FEM$\bigcirc\!\!\!\bigcirc$BEM 3.0

A Parallel Solver for Linear and Nonlinear Coupled FE/BE-Equations

G. Haase, B. Heise, M. Jung, M. Kuhn, O. Steinbach

Bericht Nr. 94–16

Institut für Mathematik, Altenberger Str. 69, A-4040 Linz
Tel.: +43-(0)732 2468-9164

Koordinator: Prof. Dr.-Ing. W. L. Wendland

Contents

Chapter 1

Introduction

At the present time high effort is being paid in both, developing and implementing parallel algorithms. The program FEM⚭BEM presented here is part of this ongoing work and unites several results achieved very recently concerning the development of preconditioning techniques, domain decomposition methods and the coupling of Finite Element Methods and Boundary Element Methods. The latter is of special interest if exterior problems are to be solved.
The present version of FEM⚭BEM can be used to solve 2d linear and nonlinear magnetic field problems as well as 2d linear elasticity problems. The first class of problems can be described formally by the following equations:

$$\begin{aligned}
-\mathrm{div}(\nu(x,|\nabla u|)\nabla u(x) &= f(x) \quad x \in \Omega \\
u(x) &= 0 \quad x \in \Gamma_D \\
\frac{\partial u(x)}{\partial n} &= g(x) \quad x \in \Gamma_N \\
\frac{\partial u(x)}{\partial n} &= \kappa(x)(u(x) - h(x)) \quad x \in \Gamma_3,
\end{aligned}$$

where Ω is a bounded or unbounded region in R^2 and $\partial\Omega := \Gamma_D \cup \Gamma_N \cup \Gamma_3$ is the boundary of Ω. We assume Γ_D not to be empty. As indicated above nonlinear equations can be solved, where the term $\nu(x,|\nabla u|)$ causing the nonlinearity must satisfy certain conditions to guarantee the existence and uniqueness of a solution (see chapter 2).
In plane linear elasticity problems we seek the displacement $u(x) = (u_1(x), u_2(x))^T$ which satisfies formally the system of Lamé equations

$$\begin{array}{l}
-\mu(x)\Delta u(x) - (\lambda(x) + \mu(x))\mathrm{grad\ div}u(x) = f(x) \quad \text{in } \Omega \\
u(x) = 0 \text{ on } \Gamma_D, \quad \sum_{l=1}^{2} \sigma_{kl}(u(x))n_l = g_k(x) \text{ on } \Gamma_N, \ (k = 1, 2)
\end{array} \tag{1.1}$$

where Ω is a bounded Lipschitz domain, $\sigma_{kl}(u)$ are the components of the stress tensor $\sigma(u)$ and $n(x) = (n_1(x), n_2(x))^T$ is the outward normal vector to $\Gamma_D \cup \Gamma_N = \Gamma := \partial\Omega$ ($\Gamma_D \neq \emptyset$) and λ and μ, $\lambda, \mu > 0$, are the Lamé coefficients of the elastic material. In (2.5), $f = (f_1, f_2)^T$ is the vector of volumic forces, $g = (g_1, g_2)^T$ is the vector of boundary tractions. A selection of examples for both classes of problems can be found in chapter 2.
The main goal of FEM⚭BEM is to provide an efficient solver for linear and nonlinear systems arising from discretizing the according partial differential equation. The documentation contains a detailed description of the libraries and of important routines allowing the implementation of the program, or in particular of the solver, into pre- and post-processing packages.

The program has been developed for MIMD computers and the results being presented in section 2.3 have been obtained on a PowerXplorer and on a GC-PowerPlus *(PARSYTEC)*.
This documentation gives an overview of the libraries belonging to FEM⊗BEM 3.0 . The libraries have been developed by several people, so that problems which may occur should be addressed directly to the appropriate person. A list of the people involved in developing the program can be found in appendix D.
The paper continues with general remarks on the mathematical background and on the program (including several examples) in chapter 2. Then, in chapter 3, the data structures and the input files required for running the program are being described. Appendix A gives a short introduction to the preprocessor ADDPre [11] which realizes automatically the DD-preprocessing.
The subsequent chapters 4-9 contain a description of the libraries in the following sequence.

- general libraries
- GiGeHi (hierarchical grid generation)
- MGSolv (multigrid algorithm)
- GSGen (generation of the (FE-) stiffness matrix)
- NewtonSolv (Newton algorithm)
- OstBEM2d (BEM matrix generation)
- GeBEM and CoBEM (BEM libraries)

An important tool, the adaptive DD–preprocessor **ADDPre**, is introduced in appendix A. It reaizes an alomost automatic decomposition of the domain into an arbitrary number of subdomains. Remarks on recent developments are given in appendix B. An overview of the pointer variables being used can be found in appendix C.
For more details concerning the mathematical theory, a complete description of the examples, pre- and post-processing we refer to the cited papers.

Chapter 2

FEM$\oplus$BEM - A Basic Description

2.1 Mathematical background

The aim of this section is to give an overview of the mathematical components being used. The details can be found in the cited papers.
The algorithm which has been implemented brings together ideas from Finite Element Methods [42], [47], Boundary Element Methods [27],[6], Domain Decompositions techniques [1, 25, 23, 22, 15], Multigrid Methods [45] and Newton or quasi-Newton Methods [39].
Formally the problems handled by FEM$\oplus$BEM can be described by the following differential equations:

$$
\begin{aligned}
-\mathrm{div}(\nu(x, |\nabla u|)\nabla u(x)) &= f(x) \quad x \in \Omega \\
u(x) &= 0 \quad x \in \Gamma_D \\
\frac{\partial u(x)}{\partial n} &= g(x) \quad x \in \Gamma_N \\
\frac{\partial u(x)}{\partial n} &= \kappa(x)(u(x) - h(x)) \quad x \in \Gamma_3,
\end{aligned}
$$

for magnetic field problems and

$$
\begin{array}{l}
-\mu(x)\Delta u(x) - (\lambda(x) + \mu(x))\mathrm{grad}\ \mathrm{div} u(x) = f(x) \quad \text{in } \Omega \\
u(x) = 0 \ \text{ on } \Gamma_D, \quad \sum_{l=1}^{2} \sigma_{kl}(u(x))n_l = g_k(x) \ \text{ on } \Gamma_N, \ (k = 1, 2)
\end{array}
\tag{2.1}
$$

for linear elasticity problems.
The problems given on a bounded or unbounded region Ω are solved via a coupled FE/BE-discretization. For this purpose the domain is being decomposed into non-overlapping subdomains, including an exterior domain if necessary. This decomposition forms the basis for our parallel algorithm [44]. If a subdomain is to be handled as FE- or BE-domain has to be specified by the user, exterior domains have to be marked as BE-domains. The boundary between interior and exterior domain is called coupling boundary. In practice it is advantageous to choose a circle as coupling boundary. After discretizing the problem locally, according to the method chosen before, we obtain a system of equations. For the first class of problems this system may be linear or nonlinear depending on the properties of the function $\nu(.,.)$. A discussion of the nonlinear problem, that is if ν depends on $|\nabla u(x)|$ can be found in [32], [33]. The linear problem systems arising within the iterative procedure for solving the nonlinear problem are being solved by the PCCG-algorithm [44] which can cope with coupled FE/BE-equations. The coupling of the two methods has been described in [43]. The efficiency of the linear solver is based upon the knowledge of good preconditioners, that is of the operators C_C, C_I and C_Λ as the occur in in the algorithm. A discussion of these problems can be found, e.g., in [20, 12, 53].

2.2 Running the program

The input data describing the discretization have to be provided in a standardized file which will be defined in section 3.1.2. These data determine uniquely the grading of the coarsest mesh. Input files can be generated using the pre-processing codes described in [11]. These tools include an semi-automatic code based on the spectral bisection method for decomposing the geometrical data.
Furthermore the components of the preconditioner within the cg-algorithm can be chosen, that is the components of the basis transformation B_I and the inner solver C_I, as they occur in [13], have to be defined. This is being done in dialog with the user. Thus, the following data have to be provided after starting the program.

Name of the file:	Enter the name of the input file (geometrical data).
Number of levels:	Number of grids in the multigrid algorithm, the coarsest grid is defined by the data given in the input file.

The components of both, the basis-transformation B_I and the inner solver C_I can be defined as follows.

MG-cycle:	Kind of the multigrid cycle (1 $\widehat{=}$ V-cycle and 2 $\widehat{=}$ W-cycle).
Number of iterations:	Number of multigrid iterations (0 is sufficient for the basistransformation and leads to one hierarchical transformation without a multigrid step, 1 is sufficient for the inner solver).
Pre-smoothing:	Kind of the pre-smoothing within the multigrid-algorithm (1 $\widehat{=}$ Gauss-Seidel-smoothing, 2 $\widehat{=}$ ω-Jacobi-smoothing).
Post-smoothing :	Kind of the post-smoothing within the multigrid-algorithm (1 $\widehat{=}$ Gauss-Seidel-smoothing, 2 $\widehat{=}$ ω-Jacobi-smoothing).
Number of pre-smoothing steps:	Number of pre-smoothing steps within the multigrid-algorithm.
Number of post-smoothing steps:	Number of post-smoothing steps within the multigrid-algorithm.
Full-multigrid:	Decision if the full-multigrid technique is to be used ($\widehat{=}$ 1) or not ($\widehat{=}$ 0).
Generalized cycle:	Decision if the generalized V-cycle (i.e. doubling the number of pre-smoothing steps at every coarser grid) is to be used ($\widehat{=}$ 1) or not ($\widehat{=}$ 0).

Finally, a file describing the problem itself (linear, nonlinear) and controlling the solver process has to be supplied.

Name of the file:	Enter the name of the input file (controlling data).

In the case of linear problems the file *linear.nc* can be used, for nonlinear problems the file must be adapted to the problem.
During the process of solving the equation every cg-iteration is confirmed by printing the current cg-rate. In the case of nonlinear calculations the newton-iteration is documented.
Finally the program can be restarted by typing "1", whereas typing "0" quits the program.
If the program has been started within the *xrun-* environment [3] it is possible to generate pictures of the mesh and of the solution as they are presented in section 2.3 or to generate postscript files. Alternatively the library *libgrapg.a* [17] may be used for graphical output. It allows an interactive post-processing.

2.3 Examples – Magnetic Field Problems

In this section there we want to give several examples demonstrating the capabilities and the performance of the program FEM$\otimes$BEM . Two examples (magnet and motor) are magnetic field problems, whereas the third example (Kawohl problem) is an optimal shape design problem which can be described by the same class of equations. In the case of the magnet we present several combinations of coupling FEM and BEM in order to demonstrate the capabilities of the program. Of course, the main advantage of coupling the two methods is, that a more realistic model can be used, in particular no artificial Dirichlet boundary has to be defined. The error bound ($(w,r)-$defect, i.e. in the discrete $KC^{-1}K$-norm with the system matrix K and the preconditioner C) of the cg-algorithm has been set to 10^{-6}. Nonlinear calculations are carried out as nested iteration.

2.3.1 The Electromagnet

Here, we want to simulate the magnetic field caused by an electromagnet as shown in figure 2.1 (see also [34, 39, 36, 37]). The problem can be described formally by the following differential equation

$$\begin{aligned} -\mathrm{div}(\nu(x,|\nabla u|)\nabla u(x)) &= f(x) \quad x \in R^2 \\ |u(x)| &\to 0 \quad \text{for } |x| \to \infty, \end{aligned}$$

where the unknown u represents the vector potential A. Usually we seek the induction B which can be obtained from $u = A$ by

$$\vec{B} = \mathrm{rot}\vec{A}, \tag{2.2}$$

where u is the solution of the corresponding weak formulation. Note, that we are given no boundary conditions, that is the domain is unbounded. That is why we are going to use a coupled Finite Element and Boundary Element model based upon domain decomposition into inner domains and one exterior domain. On the inner domain we use either FEM or BEM (the latter only if certain conditions are fulfilled) and on the exterior domain BEM is being used. The decomposition is shown in figure 2.1.
Let us consider the linear problem first, that is $\nu(.)$ depends on x but not on u or derived quantities. Let ν be defined by

$$\begin{aligned} \nu_{copper} &= 795779.0\frac{Am}{Vs} \\ \nu_{air} &= 795774.4\frac{Am}{Vs} \\ \nu_{iron} &= 1000.0\frac{Am}{Vs}. \end{aligned}$$

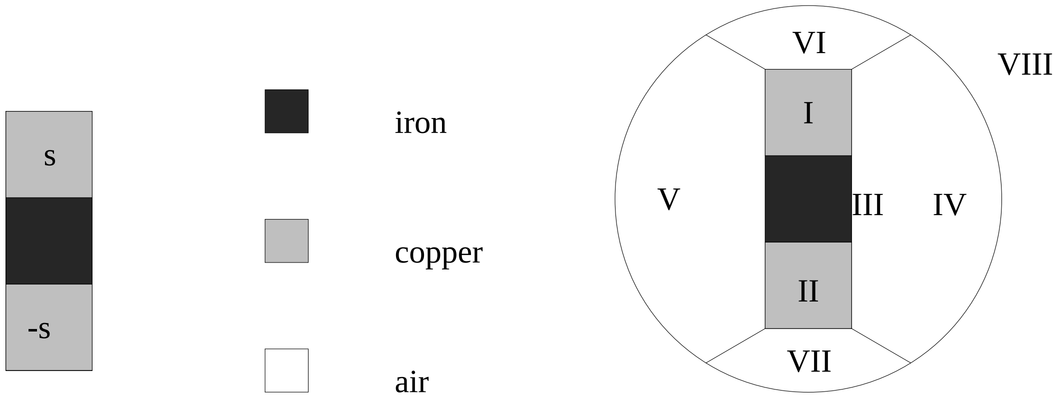

Figure 2.1: FE-coupling boundary (left) and circle as coupling boundary (right).

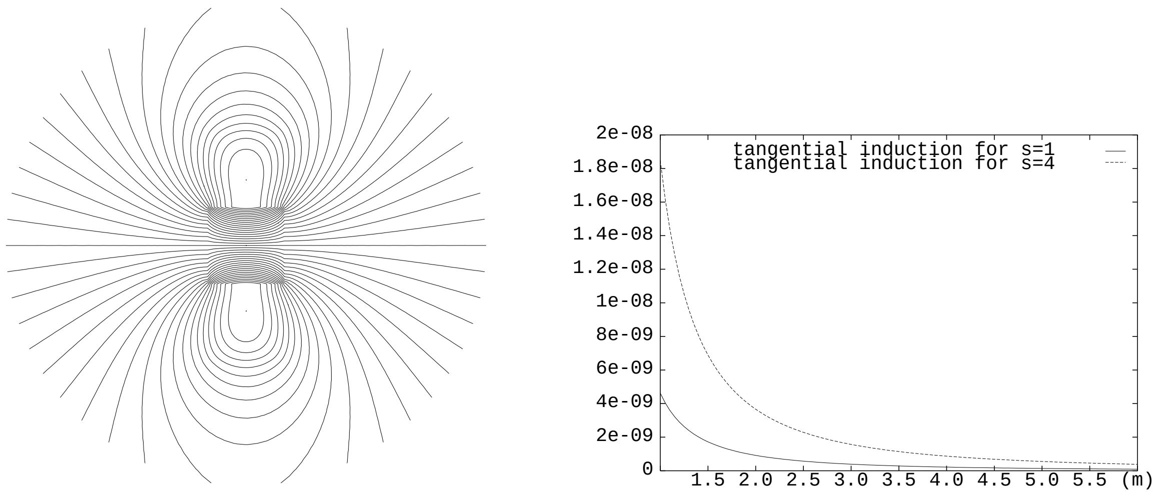

Figure 2.2: Equipotential lines and far field behaviour.

The equipotential lines of u and the far field behaviour of the tangential component of B have been drawn in figure 2.2. In the following we test the program by using different coupling models.

<u>Model 1.</u>

- 8 subdomains (FEM: I-VII, BEM: VIII)
- circle as coupling boundary

Here, we have defined the coupling boundary between interior and exterior domain to be a circle. If this circle is discretized uniformly the discrete BEM-operators become circulant matrices which are represented by the vector of their eigenvalues. By using this technique two advantages can be achieved. First, the time for generating the operators can be neglected (the eigenvalues are stored in a file) and, second, the matrix-times-vector operations can be performed via Fast Fourier Transformations.

grid	unknowns	No. it.	time (s)
1	309	16	0.7
2	1137	19	1.0
3	4353	20	1.7
4	17025	21	6.1
5	67329	24	23.5

Model 2.

- 8 subdomains (FEM: I-III, BEM: IV-VIII)
- circle as coupling boundary

grid	unknowns	No. it.	BE-system	time (s)
1	249	17	0.1	0.7
2	621	18	0.2	0.8
3	1725	19	0.7	1.3
4	5373	21	2.8	2.4
5	18429	23	11.2	7.2

Here, we demonstrate the coupling of FEM and BEM for inner domains. We observe a higher number of cg-steps caused by a deterioration of spectral equivalence constants.

Model 3.

- 4 subdomains (FEM: I-III, BEM: exterior domain)
- FE-boundary as coupling boundary

grid	unknowns	No. it.	BE-system	time (s)
1	109	14	0.1	0.2
2	337	17	0.2	0.3
3	1153	19	0.9	0.6
4	4225	20	3.7	1.9
5	16129	22	15.0	7.0

This model shows, that a non-circular coupling boundary can be used. Now, the discrete BE-operators are fully populated and do need computation time to be generated. The advantage of this model is, that no additional FE-domains (air) have to be defined. The solution at any point in the exterior domain can be obtained by applying Green's representation formula.
Finally, we want to consider the nonlinear problem, that is, we assume ν to depend on x and on u for the ferromagnetic material. Then, $\nu(.)$ is described by discrete values given in the .nc file (chapter 3) and interpolated appropriately [32].

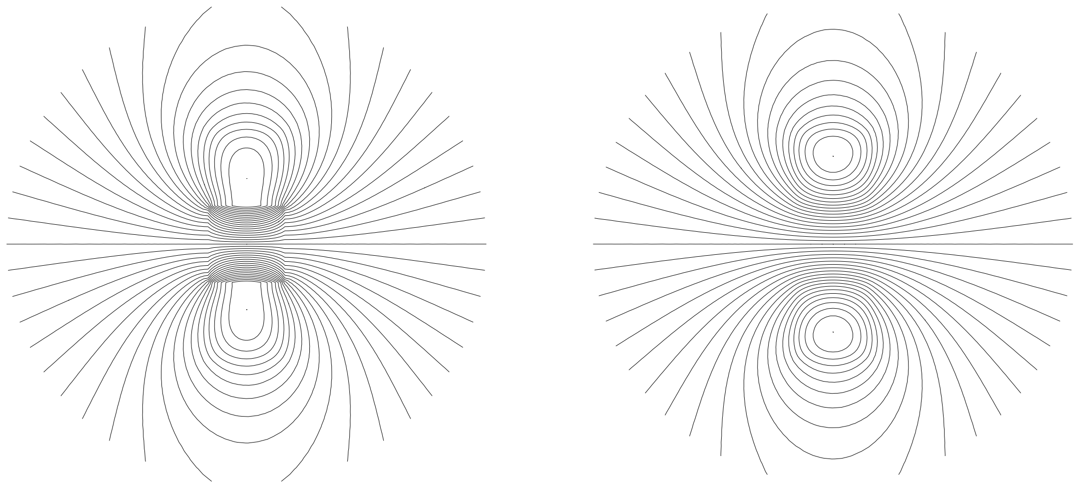

Figure 2.3: Equipotential lines for two different current densities, $1\text{A}\,\text{mm}^{-2}$ (left) and $1000\text{A}\,\text{mm}^{-2}$ (right).

Nonlinear Problem.

