACK set of linear algebra routines. This routine is based in LU decomposition. The iterative solver uses the package HUTIter described elsewhere in this document. HUTIter contains various iterative solution algorithms, including stabilized bi-conjugate gradient (BiCGStab) method, transpose free quasi minimal residual (TFQMR) method, conjugate gradient squared (CGS) method, and generalized minimal residual (GMRES) method. With a large base of experiments behind, we recommend the use of the stabilized biconjugate gradient method, at least with the ELMER Solver.

The iterative methods are almost always not usable for the linear systems in our context without preconditioning. The preconditioning method of choice in ELMER is the Incomplete LU (ILU) decomposition preconditioning. As the name implies, some part of the LU decomposition of the linear system coefficient matrix is computed. The decomposition is then used to stabilize the linear system. Instead of the original equation

$$Ax = b \tag{1.36}$$

we will try to solve the equation

$$(ILU)^{-1}Ax = (ILU)^{-1}b, \tag{1.37}$$

which will hopefully be more stable than the original equation. Experiments have shown this method to be effective. The equation (1.37) is basically the idea behind the method, not the actual algorithm used.

1.3.5 ELMER Structure

ELMER consists of the CAD Interface, the mesh generator, the solver and the postprocessor. In addition, computation of view factors needed in radiation modeling is available. The computational flow of ELMER is shown in Figure 1.1. ELMER parts are introduced in detail in the following chapters.

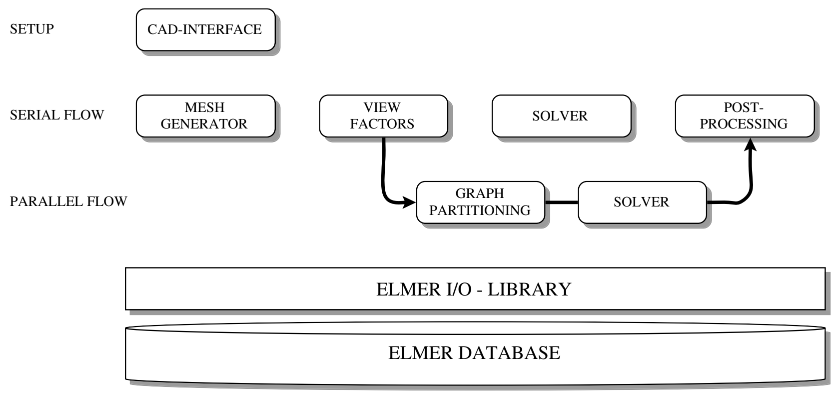

Figure 1.1: Computational flow of the ELMER program

ELMER is a collection of programs rather than a single monolithic application. Each phase of the computational process, such as mesh generation or solution of the linear equations, is handled separately. The separate processes communicate via a model database using a common set of routines. This flexible design enables the engineer to work on a small scale simulation on his desktop before performing the full simulation on an appropriate machine.

Chapter 2

The Cad interface

2.1 Introduction

Cad interface is the ELMER user interface module. With it you can set all the definitions which are needed for solving a simulation problem with other ELEMR modules.

When starting with a new problem, model geometry must be first read from an external file. This file can be a cad model which has been created with an external cad program or by other means. Currently only linear 2D cad geometries are supported. This is because current ELMER mesh generator is restricted to linear 2D geometry.

Input file can also be an external 2D or 3D mesh file. Currently supported mesh file formats are described below in context of the File menu commands.

This external file should describe the geometry of all the bodies (called also objects or materials) in the problem domain. Cad interface creates internally a so called boundary representation from the input geometry. This means that the geometry should be completely described by the body edges in the cad model or by the mesh surface elements in the external mesh file.

All bodies should form a closed and connected area. In 2D this means that the edge-loops, which define the bodies, should be closed and not intersecting with themselves or with any other edge-loops.

In 3D bodies are defined by surfaces (surface patches) and restrictions are similar as in 2D. Surfaces are described by (3D) edge-loops and again similar restrictions hold as for bodies in 2D.

Boundaries (edges or surfaces) of separate bodies can be adjacent. They form so called inner boundaries between bodies. Other boundaries are called outer boundaries.

In the following inner and outer body boundaries are often called as boundary elements. These should not be confused with the mesh elements.

When an external cad geometry is read in, it cannot be changed. The boundary elements in the cad input file are consequently the smallest level where, for example, boundary conditions can be set. On the other hand, boundary elements cannot be recombined into larger elements. Creating unnecessary many boundary elements (edges) will create a long list of separate boundary elements, although from the problem point of view these elements could form a single physical boundary. These restrictions should be taken into account when defining the geometry in the external cad program.

On the other hand, when we use an external mesh file, a typical problem is that we have too few boundaries. Without any specific boundary information, the only way to find boundaries is to devide them into internal and external boundaries based on body (material) information which is attached to each mesh element. If we have only one body, this would mean that we have only one (outer) boundary for the whole whole model. However, contrary to the cad geometry, it is possible to modify mesh boundaries. You can split and combine mesh boundaries by regrouping elements which defines these boundaries. This procedure is explained in the context of the Edit menu commands.

2.2 Model data and files

When working with cad interface, there is no need to directly manipulate the files which store the model data. However, it is useful to understand how data and files are organized.

Cad interface stores all the model definitions in a separate model file. Default name for this file is *modelname*.ecf ('ecf' for Elmer Cadinterface File). This file stores all the model information in an internal format, except the possible finite element mesh. Only mesh file name is stored in the model file.

When a model file is created and saved, all the model information which is relevant for Elmer Solver is by default stored in a separate solver input file. Default name for this file is *modelname*.sif ('sif' for Solver Input File).

If model geometry is read from an external cad file, Elmer Mesh program can be used to generate the finite element mesh. This mesh is stored in a set of files which are named using the following scheme: *modelname*.mesh.header, *modelname*.mesh.nodes, *modelname*.mesh.elements, etc.

The dircetory where these files are stored depends on the model directory and model name. For example, if model directory is /elmer/MODELS and model name is stepflow, all files will be stored by default in the directory /elmer/MODELS/stepflow.

So, model name defines the lowest level for the directory. Model name is also by default part of the individual file names.

For cad interface the model file is the most essential file. All the other files can be recreated from this file using the cad interface. But if model file is deleted, all definitions are also lost.

2.3 Using Cadinterface

Cad interface is started by entering command `ElmerCadi` in the command line. You can use the following options when starting the program:

- –model-directory=*model directory*
- –model-name=*model name*
- –settings-file=*settings file name*

For example, the following command would set model's files directory to be /elmer/MODELS/stepflow:

`ElmerCadi -model-directory=`/elmer/MODELS/ `-model-name=`stepflow

If a model file stepflow.ecf were stored in that directory, it would be automatically loaded in the cad interface.

The settings file is a way to control the default behaviour of the cad interface. If option `-settings-file` is not used, program tries to open a file named ElmerCadiSettings_*username*.txt or ElmerCadiSettings.txt, where username is the string stored in the shell variable USER (or USERNAME). These files are searched in the previous order first in the current directory and then in the directory given in the environmental variable ELMER_CADI_HOME, if this is defined.

If no settings file is found (or given as command line option) preset default values are used. The structure and available keywords in the settings files are explained in the appendix F.

When you start the cad interface, the main window with menu and command buttons is displayed as shown in Figure 2.1.

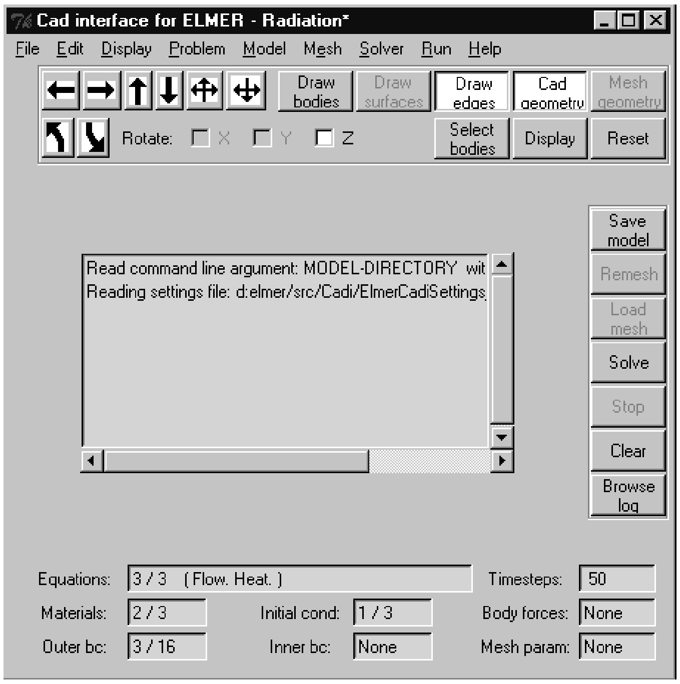

Figure 2.1: The cad interface main screen.

At the top of the window is the menu bar and two rows of command buttons. At the right side is an addional column of buttons. In the middle of the window is a message area, where short messages concerning different actions and commands are displayed. The `Clear` command button deletes all current messages in the message area. The `Browse log` command button open the latest module log file in a brwoser window.

At the bottom of the window is located a set counter fields. They display the total number of different parameters and the number of currently defined parameters. For example, you can see in Figure 2.1 that 50 timesteps have been defined, and that only one of the three bodies has currently an initial condition defined.

The main idea of the cad interface is to offer an easy to use and flexible interface for the problem definition. Consequently, there are no strict rules which should be obeyed when entering the data. However, certain basic definitions must be first given, until further data entry is possible. The following is a guideline for the recommended data entry steps when defining a new problem:

- Read the geometry from an external cad file or mesh file
- Give a name for the model (model name is needed when creating the model directory)
- Modify mesh boundaries, if needed (when starting from an external mesh file)
- Give meaningful names for bodies and boundaries
- Set coordinate system and the time dependency type for the model
- Assign equations for all the bodies
- Set other body and body element related data (material properties, boundary conditions etc.)
- Set solver parameters, timestepping rules etc.
- Save the model

Defining body equations is perhaps the most important of these tasks. On one hand, all the bodies in the problem must have an equation before any equation related data (like material properties, body forces, initial and boundary conditions) can be given for a body. On the other hand, when any equation related data (like initial or boundary conditions) is entered for a body, changing the already assigned equation could mean that these definitions must be removed from the model.

This dependency is mainly because assigning an equation also defines the type of suitable constraints for the body. For example, if their is no flow equation assigned for a body, it should not be possible to enter any flow dependent parameters for the body. On the other hand, if some parameters are set for a body, changing the body equation could introduce contradictions with the existing definitions, and the conflicting parameter settings must be removed.

In the rest of this chapter we go through more thoroughly the menus and commands and explain how they are used for the problem definition.

2.4 File menu

File menu commands are used for opening, saving and browsing the model data and related files. These commands are shown in Figure 2.2.

Commands for creating a new model or opening an existing model are the following:

Figure 2.2: The file menu.

- `Open Cad file`: read model geometry from an exernal cad file
- `Open mesh file`: read model geometry and mesh from an external mesh file
- `Open model file`: read a previously saved model file (.ecf file)

If a model was previously opened and there is some unsaved data, you will be given an option to save the old model before loading a new one.

Default file name extensions for cad geometry files are `.unv` (Ideas neutral cad file format), `.igs` (Iges file format) and `.egf` (Elmer geometry file format). The last format is explained in the appendix G.

Data in these files should describe bodies in wireframe 2D format. This is currently the only cad geometry format that is supported.

Default file name extensions for external mesh files are `.unv` (Ideas mesh files) and `.inp` (Abaqus mesh files).

The default extension for model files is `.ecf`. This extension is optional, but it is recommended for clarity.

Commands which store model definitions and related data are the following:

- `Save model file`: saves model file (.ecf file), solver input file (.sif file) and stores model geometry in the model's data files directory
- `Save model file As`: same as previous, but the name and location for the model file is asked instead of using the default name and location.

When model definitions have been modified, model data should be saved. Otherwise updated model data is not available for other ELMER modules.

If mesh is very large, it could be useful not to read the mesh automatically when a model file is opened for example just to change some material properties (this can be controlled by the cad interface settings). Or, it could be useful to remove a large mesh from the cad interface memory before starting other ELMER modules. The following commands can be used during the cad interface session to load/unload the mesh:

- `Load mesh file`: if mesh file is available, this commands reads mesh
- `Unload mesh file`: this commands removes mesh from the memory

The following commands open files in a browser window. It is not possible to edit the files using these commands.

- `Browse model file`: displays model file in a browser window
- `Browse solver input file`: displays solver input file in a browser window
- `Browse log file`: displays the log file which was created by the last activated ELMER module

Note: although model file is a normal text file, it is not meant to be edited 'offline'. The internal structure of the file is strictly predefined and any outside changes could make it unreadable for the cad interface program.

- `Save mesh in Elmer format`: by default an external mesh is saved in Elmer format when model is save. You can use this command to do it explicitely, if the default behaviour is not in use.
- `Save mesh in ElmerPost format`: saves mesh in Elmer postprocessor format

`Exit` command ends the session. If there is any unsaved data, you will be given an option to save the data before ending the program.

2.5 Edit menu

Body properties

This command opens a panel where you can edit body names and the colors which are used when bodies are displayed in the graphics window. This panel is shown in Figure 2.3.

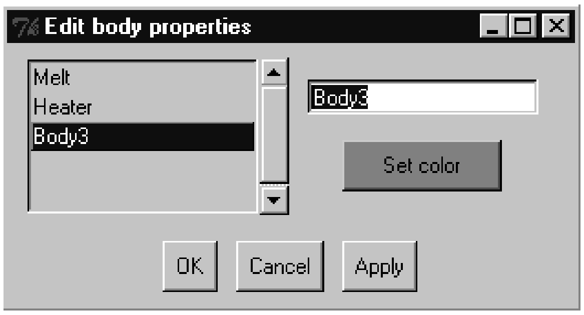

Figure 2.3: The Body properties panel.

Default body names are in the form BodyN, where N is a sequence number. Because body names are displayed in the panels where body related data is entered, given meaningful names for the bodies could make data entry easier.

When a body is selected from the list box, its name is dislayed in the name entry field. You can edit the name in this field.

The color for a selected body is shown on the `Set color` button. This color is used when the body is displayed in the graphics window.

If the graphics window is not currently visible, you can open it by pressing the `Display` button in the toolbar.

Pressing the `Set color` button opens a color palette where you can select a new color for the body.

Boundaries

This command opens a panel where you can edit boundary names. If geometry is read from an external mesh file (ie. model does not contain a cad geometry), you can also edit boundaries by splitting and combining them. This panel is shown in Figure 2.4.

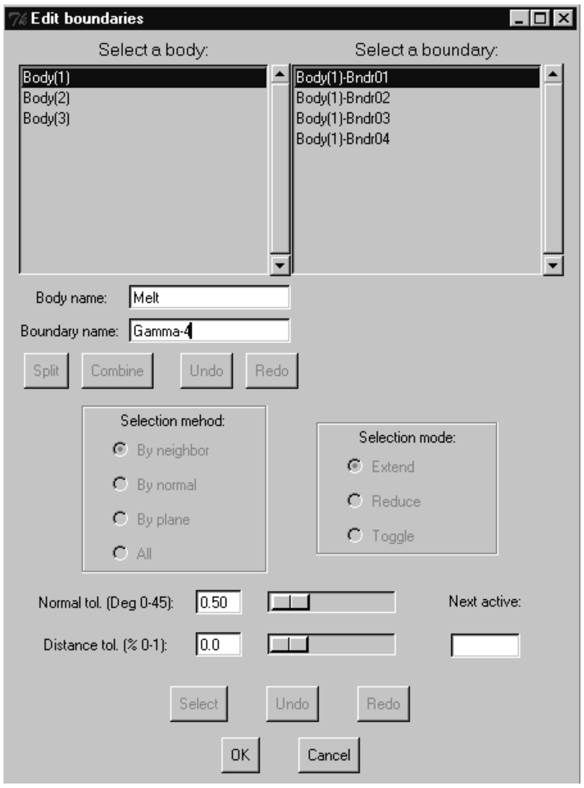

Figure 2.4: The Boundaries panel.

Default boundary names are in the form BoundaryN, where N is a sequence number. You can select a boundary from the list box, or you can select it from the graphics display. The name of the selected boundary is diplayed in the boundary name entry field, where you can edit it.

You can only edit boundaries for one body or bodypair at a time. Boundaries are split by selecting a group of mesh elements in the boundary being edited, and using the split command. Only one boundary can be split at a time. Only complete boundaries can be combined. Split and combine operations can be applied in any order and both operations can be cancelled using the undo command.

A boundary is selected for editing by double clicking the boundary either in the boundary list box or in the graphic display. If you press control key at the same time, you can select multiple boundaries.

There are two methods to select mesh elements for split operation within a boundary. You can either select elements by 'painting' them. This is done by pressing the shift key and painting with the mouse while pressing the left mouse button. If you press the control key, instead of the shift key, you can select mesh elements by clicking on them with the left mouse button. Using the right mouse button cancels selections.

The other way is to first select a reference mesh element using the mouse. After that you can extend the selection by using the `Select` button in the panel. What is selected is controlled by the criteria in the `Selection method` radiobutton group. The criteria have the following meaning:

- `By neighbour`: selection is extended by adding neighbour elements for currently selected elements. Neighbour elements are added only if their normal vector changes less than what is set by the `Normal tolerance` slider. Using this method you can quite easily and quickly extend the selection along a smooth curve or a curved surface
- `By normal`: select all elements whose normal vector differs less than the normal tolreance from the normal vector of the reference element.
- `By plane`: select all elements which are on the same plane as the reference element. Plane distance from the origin can differ by the plane tolerance which is set using the `Distance tolerance` slider
- `All`: select all mesh elements in the boundary

How the selection actually behaves, can be be controlled by the settings in the `Selection mode` radiobutton group:

- `Extend`: always extend current selection
- `Reduce`: always reduce (unselect) current selection
- `Toggle`: toggle the selection state for the elements being effected

Value displayed in the `Next active` field tells the next tolerance value which would select new mesh elements.

Remove cad geometry

If model's geometry is defined by a cad geometry, its mesh boundaries cannot be modified. This limitations is to prevent conflicts between cad and mesh geometry. However, if the original cad geometry is removed from the model, existing mesh can modified as it were a normal external mesh.

With this command you can remove the original cad geometry from the model. Note that remeshing is not possible if cad geometry is removed.

Edit solver input file

With this command you can open an existing solver input file for editing. Note that all changes will be by default overwritten when the model is saved for the next time. However, you can prevent this by setting the `Auto Save Solver Input` flag to be False in the settings.

Settings

This command opens a panel where you can modify current settings. This panel is shown in Figure 2.5.

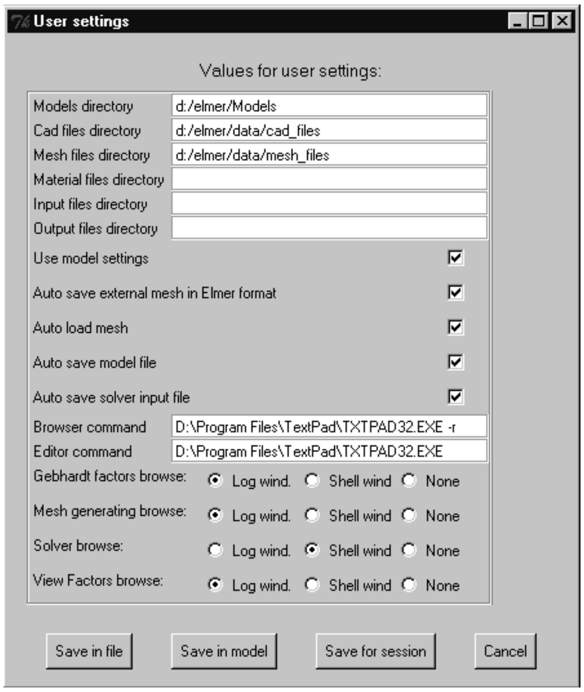

Figure 2.5: The Settings panel.

Entries in this panel correspond to the keywords in the settings file. These keywords are explained in the appendix F.

You can use the following options to save changes made in the panel:

- `Save in file`: settings are saved in the current settings file
- `Save in model`: settings are saved (when applicable) in the current model file
- `Save for session`: settings apply only in the current session

Occasionally, it could be useful to use the `Save for session` option before loading a new model. For example, if the model contains a very large mesh, and you would not like to load the mesh, you could temporarily turn off the loading. This can be done by unchecking the `Use model settings` and `Auto load mesh` checkboxes, and using then the `Save for session` command.

2.6 Display menu

Model can be displayed using the `Display` command in the `Display` menu. This command is also attached to the `Display` button. It is located in the toolbar below the menubar.

Model is displayed in a separate graphics window. It can be manipulated in the window either by dragging it with the mouse, or using the buttons in the toolbar. Dragging with the left mouse button moves the model in the window. Dragging with the right button rotates the model. You can scale (zoom) the model with the middle mouse button (or pressing left and right buttons simultaneously). `Reset` command sets the display into its initial state.

An example of the graphics window is shown in Figure 2.6. It shows the geometry of the radiation problem which is described in 7.

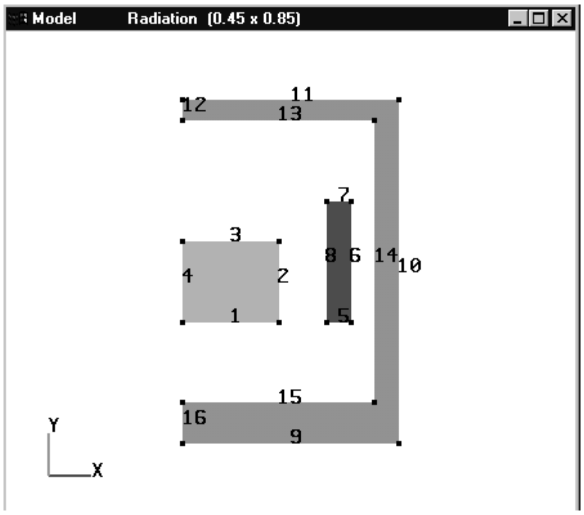

Figure 2.6: Display of the radiation example.

Bodies are displayed using different colors. All body elements (inner and outer boundaries) are uniquely numbered. These numbers are also used to identify the elements in the data entry panels. Numbering is fixed and it is internal to the system. Numbers do not have any specfic meaning and the numbering is not necessarily even continuous.

You can select bodies to be displayed using the `Select bodies` command in the `Display` menu or with the button in the toolbar. This option is not very useful for 2D models, but for complex 3D model it can be quite helpful.

In the toolbar, there are five other buttons, which control how the geometry is displayed:

- `Draw bodies`: cad geometry is drawn in solid format, mesh geometry is drawn using volume element edges
- `Draw surfaces`: mesh geometry is drawn using filled boundary elements (only for 3D models)
- `Draw edges`: cad geometry is drawn using edges, mesh geometry is drawn using boundary element edges

- `Cad geometry`: activates/deactivates cad geometry displaying (if cad geometry is available)
- `Mesh geometry`: activates/deactivates mesh geometry displaying (if mesh is geometry available)

2.7 Problem menu

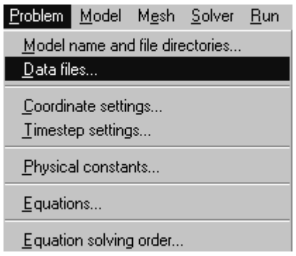

Figure 2.7: The Problem menu.

This is the menu where the actual problem definition starts. Most of the definitions concern the whole model and are not specific to bodies or body elements.

Model name and file paths

This command opens a panel where you can enter a name for the model and also the file paths which are used when reading and saving data files. This panel is shown in Figure 2.8.

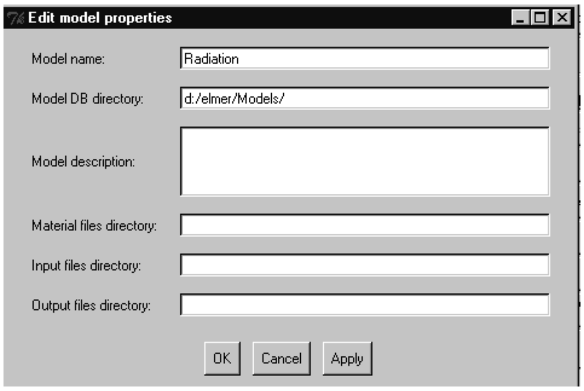

Figure 2.8: The Model name and file paths panel.

The first three fields are:

- `Model name`: used to identify model data in the database
- `Model DB directory`: root directory for the model database

- `Model description`: comment text (optional)

Model description is optional and it is displayed only in this panel.
The last three fields are for setting the input and ouput paths:

- `Input file path`: this is used as a common input path whenever a filename is entered without a path. For instance, all external files which store boundary condition definitions, could be stored in one directory. When a filename is then given as a boundary condition 'value' in the boundary condition definition panel, it would not be necessary to enter each time a path name.
- `Material file path`: this is used similarily as the input path, but only in the material properties definition panel. This way it is possible to collect all material files into one directory as a kind of a material database
- `Output file path`: the default directory for solver output (result) files.

Datafiles

This command opens a panel where you specify names for input and output files. The panel is shown in Figure 2.9

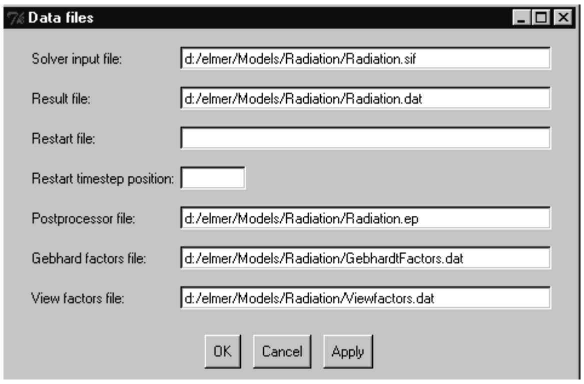

Figure 2.9: The Data files panel.

Most of the file names in the panel have a default value. This value is constructed by adding the default path to the default file name. The default path is formed by adding the model name to the model DB directory.

If you enter a path name in addition to the file name, the value entered in the field is used as it is (ie. nothing is added to entered values). If there is only the file name part in the field and the `Apply` or `Ok` button is pressed, default path is automatically added to the file name.

For example, if you want to use the default solver input file name and the current directory, you should enter: ./stepflow.sif in the `Solver input file` field. If you want change the path to the default path, you can either enter the path name explicitely, or you can edit the field so that there is only the default file name (stepflow.sif) and then press the `Apply` button.

The fields in the panel are:

- `Solver input file`: input file for the solver, deafult: *modelname*.sif
- `Output file`: solver output file, default: *modelname*.dat
- `Restart file`: if name is given, solver uses variable values in this file as initial values. This file should be in the output file format.
- `Restart timestep position`: timestep where to start in the restart file
- `Postprocessor file`: this file stores results in ELMER postprocessor format, default : *modelname*.ep. This file is also used as the input file when postprocessor is started from the Run menu.
- `Gebhardt factors file`: this file is used to store Gebhardt factors when they are calculated using the Gebhardt factors command in the Run menu, default: GebhardtFactors.dat.
- `View factors file`: this file is used to store view factors when they are calculated using the View factors command in the Run menu, default: ViewFactors.dat.

Coordinate settings

Model's coordinate system is specified in the `Coordinate settings` panel. This panel is shown in Figure 2.10.

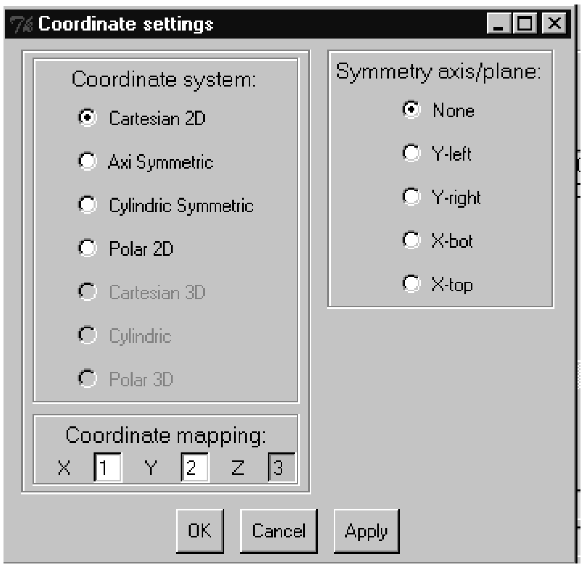

Figure 2.10: The Coordinate settings panel.

The coordinate system which can be selected depends on the dimension of the model geometry. In 2D possible systems are cartesian 2D, axisymmetric, cylindric symmetric and polar 2D. In 3D options are cartesian 3D, cylindric and polar 3D.

Values for coordinate dependent variables (like velocity) are entered in the data panels by components (like velocity-X, velocity-Y, velocity-Z). Default mapping between the components and the coordinate axes is X = 1, Y = 2 and Z = 3. This mapping can be changed entering a new index set in the `Coordinate mapping` fields. The field labels for theses entries also changes when coordinate system is changed:

- Cartesian: X, Y, Z
- Cylindric: R(ho), Z, P(hi)
- Polar: R(ho), T(heta), P(hi)

These labels are also used in data entry panels to remind you on the coordinate system and on the mapping which is in use.

If coordinate system is cartesian, it is also possible to set the symmetry axis (2D) or symmetry plane (3D). This information can be used for automatic setting of suitable boundary conditions at the symmetry axis/plane.

Timestep settings

Time dependency and timestep settings for the simulation are set in the `Timestep settings panel`. This panel is shown in Figure 2.11.

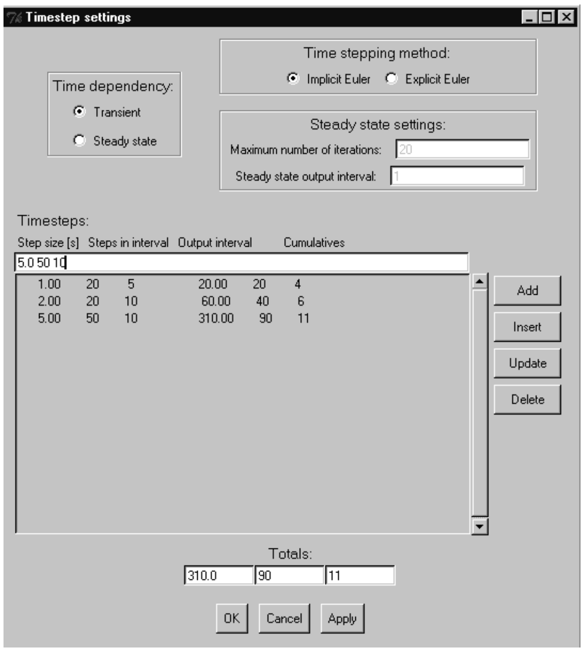

Figure 2.11: The Timestep settings panel.

The time dependency type of the problem is seleceted in the `Simulation type` box, where options are `Steady state` and `Transient`.

If `Steady state` is selected, you can use the following fields to control the number of iterations and the output frequency of the results:

- `Maximum number of iterations`: Maximum limit for the steady state iterations even if convergence criterion is not yet met.

- `Steady state output interval`: Output frequency for steady state iterations

If `Transient` problem is selected, timestepping method can be selected using the `Timestepping method` radiobuttons.

For transient problems it is also possible to use different timestep sizes. This is done by entering three numbers in the `Timestep settings` field. They have the following meaning (enter values in this order):

- `Timestep size`: step size in seconds in the interval
- `Number of steps`: number of steps in the interval
- `Output interval`: output frequency in the interval

These entries are added to the timestep list using the `Add` or `Insert` button. Insertion is done above the row which is currently selected in the list. An existing entry can be changed with the `Update` button. The `Delete` button removes the currently selected entry from the list.

For each row in the list, running cumulatives are displayed as the last three numbers. Total cumulatives are shown at the bottom of the panel in the `Totals` fields.

Physical constants

Physical constants can be changed from their default values in the `Physical constants` panel. Currently the constants are the gravity vector and the Stefan-Boltzmann constant.

Constants are expressed in standard units. The default value for the gravity is 9.82 m/s^2 in the direction of the negative y-axis. If the input data for the problem is expressed in non-standard units, it is also necessary to change the values of the physical constants accordingly.

Equations

All bodies in the model need an equation definition. Otherwise the simulation problem cannot be solved. Equations for the bodies are set using the `Equations` command in the `Problem` menu. This command opens a panel which is shown in Figure 2.12.

In this example the equations for the radiation problem (see 7) are being defined. The body with the flow equation is currently selected.

Body name is shown in the `Body name` field. Note that this name is not used in the body listbox.

Equation definitions are displayed in the listbox which is to the right of the body listbox. The number at the beginning of each line is the equation number. When an equation is applied to a body, this number is displayed in the body listbox rows after the 'Eq' text. So text Body1-Eq3 tells that equation number 3 is applied to the body number 1.

You can give a name for an equation by selecting it from the equation list and entering the name in the `Parameter name` field. Press the enter key to make this name effective. Each time you select an equation, its name is displayed in this field.

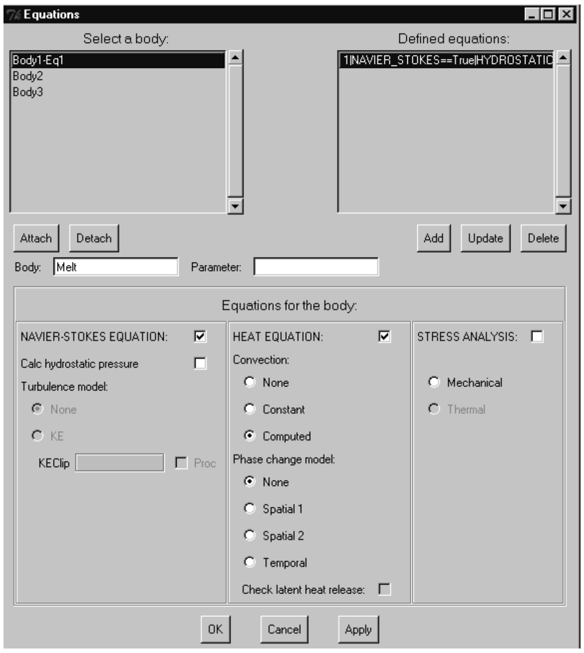

Figure 2.12: The Equations panel.

When you want to attach an equation to a body, select first the body from the body list, then select the equation from the equation list and press the `Attach` button.

If a selected body already has an equation defined, this equation is automatically marked selected in the equation list, but you can freely select an other equation and attach it to the body.

New equations are added to the equation list by entering equation parameters in the data entry area in the lower part of the panel. Press the `Add` button below the equation listbox to add the new equation to the list.

The buttons located in the middle of the panel have the following functions:

- `Attach`: apply the currently selected equation to the currently selected body
- `Detach`: remove any equation definition from the currently selected body
- `Add`: add a new equation to the equation list with the values in the data entry fields
- `Update`: update the currently selected equation with the values in the data entry fields
- `Delete`: delete the selected equation from the equation list and remove equation definition from the bodies it possibly was applied

When you press the `Ok` button, all changes are checked and accepted if correct, and the panel is closed. Note however, that you still have to save the model before changes are stored permanently. If you press the `Cancel` button, all current changes in the panel are discarded.

When data is entered, its correctness is checked. Checking rules come partly from the physics of the problem, partly from the limitations in the solver. They are the following:

- `Heat equation`: selecting `Heat equation` includes automatically heat conduction. You can add `Constant` convection (values are given as material parameters for the body!) or `Computed` convection, but the latter only if laminar flow is also selected. Phase change settings are only available if `Heat equation` is first selected.
- `Stress`: `Thermal stress` is possible only if `Heat equation` is also selected. You can select either `Mecahnical stress` or `Thermal stress`, but not both.

The physical meaning and the computional procedures for these items ares explained in more details in 5.