- 8 subdomains (FEM: I-VII, BEM: VIII)
- circle as coupling boundary

Table 2.1: Results for a current density $1\text{A}\,\text{mm}^{-2}$.

grid	unknowns	No. of Newton it.	No. cg-it. for each Newton step	time (s)
1	309	3	6,6/12	0.8
2	1137	2	6,7	0.6
3	4353	2	6,7	1.1
4	17025	3	6,8/15	6.8

Total time: 12.2

2.3.2 The Electric Motor

As a more challenging example for an exterior problem we consider an electric motor. It consists of four permanent magnets of alternating magnetization, a rotor with 4 wings, regions of air and an iron coating. figure 2.4 shows the mesh being used. The equipotential lines of B have been drawn in figure 2.4 (see also [35, 34, 37]). Again, we solve the following partial differential equation in a weak sense

$$\begin{aligned}
-\mathrm{div}(\nu(x,|\nabla u|)\nabla u(x)) &= S_z - \frac{\partial H_{0y}}{\partial x} + \frac{\partial H_{0x}}{\partial y} \quad \forall x \in \Omega_- \\
-\mathrm{div}(\nu(x)\nabla u(x)) &= 0 \quad \forall x \in \Omega_+ := R^2 \setminus \overline{(\Omega_- \cup \Omega_0)} \\
u(x) &= 0 \quad \forall x \in \Gamma_D := \partial\Omega_0 \\
|u(x)| &\to 0 \quad \text{for } |x| \to \infty,
\end{aligned}$$

where $\nu(.)$ depends on $|\nabla u|$ for ferromagnetic materials and $diam(\Omega_- \cup \Omega_0) < 1$.

Table 2.2: Results for a current density $1000\text{A}\,\text{mm}^{-2}$.

grid	unknowns	No. of Newton it.	No. cg-it. for each Newton step	time (s)
1	309	8	6,6,6,6, 6,6,6/11	2.8
2	1137	2	5, 7	0.5
3	4353	2	5,7	1.0
4	17025	3	6,8/15	7.7

Total time: 15.1

Motor (Dirichlet Boundary conditions).

- Homogeneous Dirichlet conditions are set at the boundary of the machine.
- 64 subdomains (FEM: 1-64)

Table 2.3: Motor with Dirichlet boundary (FEM).

grid	unknowns	No. of Newton it.	No. cg-it. for each Newton step	time (s)
1	1460	4	6, 11, 15, 11	5.1
2	5864	2	9, 11	1.6
3	23576	2	10, 12	2.0
4	94616	2	11, 13	3.2
5	378136	4	11,15,9,16	17.6

Total time: 37.6

Motor (with infinite outer domain).

- the infinite outer subdomain is discretized by BEM
- 64 subdomains (FEM: 1-63, BEM: 64)
- circle as coupling boundary

2.3.3 The Kawohl problem

This example is to show that the program FEM⚭BEM is efficient as well for nonlinear problems arising in quite other fields of research.
Consider the problem in $\Omega = (0,1) \times (0,1) \subset \mathcal{R}^2$:
Find $u \in V = \overset{\circ}{W}{}_2^1(\Omega)$ *such that*

$$\int_\Omega \psi(|\nabla u|)\ \nabla^\top u\ \nabla v\ dx = \int_\Omega v\ dx \quad \forall v \in V \tag{2.3}$$

Table 2.4: Motor with infinite outer domain, circular coupling.

grid	unknowns	No. of Newton it.	No. cg-it. for each Newton step	time (s)
1	1501	4	6, 11, 16, 13	6.0
2	5820	2	9, 11	1.9
3	22936	2	10 , 13	2.5
4	91080	2	11, 14	4.1
5	366088	4	11,16,11,16	19.9

Total time: 43.4

where

$$\psi(z) = \begin{cases} 2 & \text{if } z^2 \leq 0.008 \\ 1 & \text{if } z^2 \geq 0.032 \\ \sqrt{0.032}/z & \text{else.} \end{cases} \tag{2.4}$$

The problem arises in the theory of elasticity as follows (see [41, 10, 31, 34, 37]). We want to construct an infinitely long elastic bar of given cross–section Ω with maximal torsional rigidity from two different linearly elastic materials of given shear moduli. The proportions of these materials are prescribed. The problem has nearly the same form as the problem of nonlinear magnetism but it has not the same monotonicity properties, cf.[32].

Kawohl problem.

- 64 subdomains (FEM: 1-64)

grid	unknowns	No. of Newton it.	No. cg-it. for each Newton step	time (s)
1	6161	45		53.4
2	24833	2	12, 15	2.1
3	99809	2	12, 7	3.1
4	400289	2	13, 25	14.4
5	1603361	7	21, 47, 64, 111, 66, 77, 43	490.1

Total time: 563.1

In figure 2.5 the "homogenized area", i.e. the area (the finite elements) where

$$0.008 \leq (\nabla u)^2 \leq 0.032$$

holds, is shown. This is a typical picture for this problem.

2.4 Examples – Linear Elasticity Problems

We now want to present examples arsing in linear elasticity. For this purpose the components of our algorithm have been adapted to vector valued solutions (displacement). Again we discuss several combinations of FE/BE-discretizations, but we restrict ourselves to problems on bounded domains. As before, the error bound ($(w, r)-$defect) of the cg-algorithm has been set to 10^{-6}.

2.4.1 The Dam Problem

As a test problem we consider a model of a dam filled with water as sketched in figure 2.6 (see also [15]). The displacement $u(x) = (u_1(x), u_2(x))^T$ satisfies formally the system of Lamé equations

$$\begin{array}{l} -\mu(x)\Delta u(x) - (\lambda(x)+\mu(x))\mathrm{grad}\ \mathrm{div}u(x) = 0 \quad \text{in } \Omega \\ u(x) = 0 \text{ on } \Gamma_D, \quad \sum_{l=1}^{2} \sigma_{kl}(u(x))n_l = g_k(x) \text{ on } \Gamma_N, \ (k=1,2) \end{array} \tag{2.5}$$

where Ω is a bounded Lipschitz domain, $\sigma_{kl}(u)$ are the components of the stress tensor $\sigma(u)$ and $n(x) = (n_1(x), n_2(x))^T$ is the outward normal vector to $\Gamma_D \cup \Gamma_N = \Gamma := \partial\Omega$ ($\Gamma_D \neq \emptyset$) and λ and μ, $\lambda, \mu > 0$, are the Lamé coefficients of the elastic material. In (2.5), $g = (g_1, g_2)^T$ is the vector of boundary tractions.
As indicated in figure 2.6, boundary conditions are given on Γ_D (zero displacement) and on Γ_N (zero or according to the water pressure, respectively). The Lamé constants are given for rock (I-II) by $\mu_r = 7.265e4$MPa, $\lambda_r = 3.743e4$MPa and for concrete (III-VIII) by $\mu_c = 9.2e6$MPa, $\lambda_c = 9.2e6$MPa.
The BE discretization of the 1^{st} level and the FE discretization of the 2^{nd} level (deformed mesh) are shown in figure 2.6.

	BEM: I-VIII		FEM: III-VIII BEM: I-II		FEM: I-VIII	
l	I(ε)	CPU	I(ε)	CPU	I(ε)	CPU
1	36	3.1	29	2.2	21	1.6
2	37	4.1	32	3.8	25	2.7
3	40	9.1	35	9.0	29	6.2
4	40	25.5	39	28.9	33	19.4
5	42	89.8	41	108.7	37	74.1
s(5)	46.4		57.2		58.8	
N(5)	6470		78130		119318	

Table 2.5: Levels (l), pure solution time (s), number of unknowns (N), iteration count ($I(\varepsilon)$, $\varepsilon = 10^{-6}$), CPU time in seconds for the dam-problem. The experiments were carried out on a Power-XPlorer using 8 processors.

In Table 2.5 we discuss several combinations of FE/BE discretizations. Comparing the results given here with those of the previous section we observe higher iteration numbers which is, first of all, a result of the preconditioners being used.
Looking at the CPU-time we observe that the FE discretization (column 3) leads to the best results. However, if we are interested in the pure solving time $s(.)$ ($s(5)$ for the 5^{th} level is given in Table 2.5) the BE discretization (first column) is of advantage.

2.4.2 The Plain Strain State Problem

Now we consider a thin linearly elastic plate with a hole in the middle . We suppose the plate to be a homogeneous isotropic elastic body. The tension forces act on the left and right border of the plate into opposite directions, see figure 2.4.2.
Describing a plain elasticity problem in terms of the displacement u, the inner forces f, the traction t and the stress tensor $\sigma = \sigma(u)$ (Hook's law) we have

$$-\mu \triangle u(x) \ - \ (\lambda + \mu)\, \mathrm{grad}(\mathrm{div}\, u(x)) \ = \ f(x) \qquad \text{in } \Omega$$

$$\begin{aligned} u_\imath(x) &= g_\imath(x) \qquad \text{on } \Gamma_{1,\imath} \quad {\scriptstyle \imath=1,2} \\ \sum_{\jmath=1}^{2} \sigma_{\imath,\jmath}\eta_\jmath &= t_\imath(x) \qquad \text{on } \Gamma_{2,\imath} \quad {\scriptstyle \imath=1,2} \end{aligned} \quad .$$

Here we are interested in the behaviour of our algorithm for $\nu \to 0.5$. We compare a pure BEM calculation with 4 subdomains (see figure 2.8 on the left) with a FEM calculation with 16 subdomains – the deformed mesh is shown in figure 2.8 on the right.

Pure BEM calculation.

- 4 subdomains (BEM: 1-4)
- CPU time on grid 6 for $\nu = 0.29$: 84 seconds.

level	1	2	3	4	5	6
# unknowns	113	223	443	883	1 763	3 523
$\nu = 0.29$	26	30	29	32	33	34
$\nu = 0.4$	26	30	28	33	34	36
$\nu = 0.49$	27	31	32	35	38	40
$\nu = 0.499$	28	32	33	36	38	42

Pure FEM calculation.

- 16 subdomains (FEM: 1-16)
- CPU time on grid 6 for $\nu = 0.29$: 70 seconds.

level	1	2	3	4	5	6
# unknowns	486	1 840	7 158	28 234	112 146	447 000
$\nu = 0.29$	27	27	29	30	31	31
$\nu = 0.4$	36	51	33	34	34	35
$\nu = 0.49$	83	51	58	62	65	68
$\nu = 0.499$	157	141	154	163	165	178

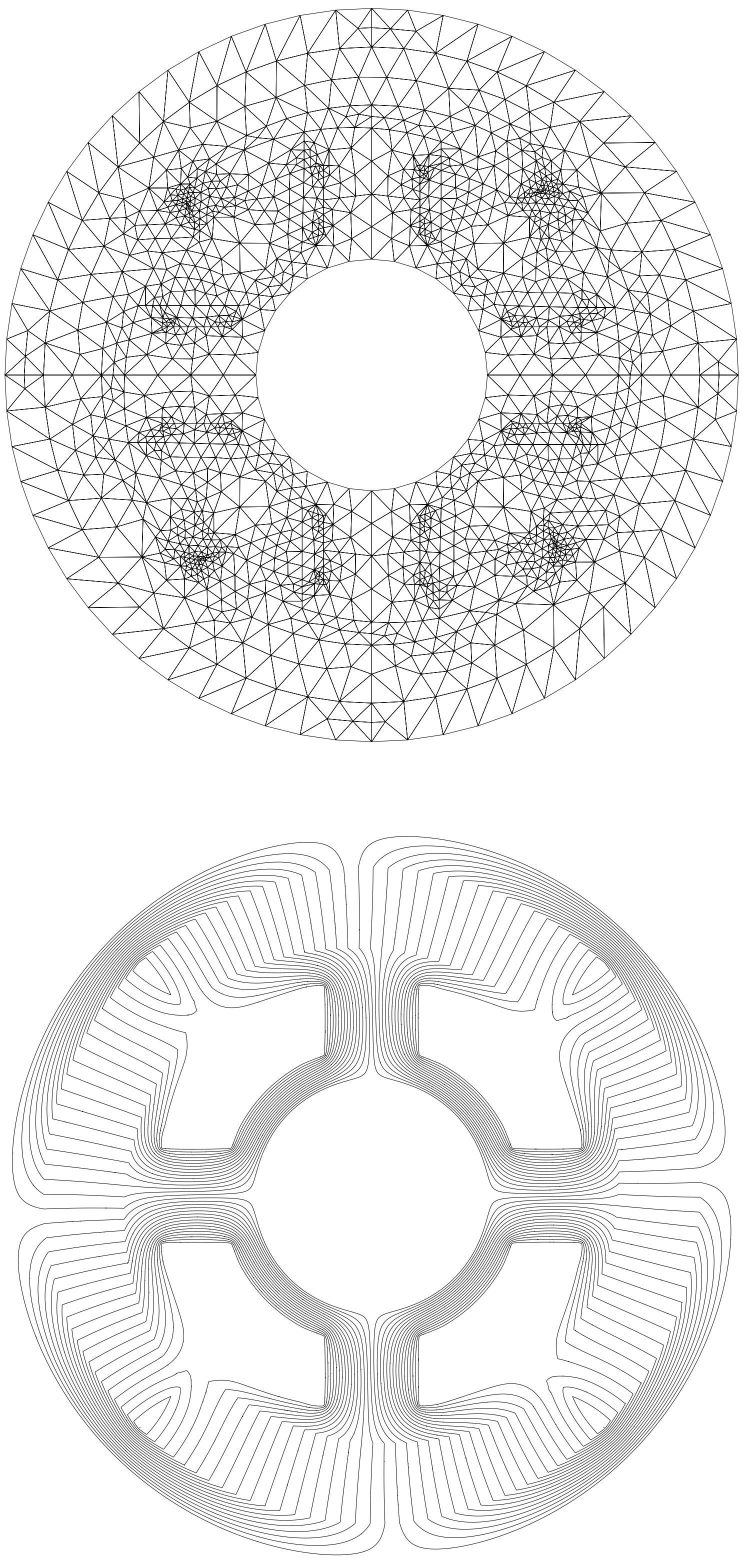

Figure 2.4: The FEM discretization of the motor and the equipotential lines of the solution.

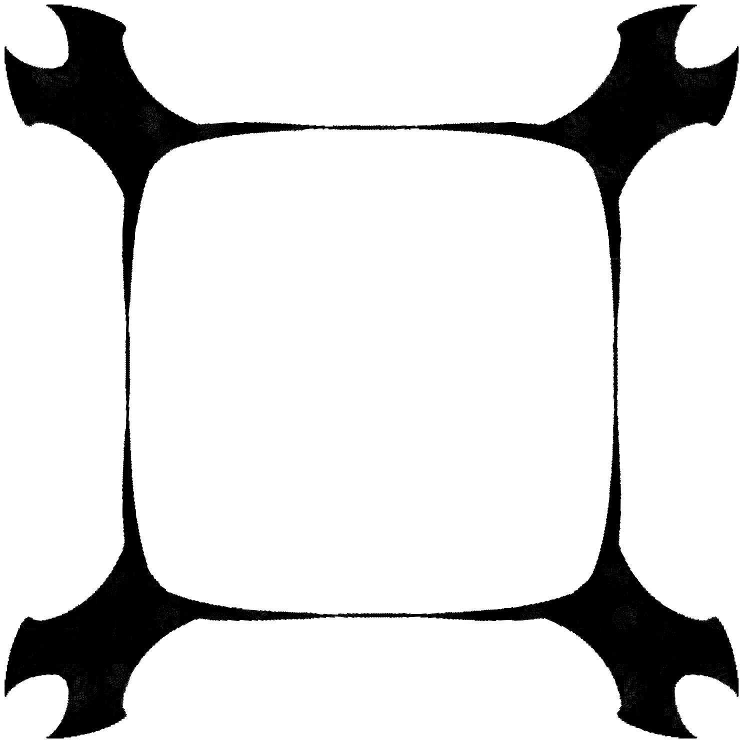

Figure 2.5: Homogenized area.

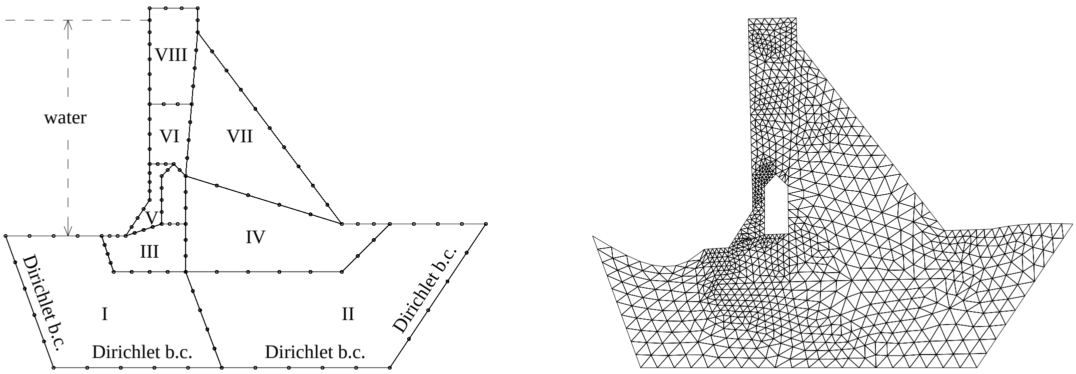

Figure 2.6: The dam problem: the subdomains and the BE discretization of the 1^{st} level (left) and the deformed (magnification factor 100) FE grid of the 2^{nd} level (right).

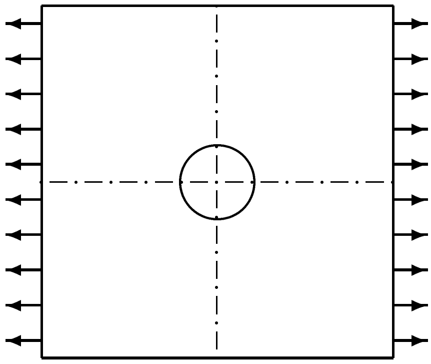

Figure 2.7: The plain strain state problem.

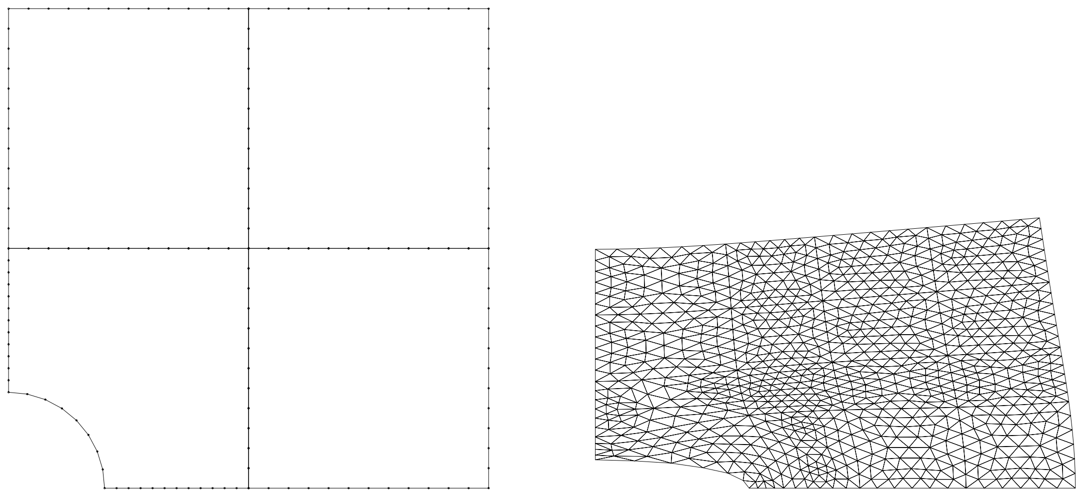

Figure 2.8: The plain strain state problem: the BE discretization (4 subdomains) of the 3^{rd} level (left) and the deformed (magnification factor 50000) FE grid (16 subdomains) of the 2^{nd} level (right).

Chapter 3

Input Data

3.1 Geometrical Data

3.1.1 Terminology

In order to solve the differential equation over a domain Ω in parallel via domain decomposition methods we have to find a way of describing the geometry globally (Ω) and locally (subdomains), which allows us, additionally, to control the communication between the processors, e.g. within the pccg-algorithm.

For this purpose we use the terminology of geometrical nodes, geometrical edges, geometrical faces and subdomains. The hierarchy is such that the **geometrical nodes** have to be defined directly by their coordinates whereas higher order structures are defined by lower order ones. That is the **geometrical edges** are defined by two or three geometrical nodes and the geometrical edges themselves define **geometrical faces**. In the 2d-case these faces coincide with the subdomains whereas in the 3d-case the **subdomains** are defined by their faces. The union of all subdomains gives the **domain** the differential equation is to be solved on.

$$\boxed{\text{geometrical nodes}}\\ \Downarrow\\ \boxed{\text{geometrical edges}}\\ \Downarrow\\ \boxed{\text{geometrical faces}}\\ \Downarrow\\ \boxed{\text{subdomains}}\\ \Downarrow\\ \boxed{\text{domain}}$$

For solving our problem numerically we have to descretize the domain or, more precisely, each subdomain. This process generates (ordinary) **nodes** at which the numerical solution is sought. Geometrical nodes which are starting points and end points of the edges belong to these ordinary nodes and they are called **crosspoints.** Geometrical nodes that have been defined as midpoints of edges do not belong necessarily to the set of ordinary nodes. The remaining nodes on the edges (which are no crosspoints) are called **pure coupling nodes.** The set of all nodes on the boundary of a subdomain is called **coupling nodes** and the set of all nodes remaining is called **inner nodes.** Globally we distinguish between the **boundary** of the domain Ω and the **inner boundary**, that consists of edges lying between two subdomains.

3.1.2 The standard input file

The communication between the program FEM⊕BEM and other programs that may perform pre- and postprocessing routines is realized via a standardized file, which we are going to describe in this section. The file contains geometrical data as well as data defining the problem which is to be solved.

The structure of the file

The codewords the blocks start with are eight letters long, where (␣) stands for space bar.

(1) Codeword: NKONST␣␣
Type: I
Length: 4
This block contains global data in the sequence given below.
1.) Number nGeom of geometrical nodes.
2.) Number nEdges of geometrical edges.
3.) Number nFaces of geometrical faces.
4.) Number nDomain of subdomains.
5.) Dimension nDim of the coordinate system.
6.) Maximum number nPoly of geometrical edges describing a subdomain.
7.) Number of degrees of freedom per node.
8.) Maximum number nSub of faces belonging to one subdomain.
9.) Number nBound of edges with boundary conditions.
10.) Number nMat of materials.

(2) Codeword: NODES␣␣␣
Type: R
Length: 4 (or 8)
Size: 4 $*$ number nGeom of nodes
This block contains the coordinates of the geometrical nodes in the following order.

number of the node
1st coordinate of the ith node
2nd coordinate of the ith node
3rd coordinate of the ith node

Here, i runs from 1 to nGeom. Note that the nodes must be ordered in such a way, that the numbers i=1,...,nCrP stand for the crosspoints and i=nCrP+1,...,nGeom stand for the midpoints that may be present.

(3) Codeword: EDGES␣␣␣
Type: I
Length: 4
Size: 6 $*$ number nEdges of the geometrical edges
The block defines the geometrical edges via the geometrical nodes.

Number of the edge
starting crosspoint of the ith edge
end crosspoint of the ith edge
midpoint of the ith edge
type of the edge
pointer

The variable i runs from 1 to nEdges. The following types of edges are defined:

1	:	straight line
2	:	arc of a circle

(4) Codeword: FACES␣␣␣
Type: I
Length: 4
Size: (3+nPoly) $*$ number of subdomains

The block defines the geometrical faces via the geometrical edges.

number of the subdomain
number of the edges describing the subdomain
number edge(1) ,..., Number edge(nPoly)
original material-id

Here, i runs from 1 to nFaces. If the number of edges describing the face is less than nPoly, then the remaining numbers are set to zero.

(5) Codeword: MAPPING2
Type: I
Length: 4
Size: (3+nSub) $*$ number nDomain of subdomains

The block contains the mapping of the subdomains onto the array of processors and classifies the kind of the problem which is to be solved on the subdomain.

number of the processor
number of faces belonging to this subdomain
number face(1),...,face(nSub)
classification of the problem to be solved in the ith subdomain

Here, i runs from 1 to nDomain. If the number of faces describing the subdomain is less than nSub, then the remaining numbers are set to zero. The classification is defined by:

-2	:	BEM-computation, exterior domain
-1	:	BEM-computation, interior domain
1	:	FEM-computation.

(6) Codeword: BOUNDARY
Type: R
Length: 4 (or 8)
Size: (3+6*nFhg) $*$ number nBound of the geometrical edges with b.c.

The block contains the values which are necessary to describe the boundary conditions along the edges.

number of the edge (see EDGES) of the ith edge with b.c.
kind of the boundary conditions
pointer
value1 starting node, value1 end node, value1 midpoint
value2 starting node, value1 end node, value2 midpoint

The variable i runs from 1 to nBound. The value of kind ranges between 1 and 5 and is defined as:

1	:	essential boundary conditions
2	:	natural boundary conditions (flowing condition)
3	:	natural boundary conditions (transition condition).

The variable pointer can be used to define a more complicated behaviour of the values of the boundary conditions than the linear one, which is assumed as standard. The second values are the transition coefficients in the case of boundary conditions of the 3rd kind. Otherwise they are not being used. The values of the midpoint will be ignored if the edge is a straight line.

(7) Codeword: MATERIAL
Type: R
Length: 4 (or 8)
Size: 8 ∗ number nMat of materials

The block contains the coefficients of the differential equation, which are constant over the subdomains

$$-\nabla^T \underline{\mathbf{u}}(x) \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} \nabla \underline{\mathbf{u}}(x) + (b_1, b_2)\nabla \underline{\mathbf{u}}(x) + c\underline{\mathbf{u}}(x) \;=\; f \qquad .$$

k_{11}, k_{12}, k_{21}, k_{22}, b_1, b$_2$, c, f

Here, i runs from 1 to nMat (=nDomain).

(8) Codeword: REFINE0␣
Type: R
Length: 4 (or 8)
Size: 5 ∗ number nEdges of geometrical edges

This block defines the a-priori strategy for refining the edges.

number of the edge (see EDGES)
kind of discretization of the ith edge
number of inner points on the ith edge
ratio 1
ratio 2

Here, i runs from 1 to nEdges.

(9) Codeword: MAIDX␣␣␣
Type: I
Length: 4

This block contains a integer field MAIDX(3,nDomain), where nDomain is the number of subdomains.

For subdomain i there is given:
Number of the element subroutine to be used for the ith subdomain.
Number of data of the field DATMA [14] necessary for it.
Start index of data in the array DATMA.

Here, i runs from 1 to nMat.

3.1.3 Global and local data

In order to compute a numerical solution over the domain Ω we need, of course, communication between the processors and thus global data, e.g. a global numbering of the geometrical nodes and edges. On the other hand the benefit of parallelism is based upon the fact that the majority of the data is local. This applies to the computational quantities as well as to the geometrical data. The latter are being stored in two common blocks *GeomGlo* (global data) and *GeomLoc* (local data).

The two common blocks

The common block *GeomGlo* (file geomglo.inc) contains several global constants and variables. The global constants are also being used in the common block *GeomLoc* (file geomloc.inc).

Constants :

constant	:	description
DimEdg	:	Length of an entry for describing the edges.
DimMap	:	Length of an entry for describing the mapping of the subdomains onto the array of processors and classification of the problem (FEM, BEM).
DimRef	:	Length of an entry for describing the integer values, which are used by mesh generator.
DimBC	:	Length of an entry for describing the number of an edge, the kind of the boundary conditions and additional information concerning the boundary conditions.
DimMat	:	Length of an entry for describing the material dependent coefficients on a subdomain.

Variables :

variable	:	description
nGeom	:	Number of global geometrical nodes including the midpoints of the edges.
nCrP	:	Number of global crosspoints.
nEdges	:	Number of geometrical edges.
nFaces	:	Number of geometrical faces (in the 2d case equal to nDomain).
nDomain	:	Number of subdomains.
nDim	:	Dimension of the domain (2 or 3).
nPoly	:	Maximum number of geometrical edges for describing one face.
nFhg	:	Number of degrees of freedom per node.
nBound	:	Number of geometrical edges with boundary conditions.

Temporary arrays :

array	:	description
XCross(nDim, nCross)	:	Coordinates of the geometrical nodes.
IEdges((DimEdg, nEdges)	:	Description of the edges.
IFace(nPoly+2, nFaces)	:	Description of the faces.
IMapping(DimMap, nDomain)	:	Mapping of the subdomains onto the array of processors.
IRefine(DimRef, nEdges)	:	Integer information for the mesh generator.
RRefine(nEdges)	:	Real information for the mesh generator.
IBound(DimBC, nBound)	:	Integer information concerning the boundary conditions.
RBound(2*3*nFhg, nBound)	:	Real information concerning the boundary conditions.
RMat(DimMat, nDomain)	:	Material dependent coefficients.

The common-block *GeomLoc* contains the local constants, variables and arrays for describing the local geometry, the initial data for the local mesh generator, the relations between the subdomains and the local coefficients of the differential equation.
Constants :

constant	:	description
MaxEdges	:	Maximum number of local geometrical edges.
MaxVertex	:	Maximum number of local geometrical nodes.
DimIE	:	Column dimension of the array IE.
MaxFhg6	:	Maximum number of degrees of freedom per node multiplied by 6.

Variables :

variable	:	description
NrDomain	:	Number of the subdomain handled by the processor.
NumVertex	:	Number of local geometrical nodes.
NumEdges	:	Number of local geometrical edges.
KindFace	:	Kind of the subdomain (3d only).
KindDisk	:	Kind of discretization (FEM=1 / BEM(int)=-1 / BEM(ext)=-2)
NV	:	Number of local crosspoints.
NE	:	Number of nodes on the local edges without the crosspoints.
NC	:	Number of local coupling nodes (= NV+NE).
NI	:	Number of inner nodes (locally).
nNode	:	Number of nodes in the subdomain (= NC+NI).
nElem	:	Number of elements (FEM) in the subdomain.

Arrays :

array	:	description
NrVerL2G(MaxVertex)	:	Global numbering of the local geometrical nodes.
NrEdgL2G(MaxEdges)	:	Global numbering of the local geometrical edges.
XCrole(2, MaxVertex)	:	Coordinates of the local geometrical nodes.
IEdgle(DimEdg, MaxEdges)	:	Description of the local edges.
IRefle(DimRef, MaxEdges)	:	Integer information for the local mesh generator.
RRefine(MaxEdges)	:	Real information for the local mesh generator.
IBoundle(MaxEdges)	:	Integer information concerning local boundary conditions.
RBoundle(MaxFhg6,MaxEdges)	:	Real information concerning local boundary conditions.
RMatle(DimMat)	:	Local material dependent coefficients.
IDirle(MaxVertex)	:	Kind of boundary conditions at the vertices.
RDirle(MaxVertex)	:	Contains the function value at the vertex in the case of essential boundary conditions.

The structure of the local data

In this section we describe the local arrays which define the geometry and the differential equation. All numbers, names and quantities mentioned are restricted to the subdomains only, that is they are local data.

Arrays describing vertices.

XCrole $\longrightarrow$ Coordinates of the geometrical nodes :

XCrole(2, i) [Real] = | x-coordinate | y-coordinate |

Note that the nodes must be ordered locally in such a way, that the numbers i=1,...,NV stand for the crosspoints and i=NV+1,...,NVertex stand for the midpoints that may be present.

IDirle, RDirle $\longrightarrow$ Essential boundary conditions at the vertices :

$$\text{IDirle(i) [Integer]} \quad = \quad \boxed{\begin{cases} 1 & \text{if essential b.c. are given at the vertex} \\ 0 & \text{otherwise} \end{cases}}$$

$$\text{RDirle(i) [Real]} \quad = \quad \boxed{\begin{cases} \text{value of b.c.} & \text{if IDirle(i)} = 1 \\ 0.0 & \text{otherwise} \end{cases}}$$

Here, i runs from 1 to NV.

NrVerL2G $\longrightarrow$ Mapping onto the global numbering :

NrVerL2G(i) [Integer] = | global number of the ith crosspoint |

Here, i runs from 1 to NumVertex.

Arrays describing the edges.

IEdgle $\longrightarrow$ description of the edges :

IEdgle(5, i) [Integer] =

$$\underbrace{\boxed{\text{starting point}}\boxed{\text{end point}}\boxed{\text{midpoint}}}_{\text{geometrical nodes of the ith edge}}\boxed{\text{type of the edge}}\boxed{\text{pointer}}$$

Here, i runs from 1 to NumEdges. The midpoint is being ignored if the edge is a straight line. The following types of edges are defined :

1	:	straight line
2	:	arc of a circle
3	:	parabola.

More general types of edges can be defined using the variable pointer.

IE $\longrightarrow$ description of the edges :

IE(5, i) [Integer] =

$$\underbrace{\boxed{\text{starting node}}\boxed{\text{end node}}}_{\text{of the ith edge}}\boxed{\text{neighboured process}}\underbrace{\boxed{\text{starting vertex}}\boxed{\text{end vertex}}}_{\text{crosspoints}}$$

Here, i runs from 1 to NumEdges. It is necessary that the nodes along an edge (without the crosspoints) are numbered increasingly according to their distance (measured along the edge) to the first node on the edge. By naming the starting crosspoint and end crosspoint a one-to-one mapping of the coupling edges between the subdomains is guaranteed. The numbering of the processes starts with 0. If the entry "neighboured process" is less than zero, then the edge being considered belongs to the boundary of the domain and the entry multiplied by (-1) gives the kind of boundary conditions on that edge.

REdgle $\longrightarrow$ weighted coefficient along the edge :

$$\text{REdgle(i) [Real]} = \boxed{\begin{cases} 0.5[Mat(process) + Mat(\text{IE(3,i)})] & \text{if IE(3,i)} \geq 0 \\ Mat(process) & \text{otherwise} \end{cases}}$$

Here, i runs from 1 to NumEdges. These data are required for the preconditioning operator along the coupling edges. $Mat(process)$ is set equal to RMatle(1).

NrEdgL2G $\longrightarrow$ mapping onto the global numbering of the edges :

NrEdgL2G(i) [Integer] = $\boxed{\text{global number of the ith geometrical edge}}$

Here, i runs from 1 to NumEdges.

<u>Arrays describing boundary conditions.</u>

RBoundle $\longrightarrow$ boundary conditions along the edges :

RBoundle(6*nFhg, i) [Real] =

$$\boxed{\text{value1-starting point}}\boxed{\text{value1-end point}}\boxed{\text{value1-midpoint}}$$
$$\boxed{\text{value2-starting point}}\boxed{\text{value2-end point}}\boxed{\text{value2-midpoint}}$$

Here, i runs from 1 to NumEdges. The dimension of the array RBoundle depends on the number of degrees of freedom per node, here we assume nFhg = 1. These values are zero for real coupling edges (i.e. inner boundaries). The values of the midpoint are ignored of the edge is a straight line and the 2nd values are ignored in the case of 1st/2nd kind boundary conditions.

IBoundle $\longrightarrow$ boundary conditions along the edges :

IBoundle(i) = $\boxed{\text{pointer to computation routines}}$

This pointer is used if non-standard (i.e. not linear interpolated) values are to be used at the nodes on the edge.

Material dependent coefficients.

RMatle $\longrightarrow$ coefficients :

RMatle(8) = | k_{11} | k_{12} | k_{21} | k_{22} | b_1 | b_2 | c | f |

The coefficients are assumed to be constant over the subdomain. They refer to the differential equation as follows:

$$-\nabla^T \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} \nabla u(x) + (b_1, b_2) \nabla u(x) + c\, u(x) \;=\; f \quad .$$

Arrays controlling the mesh generator.

IRefle, RRefle $\longrightarrow$ discretization of the edges :

IRefle(2, i) [Integer]	=	kind of discretization	number of inner nodes
RRefle(2, i) [Real]	=	ratio 1	ratio 2

Here, i runs from 1 to NumEdges.

3.2 Controlling data

3.2.1 The Newton control (.nc) file

The input of several data controlling the solver process is realized via a standardized file, the Newton control (.nc) file. It contains information about the type of the solver (Newton solver for nonlinear problems or linear solver), about the accuracy to be achieved, and a choice between different realizations of matrix generation and preconditioning in the BEM. In the case of the Newton solver it contains additional information about the type of the nonlinear problem, its nonlinear coefficient functions and the control of the Newton iteration process. It has the same design as the input file containing the geometrical data and consists of several blocks as described in section 3.1.2.

The codewords the blocks start with are eight letters long, where (␣) stands for space bar.

(1) Codeword: COCTRL␣␣
Type: I
Length: 4

This block contains control data in the sequence given below.

1.) Type IPROB of the solver.
 1 : Linear solver.
 2 : Newton solver for nonlinear problems.
2.) Maximal number NEN1 of nodes per edge (2).
3.) Maximal number NEN2 of nodes per finite element (3).
4.) Maximal number NDF of degrees of freedom per node.
5.) Maximal number NNDF of degrees of freedom for coupled problems.
6.) Index NINT1 of the 1D quadrature formula for finite elements (1).
7.) Number NQP1 of 1D quadrature points for finite elements (1).
8.) Index NINT2M of the 2D quadrature formula for the f.e. matrix (1).
9.) Number NQP2M of 2D quadrature points for the f.e. matrix (1).
10.) Index NINT2R of the 2D quadrature formula for the f.e. load vector (1).
11.) Number NQP2R of 2D quadrature points for the f.e. load vector (1).
12.) Reserve.
13.) Type IVERFT of the hierarchical refinement method (1).
14.) Type of the coupling boundary preconditioning.
15.) Type of the b.e. trial functions (3-bitwise):
 bit 1–3 (for tractions):
 0 : piecewise constant tractions
 1 : piecewise linear tractions
 bit 4–6 (for displacement):
 0 : piecewise constant displacement
 1 : piecewise linear displacement
 2 : piecewise constant displacement, nonconforming.
16.) Type of the b.e. local preconditioning method.
 0 : Integral operator (linear tractions) or multigrid (constant tractions)
 1 : scaled circle circulant
 2 : integral operator (linear tractions)
 3 : multigrid (constant tractions)
 4 : direct inversion (for test purposes only)
 5 : linear Toeplitz
 6 : algebraic Toeplitz
 13 : linear circulant.

In the present version, for the parameters 2.) ... 14.) the numbers in parentheses are the only options.

(2) Codeword: NWCTRL␣␣
 Type: I
 Length: 4

This block contains integer control data in the sequence given below.

4.) Type of Newton convergence criterion (2).
 1 : correction criterion.
 2 : defect criterion or correction criterion.
5.) Save time by omitting last matrix and defect calculation (0).
 0 : no.
 1 : yes, but not after only one Newton iteration.
 2 : always if possible.

6.) Apply the non-damped Newton method (1).
0 : no.
1 : yes, but not on the first level.
2 : always if possible.
7.) Maximal number of Newton iterates on coarsest level 1 (40).
8.) Maximal number of Newton iterates on finest level l (20).
10.) Maximal number of Newton iterates on auxiliary levels $2, \dots, l-1$ (3).
14.) Generate new systems for coarse grids (0).
0 : no.
1 : yes.
15.) Apply so-called "modified" Newton method (0).
0 : no, always generate new matrices.
1 : yes, but not on the coarse level 1.
2 : yes, do not generate matrices.

The numbers at positions 1.), 2.), 3.), 9.), 11.), 12.), 13.) are not used. A recommended choice is given in parentheses. The block is not necessary for the linear solver.

(3) Codeword: NWEPS␣␣␣
Type: R
Length: 8

This block contains real control data in the sequence given below.
1.) Maximal relative error EPS1 for defect criterion (1.0D-6).
2.) Maximal relative error EPS2 for correction criterion (1.0D-6).
7.) Parameter for increasing the damping parameter (2.0).
8.) Parameter for reduction of the damping parameter (0.25).
9.) Starting value for Newton damping parameter (1.0).
10.) Starting value for Newton damping parameter on finer levels $2, \dots, l$ (1.0).
15.) Control constant for changing accuracy of linear solver (4.0).
16.) Initial value for accuracy of linear solver (1.0D-2).
In case of solving a linear problem: accuracy (1.0D-6).

The numbers at positions 3.) ... 6.) and 11.) ... 14.) are not used. A recommended choice is given in parentheses. The block is necessary for the linear solver, too. See 16.).

(4) Codeword: MAIDX␣␣␣
Type: I
Length: 4

This block contains a integer field MAIDX(3,nDomain), where nDomain is the number of subdomains. For subdomain i there is given:
$MAIDX(1,i)$ Number of the element subroutine to be used.
$MAIDX(2,i)$ Number of data of the field DATMA necessary for it.
$MAIDX(3,i)$ Start index of data in the field DATMA.

(5) Codeword: DATMA␣␣␣
Type: R
Length: 8

This block contains the real*8 field DATMA used by the element subroutines.
Note that DATMA(1) ... DATMA(8) are overwritten by the data given in the block MATERIAL of the standard file. This allows to run the program FEM∞BEM for linear problems without the block DATMA. In this case the element subroutine ELEM07

is always called. Then the block MAIDX must contain the following data
$MAIDX(1,i) = 7$,
$MAIDX(2,i) = 8$,
$MAIDX(3,i) = 1$ for $i = 1, \ldots, nDomain$.