IMPORTANT: You should note that each body should have an equation definition before the model can be solved. Model file can be saved although all definitions are not complete, but model is still not solvable. When existing equation definitions are changed and if any equation related data (material parameters, conditions etc.) has already been entered for bodies or body elements, all conflicting definitions will be automatically removed. So, you should carefully check all previously entered data, if you change equation definitions!

Equation solving order

In this panel you can specify the solving order for the equations. Default order is Heat equation, Navier-Stokes, KE Turbulence and Stress Analysis. You can modify this by entering the new order numbers in the panel. Order numbers shoul be unique, start from one and be consecutive.

2.8 Model menu

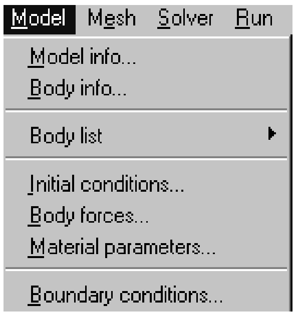

Figure 2.13: The Model menu.

When all body equations and other general problem data has been entered, you can enter the model parameter specifications. It is done using the commands in the `Model` menu.

Model info

This command opens a panel which displays general model info. You cannot edit the data in this panel. Panel is shown in Figure 2.14.

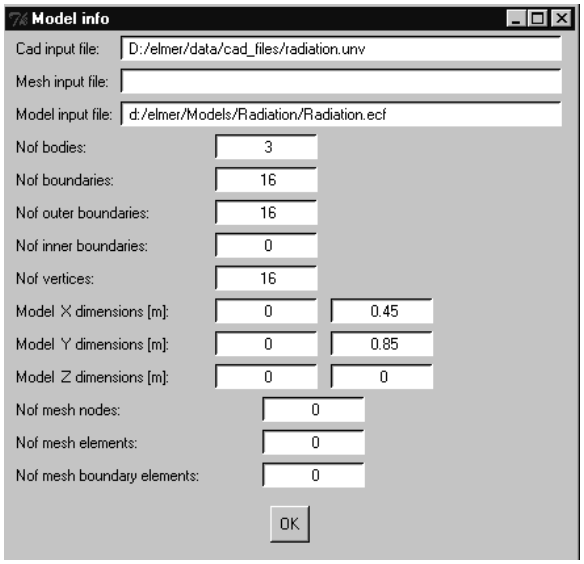

Figure 2.14: The Model info panel.

Fields in the panel are as follows:

- `Cad input file`: the name of the original cad source file
- `Mesh input file`: the name of the mesh input file
- `Model input file`: the name of the model file
- `Nof bodies ... Nof vertices`: counters for different objects defining the geometry
- `Model dimensions`: minimum, maximum and size for x,y and z coordinates, ie. model's bounding box
- `Number of mesh elements`: number of mesh volume elements (0 if mesh does not exist)
- `Number of mesh boundary elements`: number of bounary type mesh elements
- `Number of mesh nodes`: total number of nodes in the mesh

Body info

This command opens a panel where you can check dimensions and number of mesh elements for each body. You cannot edit data in this panel.

Body list

This command opens a submenu which displays a list of all bodies in the model. Buttons are colored with the body colors and body names are used as button labels. This is simply meant as a quick info on the body colors and names, you cannot activate any further commands from this menu.

Initial conditions Body forces Material parameters

In the `Initial conditions` panel you enter initial values for variables. External body forces are set in the `Body forces` panel. In the `Material parameters` panel you enter values which describe physical properties of the bodies.

All these panels concern data which applies to the whole body. The user interface in these panels is also very similar. In the following we explain the working with these panels, using the `Initial conditions` panel as an example.

The meaning and functioning of the buttons is also the same as those in the `Body equations` panel. These were already described in 2.7

Initial conditions panel is shown in Figure 2.15.

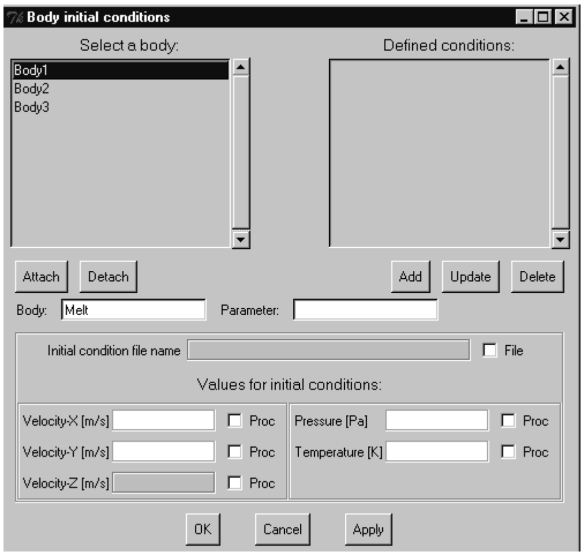

Figure 2.15: The Initial conditions panel.

The main thing to notice is that entry fields which are displayed on the panel are equation type dependent. If, for example, none of the bodies has been applied a flow equation, no flow related entry fields are displayed on the panel. On the other hand, the combination of all body equations define the fields which are displayed. This means that not all of the entry fields are necessarily applicable to all the bodies. When you select a body from the body list, only those fields are active which can be applied to this body.

Before you can enter any data, you have to select a body or a previously added initial condition. This means that all initial conditions (also body forces and material parameters) are defined for a certain equation type, because this is implied by the selected body. This also means that you can attach an initial condition only to a body which has the same equation type as what the condition has.

When you add a new condition, it is by definition applicable to the body which was selected. But knowing whether a previously defined condition is applicable to a body could be a problem. However, when you select a body, all those initial conditions which are allowed for this body, are marked with an asterix (*) in the beginning of the rows in the initial condition listbox. Any of these marked conditions can be attached to the body.

A condition is attached to a body by selecting the condition and the body in the corresponding listboxes and then clicking the `Attach` button.

If you select a body and then click `Detach` button, condition is removed from the body. The data entered in the entry fields is checked when you press the `Add` or the `Update` button. Each field has a predefined type and if you enter incorrect data (like letters in a numeric field), update is not accepted and an error box is displayed.

This data checking concerns mainly the formal correctness of the data. The checking of the logical or physical consistency of the data is quite limited.

The following entry fields need a bit more explanation:

- `Initial condition file name`: in this field you can enter the file where you have stored previously defined initial conditions (body forces, material properties). The file format is described in H. When you click the `File` checkbutton this field is activated. If you enter only a file name in this field, the `Input file path` from the `Problem/Model name and file paths` panel is added as the path name. If you add the path explicitely, data in the field is used as it is. If you click off the `File` checkbutton, the data entry in the field is disabled.

- `Proc` checkbutton: clicking this checkbutton changes the corresponding entry field to a text entry field. You can then enter two strings in the field: a module name and a function name. These define the procedure which should be used to calculate the variable values in the in the solver. When you click off the checkbutton, the entry field returns to it original role.

Using a file as a source for the parameters can be a quite useful option. You can enter more complicated definitions for initial conditions (and for body forces, material parameters and boundary conditions as well) in a file than what is currently possible from the cad interfce panels. In the panels you can enter only constant values, in a file it is possible to enter parameters which depend on temperature, coordinates or any other problem variable.

If a filename is entered, it will be located before any entry field values in the solver input file. This means that entry field values will overwrite possible definitions in the parameter input file. This way you can store 'permanent' values in files and make quick experimental changes using the entries in the panel.

Boundary conditions

This panel differs from the previous panels because boundary conditions are applied to boundaries (body elements). So, in addition to the body and boundary condition listboxes, this panel contains a third, body element listbox. The panel in Figure 2.16.

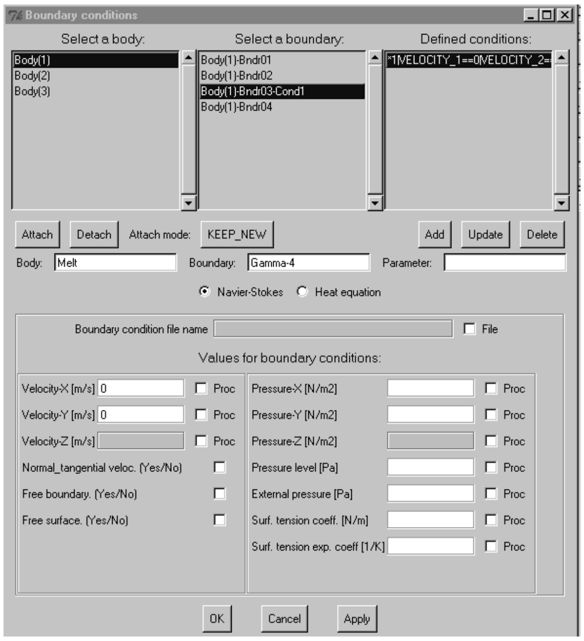

Figure 2.16: The Boundary conditions panel.

Entry fields are grouped by the equation type in this panel. If the problem contains more than one equation type (Navier-Stokes, Heat or Stress), you can use the radiobuttons in the middle of the panel to select the equation.

In our example we have radiobuttons for Navier-Stokes and Heat equation. Currently the Navier-Stokes equation is selected and we have the entry fields related to this equation displayed on the panel.

Boundary elements are numbered using the same numbers which are displayed in the graphics window. If two bodies have a common boundary, they also share the boundary elements which form this common boundary.

These common boundaries are grouped as body-pairs in the body listbox. In practice this means that, when selecting a single body from the body listbox, you will see all the outer boundary elements for that body. When selecting a body-pair, you will see all the (common) inner boundary elements for those two bodies.

Boundary elements are displayed by first selecting a body. All the boundary elements of the body (or a body-pair) are now displayed in the element listbox. If you now attach a boundary condition to the body, it is automatically applied to all those boundary elments which do not yet have a boundary condition. So, there is no need to explicitely select all elements, if you want to apply a condition to all boundaries of a body.

If you want to select a specific element, you can do that by clicking with the left mouse button in the element listbox. If you want to select multiple elements at the same time, you have to press the control key while clicking with the mouse. By pressing the shift key and the left mouse button, you can select by dragging with the mouse a continuous group of listbox rows.

Similarily as in the initial condition panel, those boundary conditions which match the equation type of a body are marked with an asterix in the beginning of the condition rows. The equation type of a body-pair is formed by combining the equation types of the both bodies.

There is one additonal button in the boundary condition panel:

- `Attach mode`: this button has two modes, KEEP_NEW or KEEP_OLD. You can change the mode by pressing the button. Mode is used when deciding how a conflict between different boundary conditions should be solved. This conflict could arise, for example, if two edges have a common vertex and the boundary condition contains a Dirichlet type constraint (like a temperature or velocity condition). In 3D these conflicts could arise byn common edges for surface elements. Selecting KEEP_NEW means that the last condition applied will win in conflicts. Selecting KEEP_OLD protects always the condition which was applied first.

2.9 Mesh menu

The only command in this menu is `Mesh parameters`. It opens a panel where you can set the mesh parameter for the whole model, selected bodies or body elements. This panel is shown in Figure 2.17

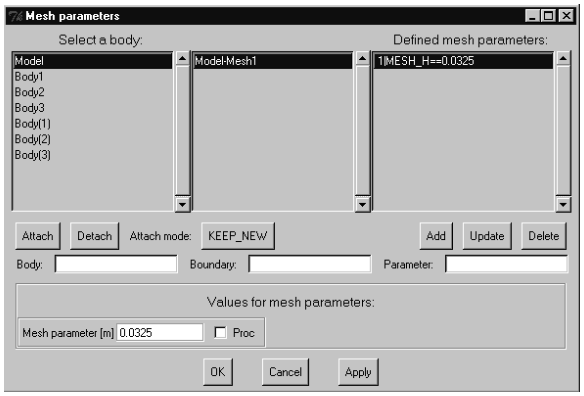

Figure 2.17: The Mesh parameters panel.

In this panel there is currently only one entry field, the `Mesh parameter`. This parameter controls how fine or coarse the generated mesh will be. It is related to the size of the smallest elements that will be created. For this reason, the dimensions of the model should be taken into account, when the value of this parameter is set.

In the object listbox we have three different type of objects.

Model object concers the whole geometry. If the mesh parameter is given only for the model, all parts in the model are handled equally when the mesh is generated.

Body objects are displayed in the BodyN format. You can give a general model level parameter and still specify different values for selected bodies.

Body elements can be selected from objects displayed in the Body(N) format. This is also the smallest level of geometry, where the mesh generating can be controlled.
NOTE: Currently only model level mesh parameter is supported. So, you should always enter a parameter at the model level. Other parameters do not currently have any effect on the mesh generating!

2.10 Solver menu

Figure 2.18: The Solver menu.

Solver parameters

In this panel you set the solver parameters for each equation system to be solved.

You should first select an equation system using the radiobuttons in the upper part of the panel.

The solver panel is shown in Figure 2.19 and settings for the Navier-Stokes equation are activated.

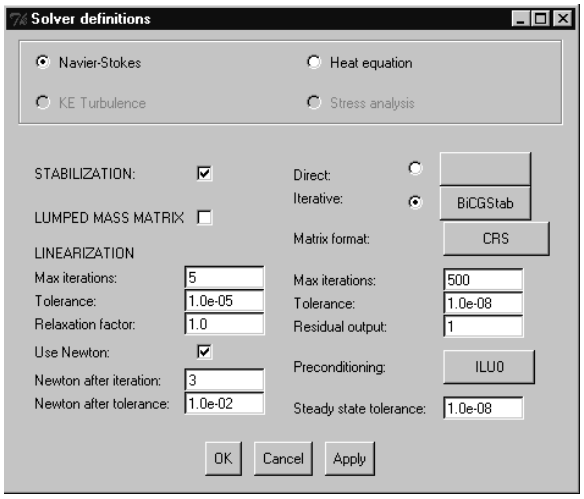

Figure 2.19: The Solver parameters panel.

All fields in the panel have default values.

Entry fields in the left hand side of the panel define general settings for the solver. Fields are the following:

- `STABILIZATION`: when this button is checked, solver uses stabilization. It is a recommended option for the Navier-Stokes equation.

- `LUMPED MASS MATRIX`: use this check button if you want that matrix which corresponds the mass matrix is lumped in the equation
- `Max iterations`: linearization is based on subiterations and this number controls the maximum number of iterations.
- `Tolerance`: if the error norm for the linearization is smaller than this number, linearization iteration is stopped.
- `Relaxation factor`: default value is 1.0, but a somewhat larger value could speed up linearization in some cases. On the other hand, sometimes a value less than unity is needed to get any solution.
- `Use Newton`: when this button is checked Newton's linearization method is used. It is faster than the default method (Picard), but is useful only when we are near enough the actual solution. The following fields control when Newton's method is started.
- `Newton after iteration, Newton after tolerance`: Newton's method is not started until at least one of these criteria is fullfilled.

The entry fields on the right hand side of the panel control how the linear system, which is formed by discretizing the original equations, is solved:

- `Direct`: this radiobutton selects direct solving method. Currently the only available direct method is BANDED solver. For relatively small problems direct solver is normally faster than an iterative solver, but for any larger problem it is propably better to use an iterative solver.
- `Iterative`: this button selects iterative solving method. You can select the actual method by double clicking in the listbox which opens when you press the button to the right of the radiobutton. Available options are:
  - `BiCGStab`: BiConjugate Gradient Stabilized method
  - `TFQMR`: Transpose Free Quasi-Minimal Residual method
  - `CGS`: Conjugate Gradient Squared method
  - `CG`: Conjugate Gradient method
  - `GMRES`: Generalized Minimal Residual method
- `Matrix format`: matrix storage format
  - `CRS`: compressed row stroage
  - `Band`: banded matrix
  - `Symmetric Band`: symmetric banded matrix
- `Max iterations`: maximum number for iterations for the iterative solver
- `Tolerance`: if the error norm is smaller than this number, iteration is stopped
- `Residual output`: given as integer N; linear system residual is output for each Nth iteration (value zero disables the output)

- `Preconditioning`: using preconditioning normally speeds up the iteration, but with the cost of some overhead. Options here are:
  - `None`: no preconditioning
  - `Diagonal`: Diagonal preconditioning matrix. Fast to construct, but not very effective
  - `ILU`: Different level of Incomplete LU decompositions. ILU0 is the recommended choice
- `Steady state tolerance`: if the error norm is smaller than this number, the steady state iteration is stopped. If we have multiple, connected equation systems, you can use this parameter to select which of the systems stops the iterations for a steady state problem.

Processor settings

This command opens a panel where you can enter the number of processors that are available for solving the problem. This parameter has meaning only in a parallel processing environment. The default value for the parameter is one. This panel is shown in Figure 2.20

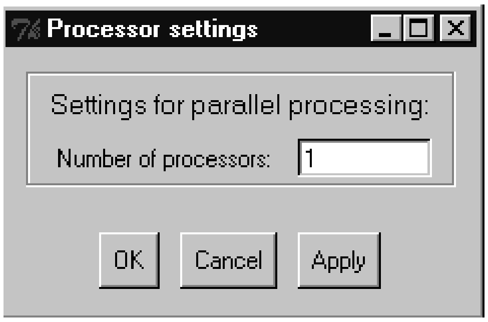

Figure 2.20: The Processor settings panel.

2.11 Run menu

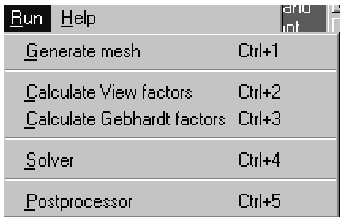

Figure 2.21: The Run menu.

You can start other ELMER modules from this menu. Before a module can be started all the necessary definitions and settings must have been done. If this not the case, a message box is displayed. If the missing data is essential for the module, you cannot continue with the process. Otherwise you can continue, but with the risk that some data is not up-to-date.

Commands in the menu are:

- `Generate mesh`: starts the ELMER mesh generator. NOTE: if the `Display mesh` button is selected, the mesh is automatically read from the database and displayed!
- `Calculate View factors`: starts ELMER Viewfactors program. It stores the data in the `View factors file` defined in the `Problem/Datafiles` panel. NOTE: this can be a lengthy process for large meshes!
- `Calculate Gebhardt factors`: starts the ELMER GeghardtFactors program. It stores the data in the `Gebhardt factors file` defined in the `Problem/Datafiles` panel. NOTE: this can also be a lengthy process!
- `Solver`: starts the ELMER solver using the solver input file defined in the `Problem/Datafiles` panel
- `Postprocessor`: starts the ELMER postprocessor using the postprocessor file defined in the `Problem/Datafiles` panel as the input file

Chapter 3

Elmer Mesh Generation

FEM-simulation always requires that the computational domain is discretized. Discretization, i.e., construction of a valid mesh covering the computational domain, is a laborious and error prone task if done by hand. Therefore, this process is usually automatized. Moreover, we normally want to control the characteristics of the mesh which have an impact on the accuracy of the simulation. Perhaps the most important of these characteristics is the mesh parameter which indicates the scale of resolution of the solution as the size of the elements.

For any mesh generation to be successful, the user has to define at least:

- the geometry of the problem, including the features relevant to the simulation such as material boundaries inside bodies, and
- the mesh characteristics.

In theory, the mesh characteristics could be derived from the geometric features but this approach is not often sensible for flow simulations in particular.

3.1 Capabilities of the Mesh Generator

The current version of ElmerMesh, the ELMER mesh generator, is an automatic 2D-mesh generator. The valid geometries are those handled by the CAD-interface. Note that curved boundaries are linearised in the conversion from CAD to mesh geometry. The mesh size is given in the form of the mesh parameter, h. No other mesh characteristics can be defined.

The generated meshes are triangular, that is, the elements of the mesh are all triangles. Once generated, the elements can be either linear or quadratic. Also the dual of the mesh is generated. The dual is used in the partitioning phase if parallel solvers are to used in the simulation. In Figure 3.1 a simple mesh with four elements is shown with its dual (dashed lines). Note that the dual is not a triangulation. However, by partitioning the dual graph we partition the elements and not the nodes which is appropriate for FEM.

3.2 Mesh Generation Strategy

Mesh generators are based on either of the two approaches: the advancing front method and the Delaunay method. Both approaches have their pros and cons

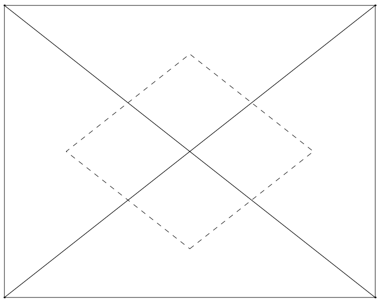

Figure 3.1: A simple mesh and its dual.

and in general it is not possible to explicitly show one superior over the other. ElmerMesh is a Delaunay type mesh generator and more specifically, a Voronoi-Segment type mesh generator.

The Delaunay property of a triangulation of points is that none of the points of the triangulation are inside a circle defined by the three points forming a triangle in the triangulation. The triangulation process is incremental in the sense that by definition we start from a valid triangulation and modify it whenever new points are added. So, the actual mesh generation consists of a sequence of two steps: point generation and triangulation, which terminates only when the desired mesh has been obtained. A stopping criterion can be for instance that all triangles are of the size defined by the mesh parameter. Figure 3.2 shows the changes induced by the introduction of two points. Note that the first two triangles are not preserved after two insertions but the second insertion does not destroy all existing triangles, the two at the lower left corner remain the same.

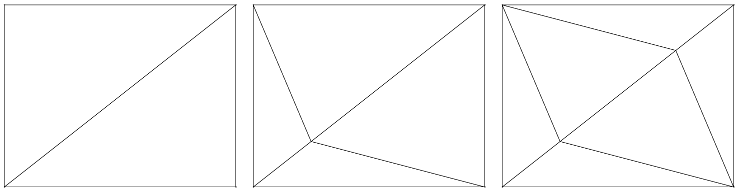

Figure 3.2: Adding points to a Delaunay triangulation.

3.3 Practical Issues

Since the Delaunay property applies only to the set of points, we say it is a connection property, the concept of boundaries is alien to the actual triangulation process. First we discretize the boundaries and connect them before the interior of the domain is discretized.

The tracking of the boundary integrity is the foremost problem in Delaunay type mesh generators. If the geometry has some features of the same order as the mesh parameter, it is possible that the boundaries are not recovered, e.g., the mesh covers the whole domain, but the internal material boundaries are overlapped by some triangles. If case this happens, ElmerMesh tries to capture the boundaries automatically, but this "recovery"-process may lead to elements which have very much different characteristics than those requested by the user.

3.4 When the System Takes Over the Control

There are two cases when the generator changes the mesh without user interaction. If the boundaries are not recovered by the standard algorithm as explained above the generator adds points to the missing boundary segments and thus effectively changes the local mesh parameter. This may lead to unexpected features in the mesh. This is illustrated in Figure 3.3, where the computational domain is of the shape of letter j. Simply connecting the geometry defining nodes is not enough, the right-hand side bar is not captured in the mesh so an additional point is inserted in the middle of the missing edge.

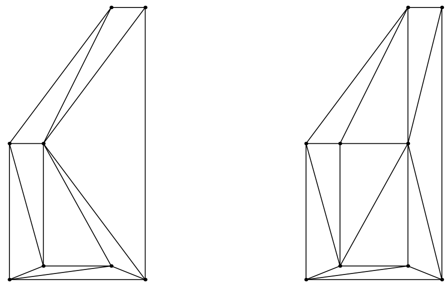

Figure 3.3: Automatic boundary recovery.

For free-boundary problems it is quite natural that the user is not required to change the mesh manually after every iteration. Those boundary nodes which are moved in the simulation are moved in the direction of the vector field and the mesh is smoothened using spring-equilibrium techniques. This approach is often referred to as the r-method.

3.5 Guidelines

The meshes generated by ElmerMesh tend to retain a relatively rigid structure away from the boundaries. This structure is then inevitably fitted to the boundaries. In 2D-geometries, this is the only region where elongated elements with high aspect ratios may result. No regularization is done, however, and the non-desired elements are used in the simulation.

The algorithm always starts from the first boundary segment of the description of the body. If mesh alignment is necessary it can be induced by changing the geometry description appropriately.

It is often advisable to generate meshes with different mesh sizes also purely from the mesh generation point of view. Paradoxically, the most difficult problem is to create small meshes due to boundary integrity issues. If the automatic boundary recovery is activated, as a rule of thumb we can say that the mesh parameter should be smaller.

Chapter 4

Elmer Graph Partitioning

ElmerSplit is the graph partitioning module of ELMER. The purpose of ElmerSplit is to distribute the mesh evenly between multiple CPUs in a parallel environment to help optimize performance of the solver.

When only a single CPU is used, there is no need for partitioning mesh. The mesh generator has already created one "partition" which is required. This means that only one solver process can be used.

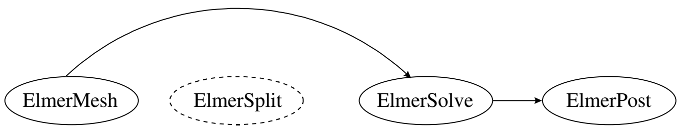

Figure 4.1: Single CPU configuration

If multiple CPUs are available, through-put time can be reduced by using multiple solver processes, each running in its own CPU as shown in figure 4.2. To work efficiently each solver -process must be able to use its local data as much as possible. ElmerSplit makes this possible by splitting the original mesh into smaller pieces so that connections between different pieces are kept at minimum.

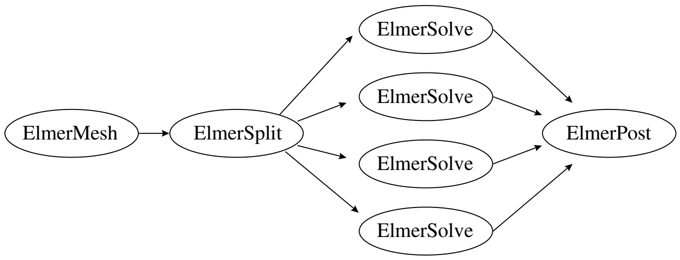

Figure 4.2: Configuration with multiple CPUs

4.1 ElmerSplit Capabilities

4.1.1 How Partitioning Works

Partitioning algorithms used in ElmerSplit are based on *Metis* -library by *George Karypis* and *Vipin Kumar* from University of Minnesota. Metis implements a collection of multilevel graph partitioning algorithms.

The Idea of multilevel graph partitioning is to approximate the original graph by a sequence of smaller graphs. The smallest graph is then partitioned using some suitable method and results are projected back to the original graph. The advantage of multilevel approach is that the graph coarsening can be done in time proportional to the number of edges, while the complexity of partitioning increases exponentially with the number of nodes.

ElmerSplit uses so-called dual-graph, which is based on connections between elements in the mesh. Each vertex of the graph represents one element and edges are connections between elements. Dual-graph is feed to the partitioning algorithm and result tells for each element in which partition it belongs. Since the original graph has more degrees of freedom, some refining is usully made at this point to further decrase the number of connections between partitions.

4.1.2 Mesh coarsening

There are basicly two kinds of graph reduction algorithms. Some algorithms, like *Random Matching* (RM), just match random vertices together, while others, like *Heavy Edge Matching* (HEM) and *Heavy Clique Matching* (HCM), use the connectivity information to find a groups of tightly connected vertices.

Random algorithms work quite well when degrees of vertices are close to the average degree of the graph. Graphs from finite element applications (FEM) are typically in this category. However, if a graph has tightly connected components it is usually better to keep those components together and use something like HEM or HCM. Splitting tightly connected components into different partitions typically leads to unnecessary increase in the connection cost of partitioning.

Since reduction algorithms have roughly similar complexity, it is usually better to use some more advanced algorithm like HEM or HCM than try to save a little time using RM. Carelessly done reduction can harm partitioning algorithm and lead to significantly higher cost partitions.

Random Matching

Random Matching (RM) visits vertices in random order. If vertex has not been matched yet, we randomly select one of its unmatched neighbors and mark both the vertices and the edge between them as matched. If vertex has no unmatched neighbors, it remains unmatched. This continues until no more vertices can be matched. After that, matched vertices are joined together and are marked as unmatched for the next reduction step.

Heavy Edge Matching

Heavy Edge Matching (HEM) visits vertices in random order, same way as RM did. However, when selecting a neighbor to match with, it chooses the one that is connected with the heaviest edge. Naturally only such neighbors that are not already matched are considered.

The idea of HEM is to minimize the edge weights of the reduced graph. Smaller edge weight typically leads to smaller connection cost when reduced graph is partitioned. This algorithm doesn't guarantee that the edge weight of the reduced graph is minimized, but experience has shown that it works very well.

Modified Heavy Edge Matching

Modified Heavy Edge Matching (MHEM) tries to minimize the average degree of the graph. Again vertices are visited in random order and matched with the neighbor that has the heaviest connection. If there are more than one vertex to choose from, the vertex that has most connections from it's neighbors to a matching vertex, is chosen. Analysis of the multilevel bisection algorithm in shows that a good edge-cut of a coarser graph is closer to a good edge-cut of the original graph if the average degree of the coarser graph is small and/or the average weight of the edges in the coarser graph is small.

Light Edge Matching

Light Edge Matching (LEM) is like HEM, but instead of matching heavily connected vertices it matches neighbors that has the lightest edge between them. Reduced graphs produced by LEM have typically much higher average degree than the original graphs. This kind of graphs are easier to handle for some partitioning algorithms like Kernighan-Lin. The choice between HEM and LEM depends on what kind of partitioning algorithm is selected for the reduced graph.

Heavy Clique Matching

Heavy Clique Matching (HCM) tries to find subgraphs that are fully or almost fully connected. The idea is very similar to the HEM but instead of just matching vertices with the heaviest edge between them, HCM joins vertices that have the highest edge density.

For a pair of vertices (u, v) edge density is defined as follows :

$$EdgeDensity = \frac{2(CE(u) + CE(v) + EW(u,v))}{(VW(u) + VW(v))(VW(u) + VW(v) - 1)}$$

where $VW(x)$ is the weight of vertex x, $EW(x,y)$ is the weight of edge between vertices x and y, and $CE(x)$ is the total weight of edges already collapsed into a vertex x. Vertices that are not connected in any way have edge density of 0 and vertices that form a clique have edge density of 1.

4.1.3 Creating partitions

After the graph has been coarsened down to desired size, it's time to create partitions. Since the graph partitioning problem is NP-complete, an exhaustive search for finding the best partitioning is usually impossible because of a vast number of possible solutions.

Metis offers four different partitioning algorithms that are able to find good partitionings in a reasonable time. In paractice results are quite good, yet there is no quarantee that results will be optimal,

Graph Growing

Graph Growing (GGP) algorithm starts from a random vertex and grows a region around it in breath-first fashion, until desired number of the vertices or vertex weight has been included. Partitions generated by graph growing are, by definition, always connected. However, the quality of partitions depends totally on a chose of the initial vertex.

Greedy Graph Growing

Greedy Graph Growing (GGGP) is a modification of GGP. Instead of growing a region in a strict breath-first fashion we can compute a gain value for each vertex on a region boundary. Gain value tells how the costs of the partitions will change if a vertex is added to a growing region. Vertices are then sorted by their gains and the vertex that has the biggest decrease (or smallest increase) of partition cost is inserted first. Then the gain values of adjacent vertices are updated and new vertices are joined to region boundary.

GGGP has the same problem as basic GGP. The quality of partitions depends on the choice of initial vertex. However this dependency is not as strong as in GGP. Usually GGGP gives better partitions with less work than GGP.

Graph Growing with Kernighan-Lin

Kernighan-Lin (KL) algorithm starts with initial partitioning of the graph. In each iteration it tries to find a subset of vertices from each partition of the graph such that swapping them leads to better partitions. If such subset exists, swap is performed and new iteration is performed for changed partitions. Algorithm stops when no such subsets can be found. In this scheme Graph Growing is first used to create initial partitions which are further improved by running KL on partition boundaries.

Kernighan-Lin algorithm has been effective in finding locally optimal partitionings when it starts with a fairly good initial partitions and when the average degree of of the graph is large.

Spectral Bisectioning

Spectral Bisectioning (EIG) is based on the properties of second eigenvector of Laplacian matrix associated with the partitioned graph. Unlike other algorithms, Spectral Bisectioning invoves a lot of floating point arithmetics and for that reason can be much slower than other algorithms. Typically partition quality is similar other algorithms or little better.

4.1.4 Refining partitions

After the reduced graph is partitioned the results have to be projected back to the original graph. This can be done by simply assigning all vertices to the same partition as their parent in the reduced graph. However, since the original graph is much finer and has many more degrees of freedom than the reduced one, these projected results can usually still be improved by swapping some vertices from one partition to another.

Metis offers six refinement algorithms, three basic algorithms and so-called boundary versions of each. Partition refining can also be turned off, if desired.

Kernighan-Lin Refinement

Kernighan-Lin refinement (KLR) simply runs KL partitioning algorithm with projected partitions. Since those partitions are already quite good, algorithm converges fast, typically within three to five iterations.

Greedy Refinement

Experiments show that the largest gain is obtained during the first iteration step. *Greedy refinement* (GR) runs only a single iteration of KL algorithm. Iteration is stopped immediately when no more swaps with positive gain are found. This reduces the complexity of refinement phase. Unfortunately the number of swapped vertices and total running time does not change in asymptotic terms because a lot of work has to be done while building appropriate data structures before iteration.

Combination of GR and KLR

The Idea of combining *Greedy* and *Kernighan-Lin* refinement algorithms is to use KL as long as the graph is small, and switch to GR when the graph is large. The motivation of this scheme is that single vertex swaps in the coarser graph can lead to larger improvement in partition quality than in the finer graph. By using KL at coarser graphs better refinement is achieved, and because these graphs are very small compared to the orginal graph, KL algorithm doesn't require a lot of computing time.

Boundary Algorithms

Almost all of the swaps in refinement phase are done between vertices on a partition boundary. *Boundary refinement* uses this information to skip unnecessary work. The idea is to focus on those vertices that are on a partition boundary and forget all other vertices. After every iteration algorithm has to check for new vertices that might have got onto a boundary due to swapping of vertices that were already on the partition boundary. Boundary-idea can applied to all basic refining algorithms.

4.2 Using the ElmerSplit

4.2.1 Command-line syntax

By default ElmerSplit uses partitioning strategy which suits well on most cases. However if needed there are also various parameters which can be tuned for optimal performance and partition quality.

```
elmersplit --model-name=NAME --model-directory=DIR
           --nproc=PARTITIONS [--coarsento=NODES]
           [--ptype=PARTITIONING_ALGORITHM]
           [--mtype=MATCHING_ALGORITHM]
           [--rtype=REFINING_ALGORITHM]
           [--verbose] [--help]
```

Figure 4.3: Elmersplit command-line syntax

–model-name Compulsory parameter that defines the name of the model.

–model-directory Compulsory parameter that defines the location of the model database.

–nproc Compulsory parameter that defines the number of partitions to be generated. Value of this parameter must be > 1.

–coarsento Optional parameter that defines the number of elements in coarsened mesh. This must be greater than *–nproc*. If not defined, the default value of 100 elements is used.

–ptype Optional parameter that selects the partitioning algorithm.

1	GGP	Graph growing partitioning
2	GGGP	Greedy graph growing partitioning
3	EIG	Spectral bisectioning
4	GGPKL	Graph growing with boundary Kerninghan-Lin

If not defined, the default algorithm (*GGPKL*) is used.

–mtype Optional parameter that selects the matching algorithm.

1	RM	Random
2	HEM	Heavy-edge
3	LEM	Light-edge
4	HCM	Heavy-clique
5	HEM*	Modified Heavy-edge
11	SRM	Sorted Random
21	SHEM	Sorted Heavy-edge
51	SHEM*	Sorted Modified Heavy-edge

If not defined, the default algorithm (*HCM*) is used.

–rtype Optional parameter that selects the refining algorithm.

1	GR	Greedy
2	KLR	Kernighan-LIN
3	GKLR	Combination of GR and KLR
11	BGR	Boundary Greedy
12	BKLR	Boundary Kernighan-Lin
13	BGKLR	Combination of BGR and BKLR
20	NR	No refinement

If not defined, the default algorithm (*BKLR*) is used.

–verbose Selects verbose output.

–help Prints short on-line help message.

4.2.2 General guidelines

To select the best partitioning scheme for given problem you must first decide whether are you trying to optimize partition quality or computing time used in generating partitions. Ie. if you are going to use the same mesh over and over again you should propably focus on partition quality and use more computing time on partitioning. On the other hand, if the structure of your mesh will change frequently it is not a worthwhile to use alot of time to get marginal improment in quality.

In most cases default values will suit just fine. However there are some points that should be noticed when choosing parameters :

- Matching algorithm. Out of available matching algorithms *HEM*, *HCM* and *HCM** perform consistently better than *RM* or *LEM*. *HEM* suits especially well for 3D element meshes. Also when *LEM* is used for coarsening, both the time spent for coarsening and refining is fairly high.

- Size of coarsened mesh. To preserve the original structure of mesh it is not a good idea to coarsen mesh too much. Ie. If 1.000.000 elements are coarsened down to 10 nodes, partitioning phase will be very fast but alot of extra work have to made during coarsening and refining. Also the structure of the original mesh could be lost which leads to badly formed partitions. Size of coarsened mesh should always be significantly larger than number of created partitions to allow partitioning algorithm do some real optimization in addition to coarsening. Also if mesh has alot of small details, a larger number of elements should be left in coarsened mesh.

- Partitioning algorithm. *GGP* is the fastest algorithm but it tends to create worse partitions than other algorithms. *EIG* tends to give worse partitions than *GGGP* and *GGPKL* and require more time. Also if the coarsened mesh is quite large, *EIG* can require much more computing time than others.

- Refining algorithm. Generally the boundary versions of algorithms outperform their non-boundary counter-parts in both speed and partition quality. *BGR* is the fastest algorithm but quality of partitions is not very good. If the number of partitions is large (ie. $>$ 64 partitions) *BGKLR* is a good choise. With a smaller number of partitions *BKLR* tends to perform little better than *BGKLR*.

Chapter 5

Overview of The Solver

The solver is a program that uses user defined model data, hopefully describing a meaningful mathematical or physical problem, to solve field variables (velocity, displacement, temperature, etc.) at mesh nodal points, or more accurately the coefficients of the piecewise, interpolating polynomials used to approximate the field variables. The user provided data includes initial and boundary conditions, material parameters, equation definitions, and the mesh. The equations and boundary conditions the solver currently can handle have been described earlier in Chapter 1.

However, the solver is readily extensible as the source code is provided in addition to the prelinked executable program. One can also extend the solver by providing dynamically linkable user defined functions instead of directly modifying the base code.

The building blocks of the solver are, in principle, not dependent on any specific equation or set of equations, but contain a quite general finite element package.

The solver is coded in Fortran 90, and to make full use of the features in more challenging problems that require user routines or modification of the code, some understanding of the peculiarities of this programming language will be needed.

5.1 Solver Capabilities

As was told in Chapter 1, the solver is capable of solving steady state or time dependent, coupled flow and heat equations. Also a thermal stress analysis type is provided. The solver is capable of handling these equations in several different coordinate systems: cartesian 2D and 3D coordinates, polar 2D and 3D coordinates, cylindrical coordinate system, including axial symmetry of geometry and flow field or just that of the geometry.

A coordinate system is given by writing a few routines returning values of the coordinate system metric, Christoffel symbols, and derivates of Christoffel symbols at given coordinate point. This makes it really easy to include new coordinate systems as rest of the code, specifically the code for the differential equations, is independent of the coordinate system. At the moment this requires modification of the base code however (there are no user routines for this purpose).

Heat is transferred inside liquid or gaseous material by conduction and convection, and inside solid material by conduction. In the ELMER solver, heat transfer mechanisms may also include radiation, either idealized radiation (a physical body loses or gains energy on a boundary depending on the temperature difference of the body on the boundary and 'external' temperature) or diffuse gray radiation, modelling exchange of energy between various surfaces of a geometrical model. When using the diffuse gray radiation model, geometrical information (called view factors) of how much different parts of the model 'see' each other needs to be known. In 3D this is a though question, potentially needing a lot of computer resources. A view factor determination program based on ray tracing is provided separately from the solver. For cylindrically symmetric geometries there is a different program, originally developed by Katajamäki. Refer to appendix C for more information about the radiation modelling.

Nonlinear Navier-Stokes equations and the heat equation with the nonlinear radiation term are solved with Newton's method, possibly with a few Picard iterations to start with. Nonlinarities resulting from nonlinear materials are solved with Picard iteration.

5.2 Structure of a Simulation Run

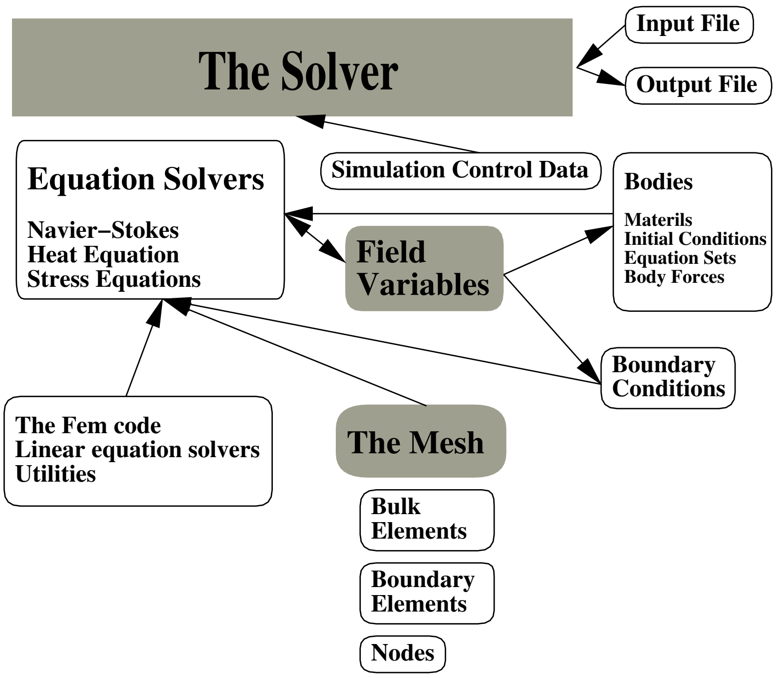

Figure 5.1: The solver picture of a simulation.

The model input data for the solver consists of the following elements:

- bodies,

- body forces,
- boundaries,
- boundary conditions,
- equations,
- initial conditions,
- materials,
- solvers,
- case control data,
- mesh description including boundaries and mapping of boundary conditions to boundaries.

A body is a geometrical entity which has boundaries, governing equations, initial coditions, body (mass) forces and material parameters. Note that a physical body may consist of several simulation bodies, if boundary conditions between different parts of the physical body are needed. Equations for a given body may be any set of the equations implemented in the solver. Initial conditions contain starting values of the field variables. Body material contains material parameter values or pointers to data from which to compute them, possibly with using user written functions. Body force definitions are means to give additional force terms for the equations (for example gravity for the stress analysis or a magnetic field generated force for the flow momentum equations). When the equation solver is computing local matricies for any given element it uses the elements knowledge of the body it belongs to get hold of the values of the needed parameters.

Boundary conditions are organized as a set of values (for all of the equations) for one boundary or a set of boundaries of model bodies. The mapping of boundary conditions to boundaries is also given in the solver input file. For more information on this issue and creation of the mesh in general refer to Chapters 3 and 2.

The solver field of the input file contains variables for controlling the nonlinear equation solver options and the linear equation solver options. Each solver is assigned an equation type for which this solver is to be used.

The definitions of all control parameters may be set from the CAD Interface program or from the solver input file. The format of the whole solver input file is described in appendix H.

5.3 Structure of the Equation Solvers

The current solver contains four specific equation solvers: the Navier-Stokes equations, the heat equation, the magnetic induction equation, and the elastic stress equations.

The Navier-Stokes equations may be coupled to the heat equation by the temperature variations of the density in the force term (the so called Bussinesq approximation), temperature variations of the surface tension coefficient and the temperature dependancy of viscosity when using the solidification phase change model, where the variable viscosity method is used. Also one may specify special body forces or material parameters, which may depend on field variables, for example temperature. Thermal stress analysis also uses the solved temperature field.

The heat equation is coupled to the Navier-Stokes equations by the convection term. Both the Navier-Stokes equations and the heat equatiation are both coupled to the magnetic induction equation, and the induction equation is coupled to the Navier-Stokes equation.

The element mesh describing the computational domain is divided into bulk elements and boundary elements, which are stored inside the program basically the same way. They are also handled identically by the equation solvers, except that the equations for the boundary elements differ from the bulk element equations.

Material and solver control parameters are not statically defined, but the solver routines ask for the values of the parameters by name when needed. The values are provided by utility routines which also check, if the values are defined to be constant, varying piecewise linearily with some field variable or provided by user functions. Within one timestep or coupled system iteration the equation solvers iterate the nonlinearities within equations until a user specified convergence criterion is met or iteration count is exceeded.

5.4 Equation Discretization at the Element Level

The FEM part provides ready made discretizations at the element level (called local matrix generation) of stabilized Navier-Stokes equations, a general scalar diffusion-convection equation and elastic stress equations and their boundary conditions. All the material parameters (for example heat conductivity, density, viscosity) may be provided to these routines at element nodes, so that they may vary in space (depend on temperature, for example). Also some of the parameters may be given as tensors (for example heat conductivity), instead of scalar values. This is required if the code is to handle anisotropic materials. All the equations are written in tensor form anyway, so the usage of tensor valued material parameters is no problem.

5.5 The Structure of the Base FEM Code

The base FEM code is quite general. It can handle line, surface and volume elements in one, two and three dimensions, and in several different coordinate systems, as was mentioned earlier. Elements may also be curved. All the elements are considered to be isoparametric, i.e. the same basis functions are used to approximate both the coordinates and the field variables. The FEM code provides the following services:

- Gaussian integration points for different element types and desired accuracy.
- Value of a quantity given at element nodes inside the element.
- First and second derivates with respect to element coordinate system.

- First and second derivates with respect to global coordinate system.
- Metric tensor of the global coordinate system and square root of its determinant.
- Christoffel symbols of the global coordinate system and space derivates thereof.
- Metric tensor of the element coordinate system and determinant of the element coordinate systems Jacobian, or more generally square root of the determinant of the metric tensor of the element coordinate system.
- Length, area or volume of a line, surface or volume element respectively.
- Measure of 'size' of an element.
- Normal vector at given point inside of an surface or line element
- Routines to determine whether a given point in space is inside an element.

You may use several different element topologies and basis functions for discretization of the equations. The solver recognizes linear and quadratic basis functions in line, triangle, quadrilateral, tetrahedron and heaxhedron topologies (Figures L.1-L.4). The available element types are described in more detail in Appendix L. What element types to use in a specific simulation depends on both the application and the mesh generator. The ELMER mesh generator, for example, doesn't include quadrilateral elements at all. Choosing the basis function degree depends on the application, and to some degree also on personal taste. Using the quadratic basis functions instead of the linear ones will increase both the assembly time of the local matrices and the time spent in solving the linear systems. On the other hand, the results of the computations, at least in principle, are more accurate or you may use less elements to achieve the same accuracy.

5.6 Utilities

Besides the equation solvers and the FEM code, the solver code includes various additional utilities:

- Bandwidth optimization. This utility is obviously needed for the band matrix solver, but the bandwidth optimization will also benefit the iterative solver: the reduction of bandwidth will also reduce the potential fill-in when preconditioning with the incomplete LU decomposition (which is almost a must with our linear systems), this will in turn have an effect on the quality of the preconditioner.
- LU decomposition and matrix inverse for full matrix.
- Utility routines to handle sparse matrices in Compressed Row Storage (CRS) format. These routines include creating the matrix, initializing the nonzero structure of the matrix, zeroing of the matrix, zeroing a row of the matrix, setting value of an element, adding value to an element, adding element local matrix to the global matrix, matrix vector multiply, diagonal

and ILU precoditioner, and a LU solver. Also a sparse approximate inverse preconditioner is included, but experience with this preconditioner is still limited and it can't be used without modifying the code.

- Storage of material parameters, boundary and initial condition parameters, field variables and solver control variables in list structures and manipulation of these lists. This includes adding name/value pairs to the lists, searching for values in the lists. If a parameter is defined to vary with some field variable, these routines will also do the interpolation, and if the parameter is given as a user routine, they will call that routine.
- Loading and calling of the dynamically linkable user routines.
- Input file scanner, output and restart file writer and reader and post processing file writer.
- Sort and search functions.
- A ray tracer module for the view factor computations. The ray tracer includes hierachical volume subdivision for speeding up the intersection calculations. Besides the usual elements, the ray tracer is able to check for intersections with bounding boxes, spheres, polygons, general rotational quadrics including cylinders, cones, etc.
- Some utilities for manipulation of bilinear, biquadratic and bicubic Bezier surfaces. This module is also for the view factor computations.

5.7 Using the Solver

The solver must be run separately after the model has been defined with the aid of ELMER CAD Interface or by other means. How to execute the solver from the CAD Interface has been described in Chapter 2.

When modelling radiation there is three steps to take after the model input data and mesh have been defined:

- compute view factors,
- compute Gebhardt factors,
- execute the solver.

All these tasks may be done from the CAD Interface program or from the command line.

The CAD Interface holds a menu entry to execute the solver. Before using the solver the model definitions must be saved to a disk file. This file can also be edited by hand and extended to hold time, space, temperature, etc., varying material parameters, initial conditions, boundary conditions, and solver control variables. The varying parameters may be specified as piecewise linear splines by giving data points or by using user defined functions to compute the values at any given time. The user functions may also access all the model data, the current solved field variables and use the utility routines from which the solver is built. The solver input file format is described in appendix H, where also a complete case input file is explained in detail.

Chapter 6

Elmer Post Processing

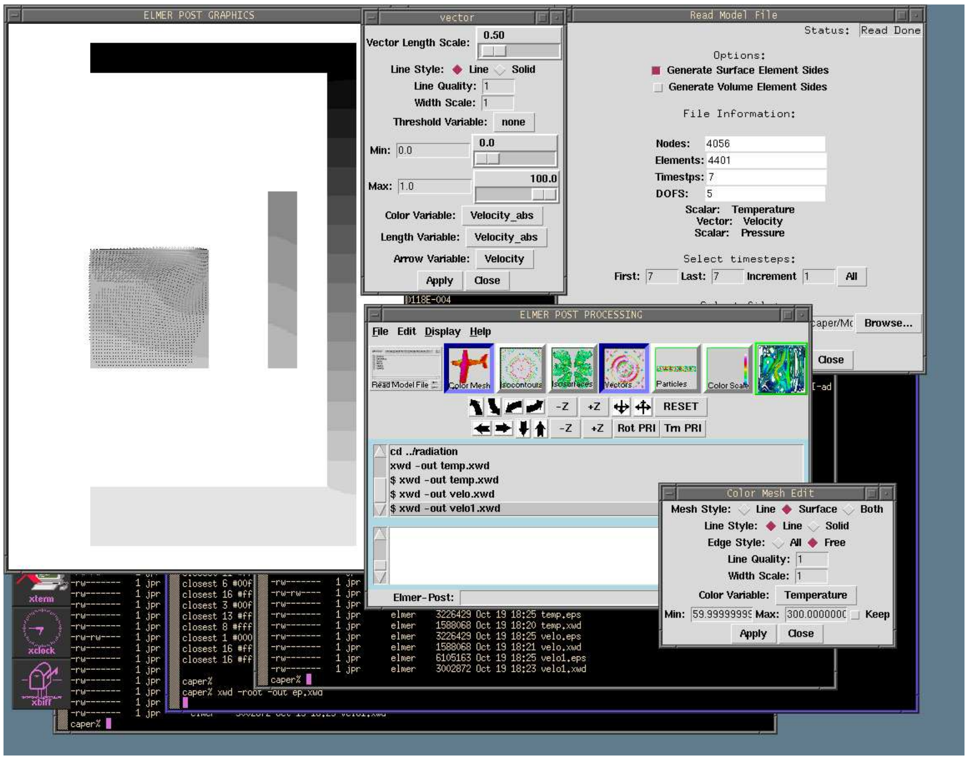

Figure 6.1: Elmerpost in use.

The post processing part of ELMER is called Elmerpost. Elmerpost is a program for displaying and analyzing results from time dependent finite element method simulations. Elmerpost can handle both 2D- and 3D- element models. Elmerpost is only loosely connected to the rest of the ELMER package, and might be used in a totally different context.

The basic building blocks of the program are: TCL/TK for user interface, OpenGL graphics library and C language. A software package called Mesa is a free implementation of OpenGL. These are all tools that are portable to various platforms including all Unix versions that we have tried, Windows NT, and potentially also Macintosh computers.

In the following a brief description of the software is given. For more specific information about this software refer to the Elmerpost documentation.

6.1 Elmerpost Capabilities

The program provides several options for displaying vector and scalar field variables:

- contour line display
- color coded meshes
- contour surfaces
- vector display
- sphere display
- particle and stream line display
- selecting arbitrary cuts from the model

In addition to being able to display the data there is more to Elmerpost. Several additional features include

- A matrix language MATC, which can be used to manipulate data with
  - Basic matrix and vector operations
  - Dynamically sizeable variables
  - A programming language with functions, flow control, loops, I/O, etc.
  - Derivative operators for the finite element model: gradient (∇), divergence ($\nabla \cdot$), and curl ($\nabla \times$)
- Both a graphical and a command line user interface
- The TCL/TK environment can be used to full extent, providing programmable user interface to extend the program to specific needs
- Colormap editor, surface material editor, bacground color editor
- Unlimited number of user controllable views of the model
- A versatile clipping ability
- Element grouping
- Online help in HTML format (a simple basic HTML browser is also included)

Manipulation of the data is possible through a matrix language called MATC included in the program. One might, for example, want to compute the stress tensor from a displacement field. Field variables from the simulation are automatically seen as MATC variables, after the FEM model and simulation data are read in to Elmerpost.

Having two programming languages, TCL and MATC, within a single application might seem a little complicated, but at the same time it adds a lot to the functionality of the program. The scope of the two are also quite different, TCL being a command/script language, and MATC on the other hand being a matrix programming language. Thus they complement each other.

Elmerpost has also a built-in ability to execute user provided extension programs. These extension programs may, for example, draw to the Elmerpost graphics window using OpenGL graphics calls or manipulate directly the MATC variables, etc.

6.2 Elmerpost Element Types

The element types Elmerpost is aware of are the same as in the solver. These have been described in the Appendix L.

6.3 Elmerpost Input Format

The input file format of Elmerpost has been described in Appedix J. In addition to this format, Elmerpost is able to read the FIDAP neutral file format. The mesh of the model must be given to Elmerpost using either of these formats, but data might be read in separately, using the MATC input routines, both via the `load` command and a set of routines which resemble the C language stdio routines.

Chapter 7

Some Walk Through Examples

In this chapter three case studies are used to demonstrate the usage of ELMER in detail, namely the backward-facing step problem, Rayleigh-Bénard convection, and radiation in axisymmetric enclosure.

7.1 The Backward Facing Step

The geometry of this problem is shown in 7.1. This case is a isothermal steady-state laminar flow problem in 2D. The flow is driven by the inflow velocity boundary condition, where the velocity profile is set to constant value in x-coordinate direction and zero in the y direction. The inflow boundary is the leftmost boundary. At outflow on the right, the flow field is assumed to have developed fully, so that the velocity profile at outflow should be parabolic.

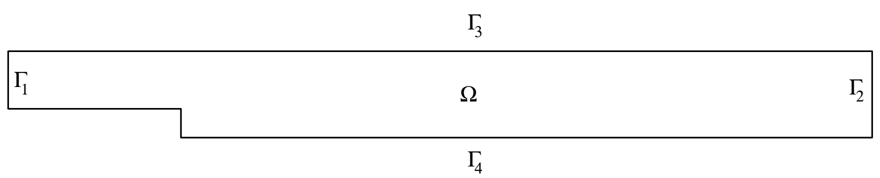

Figure 7.1: Geometry of the Backward-Facing Step Problem.

7.1.1 The Equations to be Solved

The mathematical description of the problem is as follows

$$\begin{aligned}
-\nabla \cdot (2\mu\overline{\overline{\varepsilon}}) + \rho(\vec{u}\cdot\nabla)\vec{u} + \nabla p &= 0 \quad \text{in } \Omega \\
\nabla \cdot \vec{u} &= 0 \quad \text{in } \Omega \\
u_1 = 1,\ u_2 &= 0 \quad \text{on } \Gamma_1, \\
u_2 &= 0 \quad \text{on } \Gamma_2, \\
\vec{u} &= 0 \quad \text{on } \Gamma_3, \Gamma_4.
\end{aligned}$$

The material parameters, in non-dimensional units, are

$$\begin{aligned}
\rho &= 1.0, \\
\mu &= 5 \cdot 10^{-3}.
\end{aligned}$$

7.1.2 Defining the problem

We start the problem definition by reading the cad file describing the geometry of the step problem. You can open this file selecting the `Open Cad file` in the `File` menu. The file for this problem is .../ELMER/Demo/step.unv. The cad geometry is shown in Figure 7.2.

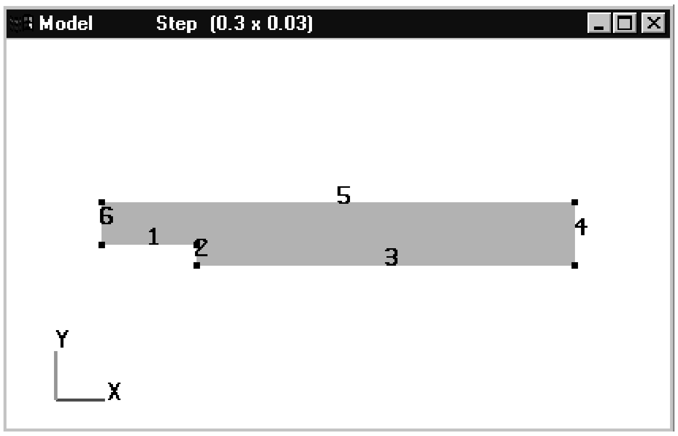

Figure 7.2: The cad geometry of the Step problem.

If you compare this display with Figure 7.1, you can see that boundaries are numbered differently. In Figure 7.1 boundaries are identified according to their role in the problem. In Figure 7.2 boundary elements are created and numbered according to how they are represented in the original cad model. When you define boundary conditions, you can group boundary elements so that they match boundaries in Figure 7.1. Grouping here means simply that you apply the same boundary condition to all elements which define a single 'physical' boundary. For example the boundary Γ_4 consists of the elements 1,2 and 3 in Figure 7.2.

As the very first step we should give a name to the model and define a path for the model files. Use the `Problem/Model` command to open the panel for this data. If you enter text STEP in the `Model name` entry and text /ELMER/MODELS in the `Model DB path` entry, model files are stored in /ELMER/MODELS/STEP directory. If this directory does not exist, it is created.

There is no need to define any input or output paths, because we do not need any specific external files for this problem.

You can also change the default body name (Body1) to something more meaningful and change the color which is used to disaply the body in the graphics window. To do this, use the `Body properites` command in the `Edit` menu.

Also, you could rename the boundaries using the names in Figure 7.1. This would make entering the boundary somewhat conditions easier. Boundary names can be given in the `Edit/Boundaries` panel.

Step problem is a 2D steady state problem in cartesian coordinate system. This means that you do not have to change the default coordinate settings, but the `Simulation type` must be changed to `Steady state`. This can be done in the `Problem/Timestep settings` panel.

You can also set the values of the maximum number of iterations and output frequency in this panel. Default walues are 20 and 1, the latter means that all iterations will be ouput to the solver output file.

Next you should set the equations to be solved. To do this, open the `Problem/Equations` panel. We are solving an isothermal laminar flow problem. To set this equation, click the `Navier-Stokes` checkbutton, because the default value is laminar flow. The equation definition for the Step body is shown in Figure 7.3.

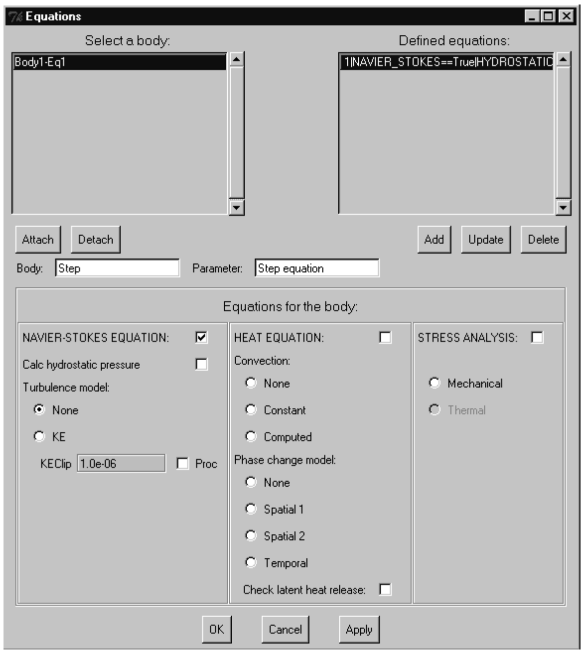

Figure 7.3: Equation definition for the Step problem.

General problem settings are now done and we can continue with the model settings. Because there are no body forces and for this problem we do not have any initial values, you can start from the material properties. Open this panel using the `Model/Material parameters` command and enter the values for density and viscosity as is shown in Figure 7.4

You can enter the viscosity value (as any other real number) in the exponential format like 5.e-3. This is equivalent to 0.005. The latter format is used for display when the data has been checked, ie. when you have pressed the `Add` or `Update` button. Clicking the `Ok` button will save the material parameters, and you can continue with the boundary conditions.

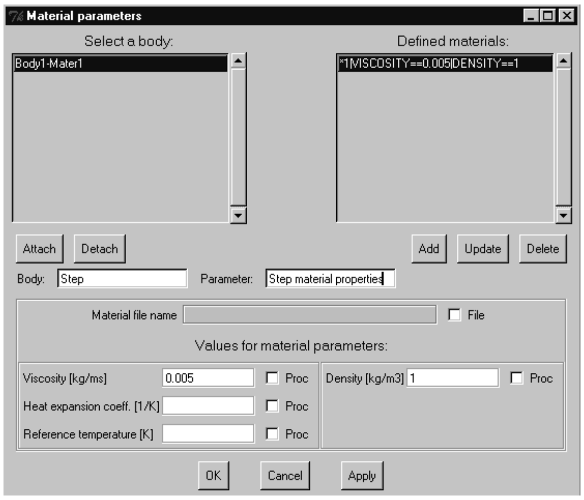

Figure 7.4: Material parameters for the Step body.

The boundary conditions panel is open with the `Model/Boundary conditons` command. This panel is shown in Figure 7.5.

Element Body1-Elm6 is the inflow boundary Γ_1. Here the flow should be strictly in x-direction, so enter 1.0 in the Velocity_X and 0 in the Velocity_Y field. This is a 2D problem and so the Velocity_Z field is disabled.

Element Body1-Elm4 is the ouflow boundary Γ_2. Also here the flow should be in x-direction, but now we do not know the magnitude, it is calculated. You should enter again 0 for the Velocity_Y, but nothing for the Velocity_X. An empty field means no value and for the unknown variable in the problem it means that it is free, calculated.

Boundaries Γ_3 and Γ_4 are no-slip wall boundaries and there you should set both velocities to zero. In Figure 7.5 we have called this condition as the Wall condition and it is applied to the boundary elements 1,2,3 and 5, which together form the boundaries Γ_3 and Γ_4.

Problem is now well defined. However, mesh parameters should be set before you can generate the mesh for the problem. For this simple geometry a modelwide parameter is enough. Open the `Mesh/Mesh parameters` panel and add a parameter with the value 0.005 (of course you can try other values too) to the parameter listbox and apply the parameter to the model object. To save the setting press the `Ok` button.

If you want, you can experiment with the solver settings in the `Solver/Solver parameters` panel, but the default values for the Navier-Stokes equation should

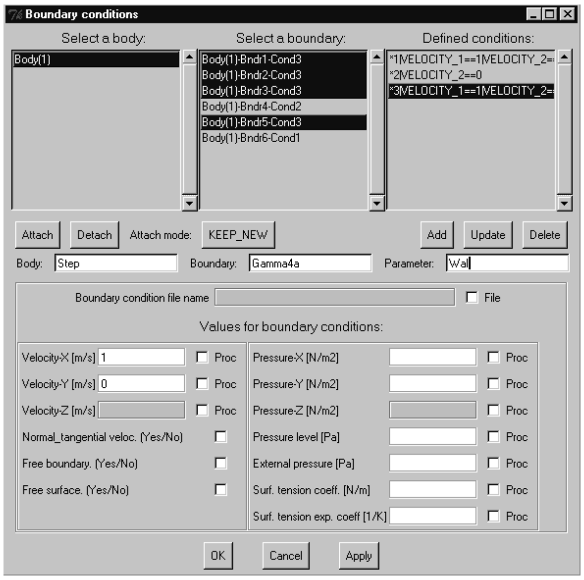

Figure 7.5: Boundary conditions for the Step problem.

be enough for this problem.

Before continuing you should save the model. Selecting `File/Save model As` will save the model file and also create the solver input file. Selecting `File/Store model in ElmerDB As` will create the database files needed for the ELMER mesh generator.

You can now create the mesh, solve the problem and display the results selecting the following commands in the `Run` menu:

- `Generate mesh`
- `Solver`
- `Postprocessor`

7.1.3 Results

The velocity and pressure fields are shown in Figure 7.6. We can see the convection roll below the step in the velocity field. Also, the outflow boundary condition has been satisfied, .i.e. the velocity profile at outflow is parabolic and the pressure gradient vanishes.

7.2 Rayleigh-Bénard Convection

The Rayleigh-Bénard convection is a time dependent coupled heat and flow problem in a rectangular area (Figure 7.7), where the flow is driven by a temperature difference between the top and bottom boundaries. In this example the temperature of the bottom boundary was set to unity. The temperature of the top boundary was set to zero. On the right boundary the velocity is set to zero. The left boundary is a symmetry boundary so that, for symmetry reasons,

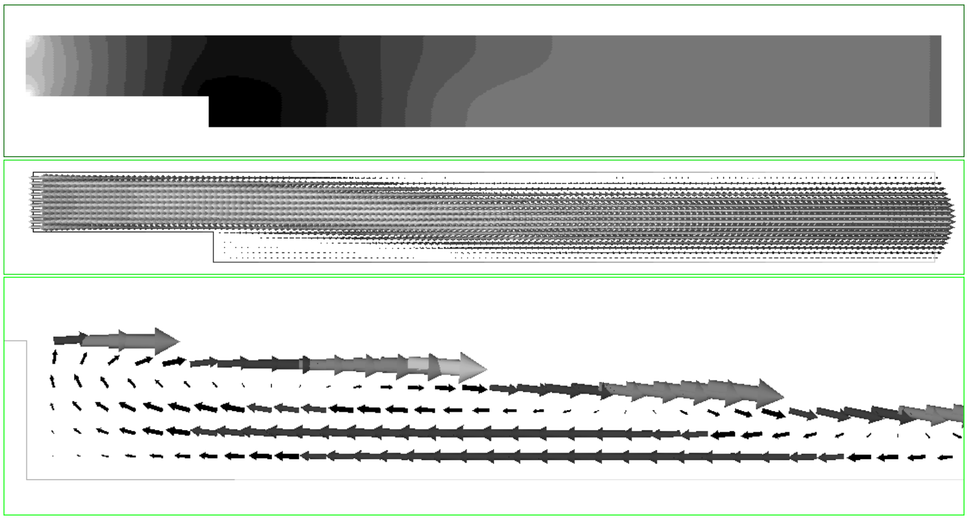

Figure 7.6: Results for the backward-facing step example. We show the isocontours of the pressure field and the velocity vectors.

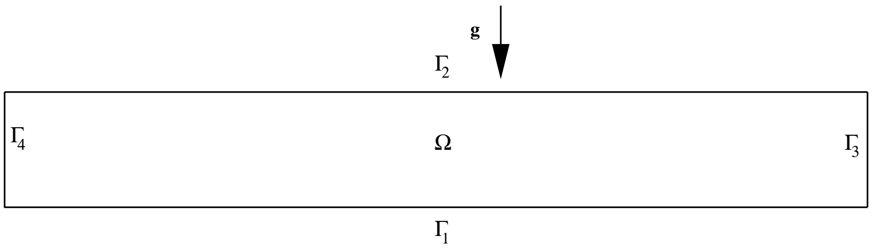

Figure 7.7: Geometry of the Rayleigh-Bénard convection example.

the x-velocity is set to zero, and the y-velocity is free to change. Here the time history of the flow and temperature distribution will be computed. The liquid is initially at rest.

7.2.1 The Equations to be Solved

The mathematical description of the problem is

$$
\begin{aligned}
\rho\frac{\partial\vec{u}}{\partial t} - \nabla\cdot(2\mu\bar{\bar{\varepsilon}}) + \rho(\vec{u}\cdot\nabla)\vec{u} + \nabla p &= \rho\vec{g}(1-\beta(T-T_0)) \quad \text{in } \Omega_1 \\
\nabla\cdot\vec{u} &= 0 \quad \text{in } \Omega \\
\rho c_p\frac{\partial T}{\partial t} + \rho c_p\vec{u}\cdot\nabla T - \nabla\cdot(k\nabla T) &= 0 \quad \text{in } \Omega, \\
\vec{u}(0) = 0, \quad T(0) &= 0, \quad \text{in } \Omega, \\
\vec{u} = 0, \quad T &= 1 \quad \text{on } \Gamma_1, \\
\vec{u} = 0, \quad T &= 0 \quad \text{on } \Gamma_2, \\
\vec{u} = 0, \quad -k\frac{\partial T}{\partial n} &= 0 \quad \text{on } \Gamma_3, \\
u_1 = 0, \quad -k\frac{\partial T}{\partial n} &= 0 \quad \text{on } \Gamma_4.
\end{aligned}
$$

The following material parameters are those of water at room temperature

$$
\begin{aligned}
\rho &= 998.2 \quad \frac{\mathrm{kg}}{\mathrm{m}^3}, \\
\mu &= 9.93\cdot 10^{-4} \quad \frac{\mathrm{kg}}{\mathrm{ms}}, \\
c_p &= 4182.5 \quad \frac{\mathrm{J}}{\mathrm{kgK}}, \\
k &= 0.597 \quad \frac{\mathrm{W}}{\mathrm{mK}}, \\
\beta &= 2.1\cdot 10^{-4} \quad \frac{1}{\mathrm{K}}.
\end{aligned}
$$

7.2.2 Defining the problem

The cad geometry of the Rayleigh-Bénard problem is defined in the .../ELMER/Demo/rb.unv file. This geometry is displayed in Figure 7.8.

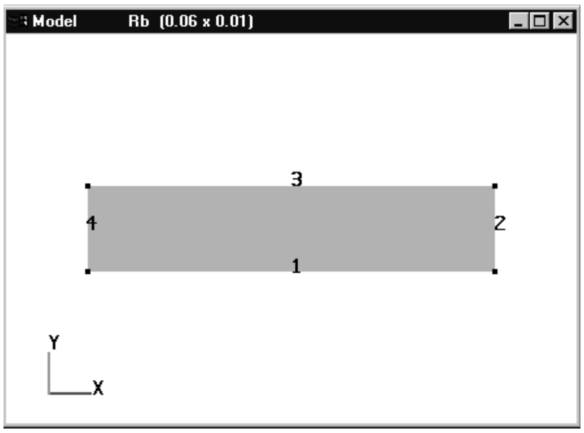

Figure 7.8: Cad geometry for the Rayleigh-Bénard problem.