For nonlinear magnetic field problems the necessary data can be generated from the B - H - table applying the auxiliary program MONSPI.

(6) Codeword: MRIDX␣␣␣
Type: I
Length: 4

This block contains a integer field MRIDX(3,nDomain), where nDomain is the number of subdomains. For subdomain i there is given:

$MRIDX(1,i)$	Number of the boundary subroutine to be used.
$MRIDX(2,i)$	Number of data of the field DATMR necessary for it.
$MRIDX(3,i)$	Start index of data in the field DATMR.

(7) Codeword: DATMR␣␣␣
Type: R
Length: 8

This block contains the real*8 field DATMR used by the boundary subroutines.

3.3 Triangular based data

The preprocessor ADDPre can use an additional type of files which are based on a description of the geometry using triangles. Those $*.tri$ files contain at least 4 data blocks which are a block of constants, the coordinates of the nodes, the description of the edges and the description of the triangles. Additionally these files may contain a block describing the boundary conditions.
The file starts with 7 integer numbers which have the following meaning:

1.)	np	number of points
2.)	ne	number of edges
3.)	nt	number of triangles
4.)	nf	number of degrees of freedom
5.)	nd	number of edges with Dirichlet boundary conditions
6.)	nn	number of points with Neumann boundary conditions
7.)	ns	number of subdomains.

What follows is the description of the nodes, edges, triangles and, possibly, boundary conditions in the following form.

(1) **NODES** This block contains the coordinates (double precision) of all nodes.

number of the node
1st coordinate of the ith node
2nd coordinate of the ith node

Here, i runs from 1 to np. The nodes are ordered such that the vertices of the triangles are given first followed by the mid-points defining curved edges that may occur.

(2) **EDGES** This block describes the edges via their bounding nodes.

Number of the edge
starting point of the ith edge
ending point of the ith edge
midpoint of the ith edge
type of the edge

Here, i runs from 1 to ne. The edges can be straight lines ($type = 0$, midpoint=0) are may be arcs of some circle. In the latter case ($type = 1$) the edges are being defined uniquely by giving their mid-points.

(3) **TRIANGLES** This block describes the edges via their bounding edges.

Number of the triangle
1st, 2nd, 3rd edge describing the ith triangle
subdomain-id of the ith triangle
(original) material-id of the ith triangle

Here, i runs from 1 to nt.

(4) **BOUNDARY** This block describes given boundary conditions.

Number of the edge
value at the starting point
value at the ending point
value at the mid-point

Here, i runs from 1 to $nd + nn$.

Chapter 4

General Libraries

4.1 The library libreadgeom.a

The library libreadgeom.a is a collection of subroutines for reading the standard input file describing the geometry and the differential equation and other files controlling the nonlinear part of the program. Here we will give just a short overview, for a detailed description see the source files or the documentation file *filework.doc*, respectively.

4.1.1 Initializing the geometry

For initializing the geometry in the first lines of a program use the subroutine

Call GetGeom(H,hmax)

H	(Real*8)	$\longrightarrow$	Auxiliary vector
hmax	(Integer)	$\longrightarrow$	Length of the auxiliary vector.
/geomloc/	(Common)	$\longleftarrow$	geometry and edge description of the subdomain

in which processor number 0 reads the geometry and differential equation from the standard input file (see 3.1.2) and distributes the global information to all processors. After each processor has taken his local and a few global information the processor stores the data in the common block /geomloc/ (see 3.1).

4.1.2 Short description

GETGEOM	Reads the global geometry from the standard input file into an auxiliary array (just processor number 0 !), distributes it to all processors and each processor takes his part of the geometrical description.
OPREFL	Opens a file for reading.
NKONST	Reads the data block of some constants.
GETDAT	Reads the rest of the data blocks.
GIDATA	Reads the rest of the data blocks incoming from an ASCII-file.
OVREAD	Dummy reading of a not-used data block in the file.
GETLOCALGEOM	Takes the local data from the global data and initialize the common block /geomloc/.

4.1.3 The tree structure of libreadgeom.a

GETGEOM

↪ ViCopy (libvbasmod.a)

↪ nKonst

  ↪ OPREFL

↪ Tree_Down (libCubecom.a)

↪ GetDat

  ↪ OPREFL

  ↪ GIDATA

↪ GetLocalGeom

4.2 The library libmathutil.a

The library libmathutil.a is a collection of useful functions and subroutines which are used in the main program and some libraries connected with the preconditioning. Here we will give just a short overview, for a detailed description see the source files or the documentation file *mathutil.doc*, respectively.

4.2.1 Matrix routines I

The following matrix routines are valid for symmetric matrices stored in the Compressed Row Storage (CRS) with the row pointer to the last element of a row, e.g. just the non-zero elements of the upper and diagonal part of the matrix are stored. For notation purposes let A be the matrix and w, u are vectors.

KLZBL	Multiplies a block of the upper matrix part with a vector. $w_m := A_{m,n}\, u_n$
KLZBL_T	Multiplies a block of the lower matrix part with a vector. $w_n := A_{n,m}\, u_m$
KLZQS	Multiplies a square block of the matrix with a vector. $w_n := A_{n,n}\, u_n$
KLZB_X	Multiplies a block of the upper matrix part with a scaled vector and adds it to another vector. $w_m := w_m + \alpha\, A_{m,n}\, u_n$
KLZBL_TX	Multiplies a block of the lower matrix part with a scaled vector and adds it to another vector. $w_n := w_n + \alpha\, A_{n,m}\, u_m$
KLZQS_X	Multiplies a square block of the matrix with a scaled vector and adds it to another vector. $w_n := w_n + \alpha\, A_{n,n}\, u_n$
KLZQS_C	The same as KLZQS written in C.

4.2.2 Matrix routines II

The following matrix routines are valid for symmetric matrices stored in the variable band-width storage (german : VBZ) with the row pointer to the last element of a row, just the upper and diagonal part of the matrix are stored. For notation purposes let A be the matrix and w, u are vectors.

VBZUP	Multiplies a block of the lower matrix part with a vector. $w_n := A_{n,m}\, u_m$
VBZLO	Multiplies a block of the upper matrix part with a vector. $w_m := A_{m,n}\, u_n$
VBZQS	Multiplies a square block of the matrix with a vector. $w_n := A_{n,n}\, u_n$
CholErz	Cholesky factorization of the matrix. $A = U^T U$
CholSubst	Solves $U^T U\, u = w$.

4.2.3 Interpolation / Extension

LinearInt	Piecewise linear interpolation (1D), arbitrary distribution of the nodes.
LinearInt_T	Transposed operation to LinearInt.
LinrInt	Piecewise linear interpolation (1D), equidistant distribution of the nodes.
LinInt_T	Transposed operation to LinInt.
SplineInt	Piecewise cubic-spline interpolation (1D), arbitrary distribution of the nodes.
LinExt	Homogeneous extension of two points into an edge (straight line).
LinExt_T	Transposed operation to LinExt.
LinExt3	Linear interpolation into a 3-node-triangle.
LinExt3_T	Transposed operation to LinExt3.

4.2.4 Mappings / Geometry

ArcVec2	Calculates the directed angle between two vectors.
DIST	Euclidean distance between two points in the plain.
VecRect2	Returns the perpendicular vector to a given vector (2D).
Circle2To1	Maps a set of points on a circle to a 1D-line.
Curve2To1	Maps a set of points on an arbitrary curve to a 1D-line.
Line2To1	Maps a set of points on a 2D-line to a 1D-line.
Points2One	Calculates the height of some points using a straight line as ordinate and it's starting point as origin.
MidPoint	Calculates the midpoint of a circle given by three points.
LinesX2	Calculates the point of intersection of two straight lines given in point-vector notation.
fLinesX2	Calculates the point of intersection of two finite lines (straight lines) given in two-point notation. The relation whether the point belongs to the interior of the finite lines or not will be returned.
GetCircle	Approximates a number of 2D-points by a circle.
Divide	Divides a closed arranged list of edges under some criteria into two sublists of the same kind, recursively. At the end a triangular net will be generated for a given (polygonal) boundary without creating inner nodes. Written in C.
NewList	Auxiliary routine for Divide.
Regular	Tests in Divide whether a new created edge is regular in the finite element sense or not.

4.2.5 Some vector routines

iPred	Returns the cyclic predecessor of an integer in an interval.
iSucc	Returns the cyclic successor of an integer in an interval.
Equal	Tests whether the difference between two complex numbers is relatively smaller (e.g. they are equal) then 1.0E-5 or not.
LessEqual	Tests whether one complex number is smaller then or equal to the other complex number in a special sense.
Online	Tests whether a point belongs to the epsilon(=1E-05)-stripe of a finite line or not.
VcReduce	Removes all double entries in a complex array and compresses the array.
ViReduce	Removes all double entries in a integer array and compresses the array.
VdCat	Extends a vector by another vector.

4.2.6 Simple Solver

FFT	Fast Fourier Transformation (just sinus part)
SftPa	Initialization routine of the FFT.
FuSin	Sinus Fast Fourier Transformation.
Solve3	Solves a tridiagonal system of equations.
FakLU3	LU-factorization of a triangular matrix.
Subst3	Solves a system of equations with a factorized triangular matrix.

4.3 The library libdebug.a

The library libdebug.a is a collection of subroutines useful for debugging a parallel program. Here we will give just a short overview, for a detailed description see the source files or the documentation file *debug.doc*, respectively.

4.3.1 Local debugging

When writing a parallel program there is often one problem : the program runs but some data may be wrong. Therefore all of the so called "local debugging routines" have the ability to print the data from certain processors. The set of abbreviations $T = \{D, R, I\}$ stands for the data types { Double Precision, Real, Integer }.

Debug{T}	Debugs a certain vector of the given type $T = \{D, R, I\}$.
Debug{T}2	Debugs a certain 2D-array of the given type $T = \{D, R, I\}$.
PrintLocalDat	Prints the local geometry data.
FitD	Tests whether the nodes on the boundary have the same value or not.
MakeFitD	Returns the average value for each node on the boundary.

4.3.2 Others

PrintDat	Prints the global geometry data incoming from the standard input file.
LoadBal	Prints the minimal and maximal usage of the huge working array.
Statistik	Prints some statistics like cg-convergence rate, number of iterations and some measured times.
Aux	Tests whether a number (usually the rest of the huge working array) is larger then 0 or not. If not, an error message is printed and the processor stops.
Error	Prints an error number, error message and stops the program on the processor.

4.4 The library libbound.a

Library for handling the accumulation of vectors and the preconditioning of the coupling nodes via various types of preconditioning operators. The common blocks *geomloc* and *geomglo* are just used by some older routines (compatibility).

4.4.1 Variables and arrays

nCrP	INTEGER	global number of cross points
ndf	INTEGER	number degrees of freedom
LevMax	INTEGER	Number of hierarchical levels (coarsest level = 0 !!)
ISP(0:LevMax)	INTEGER	Pointer at the starting address of a level in the hierarchical list ISV
ISV(4,*)	INTEGER	hierarchical list consists of : $\boxed{\text{son}}$ $\boxed{\text{father\_1}}$ $\boxed{\text{father\_2}}$ $\boxed{\text{weight(son,father\_1)}}$
DNEIGH(*)	REAL*8	Number of domains to which a coupling node belongs
SKC4(*)	REAL*8	SQRT(4/diag(K_C)) for all coupling nodes
NrVerL2G(*)	INTEGER	Stores for all local cross points their global number
rMatle(8,*)	REAL*8	Material coefficients per face
iMatle(*)	INTEGER	Material class for each face
IDV(nCrP*ndf)	INTEGER	Pointer at the last entry per row of the coarse grid matrix
SKV	REAL*8	Coarse grid matrix (assembled, factorized)
IE(5,*)	INTEGER	Edge description per multigrid level
JE(5,numcbp,0:LevMax)	INTEGER	Edge description per hierarchical level (in preparation)
NrDEdge(*)	INTEGER	number of edges describing the faces in one domain

IFacel(MaxNrEdg,*)	INTEGER	chain of edges describing the faces in one domain

4.4.2 Additional used addresses in the pointer arrays INZEI/IZEIG and in the working array B

INZEI(101)	LevMax	number of hierarchical levels (level dep.)
INZEI(102)	LISP	pointer at ISP in B (level dep.)
INZEI(103)	LISV	pointer at ISV in B (level dep.)
INZEI(104)	LDNEIGH	pointer at DNEIGH in B (level dep.)
INZEI(105)	LSKC4	pointer at SKC4 in B (level dep.)
INZEI(106)	LL2G	pointer at NrVerL2G in B
INZEI(107)	KindDisk	kind of discretization in B
INZEI(108)	LRM	pointer at rMatle in B (non linearity !)
INZEI(109)	LJE	pointer at JE in B (level dep.)
INZEI(110)	nCrP	global number of cross points
INZEI(111)	LIDV	pointer at IDV in B
INZEI(112)	LSKV	pointer at SKV in B
INZEI(113)	LOrEdg	pointer at OrEdg in B
INZEI(114)	BEMPc	describes the BEM domain and the kind of preconditioning
INZEI(115)	LMGCW	pointer at weights in B for MG-CC preconditioning
INZEI(116)	GMGPCG	use global multigrid as preconditioner
INZEI(117)	MAXCG	max. number of cg-iterations
INZEI(120)	NumMat	number of material classes in the domain
INZEI(121)	DimMat	number of parameters per material (= 8)
INZEI(122)	MaxNrEdg	max. number of edges per face describing the domain
INZEI(123)	NumSub	number of faces (in 2D subdomains) describing the domain on one processor
INZEI(124)	LNRD	pointer at NrDEdge in B
INZEI(125)	LIFA	pointer at IFacel in B
INZEI(126)	LIMA	pointer at IMatle in B
IZEIG(62,j)	LIE	Pointer at IE(level j) in B

4.4.3 Calling the accumulation

All routines starting with *Accu* require the full parameter list, especially the number of degrees of freedom can be chosen freely, e.g.

```
C***********************************************************************
C*
C*      SUBROUTINE AccuBound(ndf,nv,NumEdges,IE,nCrP,NrVerL2G,W,H,hMax)
C*
c*      Accumulates W (Real*8) on all inner boundaries including cross points.
C*
C*      --->    NDF     : # degrees of freedom
C*      --->    nv      : # local vertices (= cross points)
C*      --->    NumEdges: # local edges
C*      --->    IE      : edge description array
C*      --->    nCrP    : # global cross points
C*      --->    NrVerL2G: maps local vertex numbers to global numbers
```

```
C*      <-->    W       : In-/Output vector.
C*      <-->    H,hmax  : auxiliary array (hmax 8-Byte-words)
C*
C*      The vector W starts with the first local cross point !! ( W(1)=Wv )
C*
C***********************************************************************
```

All routines starting with *Add* just require the actual multigrid level and get the rest (degrees of freedom, etc.) from the arrays *IZeig,InZei,B*, e.g.

```
C***********************************************************************
C*
C*      SUBROUTINE AddBound(level,W,H,hmax,IZeig,InZei,B,ierr)
C*
C*      Accumulates a Real*8-vector W on all nodes on the coupling boundary.
C*      Uses the working arrays.
C*      Uses the number of degrees of freedom stored in IZEIG !
C*
C*      --->    level   : actual multigrid level
C*      <-->    w       : vector
C*      <-->    h,hmax  : auxiliary array (hmax 8-Byte-words long)
C*      --->    IZeig,InZei,B  : working arrays
C*      <---    ierr    : error handling
C*
C***********************************************************************
```

The routines starting with *WAdd* are using the common blocks *geomloc* and *geomglo*. They are just included for the sake of compatibility. When designing new code it is recommended to use the new routines !!
Due to the fact that the *Accu*-routines just include the *VdPlus*-operation the more general routines *gAccu* with a call *VdOp* from *libvbasmod.a* as additional last parameter were created. This opens a wide funtionality to the accumulation.

4.4.4 Initializing the coarse grid system

For initializing the coarse grid system the routine

```
C***********************************************************************
C*
C*      SUBROUTINE IniCoarse(IBE,LAEB,IZeig,InZei,B,ierr)
C*
C*      Builds the coarse grid system and factories it via Cholesky.
C*      Before calling this subroutine the subroutine IniGeom
C*      has to be called.
C*      IBE will be changed !!
C*
C***********************************************************************
```

gets some necessary data from the working arrays and requires storage from the huge B-array for storing the factorized coarse grid matrix SKV and the pointer at the last row entry IDV.

4.4.5 Initializing the hierarchical list

For initializing the hierarchical list the routine

```
C*********************************************************************
C*
C*      SUBROUTINE IniHier(idx,level,IBE,LAEB,IZeig,InZei,B,ierr)
C*
C*      Initializes the hierarchical list for the hierarchical
C*      preconditioning of the edges.
C*      Additionally the diagonal preconditioning vector (SKC4) and
C*      the vector including the number of neighboring subdomains (dNeigh)
C*      will be calculated.
C*
C*      --->    idx     : method of building the hierarchical list
C*      --->    level   : multigrid level
C*      <-->    IBE     : first free address in B
C*      --->    LAEB    : last free address in B
C*      <-->    IZeig,InZei   : pointer in B
C*      <-->    B       : working array
C*      <---    ierr    : error handling
C*
C*      The global maximum of the local number of hierarchical levels
C*      has to be used as the number of levels because of the
C*      necessary vector accumulation on each level in
C*      the BPX-Preconditioning on the boundary values   !!!!!!!!!!
C*
C*********************************************************************
```

determines the number of hierarchical levels until the cross point grid (= level 0) is reached, requires the storage in B and builds the hierarchical list in the arrays IDV and IDP. The parameter *idx* has the following meaning

- idx = 1 MOD 10 : Simple construction of the list (list 1).
- idx > 10 : Uses compressed numbering of the coupling nodes and creates the array JE (necessary for the communication).

Additionally the arrays SKC4 and DNEIGH are calculated and stored in B, see 4.4.2.

4.4.6 Calling the preconditioner C_C

The general routine for calling the preconditioner C_C uses again the working arrays and gets via the parameter *VkRand* the appropriate preconditioner on the boundary.

```
C*********************************************************************
C*
C*      SUBROUTINE PreBound(VkRand,W,R,level,IZeig,InZei,B,H,hmax,ierr)
C*
C*      Solves the preconditioning system
C*
C*                             -1
C*                   w  :=  C   * r
```

```
C*                    C      C     C
C*
C*      via various Schur complement preconditioners and a coarse grid solver.
C*
C*      ---> VkRand  : Schur complement preconditioning method
C*                     (Identity,Diagonal,BPS/BPhier/BPX)
C*      <--- W       : solution vector
C*      ---> R       : right hand side
C*      ---> level   : actual grid number
C*      ---> IZeig,InZei  : pointer in B
C*      ---> B       : working array
C*      ---> H,hMax  : auxiliary array of length hMax (8-byte words)
C*
C*      The arrays w,r may occupy the same memory addresses.
C*
C***********************************************************************
```

The following Schur complement preconditioner routines are available

Identity	:	no preconditioning, e.g. $C_C \equiv I_C$
Diagonal	:	diagonal preconditiong, e.g. $C_C \equiv \operatorname{diag} K_C$
BPS	:	BPS-preconditioner with Dryja
BPhier	:	Hierarchical basis preconditioner
BPX	:	Multilevel nodal basis preconditioner.
BPX3	:	Multilevel nodal basis preconditioner. The necessary communication on each level will be performed in one communication step.
MGHyp	:	Multigrid preconditioner based on the global hyper-singular integral operator, i.e. $C_C = D\,(I_C - M_C)^{-1}$.

Take care that the appropriate routine is declared as EXTERNAL !

4.4.7 Short description of the procedures

	Accumulation of a vector
AccuBound	Accumulates all coupling nodes
AccuEdges	Accumulates all nodes on the edges (without cross points)
AccuEck	Accumulates all cross points
AddBound	Accumulates all coupling nodes
AddEdges	Accumulates all nodes on the edges (without crosspoints)
AddEck	Accumulates all cross points
gAccuBound	General accumulation of all coupling nodes (basic routine)
gAccuEdges	General accumulation of all nodes on the edges (without cross points) (basic routine)
gAccuEck	General accumulation of all cross points (basic routine)
psAccu	Pseudo accumulation (without communication)

psDist	Pseudo distribution (without communication)
WAdd	Accumulates all coupling nodes
WAddKoppel	Accumulates all nodes on the edges (without cross points)
WAddEck	Accumulates all coupling nodes
	Build the coarse grid system SKV, preconditioner
AccuCoarse	Build the coarse grid system and factorize the matrix SKV
AccuCFEM	Build the FEM-coarse grid system and factorize the matrix SKV
FEMAss	Assembles the local entries in the matrix SKV
IniCoarse	Interface to AccuCoarse
IniCFEM	Interface to AccuCFEM
GetProf3	Determines the global profile width of the matrix SKV when 3-node linear triangles are used
LastID	Returns the pointer at the last entry of the matrix SKV
Mesh0	Returns a coarse grid triangulation of a subdomain
PreEck	Solve the global coarse grid system with the factorized matrix SKV
PreEckB	Interface to PreEck
	Construction of the hierarchical list on the boundary
BuildList	Constructs the hierarchical list ISV
IniDiag	Initializes the array SKC4
IniHier	Initializes the hierarchical list ISV and the arrays SKC4, DNEIGH
LevHier	Determines the number of hierarchical levels *LevMax*
NextFree	Looks for the next proper father node
SetWeight	Sets the weigh in the hierarchical list between son and first father
	Preconditioning of the coupling nodes
BPhier	Bramble/Pasciak/Schatz and hierarchical techniques
BPS	Bramble/Pasciak/Schatz
BPX	Bramble/Pasciak and multilevel nodal basis (MNB) on the boundary
BPX3	Like BPX but less communication by the use of BPXEck
BPXEck	Special preconditioner for the coarse grid system in the BPX-case, needs less communication
Diagonal	Diagonal preconditioning on the coupling boundary
Identity	No preconditioning on the coupling boundary
MGHyp	Multigrid preconditioner using the hyper-singular integral operator
FFTVKg	Dryja-preconditioning on the edges
HierInt	Hierarchical interpolation
HierRes	Hierarchical restriction
MNBInt	MNB interpolation
MNBInt3	special MNB interpolation
MNBRes	MNB restriction
PreBound	General routine for calling the preconditioner on the coupling nodes

Set0Dir	Sets all nodes with Dirichlet boundary conditions to zero
	Usefull routines
GlobMaxD	Returns the global maximum of a double precision vector
GlobMinD	Returns the global minimum of a double precision vector
SKALAR	Return the global inner product of two vectors of different types

4.5 The library libutilmg.a

The library libutilmg.a is a collection of subroutines for connecting the main program and the preconditioner described in section 4.6 with the libraries *libparmesh.a*, *libgigehi.a*, *libgsgen.a* and *libmgsolv.a*. Because these libraries are used as black boxes this handling for the initialization of the routines is necessary. On the other hand these routines have to adapt the (slightly) different data structures. Here we will give just a short overview, for a detailed description see the source files or the documentation file *utilmg.doc*, respectively.

4.5.1 Short description

Abfrage	Reads the number of levels in the multilevel algorithm.
BasisNep	Basis transformation via multigrid with the homogeneous extension of the boundary data by S.V.Nepomnyaschikh as initial guess in a subdomain.
BasisTrafo	Basis transformation via multigrid in a subdomain.
SolvIn	Preconditioner for the interior of a subdomain.
DefMG	Read the multigrid control parameter from console.
Glob_Num	Gets the global number of degrees of freedom and the number of elements.
MgOut	Prints some information about the multigrid control parameters being used.
IniPara	Requires storage from the main array and initializes the multigrid control parameter.
ParaMesh	Adapter program for generating the local coarse net (user net) via calling *ParMesh* (libparmesh.a).
IniGlS	Adapter program for generating the hierarchical grids and the stiffness matrices by calling *GiGeHi* (libgigehi.a) and *GsGenG* (libgsgen.a).
IniMG	Adapter program for initializing certain multigrid control parameters.
MGPara	Adapter program for initializing certain multigrid control parameters on all levels.

4.5.2 The basis transformation

In the following routines some parameters cannot be chosen freely because they have fixed addresses in the huge working array B. For fixing these addresses the internal array *izeig* (/mgfem/) is used. The parameter *nlevel* stands for the finest grid.
There are two different routines for calling the basis transformation. The first version with a (usually zero) initial guess $\underline{u}_I^0$ for the multigrid cycle realizes ($img = 2$)

$$\underline{u}_I \;:=\; B_I^{-1}\,\underline{f}_I \;=\; \overline{M}_I\,\underline{u}_I^0 + \left(I_I - \overline{M}_I\right) K_I^{-1}\,\underline{f}_I \;,$$

with $\overline{M}_I$ as the multigrid iteration operator and $(img = 1)$

$$\underline{u}_I \;:=\; B_I^{-T}\,\underline{f}_I \;=\; \overset{*}{\overline{M}}_I\,\underline{u}_I^0 + \left(I_I - \overset{*}{\overline{M}}_I\right) K_I^{-1}\,\underline{f}_I \;,$$

with $\overset{*}{\overline{M}}_I$ as a multigrid iteration operator adjoint to $\overline{M}_I$ in the K_I-energy inner product.

BasisTrafo(u,f,ID,IK,SK,IBE,LAEB,IZeig,InZei,B,zero,img,ierr)

U	(Real*8)	$\longleftrightarrow$	Output vector $(B(IZeig(43, nLevel)))$
F	(Real*8)	$\longrightarrow$	Right hand side vector in the multigrid procedure $(B(IZeig(44, nLevel))))$.
ID	(Integer)	$\longleftarrow$	Pointer at the last column per matrix row.
SK	(Integer)	$\longleftarrow$	Stiffness matrix (CRS).
IBE	(Integer)	$\longleftarrow$	First free address in B.
LAEB	(Integer)	$\longleftarrow$	Available length of B.
IZEIG(150,*)	(Integer)	$\longleftarrow$	Index array in B.
INZEI(*)	(Integer)	$\longleftarrow$	Index array in B.
B	(Real*8)	$\longleftrightarrow$	Huge working array.
zero	(Logical)	$\longrightarrow$	Start with $\underline{u}_I^0 = 0$ if TRUE, otherwise start with the given vector u.
img	(Integer)	$\longrightarrow$	Decides which operator will be realized (including the inner preconditioner !) $img = \{1,2,3\} \implies \{B^{-T}, B^{-1}, C^{-1}\}$.
ierr	(Integer)	$\longleftarrow$	Error handling.
/mgpara/	(Common)	$\longrightarrow$	Control parameters for the multigrid cycle.

The second version with the hierarchical homogeneous extension of the boundary data (E_{IC}) as initial guess $\underline{u}_I^0$ for the multigrid cycle realizes $(img = 2)$

$$\underline{u}_I \;:=\; -\,B_I^{-1}\,K_{IC}\,\underline{f}_C \;=\; \overline{M}_I\,E_{IC}\,\underline{f}_C - \left(I_I - \overline{M}_I\right) K_I^{-1}\,K_{IC}\,\underline{f}_C \;,$$

with $\overline{M}_I$ as the multigrid iteration operator and the transposed operation with a (usually zero) initial guess $\underline{u}_I^0$ $(img = 1)$

$$\underline{u}_C \;:=\; -\,K_{CI}\,B_I^{-1}\,\underline{f}_I \;=\; ,E_{CI}\,\overline{M}_I^T\,\underline{u}_I^0 - K_{CI}\left(I_I - \overset{*}{\overline{M}}_I\right) K_I^{-1}\,\underline{f}_I + E_{CI}\,\overset{*}{\overline{M}}_I\,\underline{f}_I \;,$$

with $\overset{*}{\overline{M}}_I$ as a multigrid iteration operator adjoint to $\overline{M}_I$ in the K_I-energy norm. Here, the call is as follows

BasisNep(u,f,ID,IK,SK,IBE,LAEB,IZeig,InZei,B,zero,img,ierr),
where the parameters are defined as above.

4.6 The Domain Decomposition preconditioners

In the subdirectory FEMooBEM of the distribution one may find a couple of files strongly connected with a parallelized version of the conjugate gradient method and the preconditioners used therein. The cg-method and the preconditioners are described in detail in [21, 19, 16, 26] so we will restrict ourselves to a short description of the routines.

4.6.1 The cg-algorithm

The detailed description of the parallelized preconditioned version of the cg-algorithm in file *cg.f* can be found in [21]. Some of the parameters have fixed addresses in the huge working array B so they cannot be chosen freely. For fixing these addresses the internal array *izeig* (/mgfem/) is used. The parameter *nlevel* stands for the finest grid. Because of the differences between the FE- and BE-cg we decided to write two cg-routines – one for FEM-domains and a second for BEM-domains. The FEM-version has to be called as follows.

CG(R,X,S,W,F,H,hmax,IBE,LAEB,IZeig,InZei,B,cgEPS,Iter,Vorkond,VkRand)

R	(Real*8)	$\longleftarrow$	Residuum during the iteration
X	(Real*8)	$\longleftarrow$	Solution
S	(Real*8)	$\longleftarrow$	Correction during the iteration
W	(Real*8)	$\longleftarrow$	Preconditioned residuum ($B(IZeig(43, nLevel))$)
F	(Real*8)	$\longleftrightarrow$	Right hand side of the system of equations and right hand side vector in the multigrid procedure ($B(IZeig(44, nLevel))$)
H	(Real*8)	$\longrightarrow$	Auxiliary vector
$hmax$	(Integer)	$\longrightarrow$	Length of the auxiliary vector.
IBE	(Integer)	$\longleftarrow$	First free address in B.
LAEB	(Integer)	$\longleftarrow$	Available length of B.
IZEIG(150,*)	(Integer)	$\longleftarrow$	Index array in B.
INZEI(*)	(Integer)	$\longleftarrow$	Index array in B.
B	(Real*8)	$\longleftrightarrow$	Huge working array
$cgEps$	(Real*8)	$\longrightarrow$	Error barrier in the cg-algorithm measured in (W, R) inner product.
$Iter$	(Integer)	$\longleftarrow$	Number of cg-iterations.
$Vorkond$	(External)	$\longrightarrow$	Call for the preconditioning.
$VKRand$	(External)	$\longrightarrow$	Schur complement preconditioning method.

The BEM-version looks as follows:

CGBEM(R,X,S,W,F,H,hmax,IBE,LAEB,IZeig,InZei,B,cgEPS,
Iter,Vorkond,VkRand),

where the parameter are defined as above.

4.6.2 The preconditioners

All of the used DD-preconditioners have the following structure

$$C = \begin{pmatrix} I_C & K_{CI}B_I^{-T} \\ O & I_I \end{pmatrix} \begin{pmatrix} C_C & O \\ O & C_I \end{pmatrix} \begin{pmatrix} I_C & O \\ B_I^{-1}K_{IC} & I_I \end{pmatrix} \quad .$$

Applying this preconditioner to the preconditioning equation $\underline{\mathbf{w}} = C_C^{-1}\, \underline{r}$ yields the

Algorithm 1 : The ASM-DD Preconditioner
$\underline{\mathbf{w}}_C = C_C^{-1} \sum_{i=1}^{p} A_{C,i}^T \left(\underline{r}_{C,i} - K_{CI,i} B_{I,i}^{-T} \underline{r}_{I,i}\right)$ $\underline{\mathbf{w}}_{I,i} = C_{I,i}^{-1} \underline{r}_{I,i} - B_{I,i}^{-1} K_{IC,i} \underline{\mathbf{w}}_{C,i} \quad ; \; i = 1, 2, \ldots, p$
Determined by the user : $C_C = ?,\ C_I = ?,\ B_I = ?$

by defining C_I (the inner preconditioner) and B_I (the basis transformation) via multigrid. The operation $\underline{\mathbf{w}}_C = C_C^{-1} \sum_{i=1}^{p} A_{C,i}^T \hat{\underline{r}}_C$ will be performed in the subroutine *PreBound* (libedge.a). This algorithm 1 (the numbering of the algorithms follows [16]) is realized in the file *vorkmg1.f* .
Choosing C_I in dependence of B_I via multigrid leads to another interesting realization of the above preconditioner called Algorithm 3 and can be found in *vorkmg3.f.* In contradiction to the two forgoing versions of the preconditioner, based on a Additive Schwarz methods (ASM), the Algorithm 4 is derived from Multiplicative Schwarz methods (MSM) but leads to the same structure of the preconditioner C with different structures of B_I and C_I. This MSM-preconditioner can be found in *vorkmg4*.
New realizations of the preconditioners described above consist in treating the operation $\hat{\underline{\mathbf{w}}}_{I,i} = -B_{I,i}^{-1} K_{IC,i} \underline{\mathbf{w}}_{C,i}$ and the transposed one as a whole in the preconditioner by using a new hierarchical extension technique [26] to get an initial guess $\underline{\mathbf{w}}_{I,i}^0 = E_{IC,i} \underline{\mathbf{w}}_{C,i}$ for the following multigrid cycles. These preconditioners can be found in the appropriating files *vorkmg1n.f, vorkmg3n.f, vorkmg4n.f* .
The choice which preconditioner should work in the program has to be done by writing the appropriate source file in the file *makefile* in the directory FEMooBEM. All preconditioners have the same calling sequence.
Some of the parameters have fixed addresses in the huge working array B so they cannot be chosen freely. For fixing these addresses the internal array *izeig* (/mgfem/) is used. The parameter *nlevel* stands for the finest grid.

VorKondEXT(VKRand,W,R,F,H,hmax,IBE,LAEB,IZeig,InZei,B,ierr)

VKRand	(External)	$\longrightarrow$	Schur complement preconditioning method.
W	(Real*8)	$\longleftarrow$	Preconditioned output vector $(B(IZeig(43, nLevel)))$.
R	(Real*8)	$\longrightarrow$	Input vector.
F	(Real*8)	$\longleftrightarrow$	Right hand side vector in the multigrid procedure $(B(IZeig(44, nLevel))))$.
H	(Real*8)	$\longrightarrow$	Auxiliary vector.
hmax	(Integer)	$\longrightarrow$	Length of the auxiliary vector.
IBE	(Integer)	$\longleftarrow$	First free address in B.
LAEB	(Integer)	$\longleftarrow$	Available length of B.

IZEIG(150,*)	(Integer)	$\longleftarrow$	Index array in B.
INZEI(*)	(Integer)	$\longleftarrow$	Index array in B.
B	(Real*8)	$\longleftrightarrow$	Huge working array.
$ierr$	(Integer)	$\longleftarrow$	Error handling.

The suffix EXT has to be replaced by one of the following choices.

suffix	algorithm	additional components
id	Identity	No preconditioning at all
be	—	Handles the BEM-part in the preconditioning of the cg, i.e. no preconditioning for inner nodes
mg1	ASM Alg. 1	Multigrid
mg1n	ASM Alg. 1	Multigrid and hierarchical extension
mg1ns	ASM Alg. 1	Multigrid and hierarchical extension plus smoothing
mg2	ASM Alg. 2	Multigrid
mg3	ASM Alg. 3	Multigrid
mg3nsi	ASM Alg. 1b	hierarchical extension plus smoothing implying multigrid
mg4	MSM Alg. 4	Multigrid
mg4n	MSM Alg. 4	Multigrid and hierarchical extension

Chapter 5

The Library GiGeHi

5.1 Introduction

The library GIGEHI contains FORTRAN routines for generating a hierarchically built up sequence of triangulations of a plain domain starting at a given coarse grid. We allow the triangles having either straight edges or curved edges (arcs of a circle or parabola). The coarsest grid can be generated by using the program PARMESH [8].
The present version of the hierarchical mesh-generator is restricted to a strategy of regular refinement, in particular the finer mesh is obtained by subdividing each triangle into four subtriangles, where the new nodes will be defined at the midpoints of the edges of the original triangle. Future versions shall be able to adapt the meshes to singularities of the solution.
The library GIGEHI described in the following can be used on both, parallel computers with MIMD architecture and sequential computers. Using it on parallel machines requires the domain to be decomposed into subdomains, where each of them is handled by one processor. The generation of both, the coarsest mesh (cf. PARMESH) and the sequence of hierarchical meshes is then performed in parallel for all subdomains, that is the program GIGEHI has to be started on each processor. GIGEHI provides not only information describing the meshes but also information concerning father-son relations which are required if multilevel methods are to be applied.
In the following we describe the data structure used within GIGEHI, the refinement algorithm and the input and output data.

5.2 The strategy of refinement

The refinement of a given mesh is based upon the following strategy. Let us suppose that the domain which is to be discretized has been subdivided into subdomains for which a coarse grid is given. Furthermore the boundary of each subdomain is divided into coupling edges, where every coupling edge lies either between two subdomains or belongs to the boundary of the original domain. A coupling edge is bounded by two, so called, cross-points. The nodes of the coarsest grid and of all the finer grids are being subdivided into three groups: cross-points – nodes that belong to at least two coupling edges, pure coupling nodes – nodes that belong to exactly one coupling edge, and inner nodes – nodes that lie inside a subdomain (cf. figure 5.1).
The nodes belonging to the discretization $\mathcal{T}_p$, which is part of the sequence $\mathcal{T}_j$, $j = 1, 2, \ldots, l$, of discretizations, are numbered in the following order: cross-points, pure coupling nodes, inner nodes of the first subdomain, inner nodes of the second subdomain, ..., inner nodes of the p-th subdomain. This ordering implies that a fixed node belonging to $\mathcal{T}_{j_*}$ has different

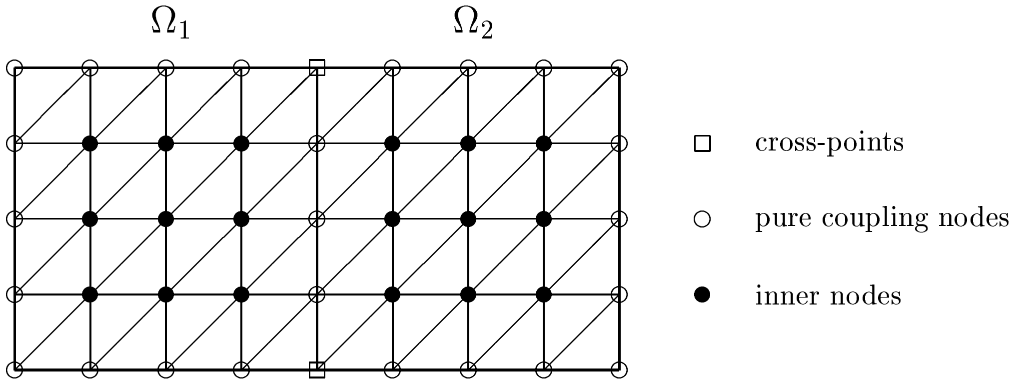

Figure 5.1: Decomposition of a domain into subdomains.

numbers in every $\mathcal{T}_j$, $j = j_*, j_* + 1, \ldots l$.
The present version of GIGEHI allows a successive refinement only by quartering the triangles. That is, we define the midpoints of the edges of the triangles belonging to $\mathcal{T}_j$ as new nodes and divide each triangle into four congruent subtriangles (cf. figure 5.2).

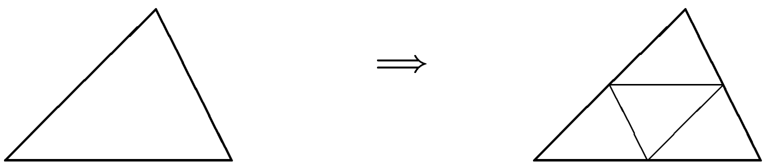

Figure 5.2: Quartering a triangle.

A future version of GIGEHI will contain routines for generating hierarchical, adapted meshes. Especially the meshes should adapt to singularities of the solution near overobtused angles of the boundary of the domain, near points at which a change of boundary conditions takes place and near interfaces.

5.3 Arrays describing a sequence of meshes

Edges are the main objects handled by the hierarchical mesh generator. The meshes are basically described by nodes, edges and triangles. The edges can be straight lines or arcs of a circle or parabola. The straight edges are characterized uniquely by their starting and ending nodes. For describing a curved edge uniquely we have to define an additional node (see array XM). Considering parts of a parabola we assume that the coupling boundary which contains the coupling edge under consideration is given by the equation $\bar{y} = a\bar{x}^2 + c$ in a local coordinate system. Here, the line connecting the cross-points, which are the starting and the ending point of the edge, is the abscissa of the coordinate system and the midpoint of this line is the origin of the local system. The triangles are characterized by their edges.

The following names will be used for quantities describing the meshes:

NUMNP	number of nodes (overall)
NUMCRP	number of cross-points
NUMCP	number coupling nodes (pure coupling nodes)
NUMINP	number inner nodes
NUMED	number of edges
NUMBED	number of edges on the boundary
NUMCBP	number of coupling edges
NUMEL	number of triangles
NUMMP	number of intermediate points
NUMWK	number of nodes having the weight $\neq 1/2$ for interpolation and restriction within the multigrid–algorithm
NDF	maximum number of degrees of freedom per node
NTR	number of grids within the hierarchy.