If you compare the numbering of the boundary elements in the graphics window with the boundaries in Figure 7.7, you should notice that the element number 2 is the boundary Γ_3 and the element number 3 is Γ_2. This should be taken into account when setting the boundary conditions.

If you rename boundaries using the names in Figure 7.7, entering the boundary conditions would be easier. Boundary names can be given in the `Edit/Boundaries` panel.

The Rayleigh-Bénard problem is a transient 2D problem in cartesian coordinate system and these are also default values. But you should set the timestepping scheme in the `Problem/Timestep settings` panel. A constant timestep size of one second is suitable for this problem. A steady state solution is reached after about 200 seconds. If you want to output every fourth timestep, you should enter the numbers 1.0 200 4 in the `Timestep settings` entry and press the `Add` button. Pressing the `Ok` button will save the settings.

The Rayleigh-Bénard problem is a coupled flow and heat problem where flow is driven by the natural convection which arises from the gravity and the temperature differences. The equations we need in this case are the laminar Navier-Stokes and the heat equation with computed convection. Necessary definitions are shown in Figure 7.9

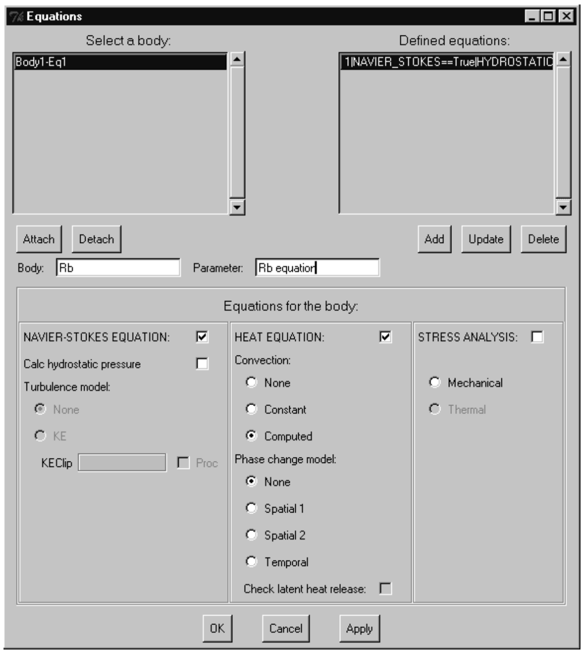

Figure 7.9: Equations for the Rayleigh-Bénard problem.

Natural convection is modelled using the Boussinesq body force. To set this force open the `Model/Body forces` panel. Select the body and click the Boussinesq checkbox. Add the definition to the body forces list and apply it to the body. Press `Ok` to save the definition.

Initially the liquid is at rest and temperature is zero. To define this open the `Model/Initial conditions` panel. Select the body and enter zeros to the

`Velocity_X`, `Velocity_Y` and `Temperature` fields. Add the definition to the initial conditions list and apply it to the body. Press `Ok` to save the definition.

This definition is shown in Figure 7.10

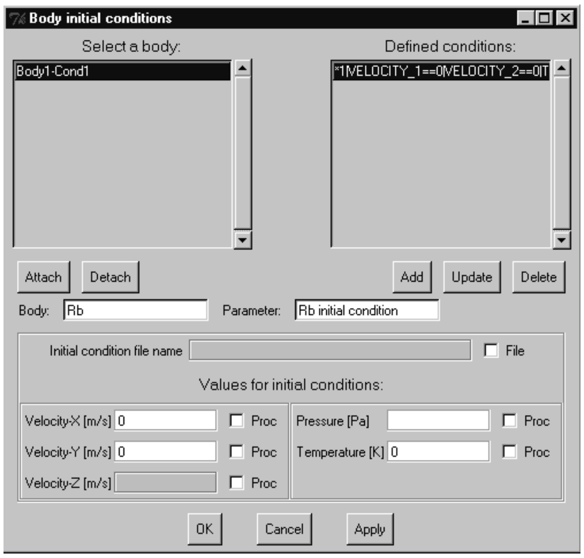

Figure 7.10: Initial conditions for the Rayleigh-Bénard problem.

Material parameters for the problem are shown in Figure 7.11.

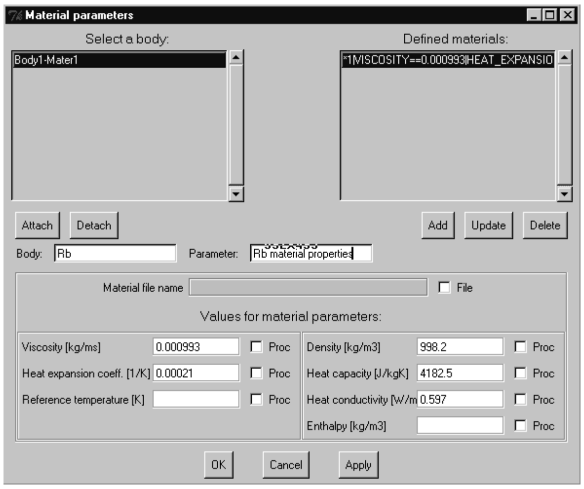

Figure 7.11: Material parametrs for the Rayleigh-Bénard problem.

To set boundary conditions we first define four conditions with the following names: the Bottom (Cond1), the Top (Cond2), the Left (Cond3) and the Right (Cond4).

All except the Left describe fixed walls and the velocities are set to zeros at these boundaries. At the Bottom `Temperature` is set to zero and at the Top to unity.

The Right wall is isolated and heat flux is set there to zero. The Left boundary is a symmetry boundary and the velocity in the x-direction is set to zero and the y-velocity is free. Because of the symmetry, heat flux is also set to zero in this boundary.

The applied conditions and the heat equation settings for the Right wall are shown in Figure 7.12. You should note the radiobuttons in the middle of the panel. They are used to select the equation and consequently the entry fields which are displayed on the panel. It is important that you check the entry values for each equation before you add a boundary condition to the list. Otherwise you could forget some old values into the entries which are not visible.

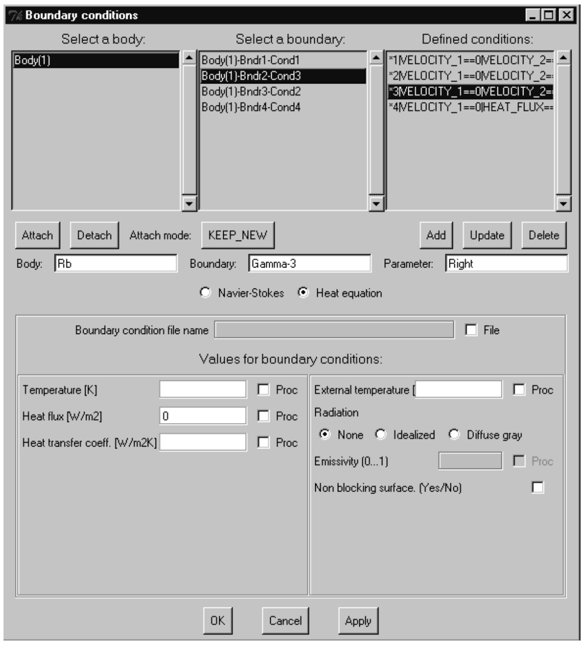

Figure 7.12: Boundary conditions for the Rayleigh-Bénard problem.

After saving the boundary conditons you can save the model, create the mesh and continue with solving the problem. You can also experiment with settings for the solver, but default values are again appropriate.

7.2.3 Results

Heat is initially transferred by conduction only (at least until 50 seconds of simulation time), and the liquid is otherwise at rest. When the local temperature differences become large enough, the liquid begins to move (from 50 secods up). At 100 seconds the flow has already formed the final eight convection rolls and at 200 seconds we have reached a steady state solution. The development of the convection rolls are shown in Figures 7.13 and 7.14.

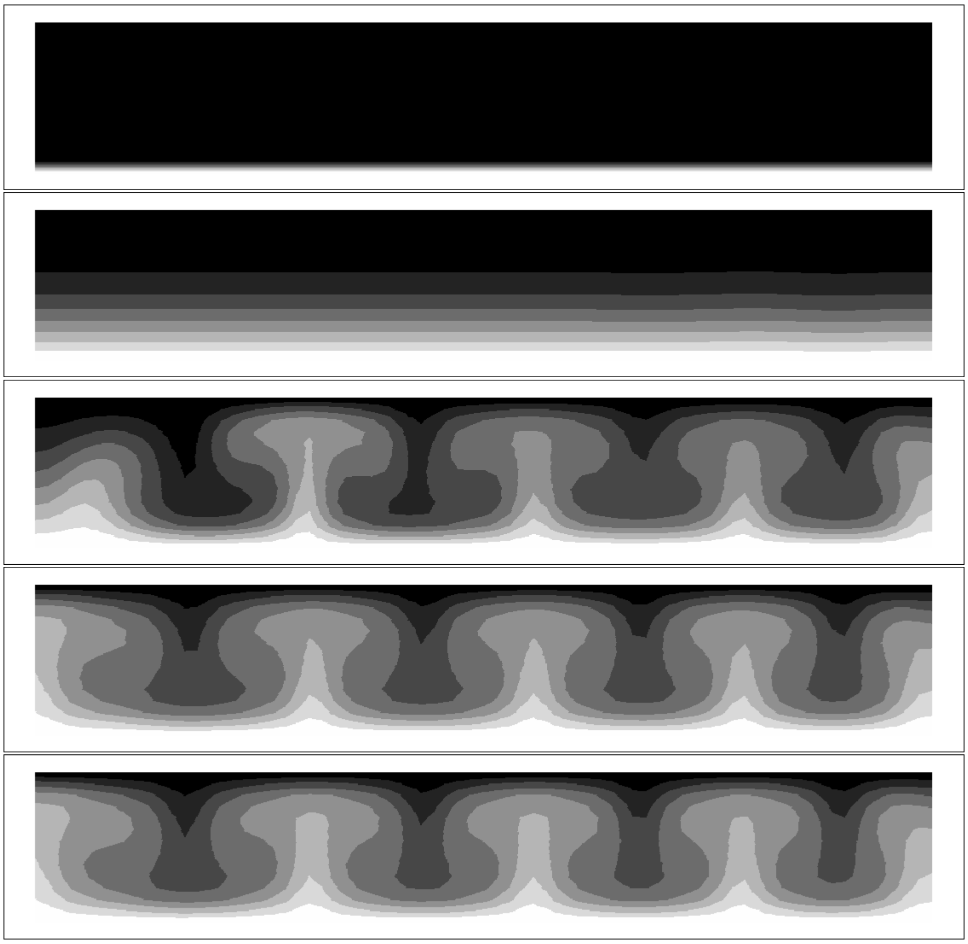

Figure 7.13: Results of the Rayleigh-Bénard convection example. Time history of the temperature field: isocontours of temperature at times $t = 0$s, $t = 50$s, $t = 100$s,$t = 150$s, and $t = 200$s.

7.3 Radiation in Axisymmetric Enclosure

This example is a steady state coupled flow and heat problem with axisymmetric geometry. The geometry of the problem is shown in Figure 7.15. Heat is exchanged between the inner boundaries of the enclosure by diffuse gray radiation. This radiative exchange of heat results in temperature differences between the boundaries of the area Ω_1, where the flow equations are applied. These temperature differences drive the velocity and pressure fields. Inside Ω_1 heat is transferred by conduction and convection, which has an effect on the boundary temperatures of this area and thus the global temperature field is connected back to flow field.

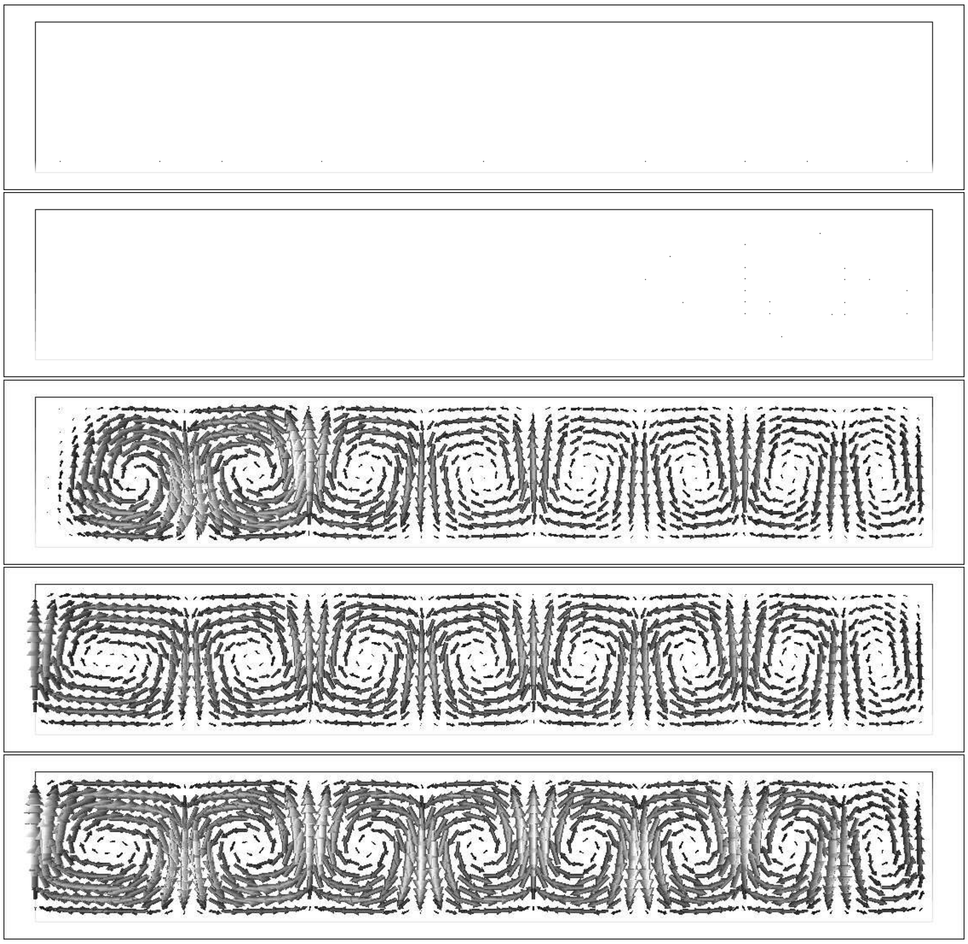

Figure 7.14: Results of the Rayleigh-Bénard convection example. Time history of the velocity field: velocity vectors at times $t = 0$s, $t = 50$s, $t = 100$s,$t = 150$s, and $t = 200$s.

7.3.1 The Equations to be Solved

The equations for this case are written below. Note that the geometry is axisymmetric, and the definition of the variables and derivative operators should

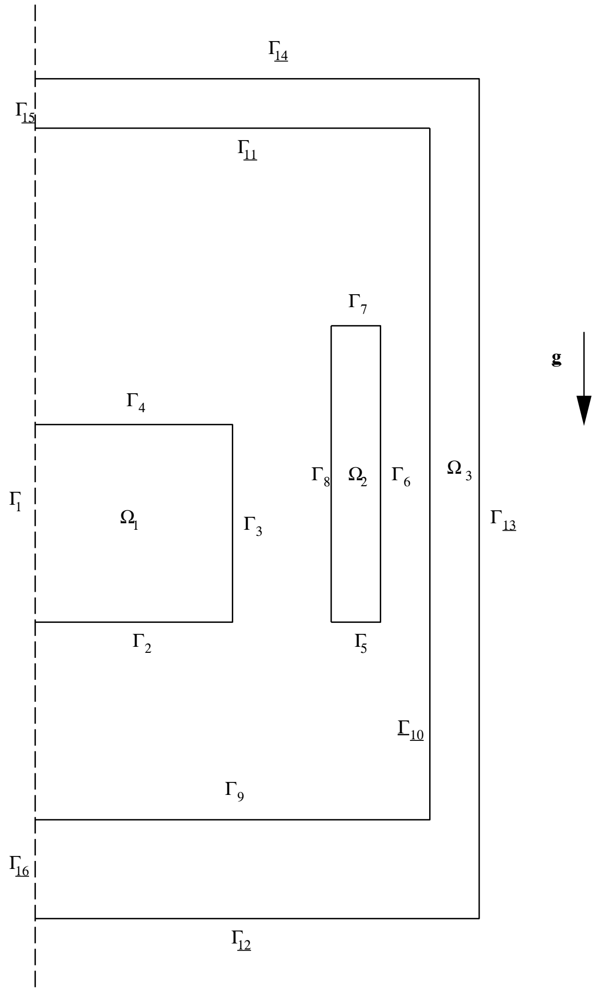

Figure 7.15: Geometry of the Radiation Example.

be taken accordingly.

$$
\begin{aligned}
-\nabla\cdot(2\mu\overline{\overline{\varepsilon}}) + \rho(\vec{u}\cdot\nabla)\vec{u} + \nabla p &= \rho\vec{g}(1-\beta(T-T_0)) \qquad \text{in } \Omega_1 \\
\nabla\cdot\vec{u} &= 0 \quad \text{in } \Omega_1 \\
\rho c_p\vec{u}\cdot\nabla T - \nabla\cdot(k\nabla T) &= 0 \quad \text{in } \Omega_1, \\
-\nabla\cdot(k\nabla T) &= 10000 \quad \text{in } \Omega_2, \\
-\nabla\cdot(k\nabla T) &= 0 \quad \text{in } \Omega_3, \\
\frac{\partial T}{\partial r} &= 0 \quad \text{on } \Gamma_1, \Gamma_{15}, \Gamma_{16} \\
u_r = 0, \quad \frac{\partial u_z}{\partial r} &= 0, \quad \text{on} \Gamma_1, \\
-k_k\frac{\partial T_k}{\partial n_k} = \sigma\varepsilon_k\Big(T_k^4 - \frac{1}{S_k\varepsilon_k}\sum_{i=1}^{N} G_{ik}\varepsilon_i T_i^4 S_i\Big), & \qquad \text{on } \Gamma_2 \ \text{ through } \Gamma_{11}, \\
u_r = 0, \quad u_z &= 0, \quad \text{on } \Gamma_2, \Gamma_3, \\
\vec{u}\cdot\vec{n} &= 0, \quad \text{on } \Gamma_4, \\
T &= 240, \quad \text{on } \Gamma_{12} \ \text{ through } \Gamma_{14}.
\end{aligned}
$$

The material parameters in the area Ω_1, in non-dimensional units, are

$$\begin{aligned} \rho &= 1.0, \\ \mu &= 2.2222 \cdot 10^{-5}, \\ c_p &= 1000.0, \\ k &= 0.5, \\ \beta &= 2.0 \cdot 10^{-5}, \\ T_0 &= 200, \\ \varepsilon &= 0.75. \end{aligned}$$

The material parameters in the area Ω_2 are set as follows

$$\begin{aligned} \rho &= 1.0, \\ k &= 0.5, \\ \varepsilon &= 0.8. \end{aligned}$$

The material parameters in the area Ω_3 are

$$\begin{aligned} \rho &= 1.0, \\ k &= 0.5, \\ \varepsilon &= 0.8. \end{aligned}$$

7.3.2 Defining the problem

The cad geometry of the radiation problem is defined in the .../ELMER/Demo/radiation.unv file. This geometry is displayed in Figure 2.6

The radiation problem is a steady-state axi-symmetric 2D problem. You should select these options in the `Coordinate settings` and the `Timesteps settings` panels.

Like the Rayleigh-Bénard problem, this is a coupled flow and heat problem where flow is driven by the natural convection. For the body Ω_1 the equations are the laminar Navier-Stokes and heat equation with computed convection. For the other two bodies you should select just the `Heat equation`, because we need to calculate only the heat condcution in these bodies.

To set the body forces on the `Model/Body forces` panel. For the flow body Ω_1 you should apply the `Boussinesq` body force The body Ω_2 a heater. Create a heat source body force with the value 10 000 $\mathrm{J/m^3s}$ and apply it to Ω_2.

For the flow body (Ω_1) we have the same material parameters as in the Raleigh-Bénard problem. Note that the value for the `Reference temperature` is now 200. For the other two bodies you have to define only values for density and heat conductivity.

Note that emissivity (ϵ) is not defined as a material parameter, but as a boundary condition parameter for the radiation boundaries.

Radiation is the factor which makes this problem more complicated than the previous examples. To make the setting of the boundary conditions easier, it is better to group the boundaries into homogenous groups.

As in previous examples, you could rename the cad boundaries using the names in Figure 7.15. Use the panel `Edit/Boundaries` to rename the boundaries.

First, the boundaries at the symmetry axis (Γ_1, Γ_{15}, Γ_{16}) are nearly similar. At the symmetry axis heat flux is zero. Because Γ_1 belongs to the flow domain, we have to also set the velocity in the r-direction to zero at the boundary.

The other condition for the boundary Γ_1 ($\frac{\partial u_z}{\partial r} = 0$) is built in the equations, so there is no need to specify it.

This gives us two different boundary conditions. We will call them the Flow symmetry and the Heat symmetry conditions.

Boundaries Γ_2 through Γ_{11} all have a radiation boundary condition, but the flow boundaries Γ_2, Γ_3 and Γ_4 are again a bit different. Boundaries 2 and 3 are walls and we have a no-slip condition there. At the boundary Γ_4 liquid velocity is set to zero in the normal direction, but otherwise the liquid can flow freely in the surface. At the boundaries Γ_5 through Γ_{11} radiation is the only condition.

This gives us three additional boundary condition groups. We call them as the Flow walls, the Flow surface and the Radiation walls conditions.

The rest of the boundaries are at the outside cover and we call them the Bottom (Γ_{12}), the Top (Γ_{14}) and the Right (Γ_{13}) conditions.

The Bottom is kept in constant temperature, the Right is isolated (no heat flux) and the Top is also kept in constant temperature.

It is better to first define all these boundary conditions and only after that apply them to proper boundary elements. This is perhaps easiest to do if you first select Ω_1 (Body1) and enter the values for the Flow symmetry, Flow walls and Flow surface conditions, because these all are related to the equation type of that body. To define the rest of the conditions it is enough to select one of the remaining bodies, because both have a similar equation definition and consequently same entry fields are available for them.

You have to again compare Figure 7.15 and Figure 2.6 in order to match boundaries and boundary elements. Remember to check values for the Navier-Stokes entries and the Heat equation entries before adding the conditions for Ω_1.

Boundary condition panel for this problem is displayed in Figure 7.16. Note how the boundary condition for the Flow surface is defined using the `Normal-tangential velocity` flag when setting the velocity condition. Check that you set the normal velocity to be zero at this boundary.

This problem should be solved using the diffuse-gray radiation model. So, select first the `Diffuse gray` radiobutton in the heat equation `Radiation` entry. After that you can enter the value for emissivity.

When all the boundary condtions have been defined, you should save the model and then solve it using the following commands in the `Run` menu:

- `Generate mesh`
- `Calculate View factors`: these should be recalculated each time the mesh is changed.
- `Calculate Gebhardt factors`: these should be recalculated when the view factors are recalculated or if the emissivity value for any boundary is changed.
- Run `Solver`
- Run `Postprocessor`

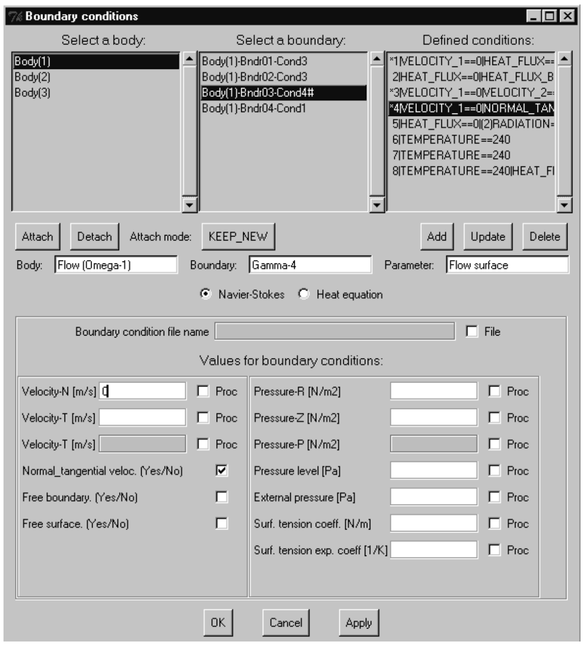

Figure 7.16: Boundary conditions for the Radiation problem.

7.3.3 Results

Results of the simulation are shown in Figure 7.17.

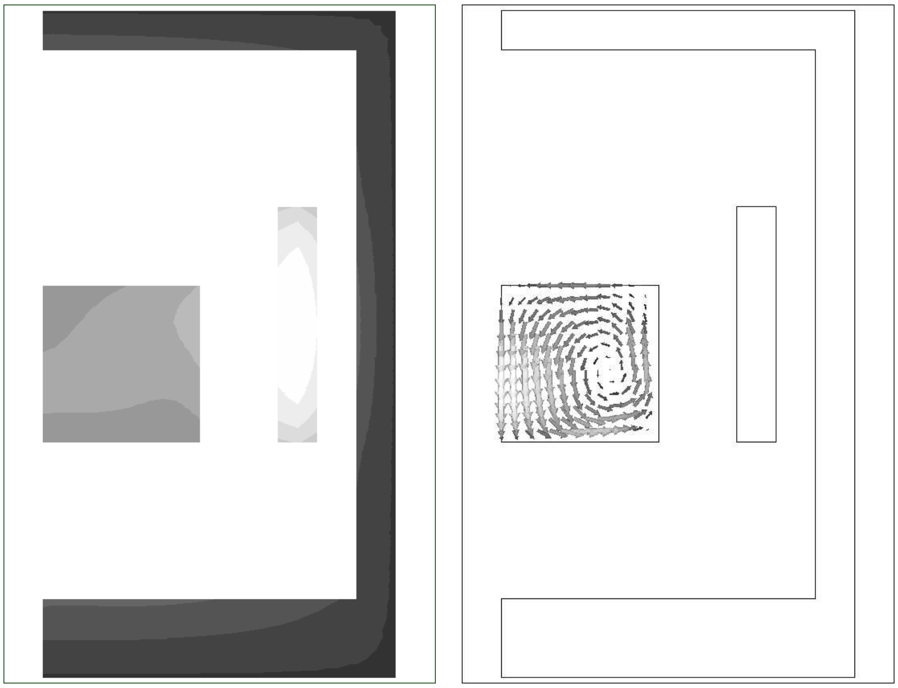

Figure 7.17: Results of the Radiation Example. We have plotted the isocontours of the temperature field and the velocity vectors

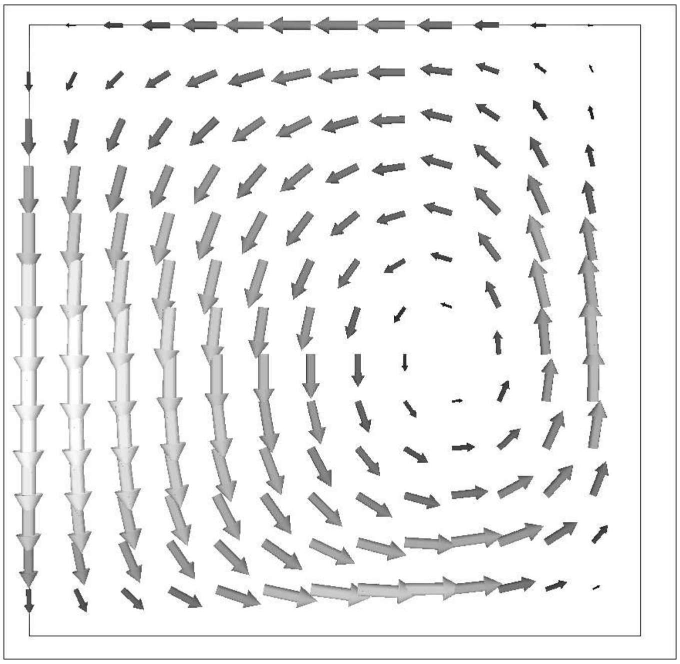

Figure 7.18: Results of the Radiation Example. The figure shows the velocity vectors of the flow field.

Appendix A

A Very Short Introduction to Tensors

In this chapter a very short practically oriented introduction to tensor formalism is given. Emphasis is placed on the computation rules for tensors.

A.1 Covariant and Contravariant Tensors

Tensors are mathematical concepts independent of coordinate systems. When expressed componentwise in a coordinate system, the components change according to specific laws when a change of coordinates is made. If a tensor dependes on n indeces, it is said to be of rank n. Scalars (temperature, for example) are rank zero tensors, vectors (pressure gradient or velocity, for example) are rank one tensors. The stress tensor is an example of a rank two tensor, depending on two indices. The components of a tensor may be covariant or contravariant. A given tensor might have both covariant and contravariant components at the same time, and thus has both a covariant rank and a contravariant rank.

Let a coordinate transformation between two coordinate systems (y^i) and (x^i) be given

$$x^i = x^i(y^1, \ldots, y^n), \quad i = 1, \ldots, n. \tag{A.1}$$

Scalars transform by invariance

$$T_X(x^1(y^1, \ldots, y^n), \ldots, x^n(y^1, \ldots, y^n)) = T_Y(y^1, \ldots, y^n).$$

Covariant components of a vector transform by the covariant law of transformation

$$v_i = \frac{\partial x^\alpha}{\partial y^i} v_\alpha. \tag{A.2}$$

This is also the usual chain rule of derivation if $\mathbf{v}$ is a gradient vector, and indeed a prototypical covariant vector is the gradient vector of a scalar quantity. The contravariant law of transformation on the other hand is

$$v^i = \frac{\partial y^i}{\partial x^\alpha} v^\alpha, \tag{A.3}$$

which, for example, can be identified as the transformation law of a vector assembled from differentials. We will follow the convention that a contravariant index of a tensor is denoted by superscript, covariant by subscript. The transformation laws generalize in an obvious way to tensors of rank higher than one.

A.2 The Metric Tensor and the Christoffel Symbols

We can metricize a given vector space by forming a measure of length (s) of a curve segment as follows

$$ds^2 = g_{ij} dx^i dx^j. \tag{A.4}$$

The length of a curve parametrized by t may now be computed as

$$s = \int_{t_0}^{t_1} \sqrt{g_{ij} \frac{dx^i}{dt} \frac{dx^j}{dt}} dt. \tag{A.5}$$

The formula (A.4) is called the first fundamental quadratic form. A space metricized as given is called a Riemannian space. The coefficients g_{ij} of the quadratic form are the covariant components of a rank two tensor, called the fundamental tensor or the metric tensor. As can be seen from the definition, the metric tensor is symmetric i.e. $g_{ij} = g_{ji}$. If the coordinate system is orthogonal, the metric tensor is diagonal i.e. $g_{ij} = 0, \quad i \neq j$. The metric tensor of the orthonormal cartesian coordinates is the unit tensor δ_{ij}, $(\delta_{ij} = 1, \quad i = j, \quad \delta_{ij} = 0$, otherwise).

Given a connection between the coordinate system at hand (y^i) and some coordinates (x^i) for which the metric h_{ij} is known

$$x^i = x^i(y^1, \ldots, y^n), \quad i = 1, \ldots, n \tag{A.6}$$

the contravariant form of the metric tensor g^{ij} can be computed with the aid of the covariant components of the metric tensor

$$g_{ij} = \frac{\partial x^\alpha}{\partial y^i} \frac{\partial x^\beta}{\partial y^j} h_{\alpha\beta} \tag{A.7}$$

by $g^{ij} = (g_{ij})^{-1}$.

Christoffel symbols of the second kind can also be computed with the aid of the covariant form of the metric tensor

$$[ij, k] = \frac{1}{2} \left(\frac{\partial g_{ik}}{\partial y^j} + \frac{\partial g_{jk}}{\partial y^i} - \frac{\partial g_{ij}}{\partial y^k} \right). \tag{A.8}$$

Christoffel symbols of the first kind are then given by

$$\left\{ \begin{array}{c} k \\ i\ j \end{array} \right\} = g^{k\alpha} [ij, \alpha]. \tag{A.9}$$

Christoffel symbols are *not* tensors and hence do *not* follow the transformation rules of tensors (unless the coordinate transformation (A.6) is affine). Both kinds of the Christoffel symbols are symmetric with respect to indices i and j. If the coordinates x^i are the orthonormal cartesian coordinates, the Christoffel symbols of the first kind can be seen to be

$$\left\{ \begin{array}{c} k \\ i\ j \end{array} \right\} = \frac{\partial^2 x^\alpha}{\partial y^i \partial y^j} \frac{\partial y^k}{\partial x^\alpha} \tag{A.10}$$

and the symbols of the second kind

$$[ij, k] = \frac{\partial^2 x^\alpha}{\partial y^i \partial y^j} \frac{\partial x^\alpha}{\partial y^k}. \tag{A.11}$$

A.3 Covariant to Contravariant and Back

The two forms of tensors can be converted to each other with the aid of the metric tensor. Covariant indices can be raised to contravariant position by

$$v^i = g^{i\alpha} v_\alpha, \quad \sigma^{ij} = g^{i\alpha} g^{j\beta} \sigma_{\alpha\beta}, \tag{A.12}$$

and similarily contravariant indices can be lowered to covariant position by

$$v_i = g_{i\alpha} v^\alpha, \quad \sigma_{ij} = g_{i\alpha} g_{j\beta} \sigma^{\alpha\beta}. \tag{A.13}$$

One can also form mixed tensors by rasing or lowering only some of the indices

$$\sigma^i_j = g_{j\alpha} \sigma^{i\alpha}. \tag{A.14}$$

A.4 Covariant Differentiation

Covariant derivate of a scalar is the usual partial derivate

$$p_{,j} = \frac{\partial p}{\partial x^j}. \tag{A.15}$$

Covariant derivate of a contravariant vector is given by

$$u^i{}_{,j} = \frac{\partial u^i}{\partial y^j} + \left\{ \begin{array}{c} i \\ k\ j \end{array} \right\} u^k, \tag{A.16}$$

Covariant derivate of a covariant vector on the other hand is

$$u_{i,j} = \frac{\partial u^i}{\partial y^j} - \left\{ \begin{array}{c} k \\ i\ j \end{array} \right\} u_k. \tag{A.17}$$

Covariant derivate of a general tensor can be written as

$$A^{i^1 i^2 \ldots i^n}_{j^1 j^2 \ldots j^m}{}_{,l} = \frac{\partial}{\partial x^l} A^{i^1 i^2 \ldots i^n}_{j^1 j^2 \ldots j^m} + \left\{ \begin{array}{c} i^1 \\ \alpha l \end{array} \right\} A^{\alpha i^2 \ldots i^n}_{j^1 j^2 \ldots j^m} + \tag{A.18}$$

$$\left\{ \begin{array}{c} i^2 \\ \alpha l \end{array} \right\} A^{i^1 \alpha \ldots i^n}_{j^1 j^2 \ldots j^m} + \ldots + \left\{ \begin{array}{c} i^n \\ \alpha l \end{array} \right\} A^{i^1 i^2 \ldots \alpha}_{j^1 j^2 \ldots j^m} - \left\{ \begin{array}{c} \alpha \\ j^1 l \end{array} \right\} A^{i^1 i^2 \ldots i^n}_{\alpha j^2 \ldots j^m} - \tag{A.19}$$

$$\left\{ \begin{array}{c} \alpha \\ j^2 l \end{array} \right\} A^{i^1 i^2 \ldots i^n}_{j^1 \alpha \ldots j^m} - \ldots - \left\{ \begin{array}{c} \alpha \\ j^m l \end{array} \right\} A^{i^1 i^2 \ldots i^n}_{j^1 j^2 \ldots \alpha}. \tag{A.20}$$

Because covariant derivate of a tensor is a tensor (of rank one higher than the original and covariant with respect to the new index) one can form second covariant derivates by differentiating covariantly again.