The following arrays are generated for describing the sequence of meshes. All the integer arrays are of the type INTEGER∗4 and all the real arrays are of the type REAL∗8.

- IED – array of edges

IED$(1,j)$	starting node of the j-th edge		
IED$(2,j)$	ending node of the j-th edge		
IED$(3,j)$	intermediate point of the j-th edge	$=0$:	for a straight edge
		$\neq 0$:	number of the intermediate point (see array XM)
IED$(4,j)$	pointer	$=0$:	inner edge
		$\neq 0$:	edge on the boundary number of the corresponding coupling edge
IED$(5,j)$	type of the edge	$=1$:	straight line
		$=2$:	arc of a circle
		$=3$:	arc of a parabola

$j = 1(1)$NUMED.

- IEDM – array containing further information concerning the edges

IEDM$(1,j)$	number of subedges of the j-th edge
IEDM$(2,j)$	number of the first subedge

$j = 1(1)$NUMED.

Remark: The subedges of an edge have to be numbered consecutively.

- IECE – array of the element-connectivity

IECE$(1,j)$	
IECE$(2,j)$	number of the edges belonging to the j-th triangle
IECE$(3,j)$	
IECE$(4,j)$	material id

$j = 1(1)$NUMEL.

Remark: This array can be used for the generation of the FE-system. The typical FE-array IX of the element connectivity via the nodes can be easily generated from the array IED. We assume the following relation between IED and IX:

$$\text{IX}(i,.) \text{ is the node opposite to the edge } \text{IED}(i,.).$$

- IBC – array for coding the boundary conditions related to the edges

IBC$(1,j)$	number of the edge	
IBC$(2,j)$	boundary material id	
IBC$(3,j)$ ... IBC$(NDF+2,j)$	$=1$:	boundary conditions of the 1st kind
	$=2$:	boundary conditions of the 2nd kind
	$=3$:	boundary conditions of the 3rd kind

$j = 1(1)$NUMBED.

- ICBN – array describing coupling edges

ICBN$(1,j)$	number of the first coupling node on the j-th coupling edge
ICBN$(2,j)$	number of coupling nodes on the j-th coupling edge
ICBN$(3,j)$	numbers of the cross-points bounding the j-th
ICBN$(4,j)$	coupling edge

$j = 1(1)$NUMCBP.

Remark: The coupling nodes on the j-th coupling edge have to be numbered consecutively beginning with the number ICBN$(1,j)$.

- X – array of the coordinates of the nodes

X$(1,j)$	x_1–coordinate of the j-th node
X$(2,j)$	x_2–coordinate of the j-th node

$j = 1(1)$NUMNP.

- XM – array of the coordinates of the intermediate nodes for curved edges

XM$(1,j)$	x_1–coordinate of the j-th intermediate node
XM$(2,j)$	x_2–coordinate of the j-th intermediate node

$j = 1(1)$NUMMP.

Remark: In order to describe a curved edge uniquely we define an additional node which is the midpoint of the corresponding arc. In the user-defined triangulation $\mathcal{T}_1$ these nodes do not have to be the midpoints necessarily. In particular these points may lie outside the edge if arcs of a circle are described. The evaluation of the intermediate nodes which are the midpoints is carried out by GIGEHI.

- ICF – array for assigning fine-grid numbers to coarse-grid numbers

ICF(j) number of the j-th coarse-grid node in the subsequent finer grid

$j = 1(1)$NUMNP (number of coarse-grid nodes).

Arrays describing "father–son–relations"

- ISONZ – array of pointers onto data about the parent-nodes in ISONN

  ISONZ(j) pointer to the last data in ISONN concerning parent-nodes of the j-th node

  $j = 1(1)$NUMNP (number of fine-grid nodes).

- ISONN – array of the parent-nodes

ISONN(i), i = ISONZ($j-1$) +1	number of parent-nodes of the j-th node ($ianz$) $= 1$: node which belongs to the coarser grid $\neq 1$: number of parent nodes
ISONN($i+1$), ..., ISONN($i + ianz$)	for $ianz = 1$: number of the j-th node according to the coarse-grid numbering for $ianz \neq 1$: numbers of the parent-nodes according to the coarse-grid numbering
ISONN($i + ianz + 1$)	pointer to the appropriate weights in the array WK if weights $\neq 1/2$ are to be used for interpolation and restriction

  $j = 1(1)$NUMNP (number of fine-grid nodes).

- WK – array of the weights for interpolation and restriction

  WK($1, j$) ... WK($3, j$) weights for interpolation and restriction

  $j = 1(1)$NUMWK.

  Remark: This array is going to be used in a future version of the hierarchical mesh generator (e.g. if the mesh is adapted to singularities).

5.4 Storage management

All the arrays describing the hierarchical sequence of meshes including the coarsest mesh are being stored on the REAL*8 working array B. The INTEGER*4 arrays IZEIG($64, NTR$) and INZEI(100) contain both, information concerning the discretization itself, e.g. number of nodes and number of elements belonging to each grid, and the addresses of the arrays, describing the meshes, in B. The parameter LAEB stands for the length of B in double words and B(IBE) is the first free element of the array B, that is IBE is the address from which on B can be used at the present state of the execution of the program.
The table below shows the content of the array IZEIG concerning the discretization.

IZEIG$(1,j)$	NUMNP	number of nodes $(j=1,\ldots,NTR)$
IZEIG$(2,j)$	NUMCRP	number of cross-points $(j=1,\ldots,NTR)$
IZEIG$(3,j)$	NUMCP	number of pure coupling nodes $(j=1,\ldots,NTR)$
IZEIG$(4,j)$	NUMINP	number of inner nodes $(j=1,\ldots,NTR)$
IZEIG$(5,j)$	NUMED	number of edges $(j=1,\ldots,NTR)$
IZEIG$(6,j)$	NUMBED	number of edges on the boundary $(j=1,\ldots,NTR)$
IZEIG$(7,j)$	NUMCBP	number of coupling edges $(j=1,\ldots,NTR)$
IZEIG$(8,j)$	NUMEL	number of elements $(j=1,\ldots,NTR)$
IZEIG$(9,j)$	NUMMP	number of intermediate points $(j=1,\ldots,NTR)$
IZEIG$(10,j)$	NUMWK	number of nodes having weights $\neq 1/2$ for interpolation and restriction within the multigrid– algorithm $(j=2,\ldots,NTR)$
IZEIG$(16,j)$	LX	address of the array X $(j=1,\ldots,NTR)$
IZEIG$(17,j)$	LXM	address of the array XM $(j=1,\ldots,NTR)$
IZEIG$(18,j)$	LIBC	address of the array IBC $(j=1,\ldots,NTR)$
IZEIG$(19,j)$	LIECE	address of the array IECE $(j=1,\ldots,NTR)$
IZEIG$(20,j)$	LIED	address of the array IED $(j=1,\ldots,NTR)$
IZEIG$(21,j)$	LICBN	address of the array ICBN $(j=1,\ldots,NTR)$
IZEIG$(22,j)$	LIEDM	address of the array IEDM $(j=1,\ldots,NTR-1)$
IZEIG$(23,j)$	LICF	address of the array ICF $(j=1,\ldots,NTR-1)$
IZEIG$(24,j)$	LISONZ	address of the array ISONZ $(j=2,\ldots,NTR)$
IZEIG$(25,j)$	LISONN	address of the array ISONN $(j=2,\ldots,NTR)$
IZEIG$(26,j)$	LWK	address of the array WK $(j=2,\ldots,NTR)$.

We demonstrate the use of the arrays IZEIG and INZEI by presenting a little program which prints the content of the array IED (array of edges) of the j-th level.

```
      SUBROUTINE UVW(J,B,IZEIG,INZEI)
      IMPLICIT NONE
      INTEGER*4  J,IZEIG,INZEI,NUMED,LIED
      REAL*8     B
      DIMENSION  B(1),IZEIG(64,1),INZEI(1)
C ***
      ... source code ...

      NUMED = IZEIG(5,J)
      LIED = IZEIG(20,J)
      CALL DRUCK(J,NUMED,B(LIED))

      ... source code ...

      RETURN
      END
C
C Subroutine for printing the content of IED on the j-th grid
C
      SUBROUTINE DRUCK(J,NUMED,IED)
      IMPLICIT NONE
      INTEGER*4  J,NUMED,IED,I,K
      DIMENSION  IED(5,1)
C ***
```

```
      WRITE(6,*) ' Output of the array of the edges of the ',j,'-th grid'
      WRITE(6,*)
      DO 1 I=1,NUMED
      WRITE(6,*) (IED(K,I),K=1,5)
    1 CONTINUE
      RETURN
      END
```

5.5 Calling the program GIGEHI

Before starting the program GIGEHI one has to ensure that the arrays IED, IECE, IBC, ICBN, X and XM (see section 5.4) have already been defined for the coarsest mesh. The arrays concerning subsequent, i.e. finer, grids will be defined by GIGEHI. Furthermore, the parameters IZEIG$(1,1)$ to IZEIG$(10,1)$ and IZEIG$(16,1)$ to IZEIG$(21,1)$ which describe the coarsest mesh (see section 5.4) must have been defined before GIGEHI is called.
The following parameters have to be defined in INZEI:

INZEI(5)	maximum number of degrees of freedom per node
INZEI(12)	number of grid in the sequence of discretizations (NTR)
INZEI(13)	type of the mesh-refinement max{INZEI(41), ... ,INZEI($39+NTR$)}
INZEI(41), ... , INZEI($39+NTR$)	kind of mesh-refinement applied to $\mathcal{T}_j$, $j=1,2,\ldots,NTR-1$ (in version 1.0 INZEI(41), ... ,INZEI($39+NTR$) must be set equal to 1.)

The program GIGEHI is called as follows:

```
CALL GIGEHI(IBE,LAEB,IZEIG,INZEI,B,IERR),
```

where

IBE	I∗4	I	lowest address from which on the array B is available for GIGEHI
		O	lowest address from which on the array B is available for other programs
LAEB	I∗4	I	length of the working array B
IZEIG	I∗4		array of pointers IZEIG$(64, NTR)$ (see also section 5.4)
		I	information concerning the coarsest grid
		O	information concerning all grids
INZEI	I∗4		information array INZEI(100)
		I	see table above

		O	unchanged input-data if IERR = 0, otherwise INZEI(12) is defined as number of the finest mesh which has been generated
B	R*8	I/O	working array B
IERR	I*4	O	error code = 0 : all refinement steps that had been demanded were performed = 10 + k : memory too small, only the grids $\mathcal{T}_j$, $j = 1, 2, \ldots, k-1$, have been generated.

The parameter marked by "I" are INPUT parameters whereas the parameters marked by "O" are OUTPUT parameters. I*4 and R*8 stand for data of the type INTEGER*4 and REAL*8, respectively.

5.6 Tree structure of the program GIGEHI

GIGEHI

- GIMIKN
  - GIMIDE
    - GIKRZR
    - GIMIPO
    - GIZWPA
      - GIINFK
- GISPZU
  - VICOPY
  - VDCOPY
- GIMADR
  - VICOPY
- GIVEFE
  - VICOPY
  - GIAEKN
  - GIVEDR
    - GIVTDR
      - GIVEKA
        - GIGEK1
        - GIGEK2
          - GIKRZR
          - GIMIPO
        - GIGEK3
      - GIDRG4
        - GIGEKI
        - GIIECE
    - GIICBN
    - GIKORR
    - GISONZ
  - GIPAMI
    - GIZWPA
      - GIINFK
- GISPFR
  - VDCOPY

5.7 A short description of the routines

GIAEKN	Determining the new numbers of the coarse-grid cross-points, coupling nodes and inner nodes assuming that all triangles have been quartered.
GIDRG4	Quartering of a triangle.
GIGEHI	Controlling routine.
GIGEKI	Generation of a new edge which is not part of an existing one.
GIGEK1	Dividing a straight edge.
GIGEK2	Dividing a curved edge (arc of a circle).
GIGEK3	Dividing a curved edge (part of a parabola).
GIICBN	Generation of the array ICBN which describes the coupling edges.
GIIECE	Definition of the array IECE for one triangle.
GIINFK	Getting the information concerning some edge from the array IEDM.
GIKORR	Correction of the numbering of the nodes, i.e. deleting non-defined nodes, and defining the array IBC.
GIKRZR	Determination of both, the centre and the radius of the circle a curved edge (arc of a circle) belongs to.
GIMADR	Adapter program for selection the refinement criterion.
GIMIDE	Computation of the intermediate nodes for curved edges from the coarsest grid.
GIMIKN	Calling the program which computes the intermediate nodes for curved edges from the coarsest grid.
GIMIPO	Computation of the coordinates of a new intermediate node on a curved edge (arc of a circle).
GIPAMI	Controlling the algorithm which computes intermediate nodes for all curved edges (parts of a parabola).
GISONZ	Compression of the array ISONN and definition of the array ISONZ.
GISPFR	Releasing memory which is no longer required.
GISPZU	Allocating memory for generating mesh-describing arrays.
GIVEDR	Controlling of one refinement step.
GIVEFE	Controlling the process of refinement.
GIVEKA	Adapter program for selecting the routine for dividing an edge depending on the type of this edge.
GIVTDR	Calling routines for quartering a triangle.
GIZWPA	Computation of an intermediate node for curved edges (parts of a parabola).
VDCOPY	Copy a REAL*8–vector.
VICOPY	Copy an INTEGER*4–type vector.

Chapter 6

The Library MGSolv

6.1 Introduction

The library MGSOLV contains FORTRAN routines which realize a multigrid algorithm including the basic components as smoothing procedures, direct solvers for the coarse-grid system and algorithms for both, interpolation and restriction. Most of the subroutines which are part of version 1.0 can be used for parallel multigrid algorithms on MIMD computers provided the data concerning the FE-discretization and the FE-system are spread over the processors according to the non-overlapping Domain Decomposition method (see e.g. [18, 23, 25]).
If the multigrid method is to be applied we have to provide a hierarchically built up sequence of triangular-based grids $\mathcal{T}_j$, $j = 1, 2, \ldots, l$ and a sequence of FE-systems

$$A_j \underline{u}_j = \underline{f}_j, \qquad j = 1, 2, \ldots, l,$$

which arise from using piecewise linear trial functions on the triangular grids.
The multigrid program makes use of the same data structure as the hierarchical mesh generator GIGEHI, chapter 5. The sequence of the FE-systems can be generated using the routines of the library GSGEN, chapter 7.

6.2 The multigrid algorithm

6.2.1 The basic scheme of the algorithm

First we will describe the basic scheme of the program performing the multigrid algorithm (l-grid algorithm) for solving the linear system $A_l \underline{u}_l = \underline{f}_l$ approximately. Note that the user has to choose the number j_g of the coarsest grid. Then, the linear system associated with the discretization $\mathcal{T}_{j_g}$ is used as coarse-grid system within the multigrid algorithm. Looking at figure 6.1 MGSMOO stands for calling a smoothing procedure of the pre- and postsmoothing with ν_1 and ν_2 smoothing steps, respectively. The smoothing procedures which are available are described in section 6.2.2.
The operator I_{j-1}^{j} defined by

$$I_{j-1}^{j}: \quad \mathcal{R}^{N_{j-1}} \longrightarrow \mathcal{R}^{N_j}, \quad j = j_g + 1, j_g + 2, \ldots, l, \tag{6.1}$$

stands for the operator of linear interpolation whereas

$$I_{j}^{j-1}: \quad \mathcal{R}^{N_j} \longrightarrow \mathcal{R}^{N_{j-1}}, \quad j = j_g + 1, j_g + 2, \ldots, l, \tag{6.2}$$

stands for the operator of the restriction. A more detailed description of these operators is given in section 6.2.3.

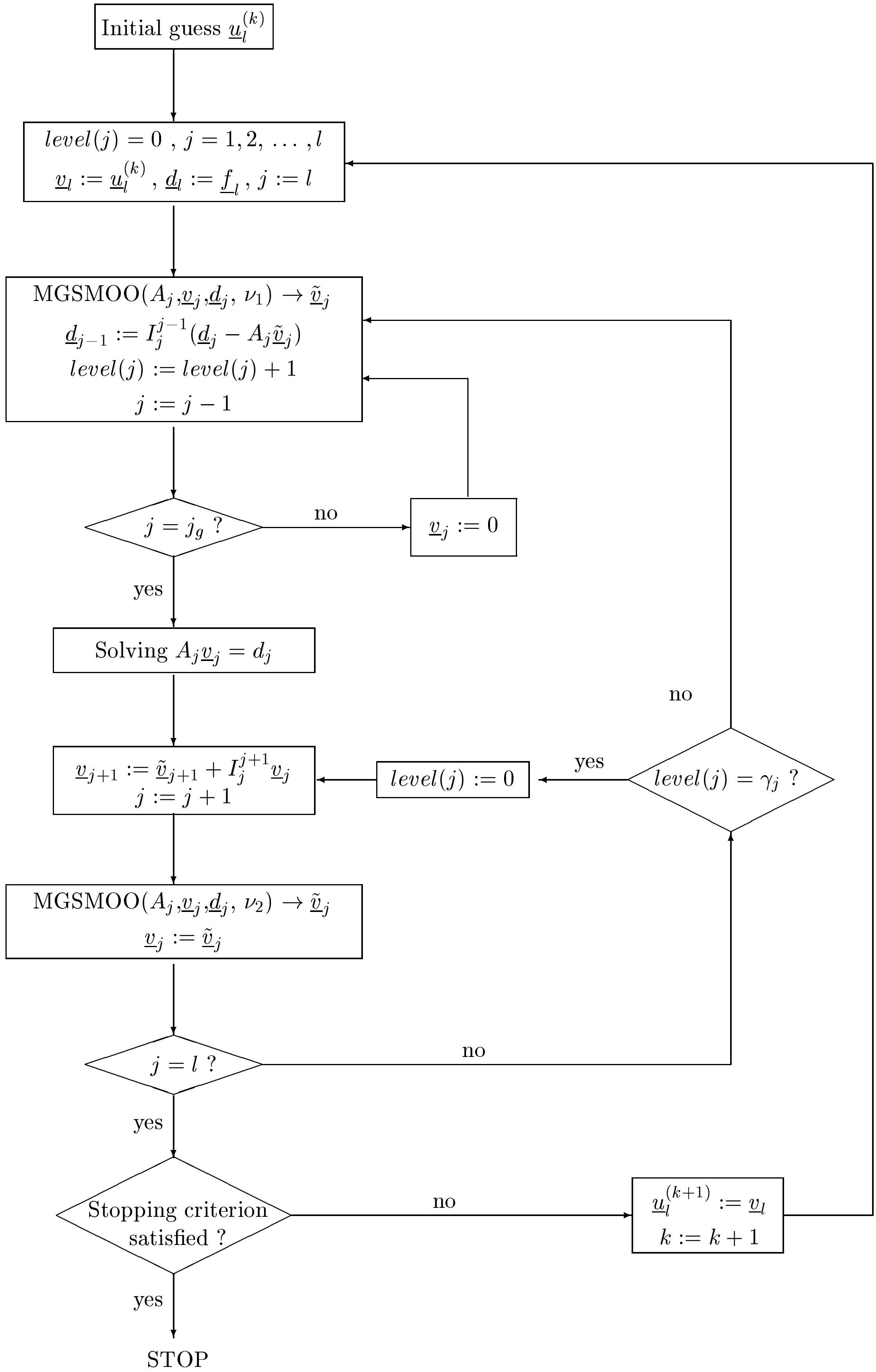

Figure 6.1: Scheme of the multigrid algorithm.

6.2.2 Solvers for the coarse-grid system and smoothing methods

The FE-system on the coarsest grid can be solved either directly or approximately by using one of the smoothing methods discussed below. In the case that the multigrid algorithm is being used for solving Dirichlet problems via Domain Decomposition methods (cf. [23, 25]) the direct method is based upon a Cholesky decomposition of the system matrix. If the multigrid algorithm is used as a global, sequential method the direct solver is realized via a LDL^T–decomposition of the system matrix.

The user can choose one of the following smoothing methods:

ITYP = 2 : ω–Jacobi–method

$$D\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.3}$$

$u^{(0)}$ is a given initial guess and $D = diag(A)$.

ITYP = 3 : Block–ω–Jacobi–method

$$D_B\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.4}$$

$u^{(0)}$ is a given initial guess and $D_B = blockdiag(A)$ ((2 x 2)–blocks).

ITYP = 4 : ω–Gauss–Seidel–method (forward)

$$(D + \omega L)\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.5}$$

$u^{(0)}$ is a given initial guess, $A = L + D + R$, L strongly lower triangular matrix, $D = diag(A)$ and R is a strongly upper triangular matrix.

ITYP = 5 : ω–Gauss–Seidel–method (backward)

$$(D + \omega R)\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.6}$$

$u^{(0)}$ is a given initial guess , $A = L + D + R$, L strongly lower triangular matrix, $D = diag(A)$ and R is a strongly upper triangular matrix.

ITYP = 6 : Block–ω–Gauss–Seidel–method (forward)

$$(D_B + \omega L)\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.7}$$

$u^{(0)}$ is a given initial guess, $A = L + D_B + R$, L strongly lower triangular matrix, $D_B = blockdiag(A)$ ((2 x 2)–blocks) and R is a strongly upper triangular matrix.

ITYP = 7 : Block–ω–Gauss–Seidel–method (backward)

$$(D_B + \omega R)\frac{\underline{u}^{(k+1)} - \underline{u}^{(k)}}{\omega} + A\underline{u}^{(k)} = \underline{f}, \quad k = 0, 1, 2, \ldots, \tag{6.8}$$

$u^{(0)}$ is a given initial guess, $A = L + D_B + R$, L strongly lower triangular matrix, $D = blockdiag(A)$ ((2 x 2)–blocks) and R is a strongly upper triangular matrix.

6.2.3 Operators of interpolation and restriction

Since we have chosen piecewise linear trial functions for our FE-discretization we now use linear interpolation within the multigrid algorithm. The operators of interpolation I^j_{j-1} are defined as follows:

$$\underline{v}_j = I^j_{j-1}\underline{v}_{j-1}, \quad \underline{v}_j = [v^i_j]_{i=1,2,\dots,N_j}, \; \underline{v}_{j-1} = [v^i_{j-1}]_{i=1,2,\dots,N_{j-1}} \tag{6.9}$$

where

$$v^i_j = \begin{cases} v^i_{j-1} & \text{if the node } p^{(i)} \text{ belongs to } \mathcal{T}_{j-1} \\ (1-\alpha)v^{i_1}_{j-1} + \alpha v^{i_2}_{j-1} & \text{if the node } p^{(i)} \text{ does not belong to } \mathcal{T}_{j-1} \text{ but} \\ & \text{lies on an edge bounded by the nodes } p^{(i_1)} \text{ and } p^{(i_2)} \\ & \text{of a triangle which belongs to } \mathcal{T}_{j-1}. \end{cases} \tag{6.10}$$

Here, α is defined according to the actual position of $p^{(i)}$ on the edge bounded by $p^{(i_1)}$ and $p^{(i_2)}$. In the case that the new nodes are generated as midpoints of the edges we obtain $\alpha = 1/2$. N_j and N_{j-1} stand for the number of nodes in $\mathcal{T}_j$ and $\mathcal{T}_{j-1}$, respectively.

The operators of restriction being used are defined by:

$$(\underline{v}_{j-1}, I^{j-1}_j \underline{v}_j)_{\mathrm{R}^{N_{j-1}}} = (I^j_{j-1}\underline{v}_{j-1}, \underline{v}_j)_{\mathrm{R}^{N_j}} \quad \forall \underline{v}_{j-1} \in \mathcal{R}^{N_{j-1}}, \, \underline{v}_j \in \mathcal{R}^{N_j}, \quad j = j_g + 1, \dots, l. \tag{6.11}$$

6.2.4 The multigrid cycles

The number γ_j of iteration steps to be performed for solving the system $A_j\underline{v}_j = \underline{d}_j$ on the j-th grid ($j = l-1, l-2, \dots, j_g+1$) approximately determines the type of the multigrid cycle. In particular $\gamma_j = 1$ leads to the V–cycle whereas the W–cycle is obtained for $\gamma_j = 2$. Figure 6.2 shows schematically both, the V–cycle and the W–cycle. Here, $*$ stands for a smoothing operation, $\circ$ symbolizes the solution of the system on the coarsest grid, $\searrow$ stands for a restriction operation and $\nearrow$ stands for a interpolation operation.

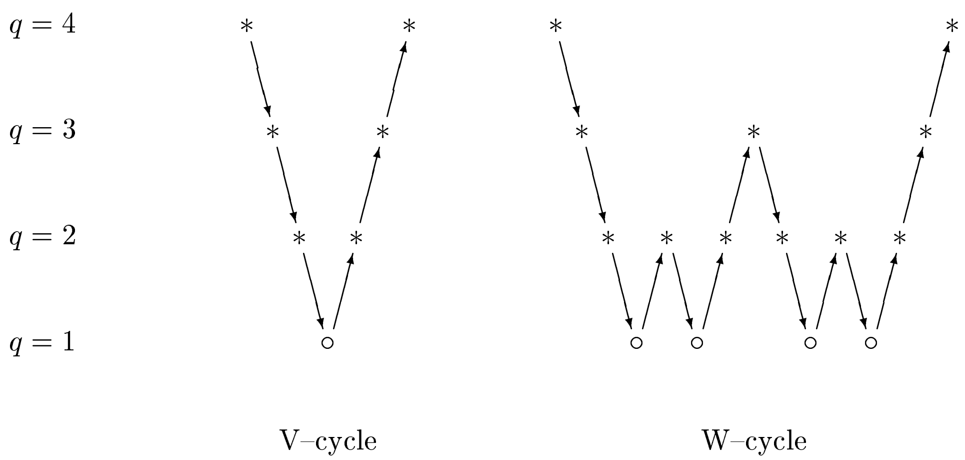

Figure 6.2: Schemes of the V–cycle and of the W–cycle.

6.2.5 Stopping criteria in the multigrid algorithm

In the case that the multigrid algorithm is being used for solving Dirichlet problems via Domain Decomposition methods (see e.g. [23, 25]) it is aborted after a pre-defined number of iteration steps. In the other case, that is if the multigrid algorithm is used as a global, sequential solver, it is aborted either if a maximum number of iteration steps has been performed or if the defect criterion

$$\| A_l \underline{u}_l^{(k)} - f_l \| \leq \varepsilon \| A_l \underline{u}_l^{(0)} - f_l \| \tag{6.12}$$

has been satisfied, where $\underline{u}_l^{(k)}$ is the k-th iterate and $\underline{u}_l^{(0)}$ is the initial guess. The error constant ε has to be defined by the user.

6.3 Preparations for calling MGSOLV

If the multigrid algorithm is to be applied one has to provide a sequence A_j, $j = j_g, j_g + 1, \ldots, l$ of stiffness matrices and the right hand side of the linear system $A_l \underline{u}_l = \underline{f}_l$ which is to be solved. We assume that piecewise linear trial functions on a hierarchical sequence of triangular grids have been used for the FE-discretization. Furthermore the multigrid algorithm requires the father–son relations describing the nodes. All the input data required by MGSOLV can be generated using the libraries GIGEHI (chapter 5) and GSGEN (chapter 7).

6.3.1 Storing the input data

All the input data (sequence of FE-systems, parameter controlling the multigrid algorithm) have to be stored on a REAL*8 working array B. The two INTEGER*4 arrays IZEIG$(64, l)$ (l = number of grids) and INZEI(100) contain both, information concerning the discretization (e.g. number of nodes, number of triangles etc.) and the addresses of the data on the array B. That is, using these addresses one can access all the data concerning the FE-systems and the discretization itself. We demonstrate the access to the solution on the finest grid in the following program.

```
      SUBROUTINE PRLOES(L,B,IZEIG,INZEI)
      IMPLICIT NONE
      INTEGER*4  L,IZEIG,INZEI,NUMNP,NDF,LU
      REAL*8     B
      DIMENSION  B(1),IZEIG(64,1),INZEI(1)
C ***
C *** Determine the number of nodes (NUMNP), the number of degrees
C *** of freedom per node (NDF) and the address of the vector of the
C *** solution (LU)
C ***
      NUMNP = IZEIG(1,L)
      NDF = INZEI(4)
      LU = IZEIG(43,L)
      CALL DRUCK(NUMNP,NDF,B(LU))
      RETURN
      END

C
C subroutine for printing the vector of the solution
```

```
C
      SUBROUTINE DRUCK(NUMNP,NDF,U)
      IMPLICIT NONE
      INTEGER*4  NUMNP,NDF,I,K
      REAL*8     U
      DIMENSION  U(NDF,1)
C ***
      WRITE(6,*) ' Print the solution on the finest grid '
      write(6,*) ' number of the node     solution '
      WRITE(6,*)
      DO 1 I=1,NUMNP
      write(6,1000) I,(U(K,I),K=1,NDF)
    1 CONTINUE
C ***
 1000 FORMAT(4X,I4,6X,2(2X,E12.5))
      RETURN
      END
```

6.3.2 Data describing the sequence of grids

In each grid of the sequence of grids the nodes have to be numbered in the following order: cross-points, pure coupling nodes, inner nodes (cf. chapter 5). The multigrid algorithm requires the following arrays which have to be provided by the hierarchical mesh generator:

- ICF – array for assigning fine-grid numbers to coarse-grid numbers

  ICF(j) number of the j-th coarse-grid node in the subsequent finer grid

  $j = 1(1)$NUMNP (number of coarse-grid nodes).

Arrays describing "father–son–relations"

- ISONZ – array of pointers onto data about the parent-nodes in ISONN

  ISONZ(j) pointer to the last data in ISONN concerning parent-nodes of the j-th node

  $j = 1(1)$NUMNP (number of fine-grid nodes).

- ISONN – array of the parent-nodes

ISONN(i),	number of parent-nodes of the j-th node ($ianz$)
i = ISONZ($j-1$) + 1	$= 1$: node which belongs to the coarser grid
	$\neq 1$: number of parent nodes
ISONN($i+1$),	for $ianz = 1$: number of the j-th node according to

..., ISONN($i + ianz$)		the coarse-grid numbering
	for $ianz \neq 1$:	numbers of the parent-nodes according to the coarse-grid numbering
ISONN($i + ianz + 1$)	pointer to the appropriate weights in the array WK if weights $\neq 1/2$ are to be used for interpolation and restriction	

$j = 1(1)$NUMNP (number of fine-grid nodes).

The array IZEIG contains both, the addresses of the array considered above and data concerning the grids. That is

IZEIG($1, j$)	NUMNP	number of nodes $(j = 1, \dots, NTR)$
IZEIG($2, j$)	NUMCRP	number of cross-points $(j = 1, \dots, NTR)$
IZEIG($3, j$)	NUMCP	number of pure coupling nodes $(j = 1, \dots, NTR)$
IZEIG($4, j$)	NUMINP	number of inner nodes $(j = 1, \dots, NTR)$
IZEIG($23, j$)	LICF	address of the array ICF $(j = 1, \dots, NTR - 1)$
IZEIG($24, j$)	LISONZ	address of the array ISONZ $(j = 2, \dots, NTR)$
IZEIG($25, j$)	LISONN	address of the array ISONN $(j = 2, \dots, NTR)$.

The following parameter have to be defined in INZEI:

INZEI(4)	number of degrees of freedom per node
INZEI(12)	number of grids in the sequence of discretizations (NTR)
INZEI(41), ... , INZEI($39 + NTR$)	kind of mesh-refinement applied to $\mathcal{T}_j$, $j = 1, 2, \dots, NTR - 1$ (in version 1.0 INZEI(41), ... ,INZEI($39 + NTR$) must be set equal to 1).

6.3.3 The sequence of the FE-systems

The sequence of stiffness matrices A_j, $j = j_g, j_g + 1, \dots, l$ and the right hand side f_l of the linear system $A_l \underline{u}_l = \underline{f}_l$ which is to be solved, are required by the multigrid algorithm. These data have to be stored on the working array B. Furthermore, storage has to be reserved for the solution vectors $\underline{u}_j$, $j = j_g, j_g + 1, \dots, l$ and for the vectors of the right hand side $\underline{f}_j$, $j = j_g, j_g + 1, \dots, l - 1$. The length of these arrays on the j-th grid is obtained from IZEIG($1, j$) $*$ INZEI(4).

The way how the stiffness matrix associated with the coarsest grid has to be stored depends on the solving method which is to be used. If the coarse-grid system is to be solved directly the matrix has to be stored in VBS-format (variable bandwidth or skyline storage), whereas if an iterative solver is to be used it must be stored in CRS-format (compressed row storage), see chapter 7 for more details. All the stiffness matrices associated with the finer grids have to be stored in CRS-format.

The coding of the boundary conditions of the first kind stored in the array IBCP is realized as follows. We assign a dual number to each node, where the $(j - 1)$-th bit is set to one if and only if boundary conditions of the first kind are given for the j-th degree of freedom at this node. The dual number is transformed into a decimal one which is then stored in IBCP at the position of the node. We will give three examples to underline this procedure.

Assume that we are given boundary conditions of the first kind for the first degree of freedom at the 5-th node. Consequently we assign the dual number 01, which is 1 in the decimal system, to the node and we have to set IBCP(5)=1. Analogously, first kind boundary conditions for the second degree of freedom at the 11-th node lead to IBCP(11)=2 (dual 10) and first kind boundary conditions for both, the first and the second degree of freedom at the 20-th node imply IBCP(20)=3 (dual 11).

The addresses of the arrays describing the sequence of systems of equations have to be assigned to components of IZEIG. That is

IZEIG(27, j)	LIBCP	address of the array IBCP for coding first-kind boundary conditions $(j = j_g, \ldots, l)$
IZEIG(40, j)	LA	address of the array A of the elements of the matrices $(j = j_g, \ldots, l)$
IZEIG(41, j)	LLAZ	address of the array LAZ, which contains the pointer to the elements on the diagonal (VBS-format) or to the last non-zero element of a row (CRS-format), $(j = j_g, \ldots, l)$
IZEIG(42, j)	LLAS	address of the array LAS, which contains the column indices required for the CRS-format, $(j = j_g, \ldots, l)$
IZEIG(43, j)	LU	address of the solution vector U $(j = j_g, \ldots, l)$
IZEIG(44, j)	LF	address of the vector F of the right hand side $(j = j_g, \ldots, l)$.

The following parameter has to be defined in INZEI.

INZEI(84) number (j_g) of the coarsest grid which is to be used in the multigrid algorithm.

6.3.4 Controlling parameter for the multigrid algorithm

The multigrid algorithm requires for its implementation controlling parameter as there are, for example, the number of smoothing steps, the type of smoothing on each grid and the kind of the multigrid cycle. We will describe those arrays stored on B which contain the controlling parameter. The arrays ITYP1, NUE1, PAR1, ITYP2, NUE2, PAR2, MART, ITMG and EPS have to be defined as it is shown in the following table.

array	length in double words	meaning of the controlling parameter
ITYP1(j)	$(l+1)/2$	for $j = j_g$: method for solving the system on the coarsest grid = 1 direct method = ITYP iterative method (see section 6.2.2) for $j = j_g + 1, \ldots, l$: = ITYP number of the method of

		pre-smoothing (see section 6.2.2)
NUE1(j)	$(l+1)/2$	for $j = j_g$: number of iteration steps if the coarse-grid system is solved iteratively, without meaning if a direct method is to be applied for $j = j_g + 1, \ldots, l$: number of pre-smoothing steps ν_1 (see figure 6.1)
PAR1(j)	l	for $j = j_g$: damping parameter ω of the iterative method for solving the coarse-grid system (see section 6.2.2), without meaning if a direct method is to be applied for $j = j_g + 1, \ldots, l$: smoothing paramter ω for the method of pre-smoothing (see section 6.2.2)
ITYP2(j)	$(l+1)/2$	for $j \leq j_g$: without meaning for $j = j_g + 1, \ldots, l$: = ITYP number of the post-smoothing method (see section 6.2.2)
NUE2(j)	$(l+1)/2$	for $j \leq j_g$: without meaning for $j = j_g + 1, \ldots, l$: number of post-smoothing steps ν_2 (see section 6.1)
PAR2(j)	l	für $j \leq j_g$: without meaning for $j = j_g + 1, \ldots, l$: smoothing parameter ω of the post-smoothing method (see section 6.2.2)
MART(j)	$(l+1)/2$	for $j \leq j_g$: without meaning for $j = j_g + 1, \ldots, l$: kind of the multigrid algorithm (see section 6.2.4)
ITMG(j)	$(l+1)/2$	for $j = 1, \ldots, l-1$: without meaning (required in a future version) for $j = l$: maximum number of iteration steps

		of the multigrid algorithm to be performed
LZYK(j)	$(l+1)/2$	for $j = 1, \ldots, l-1$: without meaning (required in a future version) for $j = l$: number of iteration steps which have actually been performed (OUTPUT parameter)
EPS(j)	2	for $j = 1$: error bound ε for the stopping criterion (see section 6.2.5) for $j = 2$: relative error in the defect norm (OUTPUT parameter)
LEVEL(j)	$(l+1)/2$	auxiliary array for controlling the multigrid algorithm

The array INZEI has to contain the addresses of the following arrays of controlling parameter.

INZEI(71)	LITYP1	address of the array ITYP1
INZEI(72)	LNUE1	address of the array NUE1
INZEI(73)	LPAR1	address of the array PAR1
INZEI(74)	LITYP2	address of the array ITYP2
INZEI(75)	LNUE2	address of the array NUE2
INZEI(76)	LPAR2	address of the array PAR2
INZEI(77)	LMART	address of the array MART
INZEI(78)	LITMG	address of the array ITMG
INZEI(80)	LLZYK	address of the array LZYK
INZEI(81)	LLEVEL	address of the array LEVEL
INZEI(82)	LEPS	address of the array EPS.

Furthermore we have to define INZEI(84):

INZEI(84)	JG	number of the grid which is to be used as coarsest grid in the multigrid algorithm

Note: In the case that the linear system on the coarsest grid is solved directly the system matrix has to be stored in VBS-format.

The routine MGADFE can be used for allocating the memory required for storing the arrays of controlling parameter. It is called by

```
CALL MGADFE(IBE,LAEB,INZEI,IERR),
```

where

IBE	I*4	I	lowest address from which on the array B is available for MGSOLV

		O	lowest address from which on the array B is available for other programs
LAEB	I*4	I	length of the working array B
INZEI	I*4		information array INZEI(100)
		I	INZEI(12) (see section 6.3.2)
		O	definition of INZEI(71) ... INZEI(82)
IERR	I*4	O	error code = 0 : sufficient memory exists = 30 : not enough memory.

The parameters marked by "I" are INPUT parameter whereas the parameters marked by "O" are OUTPUT parameter. I*4 and R*8 stand for data of the type INTEGER*4 and REAL*8, respectively.

6.4 Calling the program MGSOLV

Before MGSOLV can be started all input data described in section 6.3, that is the father–son relations (see section 6.3.2), the sequence of FE-systems (see section 6.3.3) and the controlling parameter of the multigrid algorithm (see section 6.3.4), have to be defined. Additionally the information arrays IZEIG and INZEI have to be defined as described in section 6.3.

The multigrid algorithm itself requires an auxiliary array of the length (number of nodes on the finest grid * number of degrees of freedom per node) in order to work successfully. Since the array B is used as auxiliary array the free storage capacity on B (=LAEB-IBE) must be greater than the number mentioned above at the time MGSOLV is started.
MGSOLV has to be called as follows:

```
CALL MGSOLV(IRUF,NTR,IBE,LAEB,IZEIG,INZEI,B,IERR),
```

where

IRUF	I*4	I	= 0 : the program is used as solver of a Dirichlet problem using Domain Decomposition methods = 1 : the program is used as sequential program
NTR	I*4	I	number of levels in the sequence of grids
IBE	I*4	I	lowest address from which on the array B is available for MGSOLV
		O	input unchanged
LAEB	I*4	I	length of the working array B
IZEIG	I*4		information array IZEIG(64,NTR) (definition according to section 6.3)
INZEI	I*4		information array INZEI(100)

		I	(definition according to section 6.3)
B	R*8	I/O	working array B
IERR	I*4	O	error code $= 0$: program terminated successfully $= 30 + l$: not enough memory.

The parameters marked by "I" are INPUT parameter whereas the parameters marked by "O" are OUTPUT parameter. I*4 and R*8 stand for data of the type INTEGER*4 and REAL*8, respectively.