Difference between two second covariant derivates of covariant components of a vector taken in different order

$$R^{\alpha}_{ijk} A_{\alpha} = A_{i},_{jk} - A_{i},_{kj} \tag{A.21}$$

is called the Riemann-Christoffel tensor. If the covariant differentiation is to be commutative, this tensor should vanish identically. A space where this is true is called Euclidian. The vanishing of the Riemann-Christoffel tensor is also enough to guarantee that the quadratic form (A.4) can be reduced, by a suitable change of coordinates, to

$$ds^2 = dx^i dx^i, \tag{A.22}$$

indicating orthonormal cartesian coordinates. In what follows we consider only Euclidian spaces.

A.5 The Second Fundamental Form and the Mean Curvature

Let space coordinates of a surface be given parametrically as in

$$x^i = x^i(u^1, u^2), \quad i = 1, \ldots, 3. \tag{A.23}$$

Using this definition, we may compute the metric tensor of the surface

$$a_{\alpha\beta} = \frac{\partial x^i}{\partial u^\alpha} \frac{\partial x^j}{\partial y^\beta} g_{ij}, \tag{A.24}$$

where g_{ij} is the metric tensor of the space coordinates x^i.

The normal vector to the surface may be written

$$n_i = \frac{1}{2} \epsilon^{\alpha\beta} \epsilon_{ijk} x^j_\alpha x^k_\beta, \tag{A.25}$$

where

$$x^i_\alpha = \frac{\partial x^i}{\partial u^\alpha}, \tag{A.26}$$

and $\epsilon^{12} = 1/\sqrt{a}$, $\epsilon^{21} = -1/\sqrt{a}$, and $\epsilon^{11} = \epsilon^{22} = 0$. In the same manner $\epsilon_{ijk} = \sqrt{g}$, whenever ijk is an even permutation of 123, $\epsilon_{ijk} = -\sqrt{g}$ whenever the permutation is odd, and $\epsilon_{ijk} = 0$, if two or more of the indices are alike.

A second rank tensor $b_{\alpha\beta}$ defined by

$$b_{\alpha\beta} = x^i_{\alpha,\beta} n_i \tag{A.27}$$

is called the second fundamental form of the surface. With the aid of the both fundamental forms of the surface, the mean curvature of the surface may be expressed as follows

$$H = \frac{1}{2} a^{\alpha\beta} b_{\alpha\beta}. \tag{A.28}$$

Appendix B

Discretization of the Equations

The following describes the ELMER Solver discretization of the incompressible Navier-Stokes equations, the scalar diffusion-convection equation and the elastic stress equations.

B.1 The Incompressible Navier-Stokes Equations

One possible starting point in deriving the Navier-Stokes equations is the Newtons second law of motion (for the notation refer to Appendix A)

$$\sigma^{ij}{}_{,j} = \rho \left(a^i - f^i \right), \tag{B.1}$$

where σ^{ij} is the contravariant form of the (incompressible) stress tensor

$$\sigma^{ij} = -pg^{ij} + 2\mu\varepsilon^{ij}, \tag{B.2}$$

ε^{ij} the linear strain rate tensor

$$\varepsilon^{ij} = \frac{1}{2} \left(g^{jk} u^i{}_{,k} + g^{ik} u^j{}_{,k} \right) \tag{B.3}$$

describing a Newtonian fluid, u^i the velocity vector, μ the viscosity of the fluid, ρ is density, f^i external (volume) force, a^i the acceleration

$$a^i = \frac{\partial u^i}{\partial t} + u^i{}_{,j} u^j, \tag{B.4}$$

p pressure, and g^{ij} the contravariant form of the metric tensor of the coordinate system.

The notation $u^i{}_{,j}$ denotes covariant differentiation

$$u^i{}_{,j} = \frac{\partial u^i}{\partial y^j} + \left\{ \begin{matrix} i \\ k\ j \end{matrix} \right\} u^k, \tag{B.5}$$

where the symbols with curly braces is the Christoffel symbol of the first kind.

Combining all the above we have the Navier-Stokes equations

$$\rho \frac{\partial u^i}{\partial t} + g^{ij} p,_j - 2\mu \varepsilon^{ij},_j + \rho u^i,_j u^j = \rho f^i \tag{B.6}$$

or

$$\rho \frac{\partial u^i}{\partial t} + g^{ij} p,_j - \mu \left(g^{jk} u^i,_k + g^{ik} u^j,_k \right)_{,j} + \rho u^i,_j u^j = \rho f^i \tag{B.7}$$

or taking the (incompressible) mass conservation law

$$u^i,_i = 0$$

into account

$$\rho \frac{\partial u^i}{\partial t} + g^{ij} p,_j - \mu g^{jk} u^i,_{jk} + \rho u^i,_j u^j = \rho f^i. \tag{B.8}$$

Next we will derive the variational formulation and discretize the equations.

B.1.1 Variational Formulation

We can write down the variational form of the equation (B.7) of the above section as

$$\int \left(\rho \frac{\partial u^i}{\partial t} + g^{ij} p,_j - \mu (g^{jk} u^i,_k + g^{ik} u^j,_k),_j + \rho u^i,_j u^j \right) w_i \, dV = \int \rho f^i w_i \, dV \tag{B.9}$$

($dV = \sqrt{g} dy^1 \dots dy^n$, where g is the determinant of g_{ij}) which after integrating the diffusion and pressure terms (i.e. the stress tensor part of the Navier-Stokes equations) by parts reads

$$\int \rho \frac{\partial u^i}{\partial t} w_i \, dV \quad + \quad \int \left(-g^{ij} p + \mu \left(g^{jk} u^i,_k + g^{ik} u^j,_k \right) \right) w_i,_j \, dV \tag{B.10}$$

$$+ \quad \int \rho u^i,_j u^j w_i \, dV = \int \sigma^{ij} n_j w_i \, d\Gamma + \int \rho f^i w_i \, dV \tag{B.11}$$

The variational form of the continuity equation is simply

$$\int u^i,_i w_c \, dV = 0. \tag{B.12}$$

B.1.2 Discretization of the Variational Formulation

If we now make the interpolation substitutions for the variables according to the Galerkin method

$$p = \sum_\beta \gamma_\beta p_\beta, \quad u^i = \sum_\beta \gamma_\beta u^i_\beta, \quad w_i = w_c = \gamma_\alpha, \tag{B.13}$$

where the functions γ are the piecewise defined polynomials, and the α and β indices are node indices, write the variational equation in matrix form (for one node) as

$$M \frac{\partial \mathbf{x}}{\partial t} + A\mathbf{x} = F, \tag{B.14}$$

where $\mathbf{x} = \left(u^1 \ u^2 \ u^3 \ p\right)^T$, and use Picard linearization for the convection term

$$u^i{}_{,j}\, u^j = u^i{}_{,j}\, \mathcal{U}^j, \tag{B.15}$$

we can write the diagonal matrix M as

$$M_{11} = M_{22} = M_{33} = \int \rho \gamma_\beta \gamma_\alpha \, dV. \tag{B.16}$$

Lets describe the A matrix in a procedural manner, setting the elements of the matrix to zero initially. Then we can first add the pressure terms as

$$A_{i4} \quad += \quad \int g^{ij} \gamma_\beta \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.17}$$

$$A_{k4} \quad -= \quad \int g^{ij} \gamma_\beta \left\{ \begin{matrix} k \\ ij \end{matrix} \right\} \gamma_\alpha \, dV. \tag{B.18}$$

Next we add the diffusive terms:

$$A_{ii} \quad += \quad \int \mu g^{jk} \frac{\partial \gamma_\beta}{\partial y^k} \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.19}$$

$$A_{ij} \quad += \quad \int \mu g^{ik} \frac{\partial \gamma_\beta}{\partial y^k} \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.20}$$

$$A_{il} \quad += \quad \int \mu \left(g^{jk} \left\{ \begin{matrix} i \\ lk \end{matrix} \right\} + g^{ik} \left\{ \begin{matrix} j \\ lk \end{matrix} \right\} \right) \gamma_\beta \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.21}$$

$$A_{li} \quad -= \quad \int \mu g^{jk} \frac{\partial \gamma_\beta}{\partial y^k} \left\{ \begin{matrix} l \\ ij \end{matrix} \right\} \gamma_\alpha \, dV, \tag{B.22}$$

$$A_{lj} \quad -= \quad \int \mu g^{ik} \frac{\partial \gamma_\beta}{\partial y^k} \left\{ \begin{matrix} l \\ ij \end{matrix} \right\} \gamma_\alpha \, dV, \tag{B.23}$$

$$A_{lm} \quad -= \quad \int \mu \left(g^{jk} \left\{ \begin{matrix} i \\ mk \end{matrix} \right\} + g^{ik} \left\{ \begin{matrix} j \\ mk \end{matrix} \right\} \right) \gamma_\beta \left\{ \begin{matrix} l \\ ij \end{matrix} \right\} \gamma_\alpha \, dV. \tag{B.24}$$

For the convective terms we have

$$A_{ii} \quad += \quad \int \rho \frac{\partial \gamma_\beta}{\partial y^j} \mathcal{U}^j \gamma_\alpha \, dV, \tag{B.25}$$

$$A_{ik} \quad += \quad \int \rho \left\{ \begin{matrix} i \\ kj \end{matrix} \right\} \gamma_\beta \mathcal{U}^j \gamma_\alpha \, dV. \tag{B.26}$$

We will also add the continuity equation in the matrix as

$$A_{4i} \quad += \quad \int \frac{\partial \gamma_\beta}{\partial x^i} \gamma_\alpha \, dV, \tag{B.27}$$

$$A_{4j} \quad += \quad \int \left\{ \begin{matrix} i \\ ji \end{matrix} \right\} \gamma_\beta \gamma_\alpha \, dV. \tag{B.28}$$

The force F is

$$F_i \quad = \quad \int \rho f^i \gamma_\alpha \, dV, \tag{B.29}$$

$$F_c \quad = \quad 0. \tag{B.30}$$

When the Newton linearization for the convection terms

$$u^i{}_{,j}\, u^j = u^i{}_{,j}\, \mathcal{U}^j + \mathcal{U}^i{}_{,j}\, u^j - \mathcal{U}^i{}_{,j}\, \mathcal{U}^j, \tag{B.31}$$

is used instead of the Picard linearization, we have a few additional terms

$$A_{ij} \quad += \quad \int \rho \left(\frac{\partial \mathcal{U}^i}{\partial x^j} + \left\{ \begin{matrix} i \\ kj \end{matrix} \right\} \mathcal{U}^k \right) \gamma_\beta \gamma_\alpha \, dV \tag{B.32}$$

and

$$F_i \quad += \quad \int \rho \left(\frac{\partial \mathcal{U}^i}{\partial x^j} + \left\{ \begin{matrix} i \\ kj \end{matrix} \right\} \mathcal{U}^k \right) \mathcal{U}^j \gamma_\alpha \, dV. \tag{B.33}$$

B.1.3 Stabilization

Stabilization given in Franca et.al., can be written in tensor form as (to be added to the variational formulation)

$$\sum_e \int \tau R(u, \mathcal{U}, p, t) \left(-w_{c,i} - \mu g^{jk} (w_{i,jk} + w_{j,ik}) + \rho w_{i,j} \mathcal{U}^j \right) \, dV, \tag{B.34}$$

where $R(u, \mathcal{U}, p, t)$ is the residual of the Navier-Stokes equations

$$R(u, \mathcal{U}, p, t) = \rho \frac{\partial u^i}{\partial t} + g^{ij} p_{,j} - \mu (g^{jk} u^i{}_{,jk} + g^{ik} u^j{}_{,jk}) + \rho u^i{}_{,j} \mathcal{U}^j - \rho f^i. \tag{B.35}$$

The sum above is over the elements, and integral over element interiors. Continuity equation gives an additional term

$$\int \delta u^i{}_{,i}\, g^{ij} w_{j,i} \, dV. \tag{B.36}$$

The stabilization parameters τ and δ are given as in the reference:

$$\delta = \rho ||\mathcal{U}|| h_K Re_x, \tag{B.37}$$

$$\tau = \frac{h_K}{2\rho ||\mathcal{U}||} Re_x, \tag{B.38}$$

$$Re_x = \min \left(\frac{\rho m_K h_K ||\mathcal{U}||}{4\mu}, 1 \right). \tag{B.39}$$

The parameter h_K is determined by size of the element and m_K by the interpolation degree.

B.1.4 Discretization

Lets first write the residual (without the time derivate term) in a form, where the variables (u^i, p) have been separated from each other and making the interpolation substitutions at the same time

$$
\begin{aligned}
U_i &= -\mu g^{jk}\left(\frac{\partial^2\gamma_\beta}{\partial x^j\partial x^k} - \left\{ {l \atop jk} \right\}\frac{\partial\gamma_\beta}{\partial x^l}\right) + \rho\frac{\partial\gamma_\beta}{\partial x^j}\mathcal{U}^j, && \text{(B.40)}\\
U_j &= -\mu g^{ik}\left(\frac{\partial^2\gamma_\beta}{\partial x^j\partial x^k} - \left\{ {l \atop jk} \right\}\frac{\partial\gamma_\beta}{\partial x^l}\right), && \text{(B.41)}\\
U_k &= \rho\left\{ {i \atop kj} \right\}\gamma_\beta\mathcal{U}^j, && \text{(B.42)}\\
U_l &= -\mu\left(g^{jk}\left\{ {i \atop lj} \right\} + g^{ik}\left\{ {j \atop lj} \right\}\right)\frac{\partial\gamma_\beta}{\partial x^k} - && \text{(B.43)}\\
&\quad \mu\left(g^{jk}\left\{ {i \atop lk} \right\} + g^{ik}\left\{ {j \atop lk} \right\}\right)\frac{\partial\gamma_\beta}{\partial x^j} - && \text{(B.44)}\\
&\quad \mu\left(g^{jk}\left(\frac{\partial\left\{ {i \atop lj} \right\}}{\partial x^k} + \left\{ {i \atop mk} \right\}\left\{ {m \atop lj} \right\} - \left\{ {m \atop jk} \right\}\left\{ {i \atop lm} \right\}\right)\right. + && \text{(B.45)}\\
&\quad \left. g^{ik}\left(\frac{\partial\left\{ {j \atop lj} \right\}}{\partial x^k} + \left\{ {j \atop mk} \right\}\left\{ {m \atop lj} \right\} - \left\{ {m \atop jk} \right\}\left\{ {j \atop lm} \right\}\right)\right)\gamma_\beta, && \text{(B.46)}\\
U_4 &= g^{ij}\frac{\partial\gamma_\beta}{\partial x^j}. && \text{(B.47)}
\end{aligned}
$$

If we are using Newton linearization for the convection term, there is an additional term

$$
U_j \quad += \quad \left(\frac{\partial\mathcal{U}^i}{\partial x^j} + \left\{ {i \atop kj} \right\}\mathcal{U}^k\right)\gamma_\beta. \tag{B.48}
$$

There is also a term to be added to the stabilization force term in this case, this is done below.

Ok, now we'll do the same for the weight function terms:

$$
\begin{aligned}
W_i &= -\mu g^{jk}\left(\frac{\partial^2\gamma_\alpha}{\partial x^j\partial x^k} - \left\{ {l \atop jk} \right\}\frac{\partial\gamma_\alpha}{\partial x^l}\right) + \rho\frac{\partial\gamma_\alpha}{\partial x^j}\mathcal{U}^j, && \text{(B.49)}\\
W_j &= -\mu g^{jk}\left(\frac{\partial^2\gamma_\alpha}{\partial x^j\partial x^k} - \left\{ {l \atop ik} \right\}\frac{\partial\gamma_\alpha}{\partial x^l}\right), && \text{(B.50)}\\
W_k &= \rho\left\{ {k \atop ij} \right\}\gamma_\alpha\mathcal{U}^j, && \text{(B.51)}\\
W_l &= \mu g^{jk}2\left\{ {l \atop ij} \right\}\frac{\partial\gamma_\alpha}{\partial x^k} + \mu g^{jk}\left\{ {l \atop ik} \right\}\frac{\partial\gamma_\alpha}{\partial x^j} + \mu g^{jk}\left\{ {l \atop jk} \right\}\frac{\partial\gamma_\alpha}{\partial x^i} + && \text{(B.52)}\\
&\quad \mu g^{jk}\left(2\frac{\partial\left\{ {l \atop ij} \right\}}{\partial x^k} + \left\{ {m \atop ik} \right\}\left\{ {l \atop mj} \right\} + \left\{ {m \atop jk} \right\}\left\{ {l \atop mi} \right\} + \right. && \text{(B.53)}\\
&\quad \left.\left\{ {m \atop jk} \right\}\left\{ {l \atop mi} \right\} + \left\{ {m \atop ik} \right\}\left\{ {l \atop jm} \right\}\right)\gamma_\alpha, && \text{(B.54)}\\
W_4 &= \frac{\partial\gamma_\alpha}{\partial x^i}. && \text{(B.55)}
\end{aligned}
$$

Using the above notations we can write the out the matrices. Note that no loop is indicated by the indices in W_Γ and U_Θ ($\Gamma = \Theta = i, j, k, l, 4$) below, but these are to be substituted to the equations as written before any further processing. First we'll write the M^s matrix as

$$M^s_{\Gamma i} \quad += \quad \int \tau\rho\gamma_\beta W_\Gamma \; dV, \tag{B.56}$$

the A^s matrix as

$$\begin{aligned}
A^s_{\Gamma\Theta} \quad &+= \quad \int \tau U_\Theta W_\Gamma \; dV, && \text{(B.57)}\\
A^s_{ji} \quad &+= \quad \int \delta g^{ij} \frac{\partial\gamma_\beta}{\partial x^i}\frac{\partial\gamma_\alpha}{\partial x^j} \; dV, && \text{(B.58)}\\
A^s_{jk} \quad &+= \quad \int \delta g^{ij} \left\{ {i \atop ki} \right\} \gamma_\beta \frac{\partial\gamma_\alpha}{\partial x^j} \; dV, && \text{(B.59)}\\
A^s_{ki} \quad &-= \quad \int \delta g^{ij} \frac{\partial\gamma_\beta}{\partial x^i} \left\{ {k \atop ij} \right\} \gamma_\alpha \; dV, && \text{(B.60)}\\
A^s_{kk} \quad &-= \quad \int \delta g^{ij} \left\{ {i \atop ki} \right\} \gamma_\beta \left\{ {k \atop ij} \right\} \gamma_\alpha \; dV. && \text{(B.61)}
\end{aligned}$$

Stabilization for the force vector can be written as

$$F^s_\Gamma \quad += \quad \int \tau\rho W_\Gamma \; dV. \tag{B.62}$$

and in addition to the previous term, when using Newton linearization

$$F^s_\Gamma \quad += \quad \int \tau\rho \left(\frac{\partial \mathcal{U}^i}{\partial x^j} + \left\{ {i \atop kj} \right\} \mathcal{U}^k \right) \mathcal{U}^j W_\Gamma \; dV \tag{B.63}$$

B.2 Computing the Second Derivates

In the stabilization terms of the Navier-Stokes equations we need second partial derivates of quantities with respect to the global coordinates. The computation proceeds in the following manner. Let the space coordinates of the element be given

$$x^i = x^i(u^1, u^2), \quad i = 1, \ldots, 3. \tag{B.64}$$

Let also the quantity f be given at element nodes, whereupon it may be expressed in the same way as the coordinates

$$f = f(u^1, u^2). \tag{B.65}$$

What we need is the matrix of second partial derivates of f with respect to space coordinates x^i

$$\frac{\partial^2 f}{\partial x^i \partial x^j}. \tag{B.66}$$

To this end we will first compute the second *covariant* derivates of the quantity f with respect to the element local coordinates:

$$f_{,\alpha\beta} = \frac{\partial^2 f}{\partial u^\alpha \partial u^\beta} - \left\{ \begin{matrix} \gamma \\ \alpha\beta \end{matrix} \right\} \frac{\partial f}{\partial u^\gamma} = h_{\alpha\beta}. \tag{B.67}$$

This second rank tensor may be converted to contravariant base by

$$h^{\alpha\beta} = g^{\alpha\gamma} g^{\beta\kappa} h_{\gamma\kappa} \tag{B.68}$$

As you may recall (Appendix A), the contravariant rule of transformation is

$$h^{ij} = \frac{\partial x^i}{\partial u^\alpha} \frac{\partial x^j}{\partial u^\beta} h^{\alpha\beta}. \tag{B.69}$$

If the element coordinate system metric, and thus also the Christoffel symbols, are with respect to Cartesian coordinates, the values h^{ij} contain the second partial derivates of the function f with respect to the space coordinates x^i (not necesserily Cartesian).

B.3 The Scalar Diffusion-Convection Equation

The scalar diffusion-convection equation written in tensor form, allowing anisotropic diffusion (heat conduction, for example), reads

$$C_t \frac{\partial T}{\partial t} + C_0 T + C_1 u^i T_{,i} - \left(C_2^{ij} T_{,j} \right)_{,i} = h \tag{B.70}$$

Next we will derive the variational formulation and discretize the equations.

B.3.1 Variational Formulation

We can write down the variational form of the previous equation (B.70) as

$$\int \left(C_t \frac{\partial T}{\partial t} + C_0 T + C_1 u^i T_{,i} - \left(C_2^{ij} T_{,j} \right)_{,i} \right) w \; dV = \int hw \; dV. \tag{B.71}$$

Integrating the diffusion term by parts, we get

$$\int \left(C_t \frac{\partial T}{\partial t} + C_0 T + C_1 u^i T_{,i} \right) w \; dV + \int C_2^{ij} T_{,j} w_{,i} \; dV = \int C_2^{ij} T_{,j} n_i \; d\Gamma + \int hw \; dV \tag{B.72}$$

B.3.2 Discretization of the Variational Formulation

Using the same notation as for the Navier-Stokes equations, we get the mass matrix

$$M = \int C_t \gamma_\beta \gamma_\alpha \; dV. \tag{B.73}$$

Lets describe the A matrix in a procedural manner, setting the elements of the matrix to zero initially. Then we can first add the term proportional to the zero order derivate

$$A+=\int C_0\gamma_\beta\gamma_\alpha \, dV, \tag{B.74}$$

secondly we add the diffusive terms

$$A+=\int C_2^{ij}\frac{\partial\gamma_\beta}{\partial y^i}\frac{\partial\gamma_\alpha}{\partial y^j} \, dV. \tag{B.75}$$

For the convective terms we have

$$A+=\int C_1u^i\frac{\partial\gamma_\beta}{\partial y^i}\gamma_\alpha \, dV. \tag{B.76}$$

B.3.3 Stabilization

Stabilization given in Franca et.al. may be written as

$$\sum_e\int \tau R(T,t)\left(-C_0w+C_1u^iw_{,i}-(C_2^{ij}w_{,j})_{,i}\right), \tag{B.77}$$

where $R(T,t)$ is the residual of the diffusion-convection equation

$$R(T,t)=C_t\frac{\partial T}{\partial t}+C_0T+C_1u^iT_{,i}-\left(C_2^{ij}T_{,j}\right)_{,i}-h \tag{B.78}$$

The sum above is over the elements, and the integral over element interiors. The stabilization parameter τ is given as in reference:

$$\tau \quad = \quad \frac{h_K}{2C_1||\mathcal{U}||}Pe, \tag{B.79}$$

where

$$Pe \quad = \quad \min\left(\frac{m_Kh_KC_1||\mathcal{U}||}{2C_2},1\right) \tag{B.80}$$

The parameter h_K is determined by size of the element and m_K by the interpolation degree.

B.3.4 Discretization

Lets first write the residual (without the time derivate term)

$$U \mathrel{+}= C_0\gamma_\beta, \tag{B.81}$$

$$U \mathrel{+}= C_1 u^i \frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.82}$$

$$U \mathrel{-}= C_2^{ij}\frac{\partial^2\gamma_\beta}{\partial x^i \partial x^j}, \tag{B.83}$$

$$U \mathrel{-}= \frac{\partial C_2^{ij}}{\partial x^j}\frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.84}$$

$$U \mathrel{+}= C_2^{ij}\left\{ \begin{array}{c} k \\ ij \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^k}, \tag{B.85}$$

$$U \mathrel{-}= C_2^{ik}\left\{ \begin{array}{c} j \\ jk \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.86}$$

$$U \mathrel{-}= C_2^{kj}\left\{ \begin{array}{c} i \\ jk \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^i}. \tag{B.87}$$

Ok, now we'll do the same for the weight function terms

$$W \mathrel{-}= C_0\gamma_\beta, \tag{B.88}$$

$$W \mathrel{+}= C_1 u^i \frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.89}$$

$$W \mathrel{-}= C_2^{ij}\frac{\partial^2\gamma_\beta}{\partial x^i \partial x^j}, \tag{B.90}$$

$$W \mathrel{-}= \frac{\partial C_2^{ij}}{\partial x^j}\frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.91}$$

$$W \mathrel{+}= C_2^{ij}\left\{ \begin{array}{c} k \\ ij \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^k}, \tag{B.92}$$

$$W \mathrel{-}= C_2^{ik}\left\{ \begin{array}{c} j \\ jk \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^i}, \tag{B.93}$$

$$W \mathrel{-}= C_2^{kj}\left\{ \begin{array}{c} i \\ jk \end{array} \right\}\frac{\partial\gamma_\beta}{\partial x^i}. \tag{B.94}$$

Using the above equations, stabilization term for the mass matrix is then

$$M^s = \int \tau C_t \gamma_\alpha W \; dV, \tag{B.95}$$

for the A matrix

$$A^s = \int \tau U W \; dV, \tag{B.96}$$

and for the force vector

$$F^s = \int \tau \gamma_\alpha W \; dV. \tag{B.97}$$

B.4 The Compressible Navier-Stokes Equations

We have already introduced the general Navier-Stokes equations

$$\rho \frac{\partial u^i}{\partial t} + \rho u^i{}_{,j}\, u^j - \sigma^{ij}{}_{,j} = \rho f^i \tag{B.98}$$

which apply also to the compressible case, if the (Newtonian) stress tensor is defined to contain an extra term as follows

$$\sigma^{ij} = -p g^{ij} + 2\mu\varepsilon^{ij} - \frac{2}{3}\mu u^k{}_{,k}\, g^{ij} \tag{B.99}$$

and the density is non-constant, following some state law. We will assume here the ideal gas law

$$\rho = \frac{p}{RT}. \tag{B.100}$$

The continuity equation becomes

$$\rho u^i{}_{,i} + u^i \rho_{,i} = 0. \tag{B.101}$$

The compressible flow is usually couple to the heat equation, which becomes in the compressible case

$$\rho c_v \left(\frac{\partial T}{\partial t} + u^i T_{,i} \right) + q^i{}_{,i} + T \left(\frac{\partial p}{\partial T} \right)_V u^i{}_{,i} + \tau^{ij} u_{i,j} = h \tag{B.102}$$

where τ^{ij} is the viscous part of stress tensor (check contraction!), and for ideal gas

$$c_v = c_p - R \tag{B.103}$$

and the expansion term is

$$T \left(\frac{\partial p}{\partial T} \right)_V u^i{}_{,i} = p u^i{}_{,i}\, . \tag{B.104}$$

According to Fourier's law

$$q^i = k^{ij} T_{,j} \tag{B.105}$$

B.5 The Elastic Stress Equations

The starting point for these equations is exactly the same as for the Navier-Stokes equations

$$\sigma^{ij}_{,j} = \rho(a^i - f^i). \tag{B.106}$$

The stress-strain relations naturally differ somewhat from the flow equations. For elastic isotropic material (allowing anisotropy of the heat expansion though) we have

$$\sigma^{ij} = g^{ij}\lambda d^k_{,k} + 2\mu\varepsilon^{ij} - g^{ij}\lambda\beta^{kk}(T - T_0) - 2\mu\beta^{ij}(T - T_0) \tag{B.107}$$

where λ and μ are the Lamé parameters, ε^{ij} the contravariant form of the linear strain tensor

$$\varepsilon^{ij} = \frac{1}{2}\left(g^{jk} d^i{}_{,k} + g^{ik} d^j{}_{,k}\right), \tag{B.108}$$

and β^{ij} the heat expansion tensor. For isotropic heat expansion, we may take

$$\beta^{ij} = g^{ij}\beta_0, \tag{B.109}$$

where β_0 is the scalar heat expansion coefficient.

Assuming that we may discard the acceleration a and taking T as a known quantity we may write down the elastic stress equations:

$$-g^{ij}\left(\lambda d^k_{,k}\right)_{,j} - (2\mu\varepsilon^{ij})_{,j} = -g^{ij}(\lambda\beta^{kk}(T - T_0))_{,j} - (2\mu\beta^{ij}(T - T_0))_{,j} + F^i \tag{B.110}$$

B.5.1 Variational Formulation

The variational formulation of the equation (B.110) reads

$$\int \left(-\lambda d^k_{,k} - \mu(g^{jk} d^i{}_{,k} + g^{ik} d^j{}_{,k})_{,j}\right) w_i \, dV = \tag{B.111}$$

$$\int \left(g^{ij}(\lambda\beta^{kk}(T - T_0)) + 2\mu\beta^{ij}(T - T_0)\right)_{,j} w_i \, dV + \int \rho f^i w_i \, dV. \tag{B.112}$$

Integrating by parts again

$$-\int \lambda d^k_{,k} w_i \, dV + \int \left(\mu(g^{jk} d^i{}_{,k} + g^{ik} d^j{}_{,k})\right) w_{i,j} \, dV = \tag{B.113}$$

$$\int \left(-g^{ij}\lambda\beta^{kk}(T - T_0) - 2\mu\beta^{ij}(T - T_0)\right) w_{i,j} \, dV + \tag{B.114}$$

$$\int \sigma^{ij} n_j w_i \, d\Gamma + \int f^i w_i \, dV. \tag{B.115}$$

B.5.2 Discretization of the Variational Formulation

Again using the same notation as in previous sections:

$$A_{ik} \quad += \quad \int \lambda g^{ij} \frac{\partial \gamma_\beta}{\partial y^k} \frac{\partial \gamma_\alpha}{y^j} \, dV, \tag{B.116}$$

$$A_{ii} \quad += \quad \int \mu g^{jk} \frac{\partial \gamma_\beta}{\partial y^k} \frac{\partial \gamma_\alpha}{y^j} \, dV, \tag{B.117}$$

$$A_{ij} \quad += \quad \int \mu g^{ik} \frac{\partial \gamma_\beta}{\partial y^k} \frac{\partial \gamma_\alpha}{y^j} \, dV, \tag{B.118}$$

$$A_{il} \quad += \quad \int \lambda g^{ij} \left\{ \begin{array}{c} k \\ kl \end{array} \right\} \gamma_\beta \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.119}$$

$$A_{il} \quad += \quad \int \mu g^{jk} \left\{ \begin{array}{c} i \\ kl \end{array} \right\} \gamma_\beta \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.120}$$

$$A_{il} \quad += \quad \int \mu g^{ik} \left\{ \begin{array}{c} j \\ kl \end{array} \right\} \gamma_\beta \frac{\partial \gamma_\alpha}{\partial y^j} \, dV, \tag{B.121}$$

$$A_{li} \quad += \quad \int \mu g^{jk} \frac{\partial \gamma_\beta}{\partial y^k} \left\{ \begin{array}{c} l \\ ij \end{array} \right\} \gamma_\alpha \, dV, \tag{B.122}$$

$$A_{lj} \quad += \quad \int \mu g^{ik} \frac{\partial \gamma_\beta}{\partial y^k} \left\{ \begin{array}{c} l \\ ij \end{array} \right\} \gamma_\alpha \, dV, \tag{B.123}$$

$$A_{lm} \quad += \quad \int \mu g^{jk} \left\{ \begin{array}{c} i \\ km \end{array} \right\} \gamma_\beta \left\{ \begin{array}{c} l \\ ij \end{array} \right\} \gamma_\alpha \, dV, \tag{B.124}$$

$$A_{lm} \quad += \quad \int \mu g^{ik} \left\{ \begin{array}{c} j \\ km \end{array} \right\} \gamma_\beta \left\{ \begin{array}{c} l \\ ij \end{array} \right\} \gamma_\alpha \, dV. \tag{B.125}$$

B.6 The Magnetic Induction Equation

Magnetohydrodynamics (MHD) is an approximation to the behaviour of ionized plasma under the influence of magnetic fields in moderate energy scales and relatively high densities. The presence of applied and induced magnetic fields in a flow of plasma will result in an additional force for the flow momentum equations, the Lorenz force, and also in an additional heat source for the heat equation. In the following we present the underlying equations.

The macroscopic Maxwell equations may be written as

$$\begin{aligned} D^i_{,i} &= \rho, & \text{(B.126)}\\ \epsilon^{ijk} H_{k,j} &= \frac{\partial D^i}{\partial t} + J^i, & \text{(B.127)}\\ \frac{\partial B^i}{\partial t} + \epsilon^{ijk} E_{k,j} &= 0, & \text{(B.128)}\\ B^i_{,i} &= 0, & \text{(B.129)} \end{aligned}$$

where $\mathbf{D}$ is the displacement current, $\mathbf{H}$ is the magnetic field, $\mathbf{B}$ is the magnetic induction, $\mathbf{E}$ is the electric field, $\mathbf{J}$ is the current density, and ρ the charge density. It is usually assumed (and we also assume), that the simple form of Ohms law holds

$$\mathbf{J} = \sigma \mathbf{E}, \tag{B.130}$$

where σ is the conductivity of the plasma. In laboratory frame this may be written as

$$J^i = \sigma \left(E^i + \epsilon^{ijk} u_j B_k \right). \tag{B.131}$$

What is still missing is the description of the relationship of the fields $\mathbf{E}$ and $\mathbf{B}$, and $\mathbf{D}$ and $\mathbf{H}$ respectively. We assume the simplest model

$$\mathbf{D} = \epsilon_0 \mathbf{E}, \tag{B.132}$$

$$\mathbf{H} = \frac{1}{\mu_0} \mathbf{B}, \tag{B.133}$$

where the parameters ϵ_0 and μ_0 are the electrical permittivity and magnetic permeability of free space respectively.

Next in our series of approximations is to ignore the time derivate of the displacement current $\mathbf{D}$ entirely. To see how this might be justified, lets take a look at the values of the fields. From the value of ϵ_0 for free space

$$\epsilon_0 = \frac{10^7}{4\pi c^2} \approx 10^{-11} \tag{B.134}$$

we see that the $\mathbf{E}$ field is much larger in value than the $\mathbf{D}$ field, so that in a numerical simulation we have $\mathbf{J} \gg \partial \mathbf{D}/\partial t$ in the time scales and velocities of interest in magnetohydrodynamics. The basic assumption is then, that the macroscopic collective velocities of charged particles are of the order of the fluid motion, and that the charge separation does not occur.

Using the above assumptions we may eliminate the electric field from the Maxwell equations to reach the *induction equation*

$$\frac{\partial B^i}{\partial t} - \epsilon^{ijk} g_{kl} \epsilon^{lmn} \left(u_m B_n\right)_{,j} + \frac{1}{\sigma\mu_0} \epsilon^{ijk} g_{kl} \epsilon^{lmn} B_{m,nj} = 0. \tag{B.135}$$

The force generated by the magnetic and electric fields for the flow momentum equations may be expressed as

$$F^i = \frac{1}{\mu_0}(\mathbf{J} \times \mathbf{B}) = \frac{1}{\mu_0} \epsilon^{ijk} J_j B_k, \tag{B.136}$$

and the Joule heating as

$$h = \frac{1}{\sigma\mu_0^2} \mathbf{J^2} = \frac{1}{\sigma\mu_0^2} J^i J_i. \tag{B.137}$$

The induction equation together with the Navier-Stokes and the heat equation with the above source terms are the equations of magnetohydrodynamics.

B.6.1 Discretization

For the purpose of discretizing the equation (B.135), we will first write it in a different form using the vector identities

$$\epsilon^{ijk} g_{kl} \epsilon^{lmn} B_{m,nj} = g^{ik} B^j_{,jk} - g^{jk} B^i_{,jk} \tag{B.138}$$

and

$$\epsilon^{ijk} g_{kl} \epsilon^{lmn} \left(u_m B_n\right)_{,j} = u^i B^j_{,j} - B^i u^j_{,j} + B^j u^i_{,j} - u^j B^i_{,j}. \tag{B.139}$$

From the Maxwell equations we see that

$$B^i_{,i} = 0, \tag{B.140}$$

so that the first of the equations becomes

$$\epsilon^{ijk} g_{kl} \epsilon^{lmn} B_{m,nj} = -g^{jk} B^i_{,jk}. \tag{B.141}$$

The induction equation may now be written as

$$\frac{\partial B^i}{\partial t} - u^i B^j_{,j} + B^i u^j_{,j} - B^j u^i_{,j} + u^j B^i_{,j} + \frac{1}{\sigma\mu_0} g^{jk} B^i_{,jk} = 0. \tag{B.142}$$

Multiplying by the weight functions w_i and integrating we obtain

$$\int \left(\frac{\partial B^i}{\partial t} - u^i B^j_{,j} + B^i u^j_{,j} - B^j u^i_{,j} + u^j B^i_{,j} + \frac{1}{\sigma\mu_0} g^{jk} B^i_{,jk} \right) w_i dV = 0. \tag{B.143}$$

Integrating the diffusion term by parts we get

$$\int \left(\frac{\partial B^i}{\partial t} - u^i B^j_{,j} + B^i u^j_{,j} - B^j u^i_{,j} + u^j B^i_{,j} \right) w_i dV - \tag{B.144}$$

$$\int \frac{1}{\sigma\mu_0} g^{jk} B^i_{,k} w_{i,j} dV = \int g^{jk} B^i_{,k} w_i n_j dV \tag{B.145}$$

Now we may write the discretized matrices as

$$
\begin{aligned}
A_{ij} &\ -= \int u_i \frac{\partial \gamma_\beta}{\partial x^j} \gamma_\alpha \, dV, \\
A_{ik} &\ -= \int u_i \left\{ \begin{array}{c} j \\ jk \end{array} \right\} \gamma_\beta \gamma_\alpha \, dV, \\
A_{ii} &\ += \int \gamma_\beta \left(\frac{\partial u^j}{\partial x^j} + \left\{ \begin{array}{c} j \\ jk \end{array} \right\} u^k \right) \gamma_\alpha \, dV, \\
A_{ij} &\ -= \int \gamma_\beta \left(\frac{\partial u^i}{\partial x^j} + \left\{ \begin{array}{c} i \\ jk \end{array} \right\} u^k \right) \gamma_\alpha \, dV, \\
A_{ii} &\ += \int u^j \frac{\partial \gamma_\alpha}{\partial x^j} \gamma_\beta \, dV, \\
A_{ik} &\ += \int u^j \left\{ \begin{array}{c} i \\ jk \end{array} \right\} \gamma_\alpha \gamma_\beta \, dV, \\
A_{ii} &\ -= \int \frac{1}{\sigma \mu_0} g^{jk} \frac{\partial \gamma_\alpha}{\partial x^k} \frac{\partial \gamma_\beta}{\partial x^j} \, dV, \\
A_{li} &\ += \int \frac{1}{\sigma \mu_0} g^{jk} \frac{\partial \gamma_\alpha}{\partial x^k} \left\{ \begin{array}{c} l \\ ij \end{array} \right\} \gamma_\beta \, dV, \\
A_{il} &\ -= \int \frac{1}{\sigma \mu_0} g^{jk} \left\{ \begin{array}{c} i \\ kl \end{array} \right\} \gamma_\alpha \frac{\partial \gamma_\beta}{\partial x^j} \, dV, \\
A_{ml} &\ -= \int \frac{1}{\sigma \mu_0} g^{jk} \left\{ \begin{array}{c} i \\ kl \end{array} \right\} \gamma_\alpha \left\{ \begin{array}{c} m \\ ij \end{array} \right\} \gamma_\beta \, dV.
\end{aligned}
$$

The mass matrix is the same as in other cases.

Appendix C

The Diffuse Gray Radiation Boundary Condition

We describe the diffuse gray radiation boundary condition in this appendix.

C.1 The Diffuse Gray Radiation Boundary Condition

On outer boundaries of a solid or liquid body, where the temperature is not fixed, the heat flux must be specified:

$$-k\frac{\partial T}{\partial n} = q. \tag{C.1}$$

The flux q may depend on temperatures of other boundaries of the system, if radiative exchange of heat is present. The model used here is the diffuse gray radiation model, which means that a surface radiates energy to every direction with equal intensity, and that the intensity is also independent of the wavelength of the radiation. Using these assumptions we may write the heat flux for a given point on a surface, resulting from the radiation, as

$$q(\vec{x}, T) = \sigma\left(\varepsilon(\vec{x})T^4(\vec{x}) - \int \varepsilon(\vec{y})G(\vec{y}, \vec{x})T^4(\vec{y})\; dA_y\right), \tag{C.2}$$

where σ is the Stefan-Bolzmann constant, $\varepsilon(\vec{x})$ is surface emissivity, and the integral is over all the surfaces of the model. The function G is the so called Gebhardt factor. The Gebhardt factor may be computed using the equation

$$G(\vec{y}, \vec{x}) - \int F(\vec{y}, \vec{x})(1 - \varepsilon(\vec{z}))G(\vec{z}, \vec{x})\; dA_z = F(\vec{y}, \vec{x})\varepsilon(\vec{x}), \tag{C.3}$$

where the function F is defined as follows

$$F(\vec{x}_i, \vec{x}_j) = \frac{\cos\phi_i \cos\phi_j}{\pi r^2} H_{ij}. \tag{C.4}$$

The angles are between the normals of the surfaces and the line connecting the points and H_{ij} the visibility. The function F is the so called view factor.

In practice, the surfaces of the model are divided to, say, total of N finite parts, the boundary elements. This discretization is then used to compute the view factors (refer to next section) and the Gebhardt factor for these parts. The Gebhardt factors may be computed from the equation

$$G = F(I - (I - E)F)^{-1}E, \tag{C.5}$$

where F is the view factor matrix, and E a diagonal matrix of surface emissivities.

Writing the discrete version of the boundary condition we get

$$-k_k \frac{\partial T_k}{\partial n_k} = \sigma \varepsilon_k \left(T_k^4 - \frac{1}{A_k \varepsilon_k} \sum_{i=1}^{N} G_{ik} \varepsilon_i T_i^4 A_i \right). \tag{C.6}$$

This equation is still nonlinear, and has to be linearized in order to solve the equation on a computer. To illustrate the point here, we will first show how to linearize the idealized radiation boundary condition instead of the diffuse gray radiation boundary condition. As you may recall, the idealized boundary condition reads

$$-k \frac{\partial T}{\partial n} = \sigma \varepsilon \left(T^4 - T_{ext}^4 \right). \tag{C.7}$$

We may write this somewhat differently as in

$$-k \frac{\partial T}{\partial n} = \sigma \varepsilon \left(T^3 + T^2 T_{ext} + T T_{ext}^2 + T_{ext}^3 \right) \left(T - T_{ext} \right). \tag{C.8}$$

If we now take $T = \mathcal{T}$, the temperature from the previous iteration, in first of the product terms, we have an expression linear in T. This is somewhat strange linearization, not quite a Newton method, but effective in being more insensitive to the initial guess of the temperature field than the Newton method, while still having a much better convergence rate than the fixed point method. The Newton linearization is also easy to produce by writing down the Taylor series expansion of the equation (C.7) around $\mathcal{T}$ and retaining only the constant terms and the terms linear in T:

$$-k \frac{\partial T}{\partial n} \approx \sigma \varepsilon \left(4\mathcal{T}^3 T - 3\mathcal{T}^4 - T_{ext}^4 \right). \tag{C.9}$$

This linearization has a better convergence rate than the previous one, but with the expense of being more sensitive to the initial guess. These two linearizations may be used in succession, first taking a few steps with the former linearization, and then switch to the latter when the convergence is on the way.

The linearization of the diffuse gray radiation boundary condition is very similiar. We may, for example, use the temperatures from the previous iteration to compute an 'external' temperature

$$T_{ext}^4 \approx \frac{1}{A_k \varepsilon_k} \sum_{i=1}^{N} G_{ik} \varepsilon_i \mathcal{T}_i^4 A_i, \tag{C.10}$$

and use either of the equations (C.8) or (C.9). The convergence rates of these methods are of first order, and usually they are used only to provide an initial guess for the full Newton iteration. The full Newton method may be written as

$$-k_k \frac{\partial T_k}{\partial n_k} \approx \sigma\varepsilon_k \left(4\mathcal{T}_k^3 T_k - 3\mathcal{T}_k^4 - \frac{1}{A_k\varepsilon_k} \sum_{i=1}^{N} G_{ik}\varepsilon_i \left(4\mathcal{T}_i^3 T_i - 3\mathcal{T}_i^4 \right) A_i \right). \quad \text{(C.11)}$$

One final note is due. The Gebhardt factors computed in the way presented here are assumed to be constant within elements, while the temperatures from the previous iteration are known at the element nodal points. Terms of type $\alpha G_{ik}\mathcal{T}_k^n$ are to be integrated over the element i. This is done by a one point integration rule and requires values of the powers of the temperature field at the element center. To conserve the heat flux, the powers of the temperature field will first have to be computed on nodal points, and only after that interpolated to the element center, rather than first interpolating the temperatures and computing the powers afterwards.

C.2 Computing the View Factors in 3D

A view factor between two surface patches (boundary elements) of a geometrical model (mesh) is defined as

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} H_{ij}\, dA_j\, dA_i, \quad \text{(C.12)}$$

where A_i and A_j are the areas of the patches i and j respectively, ϕ_i and ϕ_j the angles between the normals of the surfaces at given points and the line connecting the points and r the distance between the two points. The most problematic thing about the definition of the view factor integral (C.12) is the evaluation of the visibility function H_{ij}. The value of the function H_{ij} is one, if there are no obstacles between the two points, and zero otherwise. The view factors are computed for every pair of surface patches in the model, so that we eventually get a $N \times N$ matrix F.

Noting that

$$A_i F_{ij} = A_j F_{ji}, \quad \text{(C.13)}$$

allows only one half of the factors to be actually computed.

In three dimensional cases, the method the ELMER view factor computation program uses to evaluate the integral (C.12) is direct numerical integration combined with an adaptive subdivision procedure, where the elements are subdivided to smaller and smaller pieces, until user specified accuracy criteria are fullfilled. An area weighted average of the factors of the parts of the subdivided patches is then computed. The subdivision of patches is stored in hierarchical storage, so that it doesn't have to be repeated for every pair of patches. This subdivision is also used by the ray tracer, which is used for computing the visibility function. For quadratic and cubic elements the intersection computation is done with a similiar adaptive subdivision procedure combined with Newton iteration for finding the roots of the polynomial equations describing the intersection points of the rays and the surfaces.

First of the criteria to terminate the integration subdivision procedure is fullfilled when the areas of the subdivided elements are small enough. This criterion prevents the subdivision to continue forever in areas where the integrand is actually discontinuous (any sharp corner, for example). The second criterion to terminate the subdivision is fullfilled when the view factors themselves have become small enough. This criterion prevents unnecessery subdivision of areas which are either far away from each other or oriented so that the projections of their areas to each other are small.

The function H_{ij} is evaluated by sending a number of rays from one of the patches to another, and checking whether they hit some other patch of the model before hitting the target. This is potentially a very time consuming task, as in the worst case, for every pair of boundary elements, all the other boundary elements of the model must be checked for obstructing the view between the pair. The ray tracer module is designed to reduce the time spent in computing the intersections, by making a hierarchical volume subdivision of the model volume and assigning each element of the model to some of these subdivided volumes in a fixed hierarchy level. The ray tracer is then able to check whether the ray will hit the bounding boxes of the volumes, then the second level of bounding boxes, etc. before eventually going through the elements assigned to a volume in the last hierarchy level. There might be only a couple of elements assigned to that volume.

When the visibility between two surface patches is in doubt (some of the rays are obstructed, some not) an additional subdivision cycle is made. When the subdivision is eventually terminated, H_{ij} is computed as

$$H_{ij} = 1 - n_b/n, \tag{C.14}$$

where n_b is the number of the rays blocked and n the total number of rays sent.

C.3 Normalizing the View Factors

The view factors computed by, for example, the kind of procedure described in the previous section, are only approximate. For the radiative exchange of heat in finite element computations, it is sometimes important to guarantee that energy is conserved. The energy conservation requires, that in a closed system

$$\sum_{j=1}^{N} F_{ij} = 1. \tag{C.15}$$

Together with the equation (C.13) this condition may be used to normalize the factors so that energy will be conserved.

Lets begin with defining a symmetric matrix

$$S_{ij} = \frac{1}{2}(A_i F_{ij} + A_j F_{ji}). \tag{C.16}$$

This matrix we can relatively easily scale so that the row sums (and as we should preserve the symmetry, also column sums) of the matrix are equal to corresponding surface patch areas, by solving a set of equations:

$$\sum_{j=1}^{N} d_i d_j S_{ij} = A_i, \quad i = 1, ..., N \tag{C.17}$$

where the d_i:s are scale factors. The equation set is quadratic in d_i:s and may be solved by Newton iteration. The resulting linear system is diagonally dominant and very stable, so there should be no problems in solving the equation.

After the scaling is known we may compute the normalized factors by

$$F_{ij} = d_i d_j S_{ij} / A_i \tag{C.18}$$

Appendix D

Free surfaces

In this appendix a brief introduction to the methods used in tracking of a free surface is given. The methods may be used to track a free surface between two non-mixing liquids or on a liquid-gas interface. Note however, that ELMER at present doesn't include the compressible flow equations. Currently the free surface model can not be used in a transient simualtion.

D.1 The Free Surface Boundary Condition

On a free surface we must set two boundary conditions:

$$\overline{\overline{\sigma}} \cdot \vec{n} = \vec{f} \tag{D.1}$$

and

$$(\vec{u} \cdot \vec{n}) = 0. \tag{D.2}$$

The latter is to ensure that there is no flow across the boundary (i.e. ensure that the boundary is a boundary...). As for the former, refer to chapter 1.

D.2 Computing the Mean Curvature

In the specification of the normal component of the force on the boundary we have the mean curvature ($2H = 1/R$, Chapter 1). The definition of the mean curvature has been given in appendix A. The computation of the mean curvature requires second derivates of the boundary surface coordinates at the mesh nodes. This is a problem because the derivates computed from the piecewise polynomial basis functions of the elements may differ on the element-element boundaries. Also, for the linear basis functions, the second derivates are zero. For these reasons, the computation of the derivates on the nodes of the elements is done by computing an average of the elementwise derivates. The second derivates are computed by differentiating the first derivates.

Another approach to computing the derivates on nodes of the surfaces might be to fit the surface to a twice differentiable function (cubic spline in 2D, for example), and compute the derivates by differentiating this function. This approach may be used in future.

D.3 The Surface Update

The free surface is tracked by minimizing the equation

$$\int_S (\vec{u} \cdot \vec{n})^2 dS \tag{D.3}$$

on the free boundary. This is done with an iterative process, where the flow field is first solved, the free boundary tracked, the flow field solved again using the new position of the surface, etc.

The tracking of the surface within an iteration step is done by moving nodes on the free surface using an equation of type

$$y^{i+1} = y^i + \alpha(\vec{u}^i \cdot \vec{n}^i), \tag{D.4}$$

where α is to be determined, for example, so that the surface for the next iteration step will be lined with the velocity field of the current iteration step, i.e.

$$(\vec{n}^{i+i} \cdot \vec{u}^i) = 0. \tag{D.5}$$

In two dimensions the update of components of the normal vector may then be written as

$$n_x^{i+1} = n_x^i - \frac{\partial}{\partial u}\alpha(\vec{n}^i \cdot \vec{u}^i), \tag{D.6}$$

$$n_y^{i+1} = n_y^i. \tag{D.7}$$

Using the basis function approximation to the coordinate y, we may write the equation for the x component of the normal vector as

$$n_x^{i+1} = n_x^i - \frac{\partial}{\partial u}\sum_j (y_j^{i+1} - y_j^i)\gamma_j \tag{D.8}$$

$$= n_x^i - (y^{i+1} - y^i)\frac{\partial \gamma_n}{\partial u}, \tag{D.9}$$

where n is the index of the node, within the element, we are moving (it is implicit, that the y coordinate referred above is also the coordinate of the node in question and that the normal vector we are referring to is the normal vector of the surface at this point; we are moving only one node at the time). Using the equation D.5 the coordinate update is then

$$y^{i+1} = y^i + \frac{1}{u_x \frac{\partial \gamma_n}{\partial u}}(\vec{u}^i \cdot \vec{n}^i). \tag{D.10}$$

In three dimensions the process is the same. Supposing that we are moving the surface in the z coordinate direction, we may write down the update of the normal vector as

$$n_x^{i+1} = n_x^i + \frac{\partial y}{\partial u}\frac{\partial}{\partial v}\alpha(\vec{u}^i \cdot \vec{n}^i) - \frac{\partial y}{\partial v}\frac{\partial}{\partial u}\alpha(\vec{u}^i \cdot \vec{n}^i), \tag{D.11}$$

$$n_y^{i+1} = n_y^i - \frac{\partial x}{\partial u}\frac{\partial}{\partial v}\alpha(\vec{u}^i \cdot \vec{n}^i) + \frac{\partial x}{\partial v}\frac{\partial}{\partial u}\alpha(\vec{u}^i \cdot \vec{n}^i), \tag{D.12}$$

$$n_z^{i+1} = n_z^i. \tag{D.13}$$

Again using the basis function approximation to the z coordinate we get

$$z^{i+1} = z^i - \frac{(\vec{u}^i \cdot \vec{n}^i)}{\left(\frac{\partial y}{\partial u}\frac{\partial \gamma_n}{\partial v} - \frac{\partial y}{\partial v}\frac{\partial \gamma_n}{\partial u}\right) u_x + \left(\frac{\partial x}{\partial v}\frac{\partial \gamma_n}{\partial u} - \frac{\partial x}{\partial u}\frac{\partial \gamma_n}{\partial v}\right) u_y}. \tag{D.14}$$

D.4 The Mesh Update

After the surface update is done the mesh will have to be redefined, because moving the surface nodes might result in bad or degenerate or even overlapping elements. The mesh update is done by solving the equation

$$\nabla^2 y = 0, \tag{D.15}$$

where y is the coordinate whose nodes we moved using the method described in the previous section. Dirichlet boundary conditions for the boundary coordinates of fixed boundaries (and the free surface) should be set for this equation. The equation is solved using the mesh before the update of the free surface coordinates.

The diffusion equation will distribute the coordinates evenly.

Appendix E

HUTIter library

E.1 Introduction

E.1.1 General

A very general problem in linear algebra is, like in this presentation, how to solve the systems of linear equations

$$Ax = b \tag{E.1}$$

In order to have a unique solution for the equation (E.1) the *coefficient matrix* A has to be of size $n \times n$ and *non-singular*. Vectors x and b are of size $n \times 1$.

When A is formed by discretizing partial differential equations (PDEs) the structure of A is often very sparse. The ability to take advantage of this fact is very beneficial in computer based solutions.

The methods available to solve the equation (E.1) can be divided into *direct methods* and *iterative methods*. The behaviour and robustness of direct methods is predictable. This is not the case with different iterative methods [9]. There are different kind of iterative methods and some of them have many variants. The usefulness of some method depends both on the structure and characteristics of the coefficent matrix.

When the structure of the matrix A is not known then the direct methods are preferred. Unfortunately using a direct method can be very time consuming and the needed resources in a computer can be vast. If the structure of A is exploited carefully some iterative method can be very efficient. This is the key reason for their success.

In modern scientific computing parallelism is coming more and more important. That setting will pose totally new requirements for the algorithms used. The set of basic operations between vectors and matrices in iterative methods makes them much easier to parallelize.

Iterative methods are divided into two categories: *stationary* and *non-stationary*. Examples of stationary methods are *Jacobi* and *Gauss-Seidel* iterations. Non-stationary methods include Conjugate Gradients (*CG*) and variants (*CGS*) and methods based on Minimal Residuals (*GMRES,QMR*).

HUTI is an effort to make an effective and well structured library containing a collection of iterative methods. The methods implemented in the library are:

- Conjugate Gradient (CG) [3]
- Conjugate Gradient Squared (CGS) [3]
- Bi-Conjugate Gradient Stabilized (Bi-CGSTAB) [3]
- Bi-Conjugate Gradient Stabilized (2) (Bi-CGSTAB(2))
- Quasi-Minimal Residual (QMR) [3, 4, 7, 8]
- Transpose-Free Quasi-Minimal Residual (TFQMR) [6]
- Generalized Minimum Residual (GMRES) [3, 9]

This library supports both serial and parallel execution. Parallelisation has been made in a distributed memory environment using message passing as a communication method between processes. User has the same interface into the library for both of the execution models.

The name HUTI comes from Helsinki University of Technology (HUT) and Iterative solvers.

E.2 Using HUTI

E.2.1 Overall Structure

The key idea in HUTI is that all iterative solvers have the same calling conventions regardless of the selected method. All matrix related operations are done externally from the iterator library. This means that the solver doesn't need to know the exact matrix structure. Matrix can be stored for example in well-known Compressed Row Storage (CRS) or Compressed Comlumn Storage (CCS) formats. This eases the optimization of memory usage in each particular case.

In parallel setting it is on users responsibility to define the storage convention for the distribution of matrices and vectors. Well-known distribution concepts are for example block-cyclic decomposition and domain based decompositions. More information can be found from [10, 9].

The core routines (solvers) are written in pseudocode style using Fortran90 language. All the different precision and machine dependent parts are defined using preprocessor and generated from one source code. The most valuable advantage is the readability and possibility to implement new methods and their variants very quickly.

The needed layers above and below HUTI are presented in Figure E.1.

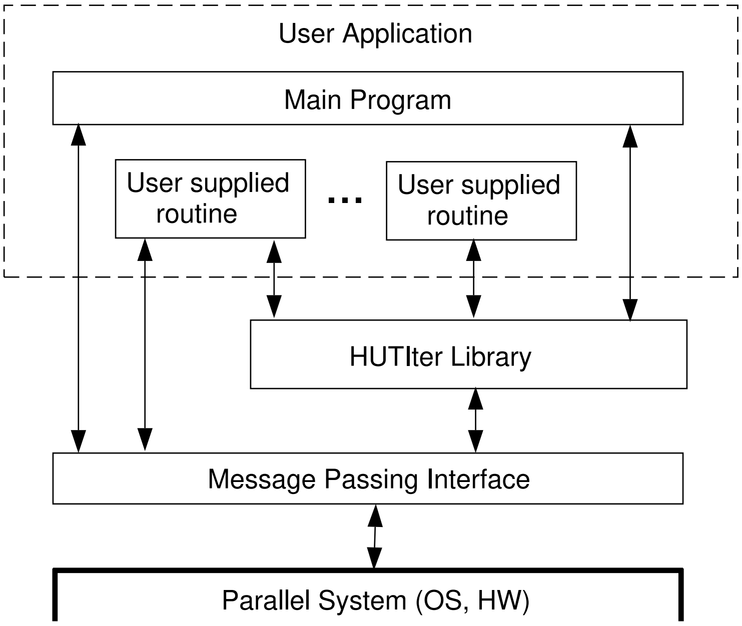

Figure E.1: The position of the HUTIter library

E.2.2 Naming Conventions

All the HUTI routine names and variables start with the

`huti_`
or
`HUTI_`

prefix. In the routine names the precision is denoted by an appropriate character: `s` for *single precision*, `d` for *double precision*, `c` for *complex* and `z` for *double complex*.

Solver routines are called in the following way:

```
CALL HUTI_*_SOLVER_TYPE ( X, RHS, IPAR, DPAR, WORK, MATVEC,
                          PCONDL, PCONDR, DOTPROD, NORM,
                          STOPC)
```

where	`*`	is either `S`, `D`, `C` or `Z` depending on the precision.
	`SOLVER_TYPE`	is either `CG`, `CGS`, `BICGSTAB`, `BICGSTAB_2`, `QMR`, `TFQMR` or `GMRES` depending on the method.

Table E.1 describes the parameters for solver routines.

Argument	Type	Description
X	vector of type *	Vector x, the current iterate
RHS	vector of type *	b, the Right Hand Side
IPAR	vector type integer	IPAR-structure, see section E.2.4
DPAR	vector of type double prec.	DPAR-structure, see section E.2.4
WORK	matrix of type *	User allocated working array, size varies depending on the method, see table E.7
MATVEC	subroutine	User supplied external routine, must perform matrix-vector product
PCONDL	subroutine	User supplied routine for left side preconditioning
PCONDR	subroutine	User supplied routine for right side preconditioning
DOTPROD	function	Used supplied routine to perform dot product
NORM	function	User supplied routine returning norm of a vector
STOPC	function	User supplied routine to perform stopping criteria testing

Table E.1: Parameters for the solver routines

The external routine `MATVEC` is the only needed routine when calling a solver. It should perform matrix-vector product. Using zeros in place of the other external routine names will force the library to use default routines applicapable to the selected execution model. For example *double complex* Conjugate Gradient method could be called from a FORTRAN program in the following way:

```
CALL HUTI_Z_CG (X, RHS, IPAR, DPAR, WORK, MATVEC, 0, 0, 0, 0, 0)
```

where `X, RHS, IPAR, DPAR` are user supplied vectors and `WORK` is the user allocated work space (an array) for the iterator. In this case the library would use BLAS-1 calls for `DOTPROD` and `NORM` if executed in serial execution mode. There would be no preconditioning applied. The `IPAR` and `DPAR` structures must contain user supplied information about the dimensions of the vectors and work array and certain control information for the iterators.

E.2.3 External Routines

This section describes external routines that can be given as arguments for the solver routine. Only `MATVEC` routine is required, others routines are optional.

These routines are called from the solver and type of arguments and order is presented for each routine.

The matrix A can be stored in any format because it is totally on the user's responsability to make it available for the external routines.

The `IPAR` structure is passed to some of the external routines and is used to carry on certain control variables from the solver routine. In the `IPAR` structure there is for example the assumed type of the matrix in external operation. This applies for both the matrix-vector operation $Au = v$ and preconditioning operations $M_1^{-1}u = v$ and $M_2^{-1}u = v$.

Matrix-Vector operation

The arguments for the external matrix-vector operation `MATVEC` are given in Table E.2. This routine should perform matrix-vector product. In the `IPAR` structure the iterator provides information about the matrix form, should it be transposed or not. Only non-transposed forms are used in CG, CGS, Bi-CGSTAB, TFQMR and GMRES methods. Only QMR will need a transposed matrix-vector product, that is $A^T u = v$.

The calling convention for the `MATVEC` is:

```
SUBROUTINE MATVEC ( U, V, IPAR )
```

Argument	Type	Description
U	vector of type *	Vector u in $Au = v$
V	vector of type *	Vector v in $Au = v$
IPAR	vector of type integer	IPAR-structure, see section E.2.4

Table E.2: Parameters for the external MATVEC subroutine

Preconditioning

The routines `PCONDL` and `PCONDR` should solve the $M_1 u = v$ and $M_2 u = v$ respectively if preconditioning matrix is splitted in two parts. If only one preconditioning matrix M is available, the `PCONDL` routine should solve $Mu = v$ and `PCONDR` should not be supplied for the solver (the argument must be zero).

The arguments for the external preconditioning operations `PCONDL` and `PCONDR` are given in Table E.3. Preconditioning routines should use the information in `IPAR` structure to apply transposed or non-transposed solve when needed. Only QMR method will need the $M^{-T}u = v$ operation.

The calling convention for the `PCONDL` is

```
SUBROUTINE PCONDL ( U, V, IPAR )
```

and for the `PCONDR`

```
SUBROUTINE PCONDR ( U, V, IPAR )
```

Argument	Type	Description
U	vector of type *	Vector u in $Mu = v$
V	vector of type *	Vector v in $Mu = v$
IPAR	vector of type integer	IPAR-structure, see section E.2.4

Table E.3: Parameters for the external PCONDL and PCONDR subroutines

Global Dot Product

The external function DOTPROD is supposed to perform global dot product for two given vectors. In the serial case this routine is by default the corresponding BLAS-1 routine. In the parallel case this is the place to do global product using for example MPI_ALLREDUCE function to sum up the local products computed using for example BLAS-1 routine.

The calling convention for the function DOTPROD is

```
FUNCTION DOTPROD ( NDIM, X, INCX, Y, INCY )
```

Argument	Type	Description
NDIM	integer	Dimension of vectors X and Y
X	vector of type *	Vector x in $x \cdot y$
INCX	integer	The increment for the elements of X
Y	vector of type *	Vector y in $x \cdot y$
INCY	integer	The increment for the elements of Y

Table E.4: Parameters for the external DOTPROD function

The function DOTPROD must return a value of the same type as the argument vectors.

Global Vector Norm

The external routine NORM is used to produce the global vector norm, usually the vector 2-norm $||x||_2$. In the serial case this routine is by default the corresponding BLAS-1 routine. In parallel case this is very similar as DOTPROD function.

The calling convention for the function NORM is

```
FUNCTION NORM ( NDIM, X, INCX )
```

Argument	Type	Description
NDIM	integer	Dimension of vector X
X	vector of type *	Vector x in $\|x\|$
INCX	integer	The increment for the elements of X

Table E.5: Parameters for the external NORM function

The function `NORM` must return a value that is either real if X is single precision (*real or complex*) or double if X is double precision (*double precision or double complex*).

Stopping Criterion

Stopping criterion can be selected from the built-in stopping criteria or it can be supplied by the user. Built-in alternatives are listed in table E.7.

The calling convention for the user supplied function `STOPC` is

```
FUNCTION STOPC ( X, B, R, IPAR, DPAR )
```

Argument	Type	Description
X	vector of type *	Current iterate x_n
B	vector of type *	The original right-hand side
R	vector of type *	Current residual vector r_n
IPAR	vector of type integer	IPAR-structure, see section E.2.4
DPAR	vector of type double precision	DPAR-structure, see section E.2.4

Table E.6: Parameters for the external STOPC function

The function `STOPC` must return a value of same type as the `NORM` function for selected precision. See the previous section.

The returned value should describe somehow how close the current iterate is the user supplied tolerance. It will be tested against the tolerance and printed if requested.

E.2.4 Iteration Parameters

IPAR -Structure

The `IPAR` structure is used to control the progress and behaviour of the iterator routine and to get status back from it. `IPAR` is also passed futher to some of the user supplied routines.

Input parameters are described in table E.7 along with their default values, output parameters are in table E.8.

A more detailed description of the various parameters and output values for each solver is listed on the corresponding reference pages.

Element	Description	Default
	General parameters	
1	Length of the IPAR structure	50
2	Length of the DPAR structure	10
3	Leading dimension of the matrix (and vectors)	
4	Number of vectors in the `work` array:	
	CG: 4	
	CGS: 7	
	Bi-CGSTAB: 8	
	Bi-CGSTAB_2: 8	
	QMR: 14	
	TFQMR: 10	
	GMRES: 7 + number of restart vectors	
5	Number of iterations between debug output	0
6	Assumed matrix type in external operations	
	0: Matrix must *not* be transposed	
	1: Matrix must be transposed	
	Iteration parameters	
10	Maximum number of iterations allowed	5000
12	Stopping criterion used:	0
	(ϵ is the tolerance given by the user, see table E.9)	
	0: $\|\|r_n\|\| < \epsilon$	
	1: $\|\|r_n\|\| < \epsilon\|\|b\|\|$	
	2: $\|\|z_n\|\| < \epsilon$	
	3: $\|\|z_n\|\| < \epsilon\|\|b\|\|$	
	4: $M^{-1}\|\|z_n\|\| < \epsilon M^{-1}\|\|b\|\|$	
	5: $\|\|x_n - x_{n-1}\|\| < \epsilon$	
	6: *upper bound* $< \epsilon$ (only with TFQMR)	
	10: Use the user supplied routine `STOPC`	
13	Preconditioning technique used:	0
	0: None	
	1: Right preconditioning	
	2: Left preconditioning	
	3: Symmetric preconditioning	
14	Initial x_0, starting vector:	0
	0: Random x_0	
	1: User supplied x_0, vector in `XVEC`	
15	Number of restart vectors in GMRES(m)	1
	Parallel environment parameters	
20	Processor identifaction number for spesific process	
21	Number of processors	1

Table E.7: IPAR-structure, input parameters

Element	Description
	General parameters
30	Status information: 0: No change 1: Iteration converged 2: Maximum number of iterations reached 10: QMR breakdown in ρ or ψ 11: QMR breakdown in δ 12: QMR breakdown in ϵ 13: QMR breakdown in β 14: QMR breakdown in γ 20: CG breakdown in ρ 25: CGS breakdown in ρ 30: TFQMR breakdown in ρ 35: Bi-CGSTAB breakdown in ρ 36: Bi-CGSTAB breakdown in $\|\|s\|\|$ 37: Bi-CGSTAB breakdown in ω
31	Number of iterations runned through

Table E.8: IPAR-structure, output parameters

DPAR -Structure

For parameters of type *double precision* there is a structure called `DPAR`. Table E.9 describes the elements of this structure.

Element	Description	Default
	General parameters	
1	Tolerance used by stopping criterion	$10e^{-6}$

Table E.9: DPAR-structure

E.2.5 Header Files

`huti_fdefs.h` and `huti_defs.h`

There are header files in preprocessor format for both Fortran90 and C languages. These header files include definitions for all of the variables described in tables E.7, E.8 and E.9. There are also definitions for possible flags of certain variables and default values.

The user should use the named definitions by including header file via `#include "huti_defs.h"` for C defines and `#include "huti_fdefs.h"` for Fortran90 defines. In that way the compatibility is guaranteed also with the later versions of the library.

Appendix F

Format of the ELMER Cad Interface settings file

In this appendix we describe the structure of the cad interface settings file. An example of the settings file is also given.

The general structure of the file is the following:

```
User Settings
    Keyword
        Data type
            Value
End
```

If a keyword is not given in the settings file, the default value for that keyword is used.

Keywords (not case sensitive) in use are:

- `Default Models Directory`: main directory for the models. Individual models will be stored in subdirectories
  default value: *current directory*

- `Default Cad Files Directory`: the directory where cad geometry files are read
  default value: *current directory*

- `Default Mesh Files Directory`: the directory where external mesh geometry files are read
  default value: *current directory*

- `Default Material Files Directory`: the default include path when material properties are read from a file. You can keep your "material database" in this directory!
  default value: *model directory*

- `Default Input Files Directory`: the default include path when other parameters are read from a file
  default value: *model directory*
- `Default Output Files Directory`: the default directory for the solver output files
  default value: *model directory*
- `Default Use Model Settings`: this flag controls if settings saved in the model file should be used when the model file is loaded
  default value: True
- `Auto Load Mesh`: this flags controls if mesh should be autmatically loaded
  default value: True
- `Auto Save Model`: this flag controls if updated model file should be saved before running any other Elmer modules.
  default value: True
- `Auto Save Solver Input`: this flag controls if solver input file should be always updated when model file is updated.
  default value: True
- `Browser Command`: command for an external browser program
  default value: *internal browser*
- `Editor Command`: command for an external editor program
  default value: *internal editor*
- `Browse Mode Gebhardt Factors`: log browse mode for Elmer Gebhardt Factors calculation module
  default value: Logfile
- `Browse Mode Mesh`: log browse mode for Elmer Mesh generator module
  default value: Logfile
- `Browse Mode Cadi2Db`: log browse mode for Elmer model to database converting module
  default value: Logfile
- `Browse Mode Solver`: log browse mode for Elmer Solver module
  default value: Logfile
- `Browse Mode View Factors`: log browse mode for Elmer View Factors calculation module
  default value: Logfile

Possible values for Browse Mode keywords are:

- Logfile : output goes to a logfile and you can use a browser to read the log
- Shell : output goes to system console
- None : no output

NOTE: All keywords starting with "Default" are keywords, which are stored and read only from the settings-file (like this) Other keyword values are also stored in the model file, and they are used when loading the model file (.ecf file). However you can prevent this by setting the keyword "Default Use Model Settings" to be untrue.

Data types in use are:

- String: if a string value contains spaces, the hole string should be inside quotation marks
- Logical: possible values are True/1 and False/0

NOTE: In the following, lines starting with the exclamation (!) mark, are comments.

```
!An example file
User Settings

  Default Models Directory
    String "c:/elmer/Models"

  Default Cad Files Directory
    String "c:/elmer/data/cad\_files"

  Default Mesh Files Directory
    String "c:/elmer/data/mesh\_files"

  Default Material Files Directory
    String "c:/elmer/material\_files"

  Default Input Files Directory
    String "c:/elmer/Models/stepflow"

  Default Output Files Directory
    String "c:/elmer/Models/stepflow"

  Default Use Model Settings
    Logical 1

  Auto Load Mesh
    Logical 1

  Auto Save Model
    Logical 1

  Auto Save Solver Input
    Logical 1

  !NOTE: read only mode (-r) for the external browser
  Browser Command
```

```
    String "C:\Program Files\TextPad\TXTPAD32.EXE -r"

  ! External editor
  Editor Command
    String "C:\Program Files\TextPad\TXTPAD32.EXE"

  !The following Browse Mode flags control the output log
  !of the ELMER Run-menu programs (default flag value is Logfile):

  Browse Mode Gebhardt Factors
    String Logfile

  Browse Mode Mesh
    String Logfile

  !Normally  no need for a log here
  Browse Mode Cadi2Db
    String None

  !Solver log can be quite long, so system console is the fastest.
  !However, a good external text editor could also be useful
  !if its updates its view quickly, anyway you can scroll it!
  Browse Mode Solver
    String Shell

  Browse Mode View Factors
    String Logfile

End
```

Appendix G

Format of the Elmer Geometry File

ELMER geometry file is one of the supported geometry formats for cad interface. The file name should be in the format *filename*.egf ('egf' for Elmer Geometry File).

Currently only linear (polygon based) 2D geometries are supported. There are no lmitations for the number of bodies defined, as long body edges define closed and non-intersecting geometries. A body can consist of multiple edge loops, as long as loops define a simply connected area. Outermost loop should always be counterclockwise (ccw) oriented. Possible inner loop(s) should be clockwise (cw) oriented

In this appendix we describe the structure of the ELMER geometry file using two example files. First file defines two adjacent squares, both with 1 meter side length. The other file defines a unit square with a square hole in the center.

```
!NOTE: lines starting with exclamation (!) mark are comments.

!Header section should be always first in the file.


!=====================================
!Example file 1, two adjacent squares
!=====================================

Header
   !Model name is optional
   Model Name
      String "Squares2"
   !Geometry dimension (currently only 2 (2D) is supported)
   Dimension
      Integer 2
End

!All model vertices should be defined before any body definition
!Vertices are implicitely indexed (1, 2, ...)
```

```
!Default value for the z-coordinate is 0.0
Vertices
   Points
      ! Here we give all coordinates, although model is 2D
      ! Size could be also 6 2, if only x and y were given
      Size 6 3
         Real

         0.0 0.0 0.0
         1.0 0.0 0.0
         1.0 1.0 0.0
         0.0 1.0 0.0
         2.0 0.0 0.0
         2.0 1.0 0.0

End

!Bodies numbered 1, 2, ...
!Name and color fields are optional
!Integer values for the color are RGBA values (0, ...,255), default Alpha is 255

Body 1
   Name
      String Red
   Color
      Size 4
         Integer 255 0 0 255
   !Body defined by four vertices
   Polygon
      Size 4
         !Vertex indices
         Integer 1 2 3 4
End

Body 2
   Name
      String "Blue"
   Color
      !Default alpha is used here
      Size 3
         Integer 0 0 255
   Polygon
      Size 4
         Integer 2 5 6 3
End


!================================================
!Example file 2, a unit square with a square hole
!================================================
```

```
Header
  Model Name "SquareWithHole"
  Dimension
    Integer 2
End

Vertices
  Points
    Size 8 3
      Real
   !Outer square (1m * 1m)
      0.0 0.0 0.0
     1.0 0.0 0.0
    1.0 1.0 0.0
    0.0 1.0 0.0
    !Inner square (0.5m * 0.5m)
      0.25 0.25 0.0
     0.75 0.25 0.0
    0.75 0.75 0.0
    0.25 0.75 0.0

End

Body 1
  Name
    String "Blue"
  Color
    Size 4
    Integer 255 0 0 255

  ! Outer square (NOTE: ccw (counterclockwise) orientation!)
  Polygon
    Size 4
     Integer 1 2 3 4

 ! Inner square (NOTE: cw (clockwise) orientation!)
  Polygon
    Size 4
     Integer 8 7 6 5

 End
```

Appendix H

Solver Input Format

In this appendix we describe the solver input file format in detail. This file is usually written by the ELMER CAD Interface, but can also be generated by other means or modified by the user to suit spesific needs.

For the programmers: the solver input file reader has no knowledge of the content of the different sections in this file except of the Header section. So in any section you can add name-value pairs whenever needed without breaking anything. These values may then be accessed by calling them by the name provided. The utility routines `ListGet*` may be used for this purpose. These utility routines have been described in appendix K.

Because of the generality of the format, the types of the parameters must be given with their values. Also there are a few generalizations of the format given below.

H.1 Array Variables

Any parameter typed `Real` or `Integer` may be given keyword `size n1 n2`, where `n2` is optional and when given, the statement defines a two dimensional array instead of a one dimensional. An example of an array variable is the gravity vector declared in the constants section:

```
Gravity
  Size 4
    Real 0 1 0 9.82
```

The above statement gives a real vector whose length is four. In this case the first three components give the direction vector of the gravity and the fourth component gives its intensity.

H.2 Parameters Depending on Field Variables or Time

Any parameter typed `Real` depending on some field variable or time, may be given as a piecewise linear spline as follows:

```
Parameter-Name
  Variable Variable-Name
    Real
      Variable-Value-1 Parameter-Value-1
      Variable-Value-2 Parameter-Value-2
                 ...
      Variable-Value-n Parameter-Value-n
    End
```

If the `size` keyword is also given, and differs from one, the parameter array values are given instead of the single parameter value. The variables on which a parameter may depend are at the moment:

- `Time`,
- `Temperature`,
- `Pressure`,
- `Velocity 1, Velocity 2, Velocity 3`,
- `Coordinate 1, Coordinate 2, Coordinate 3`,
- `Displacement 1, Displacement 2, Displacement 3`,
- `Magnetic Field 1, Magnetic Field 2, Magnetic Field 3`,
- `Electric Current 1, Electric Current 2, Electric Current 3`.

The programmer may add more variables using the utility routine `VariableAdd`.

In the Margoni convection example later in this chapter, there is a temperature boundary condition depending on x-coordinate, which is defined as follows:

```
Temperature
  Variable Coordinate 1
    Real
       0  0.5
       1 -0.5
    End
```

This says, that the value of temperature is 0.5 degrees when x-coordinate is 0, the value of temperature is -0.5 degrees when x-coordinate is 1, and that the value of temperature will be linearily varying with the x-coordinate in between these values.

H.3 User Defined Functions

Any parameter typed `Real`, `Integer` or `Logical` may be given keyword `Procedure` instead of the value of the parameter. Then a user defined function will be executed when the parameter value is needed. The syntax of this statement is:

```
Parameter-Name
  Real Procedure "Filename" "Function-name"
```

The `Filename` is the name of the file of the dynamically loadable code (called a shareable image on Unix and a DLL on WINDOWS). As an very simple example, the following Fortran 90 function would return the value of the surface tension coefficient (as in the Margoni convection example later in this chapter), when needed:

```
FUNCTION SurfTension(Model,n,T) RESULT(Tension)
  !DEC$ATTRIBUTES DLLEXPORT :: SurfTension
  USE Types
  IMPLICIT NONE

  TYPE(Model_t), POINTER :: Model
  INTEGER :: n
  DOUBLE PRECISION :: T

  DOUBLE PRECISION :: Tension

  Tension = 548*(1-T)
END FUNCTION SurfTension
```

If this function is in the file `Surf.f90`, the following command could be used to compile and link the program on a Silicon Graphics or DEC workstation (among others):

```
f90 -I$ELMER_HOME/include -o Surf -shared Surf.f90
```

On a Windows NT workstation, using the Digital Visual Fortran version 5 (the Fortran 90 comment line `!DEC$ATTRIBUTES...` is actually a compiler directive for this compiler), the command would be:

```
f90 -I%ELMER_HOME%\include -o Surf -dll Surf.f90
```

After the function has been compiled, it can be used to define the surface tension coefficient in the solver input file as

```
Surface Tension Coefficient
   Variable Temperature
     Real Procedure "Surf" SurfaceTension
```

The user function will receive as first argument the model structure holding pointers to all information about the model, solved field variables, materials, etc. It also holds a pointer to the element that is being assembled at the moment when the function is called. The second argument is the number of the node for which the function should return its value and if the parameter in question is defined to be varying with some field variable or time, the third argument will contain the value of that variable. The user functions may use the utility routines defined in the solver to get hold of other parameter values, field variable values, etc. More about the solver utility routines will be told in Appendix K.

H.4 The Format

H.4.1 The `Header` Section

The first section in the input file must be the header section defining various dimensions: number of model bodies, number of body force definitions, number of separate boundaries, number of boundary condition definitions, number of equation set definitions, number of initial condition definitions and finally number of equation solver definitions. Also the path to the mesh database is given in this section. The syntax of the header section is as follows:

```
!
! 7 entities
!
Header
  Mesh DB "Dir" "Name"
  Bodies              n
  Body Forces         n
  Boundaries          n
  Boundary Conditions n
  Equations           n
  Initial Conditions  n
  Materials           n
  Solvers             n
End
```

Note that this is the only section whose content is known to the solver input reader.

H.4.2 The `Constants` Section

The constants section must always be present and hold at least direction and intensity of the gravity vector and value of the Stefan-Bolzmann constant.

```
! 2 Entities
!
Constants
  Gravity
    Size 4
      Real  x y z abs

   Stefan Bolzmann
      Real
End
```

H.4.3 The `Simulation` Section

The simulation section gives the case control data:

- `Simulation Type`: a character string containing either the keyword `Transient` or `Steady State`.
- `Coordinate Mapping`: mapping of the mesh coordinates to the order used in the solver: (x, y, z) in cartesian coordinates, (r, z, θ) in cylindrical coordinates and (r, θ, ϕ) in polar coordinates. The mapping is a triplet of integers giving the order number of the solver coordinate variables in the mesh coordinate arrays.

- `Coordinate System`: a character string defining the coordinate system to be used, one of: `Cartesian 2D`, `Cartesian 3D`, `Polar 2D`, `Polar 3D`, `Cylindric`, `Cylindric Symmetric` and `Axi Symmetric`.
- `Gebhardt Factors`: If the model includes diffuse gray radiation, the file containing the Gebhardt factors must be given. This is written by the program `GebhardtFactors` as a preprocessing step.
- `View Factors`: If the model includes diffuse gray radiation, the file containing the view factors must be given to the program computing the Gebhardt factors. This is written by the program `Viewfactors` as a preprocessing step. The tasks of computing view factors and the Gebhardt factors have been divided, because the view factors depend only on geometry (and thus the mesh), while the Gebhardt factors also depend on the emissivities.
- `Timestep Intervals`: timesteps are given in intervals of fixed sizes, this integer array gives the number of timesteps within each of the intervals. The number of intervals must be given with the `size` specifier.
- `Timestep Sizes`: array of the same size as given with the `Timestep Intervals` keyword, giving size of the timesteps within the intervals.
- `Output File`: Name of the output file. Format of the output file is described in Section I.
- `Output Intervals`: array of the same size as given with the `Timestep Intervals` keyword, giving the update frequency of the output file within the timestep intervals. For steady state simulation, there is only one interval.
- `Post File`: If this keyword is given, mesh and results are written to the file in the format understood by Elmer Post.
- `Restart File`: The format of the restart file is the same as that of the `Output File`. The restart file gives field variable values, which will replace the values of the variables inside the solver after initial conditions have been set. The format of the file allows only some of the field variables to be given.
- `Restart Position`: Integer giving the order number of the timestep or steady state iteration output within the restart file. If this number is larger than the number of timesteps in the given file, last timestep found is used. If this keyword is not given, the first timestep from the file will be used.
- `Steady State Max Iterations`: Maximum number of steady state, coupled system iterations. If equation specific convergence tolerances are achieved before the iteration count is exceeded, they will terminate the iteration instead.

```
! 13 entities
!
```

```
Simulation
  Coordinate Mapping
    Size 3
      Integer
  Coordinate System             String [Cartesian 2D]
                                       [Cartesian 3D]
                                       [Cylindric Symmetric]
                                       [Axi Symmetric]
                                       [Cylindric]
                                       [Polar 2D]
                                       [Polar 3D]
  Gebhardt Factors              File
  Output File                   File
  Output Intervals              Integer Array
  Post File                     File
  Restart File                  File
  Restart Position              File
  Simulation Type               String [Transient]
                                       [Steady State]
  Steady State Max Iterations   Integer
  Timestep Intervals            Integer Array
  Timestep Sizes                Real Array
  View Factors                  File
End
```

H.4.4 The `Solver` Section

The solver section defines equation solver control variables:

- `Equation`: a character string containing the name of the equation this solver is intended for, one of: `Navier-Stokes`, `Heat Equation`, `Stress Equations`.
- `Linear System Solver`: a character string containing either `Iterative` or `Direct`.
- `Linear System Iterative Method`: a character string containing name of the iterative method to be used: `BiCGStab`, `CGS`, `TFQMR`, `GMRES` or `CG`. Beware, that the last of the choices is for symmetrical linear systems, which is not the case for any of the default equations in our context. Even if the equations themselves would be symmetric, symmetry is destroyed by the way the Dirichlet boundary conditions are handeled inside the solver. This keyword must be given, if the iterative solver is used.
- `Linear Solver Max Iterations`: the maximum number of iterations the iterative solver is allowed to do. If the residual of the linear system is small enough before the iteration count is met, this will terminate the iteration instead.
- `Linear Solver Preconditioning`: a character string containing the name of the preconditioning method for iterative solver, one of: `None`, `Diagonal`, `ILU`. The choice `ILU` (ILU stands for Incomplete LU decomposition) is the

recommended choice, and usually the iterative solver won't be able to solve the equations without this preconditioning.

- `Linear System Convergence Tolerance`: a threshold value giving the value of the residual to terminate the iteration. More accurately, the termination criterion is
$$||Ax - b|| < \epsilon ||b||,$$
where ϵ is the value given with this keyword.

- `Nonlinear System Convergence Tolerance`: this keyword gives a criterion to terminate the nonlinear iteration after the relative change of the norm of the field variable between two consecutive iterations is small enough
$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$
where ϵ is the value given with this keyword.

- `Nonlinear System Max Iterations`: The maxmimum number of nonlinear iterations the solver is allowed to do.

- `Nonlinear System Newton After Iterations`: Change the nonlinear solver type to Newton iteration after a number of Picard iterations have been performed. If a given convergence tolerance between two iterations is met before the iteration count is met, it will switch the iteration type instead.

- `Nonlinear System Newton After Tolerance`: Change the nonlinear solver type to Newton iteration, if the relative change of the norm of the field variable meets a tolerance criterion:
$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$
where ϵ is the value given with this keyword.

- `Nonlinear System Relaxation Factor`: Giving this keyword triggers the use of relaxation in the nonlinear equation solver. Using a factor below unity is sometimes required to achive convergence of the nonlinear system. A factor above unity might speed up the convergence. Relaxed variable is defined as follows:
$$u_i^{'} = \lambda u_i + (1 - \lambda) u_{i-1},$$
where λ is the factor given with this keyword. The default value for the relaxation factor is unity.

- `Steady State Convergence Tolerance`: With this keyword a equation specific steady state convergence tolerance is given. First of the the solvers to reach the tolerance criterion will end the coupled system iteration. The tolerance criterion is:
$$||u_i - u_{i-1}|| < \epsilon ||u_i||,$$
where ϵ is the value given with this keyword.

- `Stabilize`: If this flag is set true the solver will use stabilized finite element method when solving the Navier-Stokes equations, and heat equation with a convection term. Usually stabilization of the equations must be done in order to succesfully solve the equations. This setting has no meaning for the stress analysis type, at least not at the moment.

```
!
! 12 entities
Solver id
  Equation                                String [Navier-Stokes]
                                                 [Heat Equation]
                                                 [Stress Equations]
                                                 [Magnetic Induction}
  Linear System Convergence Tolerance     Real
  Linear System Iterative Method          String [BiCGStab][CGS]
                                                 [TFQMR][GMRES][CG]
  Linear System Max Iterations            Integer
  Linear System Preconditioning           String [None][Diagonal]
                                                     [ILU]
  Linear System Solver                    String [Iterative]
                                                 [Direct]
  Nonlinear System Convergence Tolerance  Real
  Nonlinear System Max Iterations         Integer
  Nonlinear System Newton After Iterations Integer
  Nonlinear System Newton After Tolerance Real
  Nonlinear System Relaxation Factor      Real
  Stabilize                               Logical
End
```

H.4.5 The `Body` Section

The body section defines which of the equations, initial conditions, materials, and body forces defined in the input file, should be used for the given simulation body. All these sections are described in detail below.

```
! 4 entities
!
Body id
  Body Force          Integer
  Equation            Integer
  Material            Integer
  Initial Condition   Integer
End
```

H.4.6 The `Equation` Section

The equation section is used to define a set of equations for a body:

- `Navier-Stokes`: if set to true, solve the Navier-Stokes equations.
- `Heat Equation`: if set to true, solve the heat equation.

- `Magnetic Induction`: if set to true, solve the magnetic induction equation.
- `Stress Analysis`: if set to true, solve the stress equations.
- `Convection`: a character string giving the convection type to be used in the heat equation, one of: `None`, `Computed`, `Constant`.
- `Flow Model`: This keyword specifies the flow model to be used. Currently it can only be set to `Laminar`.
- `Stress Model`: a character string containing either `Mechanical` or `Thermal`. If the latter choice is used heat equation must also be solved.
- `Phase Change`: if set to true, the solidification phase change model is activated. Note that when soldification is modelled, the enthalpy-temperature- and viscosity-temperature-curves must be defined in the material section.

```
!
! 12 entities
Equation id
  Convection             String [None][Computed][Constant]
  Flow Model             String [Laminar][Turbulent KE]
  Heat Equation          Logical
  Navier-Stokes          Logical
  Phase Change           Logical
  Phase Change Model     String  [Enthalpy]
  Stress Analysis        Logical
  Stress Model           String [Mechanical][Thermal]
  Magnetic Induction      Logical True
End
```

H.4.7 The `Body Force` Section

The body force section may be used to give additional force terms for the equations. The following keywords are recongnized by the solver:

- `Bussinesq`: a logical value, if set true, sets the Bussinesq model on.
- `Flow BodyForce 1,2,3`: may be used to give additional body force for the flow momentum equations.
- `Heat Source`: an additional heat source for the heat equation may be given with this keyword.
- `Stress Bodyforce 1,2,3`: an additional force term for the stress equations may be given with these keywords.
- `Lorenz Force`: a logical value, if set true, triggers the magnetic field force for the flow mementum equations, and the Joule heating for the heat equation.

- `Friction Heat`: a logical value, if set true, triggers use of the friction heating:

$$h_f = 2\mu\varepsilon^2. \tag{H.1}$$

```
! 10 entities
!
Body force id
  Bussinesq                                  Logical
  Flow Bodyforce 1                           Real
  Flow Bodyforce 2                           Real
  Flow Bodyforce 3                           Real
  Heat Source                                Real
  Stress Bodyforce 1                         Real
  Stress Bodyforce 2                         Real
  Stress Bodyforce 3                         Real
  Lorenz Force                               Logical
  Friction Heat                              Logical
 End
```

H.4.8 The `Initial Condition` Section

The initial codition section may be used to set initial values for the field variables. The following variables are active:

- `Pressure`
- `Temperature`
- `Velocity 1, Velocity 2, Velocity 3`
- `Displacement 1, Displacement 2, Displacement 3`
- `Magnetic Field 1, Magnetic Field 2, Magnetic Field 3`
- `Electric Current 1, Electric Current 2, Electric Current 3`

```
! 16 entities
!
Initial Condition id
  Kinetic Energy                             Real
  Kinetic Energy Dissipation                 Real
  Pressure                                   Real
  Temperature                                Real
  Velocity 1                                 Real
  Velocity 2                                 Real
  Velocity 3                                 Real
  Displacement 1                             Real
  Displacement 2                             Real
  Displacement 3                             Real
  Magnetic Field 1                           Real
  Magnetic Field 2                           Real
  Magnetic Field 3                           Real
End
```

H.4.9 The Material Section

The material section is used to give the material parameter values. The following material parameters may be set:

- `Density`
- `Enthalpy`: note that, when using the solidification modelling, an enthalpy-temperature curve must be given. The enthalpy is derived with respect to tempereratute to get the value of the effective heat capacity.
- `Viscosity`: note that, when using the solidification modelling, a viscosity-temperature curve must be given. The viscosity must be set to high enough value in the temperature range for solid material to effectively set the velocity to zero.
- `Heat Capacity`
- `Heat Conductivity`
- `Heat Expansion Cofficient`
- `Reference Temperature`: This is the reference temperature for the Bussinesq model of temperature dependancy of density.
- `Poisson Ratio`
- `Youngs Modulus`
- `Magnetic Permeability`
- `Electric Conductivity`
- `Convection Velocity 1,2,3`: Convection velocity for the constant convection model.
-
- Applied Magnetic Field 1,2,3: An applied magnetic field may be given with these keywords.

```
! 17 entities
!
Material id
  Density                                    Real
  Enthalpy                                   Real
  Heat Capacity                              Real
  Heat Conductivity                          Real
  Heat Expansion Coefficient                 Real
  Poisson Ratio                              Real
  Reference Temperature                      Real
  Viscosity                                  Real
  Youngs Modulus                             Real
  Electric Conductivity                      Real
  Magnetic Permeability                      Real
```

```
  Convection Velocity 1                     Real
  Convection Velocity 2                     Real
  Convection Velocity 3                     Real
  Applied Magnetic Field 1                  Real
  Applied Magnetic Field 2                  Real
  Applied Magnetic Field 3                  Real
End
```

H.4.10 The `Boundary` Section

In this section a mapping of boundary conditions to boundaries of the geometrical model is given. In the boundary section the following keywords may be given

- `Body 1`: With this keyword the id of the first body which the boundary belongs is given.
- `Body 2`: With this keyword the id of the second body which the boundary belongs is given.
- `Boundary Condition`: With this keyword the id of the boundary condition to be applied on this boundary is given.

```
Boundary id
   Body 1               Integer
   Body 2               Integer
   Boundary Condition   Integer
End
```

H.4.11 The `Boundary Condition` Section

The boundary condition section holds the parameter values for various boundary condition types. Dirichlet boundary conditions may be set for all the primary field variables: displacement (keywords `Displacement 1`, `Displacement 2`, `Displacement 3`), velocity (keywords `Velocity 1`, `Velocity 2`, and `Velocity 3`), temperature and pressure, and magnetic field (`Magnetic Field 1`, `Magnetic Field 2`, and `Magnetic Field 3`. The Dirichlet conditions for the vector variables may be given in normal-tangential coordinate system instead of the coordinate axis directed system using the keywords `Normal-Tangential Velocity`, `Normal-Tangential Displacement`, and `Normal-Tangential Magnetic Field`.

For the heat equation the heat flux boundary condition control variables are:

- `Heat Flux BC`: must be set to `true`, if heat flux boundary condition is present.
- `Heat Flux`: a user defined heat flux term.
- `Heat Transfer Coefficient, External Temperature`: a heat flux boundary condition of the type

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

is used. The variable α is the value given with the keyword `Heat Transfer Coefficient`, the variable T_{ext} is the value given with the keyword `External Temperature`.

- `Radiation, Emissivity`: with these keywords the radiation boundary condition is activated. With the keyword `Radiation` a character string is given, containing the type of the radiation model for this boundary, one of: `None`, `Idealized`, `Diffuse Gray`. Emissivity is the emissivity of the boundary surface. Note that, if using the diffuse gray radiation model, the file containing the Gebhardt factors must be given in the simulation section.

For the Navier-Stokes equations, the following variables control the force boundary conditions:

- `Flow Force BC`: a logical value that must be set to `true`, if there is a force boundary condition for the Navier-Stokes equations.

- `Surface Tension Coefficient, Surface Tension Expansion Coefficient`: Giving these values triggers a tangetial stress boundary condition to be used. If the keyword `Surface Tension Expansion Coefficient` is given, a linear dependancy of the surface tension coefficient on the temperature is assumed. Otherwise the value given with the `Surface Tension Coefficient` is assumed to hold the dependancy explicitely. Note that this boundary condition is the tangential derivate of the surface tension coefficient.

- `External Pressure`: a pressure boundary condition directed normal to the surface.

- `Pressure 1,2,3`: a pressure boundary condition vector.

A free surface is specified by giving the keyword `Free Surface` value `True`. The logical keyword `Free Moving` specifies whether the regeneration of mesh is free to move the nodes of a given boundary when remeshing after moving the free surface nodal points. The default is that the boundary nodes are fixed.

The force boundary condition for the stress equations is a force vector given with keywords `Force 1, Force 2, Force 3`.

```
! 33 entities
!
Boundary Condition id
  Normal-Tangential Displacement          Logical
  Displacement 1                          Real
  Displacement 2                          Real
  Displacement 3                          Real
  Emissivity                              Real
  External Pressure                       Real
  External Temperature                    Real
  Flow Force BC                           Logical
  Force 1                                 Real
  Force 2                                 Real
  Force 3                                 Real
```

```
  Heat Flux                                    Real
  Heat Flux BC                                 Logical
  Heat Transfer Coefficient                    Real
  Kinetic Energy                               Real
  Kinetic Energy Dissipation                   Real
  Pressure 1                                   Real
  Pressure 2                                   Real
  Pressure 3                                   Real
  Radiation                                    String [None][Idealized]
                                                       [Diffuse gray]
  Surface Tension Coefficient                  Real
  Surface Tension Expansion Coefficient        Real
  Temperature                                  Real
  Normal-Tangential Velocity                   Logical
  Velocity 1                                   Real
  Velocity 2                                   Real
  Velocity 3                                   Real
  Free Surface                                 Logical
  Free Moving                                  Logical
  Normal-Tangential Magnetic Field             Logical
  Magnetic Field 1                             Real
  Magnetic Field 2                             Real
  Magnetic Field 3                             Real
End
```

H.5 An Example of Using the Input File

In the following a sample solver input file, written by the ELMER CAD Interface program, will be described. The problem the solver input file applies is a steady-state coupled heat and flow simulation, where the coupling comes from the convection term in the heat equation side, and temperature dependancy of the surface tension coefficient (called Margoni convection) in the flow solver side. The geometry used in this simple example is a unit square, so there are four boundaries in this model. The boundaries were assigned boundary conditions so that bottom, top, left and right boundaries had boundary conditions id:s 1,2,3 and 4 respectively (figure H.1).

Mathematically the equations to be solved are

$$\begin{aligned}
-\nabla\cdot(2\mu\bar{\bar{\varepsilon}})+\rho(\vec{u}\cdot\nabla)\vec{u}+\nabla p &= 0 \quad \text{in } \Omega \\
\nabla\cdot\vec{u} &= 0 \quad \text{in } \Omega \\
\rho c_p \vec{u}\cdot\nabla T-\nabla\cdot(k\nabla T) &= 0 \quad \text{in } \Omega, \\
\vec{u}=0,\ \ T &= \frac{1}{2}-x \quad \text{on } \Gamma_1, \\
\sigma_\tau=\frac{\partial\gamma}{\partial\tau},\ \ \vec{u}\cdot\vec{n}=0,\ \ -k\frac{\partial T}{\partial n} &= 0 \quad \text{on } \Gamma_2, \\
\vec{u}=0,\ \ T &= \frac{1}{2} \quad \text{on } \Gamma_3, \\
\vec{u}=0,\ \ T &= -\frac{1}{2} \quad \text{on } \Gamma_4.
\end{aligned}$$

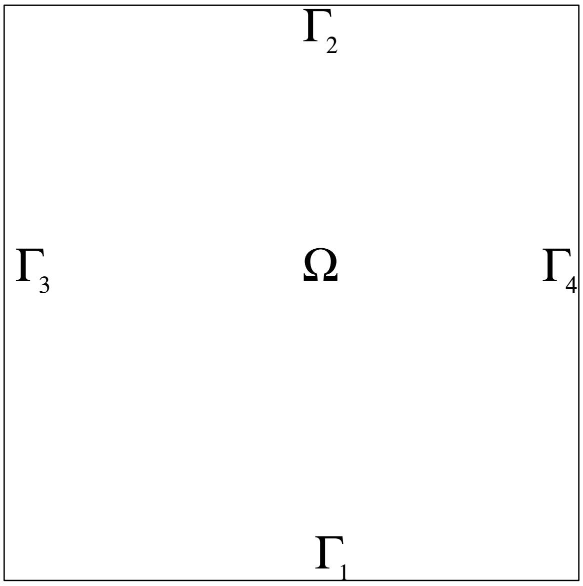

Figure H.1: Geometry of the Margoni convection example.

The material parameters, in non-dimensional units, are as follows:

$$\begin{aligned}\rho &= 1.0,\\ \mu &= 1.0,\\ c_p &= 1.0,\\ k &= 1.37,\\ \gamma &= 548.0(1-T).\end{aligned}$$

The Input File Header

First in the solver input file there is the header section. In the header section various dimensions of other sections are given. Also path to the mesh database is given here:

```
Header
  Mesh DB "Models" "Square"
  Bodies              1
  Materials           1
  Equations           1
  Solvers             2
  Body Forces         1
  Boundaries          4
  Boundary Conditions 4
End
```

Solver Control Data

Next we must give the solver the case control data

```
Simulation
  Coordinate System
    String Cartesian 2D

  Simulation Type
    String Steady State

  Steady State Max Iterations
    Integer 50

  Output Intervals
    Size 1
      Integer 1

  Output File
    File "Margoni.dat"

  Post File
    File "Margoni.ep"
End
```

The output file will contain the solved field variables for every iteration of the coupled system: temperature, velocity and pressure. The post processing file will also contain the mesh.

Body Definitions

This model has only one body which is assigned material and the equation set (to be defined later). There are no initial conditions or mass forces definition for the current case.

```
Body 1
  Equation
    Integer 1

  Material
    Integer 1
End
```

Equation Set Definitions

Next we define which equations to solve (there is only one set of equations in this example). The case where we would have more than one equation set is, for example, a case where temperature is solved in the whole model, but flow field is solved only in some parts of the model.

```
Equation 1
  Navier-Stokes
    Logical True

  Heat Equation
```

```
    Logical True

  Convection
    String Computed
End
```

Now we are ready to define solver parameters for the heat equation

```
Solver 1
  Equation
    String Heat Equation

  Stabilize
    Logical True

  Nonlinear System Max Iterations
    Integer 1

  Linear System Solver
    String Iterative

  Linear System Iterative Method
    String BiCGStab

  Linear System Preconditioning
    String ILU

  Linear System Max Iterations
    Integer 100

  Linear System Convergence Tolerance
    Real 1.0e-8
End
```

There is no nonlinearity present in the current case for the heat equation, so that we can set the number of nonlinear iterations to unity. The linear system is solved with an iterative method and system is precoditioned with an incomplete LU decomposition.

Solver Definitions

Define solver parameters for the Navier-Stokes equations

```
Solver 2
  Equation
    String Navier-Stokes

  Stabilize
    Logical True

  Nonlinear System Max Iterations
    Integer 10
```

```
  Nonlinar System Convergence Tolerance
    Real 1.0e-6

  Nonlinear System Newton After Iterations
    Integer 3

  Nonlinear System Newton After Tolerance
    Real 1.0e-2

  Nonlinear System Relaxation Factor
    Real 1.0

  Linear System Solver
    String Iterative

  Linear System Iterative Method
    String BiCGStab

  Linear System Preconditioning
    String ILU

  Linear System Max Iterations
    Integer 100

  Linear System Convergence Tolerance
    Real 1.0e-8

  Steady State Convergence Tolerance
    Real 1.0e-5
End
```

Material parameters

The material parameters are all constant in this example

```
Material 1
  Density
    Real 1.00

  Viscosity
    Real 1.00

  Heat Capacity
    Real 1.00

  Heat Conductivity
    Real 1.37
End
```

Boundaries and Boundary Conditions

The mapping of the boundary condtions to boundaries must first be given. In this case it is the trivial mapping:

```
Boundary 1
  Boundary Condition
    Integer 1
End

Boundary 2
  Boundary Condition
    Integer 2
End

Boundary 3
  Boundary Condition
    Integer 3
End

Boundary 4
  Boundary Condition
    Integer 4
End
```

The bottom boundary has a Dirichlet boundary condition for temperature, which varies linearily from 0.5 degrees on the left side to -0.5 degrees on the right side. This boundary condition has been given below as data points describing (generally) a piecewise linear spline. There are other methods to implement varying boundary conditions to be described later.

```
!****** Bottom BC: Temperature T(x) from T(0)= 0.5
!                 to T(1)=-0.5,Velocity=0
Boundary Condition 1
  Temperature
    Variable Coordinate 1
      Real
      ! x  T
        0  0.5
        1 -0.5
      End

  Velocity 1
    Real 0.0

  Velocity 2
    Real 0.0
End
```

The top boundary has a stress boundary condition. Note that the force boundary conditions must be flagged with the 'Flow Force BC' keyword in order to be recognized by the solver. The tangetial component of the force vector is the tangential derivate of the temperature dependent surface tension coefficient. The temperature dependancy of this parameter is assumed to be linear in this case. The normal component of the stress condition may be omitted as the curvature of the boundary is zero, and the level of the atmospheric pressure is immaterial and would only move the zero level of the pressure field. Instead the condition $u \cdot n = 0$ (no flow across the surface) is enforced.

```
!****** Top BC: Tangetial Stress, normal velocity zero
Boundary Condition 2
  Flow Force BC
    Logical True

  Surface Tension Coefficient
    Variable Temperature
      Real
       !  T    \gamma  (only the slope matters...)
        -1.0  548.0
         1.0 -548.0
      End

  Normal-Tangential Velocity
    Logical True

  Velocity 1
    Real 0.0
End
```

Right and left sides have Dirichlet boundary conditions for both the temperature and the velocity components.

```
!****** Right side BC: Temperature=-0.5, Velocity=0
Boundary Condition 3
  Temperature
    Real -0.5

  Velocity 1
    Real 0.0

  Velocity 2
    Real 0.0
End

!****** Left side BC: Temperature=0.5, Velocity=0
Boundary Condition 4
  Temperature
    Real 0.5

  Velocity 1
    Real 0.0

  Velocity 2
```

```
    Real 0.0
End
```

Results

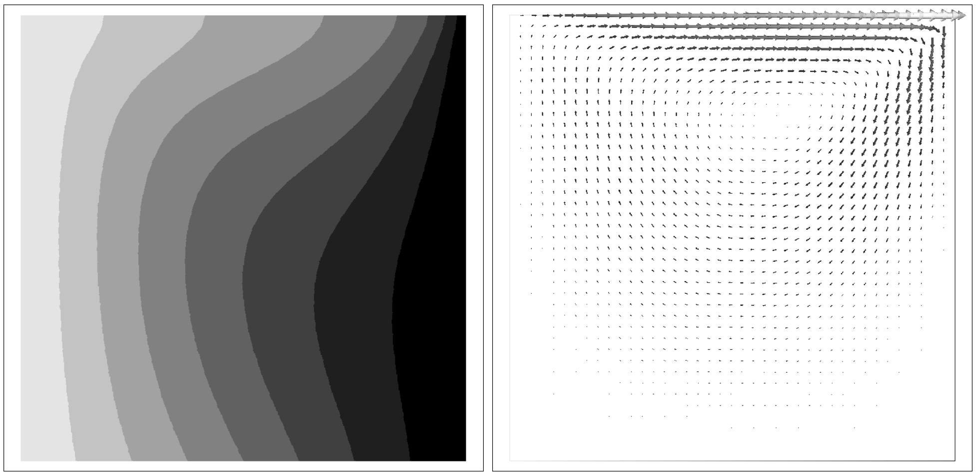

Figure H.2: Results of the Margoni convection example. Isocontours of the temperature field and the velocity vectors.

The resulting temperature and velocity fields are shown in figure H.2.

Appendix I

Format of the Solver Output File

The solver output file format is very simple. The file consist of a header and the solution data. The solver output format is also used in the restart files (refer to previous chapter for more information about the restart files).

I.1 The Header

First row in the file is the string

```
Degrees of freedom:
```

followed by names of the variables, which have been written to this file, for example

```
temperature
pressure
velocity 2
velocity 1
```

The names must each be on their own rows. The whole set of names of variables used by the solver is

- `Displacement 1`
- `Displacement 2`
- `Displacement 3`
- `Temperature`
- `Pressure`
- `Velocity 1`
- `Velocity 2`
- `Velocity 3`

- `Coordinate 1`
- `Coordinate 2`
- `Coordinate 3`
- `Magnetic Field 1`
- `Magnetic Field 2`
- `Magnetic Field 3`
- `Electric Current 1`
- `Electric Current 2`
- `Electric Current 3`
- `Velocity 1`
- `Velocity 2`
- `Velocity 3`

Following the names is the total number of degrees of freedom, for example:

```
 Total DOFs:            4
```

This ends the header section.

I.2 The Output Data

For every timestep or steady state iteration stored in output file, there is first a header row giving count of the timestep in this file, the timestep number and the current simulation time, for example

```
Time:     1    1      0.000000000000
```

Following is, for every variable stored in the file, first the name of the variable, followed by rows which contain the node number of the value and the value itself:

```
temperature
           1   300.000000000000
           2   300.000000000000
           3   300.000000000000
           4   300.000000000000
           5   300.000000000000
           6   300.000000000000
               ....
```

Appendix J

The Elmerpost File format

The Elmerpost input file format is as follows

```
nn ne nf nt scalar: name vector: name ...
x0 y0 z0
...       ! node coordinates (nn) rows (x,y,z)
xn yn zn
group-name element-type i0 ... in
...       ! element decriptions (ne) rows
group-name element-type i0 ... in
#time 1 timestep1 time1
vx vy vz p ...
...       ! nn rows
vx vy vz p ...
#time 2 timestep2 time2
vx vy vz p
...       ! nn rows
vx vy vz p ...
 ....
#time n timestepn timen
vx vy vz p ...
...       ! nn rows
vx vy vz p ...
```

J.1 The Header

The following information must be given in the header

- `nn`: number of the mesh nodal points
- `ne`: number of the elements
- `nf`: gives the total number of degrees of freedom (add three for any vector quantity and one for a scalar quantity).
- `nt`: number of timesteps stored in the file
- `scalar: name, vector: name`: list of variable names

J.2 The Mesh

First nodal coordinates are given, each triplet on its own row. There must be three coordinates even if the model would be 2D.

Element description:

- `group-name`: name of the element group (material, body or some such)
- `element-type`: numeric code giving the element type, refer to appendix L.
- The numbers `i0-in` in the element descriptions are indexes to the nodal coordinate array at the beginning of the file. First coordinate in the file has index zero.

J.3 The Solution Data

For each timestep the following information is written:

- `#time n t time`: `n` is the order number of the timestep written in this file, `t` is the simulation timestep number, and `time` is the current simulation time.
- Next the nodal values of the degrees of freedom are given, one nodal point at a time, and in the order given with the keywords `scalar:` and `vector:` in the header.

Refer to Elmerpost documentation for more information.

Appendix K

Solver Utility Routines

In this appendix various solver utility routines are described. First we must describe the structures used to hold various data used by the solver. As mentioned before, the solver is coded in the Fortran 90 porgramming language, and in order to use its utilities, this programming language must be used.

K.1 Data Structures

All the data structures are defined in a Fortran 90 module `Types` which must be included to a program using these structures with the Fortran statement:

```
USE Types
```

The executable code and some symbolic names have been defined in various other modules, which must be included in the user program similarily to the above statement, if you need services from these modules.

K.1.1 The Model Structure

The model structure holds pointers to all information about the model we are trying to solve.

```
    TYPE Model_t
!
!     Coordinate system dimension + type
!
      INTEGER :: Dimension, CoordinateSystem
```

The coordinate system code is held in the entry `CoordinateSystem`. The code may be referenced to with a symbolic name:

- `Cartesian`
- `Cylindric`
- `AxisSymmetric`
- `CylindricSymmetric`

- Polar

These codes have been defined in the module `CoordinateSystems`.

Following are the several sections read in from the solver input file described earlier. Their content is basically name-value-pairs which are stored in list structures. The basic list structure does not have to be known to the user, but is manipulated by a set of utility routines to be described later.

```
!
!     Simulation input data, that concern the model as a whole
!
      TYPE(ValueList_t), POINTER :: Simulation
!
!     Variables
!
      TYPE(Variable_t), POINTER  :: Variables
!
!     Some physical constants, that will be read from the
!     input file or set by other means: gravity direction/
!     intensity and Stefan-Bolzmann constant)
!
      TYPE(ValueList_t), POINTER :: Constants
!
!     Types  of  equations (flow,heat,...) and  some
!     parameters (for example laminar or turbulent flow or
!     type of convection model for heat equation, etc.)
!
      INTEGER :: NumberOfEquations
      TYPE(EquationArray_t), POINTER :: Equations(:)
!
!     Active bodyforces: (bussinesq approx., heatsource,
!     freele chosen bodyforce...)
!
      INTEGER :: NumberOfBodyForces
      TYPE(BodyForceArray_t), POINTER :: BodyForces(:)
!
!     Initial conditions for field variables
!
      INTEGER :: NumberOfICs
      TYPE(InitialConditionArray_t), POINTER :: ICs(:)
!
!     Boundary conditions
!
      INTEGER :: NumberOfBCs
      TYPE(BoundaryConditionArray_t), POINTER :: BCs(:)
!
!     Material parameters
!
      INTEGER :: NumberOfMaterials
      TYPE(MaterialArray_t), POINTER :: Materials(:)
!
```

```
!     Active bodies, every element has a pointer to a body,
!     body has material,ICs,bodyforces and equations
!
      INTEGER :: NumberOfBodies
      TYPE(BodyArray_t), POINTER :: Bodies(:)
!
!     Equation solver control variables
!
      INTEGER :: NumberOfSolvers
      TYPE(Solver_t), POINTER :: Solvers(:)
```

Following are the mesh dimensions and pointers to structures describing the mesh:

```
!
!     Mesh dimensions
!
      INTEGER :: NumberOfBulkElements, &
                 NumberOfNodes,        &
                 NumberOfBoundaryElements


!
!     Node coordinates + info for parallel computations
!
      TYPE(Nodes_t) :: Nodes
!
!     Max number of nodes in any one element in this model
!
      INTEGER :: MaxElementNodes
!
!     Elements
!
      TYPE(Element_t),DIMENSION(:), POINTER :: Elements
```

The model structure also holds a pointer to the element currently being assembled, so that user routines may refer to this element:

```
!
!     For reference the current element in process
!
      TYPE(Element_t), POINTER :: CurrentElement
           ...
    END TYPE Model_t
```

K.1.2 The Element, Element Type and Node Structures

The structures holding elements are as follows:

```
   TYPE Element_t
     TYPE(ElementType_t), POINTER :: Type
     INTEGER :: BodyId
```

```
    TYPE(BoundaryInfo_t), POINTER :: BoundaryInfo
    INTEGER, DIMENSION(:), POINTER :: NodeIndexes
  END TYPE Element_t
```

The first thing in this structure is a pointer to element type (the entry `Type`), where information about the element basis functions, number of nodes for the element type, local nodal coordinates, etc., are kept. The element also holds an index to the body it belongs (`BodyId`). For boundary elements there is a separate additional structure (`BoundaryInfo`). Last there is the element connectivity information (`NodeIndexes`).

The additional information for boundary elements is kept in a structure of type `BoundaryInfo_t`:

```
  TYPE BoundaryInfo_t
    INTEGER :: Constraint
    INTEGER :: LeftBody,LeftElement
    INTEGER :: RightBody,RightElement
    TYPE(Factors_t) :: ViewFactors,GebhardtFactors
  END TYPE BoundaryInfo_t
```

For the boundary element, there is first the boundary condition id (`Constraint`), indeces of the bodies to which it belongs (`LeftBody, RightBody`) and the bulk elements it belongs (`LeftElement, RightElement`). It also holds pointers to view- and Gebhardt factors if radiation modelling is defined.

The element type structure is defined as follows:

```
  !
  ! Element type description
  !
  TYPE ElementType_t
    ! this is a list of types
    TYPE(ElementType_t),POINTER :: NextElementType

    ! numeric code for element
    INTEGER :: ElementCode

    INTEGER :: BasisFunctionDegree, & ! 1=linear, 2=quadratic
               NumberOfNodes, &
               Dimension              ! 1=line, 2=surface, 3=volume

    INTEGER :: GaussPoints ! number of gauss points to use

    DOUBLE PRECISION :: StabilizationMK ! stab.param. depending on
                                        ! interpolation type

    DOUBLE PRECISION, DIMENSION(:,:), POINTER :: BasisFunctions

    DOUBLE PRECISION, DIMENSION(:), POINTER :: NodeU,NodeV,NodeW
  END TYPE ElementType_t
```

Numeric codes for the element types (the variable `ElementCode` is assigned one of them for each element type) have beed defined in the element type definition file described in appendix L. The variable `Basis Functions` holds the basis function matrix formed at the initialization phase, and finally the arrays `NodeU, NodeV and NodeW` are the element type local coordinates.

The mesh nodes are stored in the following structure, only fields relevant to a single processor run are the coordinate arrays `x, y`, and `z`. The rest of the fields are used by the parallel solver only, and will not be defined for a single processor run.

```
!
!  Coordinate and vector type definition, coordinate
!  arrays must be allocated prior to use of variables
!  of this type
!
TYPE Nodes_t
  INTEGER :: NumberOfNodes,TotalNodes
  DOUBLE PRECISION, POINTER      :: x(:),y(:),z(:)
  LOGICAL, POINTER               :: Interface(:)
  TYPE(NeighbourList_t),POINTER  :: NeighbourList(:)
  INTEGER, POINTER               :: GlobalNodeNumber(:)
  INTEGER, POINTER               :: Perm(:),INVPerm(:)
  INTEGER :: NumberOfIfNodes
END TYPE Nodes_t
```

K.2 Retrieving Model Information Within User Routines

All the model input information: material parameters, boundary condition parameters, field variable values, etc. are available to user routines inthe same way as the solver gets hold of them.

User routines receive as the first argument the current model in a structure of type `Model_t` defined previously. This pointer may be used to get the parameter values through a set of `ListGet*` routines, and field variable values through the routine `VariableGet`. To use these routines the module `Lists` must be included in the program with the `USE` statement.

After the following variable definitions

- `LOGICAL :: L`
- `INTEGER :: I`
- `INTEGER, POINTER :: J(:)`
- `CHARACTER(LEN=MAX_NAME_LEN) :: S`
- `DOUBLE PRECISION :: P,Q(n)`
- `DOUBLE PRECISION, POINTER :: R(:,:), S(:,:,:)`

have been made, the `ListGet*` functions may be called as follows:

- `L = ListGetLogical( List, 'Parameter name', Exists )`

- `I = ListGetInteger( List, 'Parameter name', Exists )`
- `P = ListGetConstReal( List, 'Parameter name', Exists )`
- `Q(1:n) = ListGetReal( List, 'Parameter name', n, NodeIndexes, Exists )`
- `C = ListGetString( List, 'Parameter name', Exists )`
- `J => ListGetIntegerArray( List,' Parameter name', Exists )`
- `R => ListGetConstRealArray( List, 'Parameter name', Exists )`
- `S => ListGetRealArray( List, 'Parameter name', n, NodeIndexes, Exists )`

The first argument to these functions is a pointer to the list from which the values are to be searched, and may be one of the following (at least):

- `Model % Simulation`
- `Model % Constants`
- `Model % BCs(i) % Values`
- `Model % ICs(i) % Values`
- `Model % Bodies(i) % Values`
- `Model % Solvers(i) % Values`
- `Model % Materials(i) % Values`
- `Model % Equations(i) % Values`
- `Model % BodyForces(i) % Values`
- `Model % Boundaries(i) % Values`

The last argument `Exists` is an optional argument of type `LOGICAL` and will return true if the parameter existed. If the argument is not given, and the parameter doesn't exist, a warning message is given. One could also add parameter definitions to a list or change existing parameter values by using a different set of utility routines. Refer to the solver code if you want to do this.

As an example, a user routine might want to know the value of the density for the nodes of the element currently being assembled:

```
USE Types
USE Lists

TYPE( Element_t ), POINTER :: Elm
INTEGER mat,n
DOUBLE PRECISION :: Density(4) ! Assume max 4 nodes in an element

! Get the element pointer
Elm => Model % CurrentElement
```

```
! Material index for the element
mat = ListGetInteger( Model % Bodies(Elm % BodyId) %  &
           Values, 'Material' )

n = elm % Type % NumberOfNodes

! And finally, density values at the element nodal points
Density(1:n) = ListGetReal( Model % Materials(mat) % Values, &
           'Density', n, elm % NodeIndexes )
```

The field variables are also held in a list structure and may be retrieved by a call to the routine `VariableGet`

```
 USE Types
 USE Lists

TYPE(Variable_t), POINTER :: Variable
Variable => VariableGet( Model % Variables, 'Variable name' )
```

The structure `Variable_t` is defined as:

```
TYPE Variable_t
  TYPE(Variable_t), POINTER   :: Next
  CHARACTER(LEN=MAX_NAME_LEN) :: Name

  INTEGER :: DOFs
  INTEGER, POINTER            :: Perm(:)
  DOUBLE PRECISION, POINTER   :: Values(:),Norm
END TYPE Variable_t
```

The entry `Values` is a double precision array holding the variable values, `DOFs` is the number of components in the array (for example, in the flow solution array the number of components is three or four, i.e. the velocity components and pressure) , and the `Perm` array contains the possible nodal renumbering of the values. As an example, the following program segment might be used to get value of the x-velocity at nodal point `n`

```
USE Types
USE Lists

TYPE(Variable_t), POINTER :: Velocity
INTEGER :: k
DOUBLE PRECISION :: xvelo

Velocity => VariableGet( Model % Variables, 'Velocity 1' )

k = n
IF ( ASSOCIATED(Velocity % Perm) ) k = Velocity % Perm(k)

xvelo = 0.0D0
IF ( k /= 0 ) xvelo = Velocity % Values(k)
```

Note that both the components of the velocity field and the flow solution array containing the velocity field and pressure are entered in the list of variables by the solver. The same applies to the displacement field. Note also that the velocity might not be defined for all the nodal points. If a variable is not defined for a nodal point, the permutation vector `Perm` will contain zero at that position.

The whole set of variables entered in the list by the solver is:

- `Time`
- `Stress Solution`
- `Displacement 1`
- `Displacement 2`
- `Displacement 3`
- `Coordinate 1`
- `Coordinate 2`
- `Coordinate 3`
- `Temperature`
- `Flow Solution`
- `Pressure`
- `Velocity 1`
- `Velocity 2`
- `Velocity 3`
- `Magnetic Field`
- `Magnetic Field 1`
- `Magnetic Field 2`
- `Magnetic Field 3`
- `Electric Current`
- `Electric Current 1`
- `Electric Current 2`
- `Electric Current 3`

The time variable contains only a single value, the current simulation time. There is obviously no permutation vector for the coordinate arrays.

The user may add variables to the current list of variables by using the routine `VariableAdd`:

```
INTEGER :: DOFs
INTEGER, POINTER :: Perm
DOUBLE PRECISION, POINTER :: Values

CALL  VariableAdd( Model % Variables,'Variable name', &
                   DOFs,Values,Perm )
```

The last argument `Perm` is optional, and when provided gives a nodal renumbering of the variable values.

Appendix L

The Solver Element Types

The default element types of the ELMER solver are

- linear and quadratic triangles (3, 6, and 7 nodal points, Figure L.1),

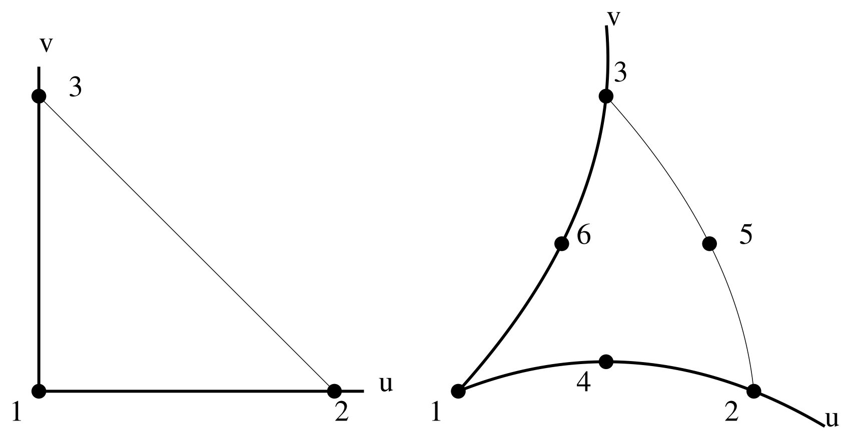

Figure L.1: The 3 and 6 node triangular elements.

- bilinear and quadratic quadrilaterals (4,5,8, and 9 nodal points, Figure L.2),
- linear and quadratic tetrahedrons (4 and 10 nodal points, Figure L.3),
- trilinear and quadratic hexahedrons (8, 20, and 27 nodal points, Figure L.4).

Further element types belonging to these basic topologies may be added to the program by editing element type definition file, without touching the solver code nor compiling or linking the program. How to create a mesh using these element types is a separate question.

The definition file is read in when the solver code is initialized and a basis function matrix is formed for every element type defined. The rest of the FEM

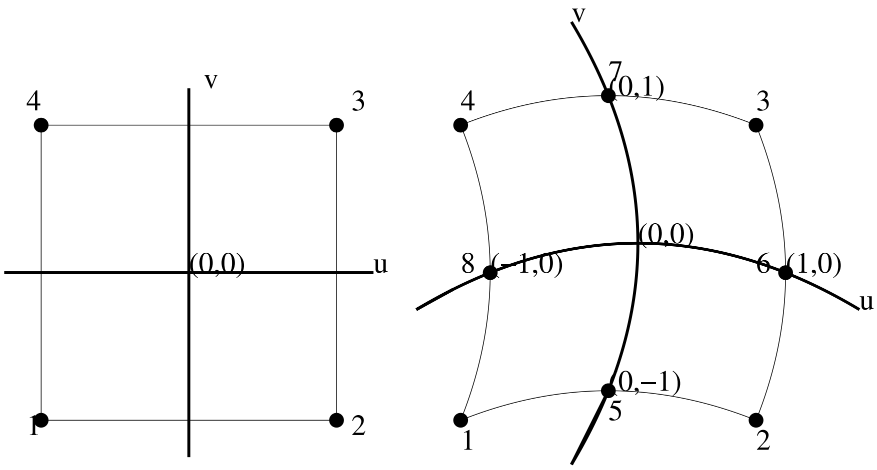

Figure L.2: The 4, and 8 node quadrilateral elements.

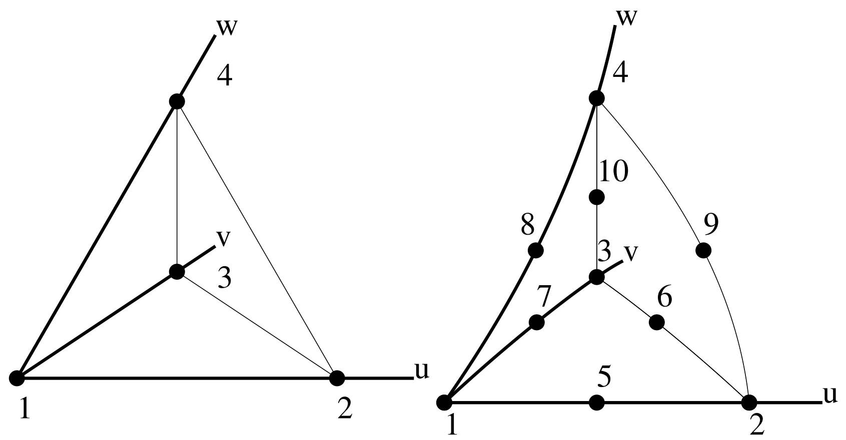

Figure L.3: The 4 and 10 node tetrahedron elements.

code (computing local and global derivates, integration of elements) is then basically independent of the element type.

The following is listing of the default element type definition file.

```
!/*****************************************************************************
! *
! *       ELMER, A Computational Fluid Dynamics Program.
! *
! *       Copyright 1st April 1995 - , Center for Scientific Computing,
! *                                    Finland.
! *
! *       All rights reserved. No part of this program may be used,
! *       reproduced or transmitted in any form or by any means
! *       without the written permission of CSC.
! *
```

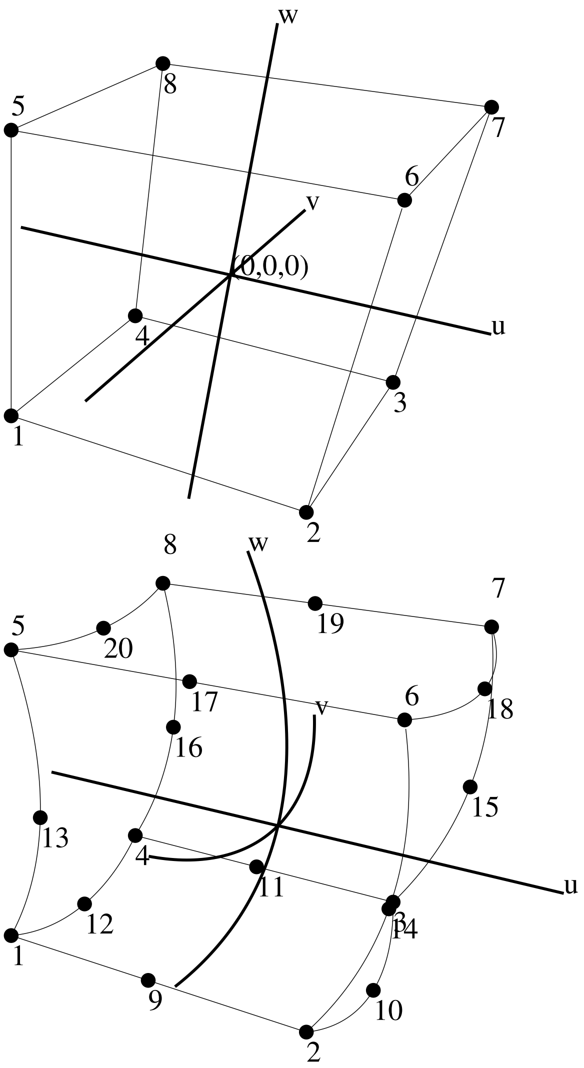

Figure L.4: The 8 and 20 node hexahedron elements.

```
! *****************************************************************************/
!
!/*****************************************************************************
! *
! *  Element type definitions, read in the initialization phase
! *
! *****************************************************************************
! *
! *                     Author       Juha Ruokolainen
! *
! *                    Address Center for Scientific Computing
! *                                Tietotie 6, P.O. BOX 405
! *                                  02101 Espoo, Finland
```

```
! *                                 Tel. +358 0 457 2723
! *                               Telefax +358 0 457 2302
! *                             EMail Juha.Ruokolainen@csc.fi
! *
! *                      Date 04 Oct 1996
! *
! *               Modified by
! *
! *      Date of modification
! *
! ******************************************************************************/


!-------------------------------------------------------------------------------

ELEMENT 1 Node Point
!
Dimension 1
Topology Point
!
Code 101
!
Nodes 1
!
Node U 0.0
!
Basis 1
!
Gauss Points 0
!
Stabilization 0.0
END ELEMENT

!-------------------------------------------------------------------------------

ELEMENT 2 Node Line
!
Dimension 1
Topology Line
!
Code 202
!
Nodes 2
!         1    2
Node U  -1.0  1.0
!

!       1  2   3
!       1  u  u^2
Basis   1 2
```

```
Gauss Points 2

Stabilization  0.333333333333333333
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 3 Node Line
!
Dimension 1
Topology Line
!
Code 203
!
Nodes 3
!         1    2   3
Node U  -1.0  1.0 0.0
!

!       1  2   3
!       1  u  u^2
Basis   1 2 3

Gauss Points 3

Stabilization  0.1666666666666666666
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 3 Node Triangle
!
Dimension 2
Topology Triangle
!
Code 303
!
Nodes 3
!        1    2    3
Node U  0.0  1.0  0.0
Node V  0.0  0.0  1.0

!
!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis   1 2 4

Gauss Points 1
```

```
Stabilization 0.33333333333333333333
END ELEMENT

!-------------------------------------------------------------------------------

ELEMENT 4 Node Triangle
!
Dimension 2
Topology Triangle
!
Code 304
!
Nodes 4
!        1    2    3            4
Node U  0.0  1.0  0.0  0.333333333333333333
Node V  0.0  0.0  1.0  0.333333333333333333
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis   1 2 4 5

Gauss Points 3    (?????)

Stabilization  0.333333333333333333
END ELEMENT

!-------------------------------------------------------------------------------

ELEMENT 6 Node Triangle
!
Dimension 2
Topology Triangle
!
Code 306
!
Nodes 6
!
!        1    2    3    4    5    6
Node U  0.0  1.0  0.0  0.5  0.5  0.0
Node V  0.0  0.0  1.0  0.0  0.5  0.5
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis 1 2 3 4 5 7

Gauss Points 4

Stabilization  0.041666666666666666
```

```
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 7 Node Triangle
!
Dimension 2
Topology Triangle
!
Code 307
!
Nodes 7
!
!        1    2    3    4    5    6           7
Node U  0.0  1.0  0.0  0.5  0.5  0.0  0.3333333333333333
Node V  0.0  0.0  1.0  0.0  0.5  0.5  0.3333333333333333
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis 1 2 3 4 5 7 9

Gauss Points 6  (??????)

Stabilization  0.0416666666666666666
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 4 Node Quadrilateral
!
Dimension 2
Topology Quad
!
Code 404

Nodes 4
!         1    2    3    4
Node U  -1.0  1.0  1.0 -1.0
Node V  -1.0 -1.0  1.0  1.0
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis   1 2 4 5

Gauss Points 4

Stabilization  0.333333333333333333
END ELEMENT
```

```
!--------------------------------------------------------------------------------

ELEMENT 5 Node Quadrilateral
!
Dimension 2
Topology Quad
!
Code 405

Nodes 5
!         1    2    3    4    5
Node U  -1.0  1.0  1.0 -1.0  0.0
Node V  -1.0 -1.0  1.0  1.0  0.0
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis   1 2 4 5 9

Gauss Points 9   (???????)

Stabilization  0.08148148148148
END ELEMENT

!--------------------------------------------------------------------------------

ELEMENT 8 Node Quadrilateral
!
Dimension 2
Topology Quad
!
Code 408

Nodes 8
!         1    2    3    4    5    6    7     8
Node U  -1.0  1.0  1.0 -1.0  0.0  1.0  0.0  -1.0
Node V  -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0   0.0
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis  1 2 3 4 5 6 7 8

Gauss Points 9   !  ( ?????? )

Stabilization  0.08148148148148
END ELEMENT

!--------------------------------------------------------------------------------
```

```
ELEMENT 9 Node Quadrilateral
!
Dimension 2
Topology Quad
!
Code 409

Nodes 9
!        1    2    3    4    5    6    7     8    9
Node U  -1.0  1.0  1.0 -1.0  0.0  1.0  0.0  -1.0  0.0
Node V  -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0   0.0  0.0
!

!       1  2   3   4   5   6     7    8      9
!       1  u  u^2  v  uv  u^2v  v^2  uv^2  u^2v^2
Basis 1 2 3 4 5 6 7 8 9

Gauss Points 9

Stabilization  0.08148148148148
END ELEMENT

!--------------------------------------------------------------------------------

ELEMENT 4 Node Tetrahedron
!
Dimension 3
Topology Tetra
!
Code 504

Nodes 4
!        1    2    3    4
Node U   0.0  1.0  0.0  0.0
Node V   0.0  0.0  1.0  0.0
Node W   0.0  0.0  0.0  1.0
!

!
! 1     2     3      4     5      6       7       8        9
! 1     u    u^2     v     uv     u^2v    v^2     uv^2     u^2v^2
!
! 10   11    12     13    14      15      16      17        18
! w    uw   u^2w    vw    uvw    u^2vw   v^2w    uv^2w    u^2v^2w
!
! 19   20     21     22    23      24       25      26         27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis  1 2 4 10
```

```
Gauss Points 4

Stabilization  0.333333333333333333
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 10 Node Tetrahedron
!
Dimension 3
Topology Tetra
!
Code 510

Nodes 10
!          1    2    3    4    5    6    7    8    9   10
Node U   0.0  1.0  0.0  0.0  0.5  0.5  0.0  0.0  0.5  0.0
Node V   0.0  0.0  1.0  0.0  0.0  0.5  0.5  0.0  0.0  0.5
Node W   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.5  0.5  0.5
!

!
! 1     2     3      4     5      6       7       8        9
! 1     u    u^2     v     uv     u^2v     v^2     uv^2     u^2v^2
!
! 10   11    12     13    14      15      16       17       18
! w    uw   u^2w    vw    uvw    u^2vw    v^2w     uv^2w    u^2v^2w
!
! 19   20     21     22    23      24       25      26       27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis 1 2 3 4 5 7 10 11 13 19


Gauss Points 10  ! (???????)

Stabilization  0.333333333333333333
END ELEMENT

!------------------------------------------------------------------------------

ELEMENT 8 Node Octahedron
!
Dimension 3
Topology  Brick
!
Code 808
!
```

```
Nodes 8
!         1    2    3    4     5    6    7    8
Node U   -1.0  1.0  1.0 -1.0  -1.0  1.0  1.0 -1.0
Node V   -1.0 -1.0  1.0  1.0  -1.0 -1.0  1.0  1.0
Node W   -1.0 -1.0 -1.0 -1.0   1.0  1.0  1.0  1.0
!

!
! 1     2     3      4     5       6        7       8         9
! 1     u    u^2     v     uv     u^2v     v^2     uv^2     u^2v^2
!
! 10   11    12     13    14       15       16       17        18
! w    uw   u^2w    vw    uvw    u^2vw    v^2w     uv^2w    u^2v^2w
!
! 19   20     21     22    23       24       25       26        27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis 1 2 4 5 10 11 13 14

Gauss Points 8

Stabilization  0.333333333333333333
END ELEMENT

!-------------------------------------------------------------------------------

ELEMENT 20 Node Octahedron
!
Dimension 3
Topology  Brick
!
Code 820
!
Nodes 20
!         1    2    3    4    5    6    7    8    9    10   11  12
Node U   -1.0  1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 -1.0 \
         -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 -1.0
!         13   14   15   16   17   18   19   20

Node V   -1.0 -1.0  1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 \
         -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0

Node W   -1.0 -1.0 -1.0 -1.0  1.0  1.0  1.0  1.0 -1.0 -1.0 -1.0 -1.0 \
          0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0
!

!
! 1     2     3      4     5       6        7       8         9
! 1     u    u^2     v     uv     u^2v     v^2     uv^2     u^2v^2
!
```

```
! 10   11    12      13    14       15      16       17        18
! w    uw   u^2w    vw    uvw    u^2vw    v^2w     uv^2w    u^2v^2w
!
! 19   20      21      22    23       24       25      26        27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis 1 2 3 4 5 6 7 8 10 11 12 13 14 15 16 17 19 20 22 23

Gauss Points 27

Stabilization  0.08148148148148
END ELEMENT

!--------------------------------------------------------------------------------

ELEMENT 27 Node Octahedron
!
Dimension 3
Topology  Brick
!
Code 827
!
Nodes 27
!          1    2    3    4    5    6    7    8    9    10   11  12
Node U   -1.0  1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 -1.0 \
         -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 -1.0  0.0  1.0  0.0 -1.0 \
          0.0  0.0  0.0
!         13   14   15   16   17   18   19   20   21   22   23   24
!         25   26   27

Node V   -1.0 -1.0  1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 \
         -1.0 -1.0  1.0  1.0 -1.0  0.0  1.0  0.0 -1.0  0.0  1.0  0.0 \
          0.0  0.0  0.0

Node W   -1.0 -1.0 -1.0 -1.0  1.0  1.0  1.0  1.0 -1.0 -1.0 -1.0 -1.0 \
          0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0  0.0  0.0  0.0 \
         -1.0  1.0  0.0
!

!
! 1     2     3      4     5       6        7       8         9
! 1     u    u^2     v     uv     u^2v     v^2     uv^2     u^2v^2
!
! 10   11    12      13    14       15      16       17        18
! w    uw   u^2w    vw    uvw    u^2vw    v^2w     uv^2w    u^2v^2w
!
! 19   20      21      22    23       24       25      26        27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
```

```
Gauss Points 64

Stabilization  0.08148148148148
END ELEMENT

!--------------------------------------------------------------------------------

ELEMENT 6 Node Wedge
!
Dimension 3
Topology Wedge
!
Code 606

Nodes 6
!         1    2    3    4    5    6
Node U   0.0  1.0  0.0  0.0  1.0  0.0
Node V   0.0  0.0  1.0  0.0  0.0  1.0
Node W   0.0  0.0  0.0  1.0  1.0  1.0
!


!
! 1     2     3      4     5       6        7       8         9
! 1     u    u^2     v     uv     u^2v     v^2     uv^2     u^2v^2
!
! 10   11    12     13    14       15       16       17        18
! w    uw   u^2w    vw    uvw    u^2vw    v^2w     uv^2w    u^2v^2w
!
! 19   20     21     22    23       24       25      26        27
! w^2 uw^2  u^2w^2  vw^2  uvw^2  u^2vw^2  v^2w^2  uv^2w^2  u^2v^2w^2
!
Basis 1 2 4 10 11 13

Gauss Points 1

Stabilization  0.333333333333333333
END ELEMENT

!--------------------------------------------------------------------------------
```

Bibliography

[1] L.P. Franca, S.L. Frey, T.J.R. Hughes, *Computer methods in Applied Mechanics and Engineering* **95** (1992) 253–276

[2] L.P. Franca, S.L. Frey, *Computer methods in Applied Mechanics and Engineering* **99** (1992) 209–233

[3] Richard Barrett et al.,*Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1993

[4] Roland W. Freund and Noël Nachtigal, *QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems*, Numer. Math. 60, 315-339, 1991

[5] Roland W. Freund, *An Implmentation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices*, SIAM J. Sci. Comput., Vol. 14, No. 1, pp. 137-158, January 1993

[6] Roland W. Freund, *A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems*, SIAM J. Sci. Comput., Vol. 14, No. 2, pp. 470-482, March 1993

[7] Roland W. Freund and Noël Nachtigal, *An Implementation of the QMR Method Based on Coupled Two-Term Recurrences*, SIAM J. Sci. Comput., Vol. 15, No. 2, pp. 313-337, March 1994

[8] H. Martin Bücker and Manfred Sauren, *A Parallel Version of the Unsymmetric Lanczos Algorithm and its Application to QMR*, Forschungszentrum Jülich, March 1996

[9] Yousef Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1996

[10] Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company Inc., 1994.