The program MGSOLV defines the following two parameter of the array INZEI.

INZEI(83)	ICC	$= 1$: The matrix associated with the coarsest grid being used in the multigrid algorithm has been decomposed. $= 0$: No decomposition of this matrix has been performed.
INZEI(85)	IABBR	Code of termination of the multigrid algorithm. $= 1$: The maximum number of iterations has been performed. $= 2$: The stopping criterion (see equation (6.12)) has been satisfied. $= 3$: The maximum number of iterations has been performed and the stopping criterion has been satisfied.

6.5 Tree structure of the program MGSOLV

```
MGSOLV
 |
 |                  ┌─ ACTCOL  ─ SCAPR
 |                  ├─ MGCCDE
 |                  ├─ MGCCLO
 |                  ├─ MGJAC1  ─ VDCOPY
 ├─ MGSMOO  ────────┼─ MGJAC2  ─ VDCOPY
 |                  ├─ MGGSB1  ─ VDCOPY
 |                  ├─ MGGSB2  ─ VDCOPY
 |                  ├─ MGGSB3  ─ VDCOPY
 |                  └─ MGGSB4  ─ VDCOPY
 |
 |                  ┌─ AXMKLZ
 |                  ├─ VDAXPY
 |                  ├─ SCAPR
 |                  ├─ MGSMOO  ─ · · ·
 └─ MGALGO  ────────┤
                    ├─ MGPRGE  ─ MGPRG1  ─ VDCOPY
                    ├─ MGPRFE  ─ MGPRF1  ─ VDCOPY
                    |           ┌─ AXMKLZ
                    └─ MGABBR  ─┼─ VDAXPY
                                └─ SCAPR
```

6.6 A short description of the routines

ACTCOL	Solving a system directly, where the matrix is being LDL^T–decomposed. The matrix has to be stored in VBS-format.
AXMKLZ	Matrix times vector multiplication, where the matrix has to be stored in CRS-format.
MGABBR	Test termination criteria of the multigrid algorithm.
MGALGO	Controlling of the multigrid algorithm.
MGCCDE	Cholesky-decomposition of a matrix given in VBS-format.
MGCCLO	Forward and backward substitiution of the Cholesky–method.
MGGSB1	ω–Gauss–Seidel–method forward, where the matrix has to be stored in CRS-format.
MGGSB2	ω–Gauss–Seidel–method backward, where the matrix has to be stored in CRS-format.
MGGSB3	Block–ω–Gauss–Seidel–method forward, where the matrix has to be stored in CRS-format.
MGGSB4	Block–ω–Gauss–Seidel–method backward, where the matrix has to be stored in CRS-format.
MGJAC1	ω–Jacobi–method, where the matrix has to be stored in CRS-format.
MGJAC2	Block–ω–Jacobi–method, where the matrix has to be stored in CRS-format.
MGPRFE	Adapter program for selecting the algorithm of interpolation.
MGPRF1	Linear interpolation with the weights 1/2.
MGPRGE	Adapter program for selecting the algorithm of restriction.
MGPRG1	Restriction with the weights 1/2.
MGSMOO	Adapter program for selecting the method of smoothing.
MGSOLV	Calling the multigrid algorithm.
SCAPR	Compute the scalar product of two vectors.
VDAXPY	Vector operation $X = Y + \alpha Z$ for REAL*8 arrays.
VDCOPY	Copy a REAL$*$8–vector.
VICOPY	Copy an INTEGER$*$4–type vector.

Chapter 7

The Library GSGen

7.1 Introduction

The library GSGEN contains FORTRAN routines for generating finite element matrices for a given triangular grid. We allow the triangles having either straight edges or curved edges (arcs of a circle or parabola). The coarsest grid can be generated by using the program PARMESH [8], a hierarchically built up sequence of triangulations of a plain domain starting at a given coarse grid can be generated by using the program GIGEHI (chapter 5). The present version of the library generates f.e. stiffness matrices for linear elliptic, symmetric boundary value problems of second order for linear shape functions. It is able to generate Jacobi matrices which are required for solving nonlinear elliptic problems (occuring, e.g., in magnetic field calculations) by the Newton method (cf. chapter 8).
The library GSGEN described in the following can be used on both, parallel computers with MIMD architecture and sequential computers. Using it on parallel machines requires the domain to be decomposed into subdomains, where each of them is handled by one processor. The generation of both, the coarsest mesh (cf. PARMESH) and the sequence of hierarchical meshes is then performed in parallel for all subdomains, that is the program GIGEHI has to be started on each processor. Then the program GSGEN can be started on all processors for the desired grids. The local f.e. matrices are generated on all processors completely in parallel.
In the following we describe the data structure used within GSGEN, the methods of matrix storage, the system generating algorithm, and the input and output data.

7.2 Generation of finite element matrices

The standard f.e. assembling procedure contains a loop over all finite elements, it generates element stiffness matrices and assembles the element matrices into the global matrix. At the same time the load vector is assembled. If it is possible, e.g., in the case of constant coefficients and constant right-hand side, integrals are solved analytically, not by quadrature formulae.
Dirichlet boundary conditions are set as follows: The row and the column corresponding to a node with a Dirichlet boundary condition have the only non-zero element `1.0D0` in the main diagonal. The load vector contains the value of the boundary condition at the corresponding position. Further non-zero elements in this row or column are eliminated immediately. Penalty methods are not applied within the library GSGEN.
When using GSGEN on a parallel machine where each of the subdomains is handled by one processor, the matrix on one processor contains only the entries due to finite elements inside

of the processor's own subdomain. This is exactly the method of storage desired for parallel domain decomposition solvers.
Curved edges of an element are treated as straight ones in all subroutines. The error caused by this is in the order of magnitude of the discretization error. Further, the experience with the program FEMGP [48] justifies this simplification.

7.3 Input data

7.3.1 Arrays describing the mesh

We assume that a sequence of meshes is given by the arrays described in chapter 5. The system generation is done on one grid independently of all other grids. If systems are required for several grids (e.g., for multilevel methods), the subroutine GSGEN1 has to be called for each grid. The subroutine GSGENG (cf. 7.6) realizes these calls.
For sake of completeness, the arrays are given here.
The following names will be used for quantities describing the meshes:

NUMNP	number of nodes (overall)
NUMED	number of edges
NUMBED	number of edges on the boundary
NDF	maximum number of degrees of freedom per node
NTR	number of grids within the hierarchy.

The following arrays are generated for describing the sequence of meshes. All the integer arrays are of the type INTEGER∗4 and all the real arrays are of the type REAL∗8.

- IED – array of edges

IED$(1,j)$	starting node of the j-th edge		
IED$(2,j)$	ending node of the j-th edge		
IED$(3,j)$	intermediate point of the j-th edge	$= 0$:	for a straight edge
		$\neq 0$:	number of the intermediate point (see array XM)
IED$(4,j)$	pointer	$= 0$:	inner edge
		$\neq 0$:	edge on the boundary number of the corresponding coupling edge
IED$(5,j)$	type of the edge	$= 1$:	straight line
		$= 2$:	arc of a circle
		$= 3$:	arc of a parabola

  $j = 1(1)$NUMED.

- IEDM – array containing further information concerning the edges

IEDM$(1,j)$	number of subedges of the j-th edge
IEDM$(2,j)$	number of the first subedge

$j = 1(1)$NUMED.

Remark: The subedges of an edge have to be numbered consecutively.

- IECE – array of the element-connectivity

IECE$(1,j)$	
IECE$(2,j)$	number of the edges belonging to the j-th triangle
IECE$(3,j)$	
IECE$(4,j)$	material id

$j = 1(1)$NUMEL.

Remark: This array can be used for the generation of the FE-system. The typical FE-array IX of the element connectivity via the nodes can be easily generated from the array IED. We assume the following relation between IED and IX:

IX$(i,.)$ is the node opposite to the edge IED$(i,.)$.

- IBC – array for coding the boundary conditions related to the edges

IBC$(1,j)$	number of the edge	
IBC$(2,j)$	boundary material id	
IBC$(3,j)$ ... IBC$(NDF+2,j)$	$= 1$:	boundary conditions of the 1st kind
	$= 2$:	boundary conditions of the 2nd kind
	$= 3$:	boundary conditions of the 3rd kind

$j = 1(1)$NUMBED.

- X – array of the coordinates of the nodes

X$(1,j)$	x_1–coordinate of the j-th node
X$(2,j)$	x_2–coordinate of the j-th node

$j = 1(1)$NUMNP.

- XM – array of the coordinates of the intermediate nodes for curved edges

XM$(1,j)$	x_1–coordinate of the j-th intermediate node
XM$(2,j)$	x_2–coordinate of the j-th intermediate node

$j = 1(1)$NUMMP.

Arrays describing "father–son–relations"

- ISONZ – array of pointers onto data about the parent-nodes in ISONN

ISONZ(j)	pointer to the last data in ISONN concerning parent-nodes of the j-th node

$j = 1(1)$NUMNP (number of fine-grid nodes).

- ISONN – array of the parent-nodes

ISONN(i), i = ISONZ($j-1$) +1	number of parent-nodes of the j-th node ($ianz$) $=1$: node which belongs to the coarser grid $\neq 1$: number of parent nodes
ISONN($i+1$), ..., ISONN($i+ianz$)	for $ianz = 1$: number of the j-th node according to the coarse-grid numbering for $ianz \neq 1$: numbers of the parent-nodes according to the coarse-grid numbering
ISONN($i+ianz+1$)	pointer to the appropriate weights in the array WK if weights $\neq 1/2$ are to be used for interpolation and restriction

$j = 1(1)$NUMNP (number of fine-grid nodes).

The following array IBCP is generated in the subroutine GSGIVO which is part of the program GSGEN.

- IBCP - Dirichlet boundary conditions stored with respect to nodes

IBCP(i)	The integer is read as binary number. The binary digit for 2^{n-1} means: =1 : Dirichlet (essential) boundary conditions for the n-th degree of freedom =0 : natural boundary conditions for the n-th degree of freedom *Example:* If IBCP(j)=5 there are Dirichlet boundary conditions for the first and the third degree of freedom, and natural boundary conditions for all other degrees of freedom in the node j.

$j = 1(1)$NUMNP (number of fine-grid nodes).

7.3.2 Coefficients for linear problems and boundary data

For linear problems, all coefficients (including the right-hand side) are taken from the array RMatle in the COMMON block GeomLoc (see subsection **??**, cf. also [16]). They are supposed as constant in a subdomain. The boundary values are taken from the array RBoundle of the same COMMON block.
Therefore the user data input can be realized via the standard file, cf. 3.1.2.

7.3.3 Coefficients for nonlinear problems

The "element subroutine" was chosen as an interface in order to keep the library GSGEN opened for new classes of linear and nonlinear problems. It is called in the subroutine GSELEM which can be extended in a simple way. The element subroutine is chosen via the material id and an index vector. An array of REAL*8 data is submitted to it. The element subroutine generates

- in the case of linear problems the element stiffness matrix and the element load vector,
- in the case of nonlinear problems (cf. chapter 8) the element Jacobi matrix and the current element defect.

The following arrays are required:

- MAIDX – material index array

MAIDX(1,j)	number of element subroutine for material id j
MAIDX(2,j)	length of the REAL*8 data array DATMAT to be used in the element subroutine
MAIDX(3,j)	pointer to the data array DATMAT in the array DATMA

- DATMA – material data array

  DATMA(MAIDX(3,j)) ... DATMA(MAIDX(3,j)+MAIDX(2,j)-1)
  REAL*8 data array DATMAT to be used in the element subroutine
  for material id j

These data arrays are read from the Newton control data file (.nc-file), cf. 3.2.1. Therefore the user data input can be realized via the (.nc) file.
The element subroutine ELEMxx (xx is substituted by 01 ... 99) is called by the line:

```
CALL ELEMxx(NROUT,MAT,MATANZ,DATMAT,S,P,
  UUL,XL,IBCL,UL,AREA,COEFF),
```

where

NROUT	I*4	I	number of element subroutine
MAT	I*4	I	material id
MATANZ	I*4	I	length of array DATMAT
DATMAT	R*8	I	data array for a material
S	R*8	O	element stiffness (resp. Jacobi) matrix S(3,3)
P	R*8	O	element load (resp. defect) vector P(3)
UUL	R*8	I	local approximate solution of the nonlinear problem, UUL(3)
XL	R*8	I	local node coordinates, i.e. array XL(2,3) of the coordinates of the 3 nodes
IBCL	I*4	I	local array IBCL(3), corresponding to the array IBC for the 3 nodes
UL	R*8	I	local approximate solution of the linear problem, UL(3)
AREA	R*8	O	area of the element
COEFF	R*8	O	array COEFF(2,2) of the coefficients

The parameter marked by "I" are INPUT parameters whereas the parameters marked by "O" are OUTPUT parameters. I*4 and R*8 stand for data of the type INTEGER*4 and REAL*8, respectively.
For subdomains with linear material behaviour, i.e., if the coefficients are independent of the solution, the parameters AREA and COEFF can be omitted. For linear problems, the parameter UUL can be omitted, too.
The current version of the adapter subroutine GSELEM is given in the following. Possible extensions are obvious.

```
      SUBROUTINE GSELEM(IZEIG,INZEI,B,IG,x,u,ul,uu,uul,
     +                  iece,ied,ibcp,maidx,datma,
     +                  AFL,BFL)
```

```
        ... source code ...

C *** Loop over elements
        do 8 i = 1,numel
C *** Getting element data

        ... source code ...

C *** Getting material id, element subroutine number, pointer & length
        mat = iece(4,i)
        if (mat.le.0) then
          nrout = 1
        else
          nrout = maidx(1,mat)
        endif
        if (nrout.le.0) nrout = 1
        matanz = maidx(2,mat)
        matanf = maidx(3,mat)
C *** nonlinear problem - Newton's method
        nlin = (inzei(1).eq.2)
c *** nonlinear material
        nlMat = (nrout.eq.3.or.nrout.eq.4.or.nrout.eq.8)
C *** Loop over element nodes
        do 4 j = 1,nen2

        ... source code ...

  4     continue
C ***
C *** Calling the element subroutine
C ***
        goto (101,102,103,104,105,106,107,108,109,110),nrout
c *** Laplace u = f
 101    call elem01(nrout,mat,matanz,datma(matanf),
    +              b(ls),b(lp),ul,xl,ibcl)
        goto 200
C *** - div (k * grad u ) = f
 102    call elem02(nrout,mat,matanz,datma(matanf),
    +              b(ls),b(lp),ul,xl,ibcl)
        goto 200
C *** nonlinear materials (iron) in nonlinear magnetic fields, without rhs
 103    call elem03(nrout,mat,matanz,datma(matanf),
    +              b(ls),b(lp),ul,xl,ibcl,uul,Area,Coeff)
        SumArea = SumArea + Area
        Call VDaXpY(4,WSumCoeff,WSumCoeff,Area,Coeff)
        goto 200
C *** Kawohl problem (see literature:Kawohl, Stara, Wittum)
 104    call elem04(nrout,mat,matanz,datma(matanf),
    +              b(ls),b(lp),ul,xl,ibcl,uul,Area,Coeff)
        SumArea = SumArea + Area
        Call VDaXpY(4,WSumCoeff,WSumCoeff,Area,Coeff)
        goto 200
C *** Nonlinear magnetic field calculations, with currents and perm. magnets
C *** but linear materials (copper windings, permanent magnets)
 105    call elem05(nrout,mat,matanz,datma(matanf),
    +              b(ls),b(lp),ul,xl,ibcl,uul)
```

```
       goto 200
C *** Linear problems: data due to RMatle on COMMON GeomLoc
 106   continue
       call elem06(nrout,mat,matanz,datma(matanf),
     +             b(ls),b(lp),ul,xl,ibcl)
       goto 200
C *** Linear problems: data due to RMatle on COMMON GeomLoc
C *** BUT ! RMatle(5) = radial magnetization w.r.t. origin
C *** (to be used for special electric machines)
 107   continue
       call elem07(nrout,mat,matanz,datma(matanf),
     +             b(ls),b(lp),ul,xl,ibcl)
       goto 200
C *** Nonlinear magnetic field problems
 108   call elem08(nrout,mat,matanz,datma(matanf),
     +             b(ls),b(lp),ul,xl,ibcl,uul,Area,Coeff)
       SumArea = SumArea + Area
       Call VDaXpY(4,WSumCoeff,WSumCoeff,Area,Coeff)
       goto 200
C *** Plane linear elasticity problems
 109   call elem09(nrout,mat,matanz,datma(matanf),
     +             b(ls),b(lp),ul,xl,ibcl)
       goto 200
c ***
 110   continue
c ***
 200   continue
c *** Assembling of matrices
       if (iass.eq.1.or.iass.eq.3) then
         call addst1(b(la),b(lial1),b(lial2),b(lf),b(ls),b(lp),
     +               loix,ibcl,nen2,ndf,afl,bfl)
       elseif(iass.eq.2) then
         call addst2(b(la),b(lial1),b(lf),b(ls),b(lp),
     +               loix,ibcl,nen2,ndf,afl,bfl)
      endif
  8   continue
c* change RMatle for nonlinear materials for better crosspoint system
c* on the coarsest grid for nonlinear materials only !
       if (IG.eq.InZei(84)) then
         if (nlMat) then
           if (SumArea.gt.0.0d0) then
      call VDDiv(4,NewRMatle,1,WSumCoeff,1,SumArea,0)
      call ChRMatle(NewRMatle)
           endif
         endif
       endif
      return
      end
```

The element subroutines of the present version are corresponding with the following boundary value problem types. (L means a linear problem, N means a nonlinear problem solved by Newton's method.)

ELEMO1 L $\int_\Omega \nabla^\top u \nabla v \; dx = \int_\Omega f v \; dx$,
with $f =$ `DATMAT(8)`,

ELEM02 L $\int_\Omega k \nabla^\top u \nabla v \; dx = \int_\Omega f v \; dx$,
with $k =$ `DATMAT(1)` and $f =$ `DATMAT(8)`,

ELEM03 N $\int_\Omega \nu \left(|\nabla^\top u| \right) \nabla^\top u \nabla v \; dx = 0$
(magnetic field problem),

ELEM04 N $\int_\Omega \nu \left(|\nabla^\top u| \right) \nabla^\top u \nabla v \; dx = \int_\Omega f v \; dx$,
with $f =$ `DATMAT(8)` and $\nu(z)$ being the function of the "Kawohl" problem [41],

ELEM05 N $\int_\Omega \nu \nabla^\top u \nabla v \; dx = \int_\Omega f v - \mathrm{H}_{0y} \frac{\partial v}{\partial x} + \mathrm{H}_{0x} \frac{\partial v}{\partial y} \; dx$,
with $\mathrm{H}_{0x} = \mathrm{H}_0 \frac{x}{\sqrt{x^2+y^2}}$, $\mathrm{H}_{0y} = \mathrm{H}_0 \frac{y}{\sqrt{x^2+y^2}}$,
$\mathrm{H}_0 =$ `DATMAT(5)`, $\nu =$ `DATMAT(1)` and $f =$ `DATMAT(8)`
(magnetic field problem, linear material with current and radial permanent magnetization),

ELEM06 L $\int_\Omega \nabla^\top u \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \nabla v + c u v \; dx = \int_\Omega f v \; dx$,
with $a_{11} =$ `DATMAT(1)`, $a_{12} = a_{21} =$ `DATMAT(2)`, $a_{22} =$ `DATMAT(4)`, $c =$ `DATMAT(7)` and $f =$ `DATMAT(8)`,

ELEM07 L $\int_\Omega \nabla^\top u \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \nabla v + c u v \; dx = \int_\Omega f v - \mathrm{H}_{0y} \frac{\partial v}{\partial x} + \mathrm{H}_{0x} \frac{\partial v}{\partial y} \; dx$,
with $\mathrm{H}_{0x} = \mathrm{H}_0 \frac{x}{\sqrt{x^2+y^2}}$, $\mathrm{H}_{0y} = \mathrm{H}_0 \frac{y}{\sqrt{x^2+y^2}}$,
$\mathrm{H}_0 =$ `DATMAT(5)`, $a_{11} =$ `DATMAT(1)`, $a_{12} = a_{21} =$ `DATMAT(2)`, $a_{22} =$ `DATMAT(4)`, $c =$ `DATMAT(7)` and $f =$ `DATMAT(8)`
(linear magnetic field problem, with current and radial permanent magnetization),

ELEM08 N $\int_\Omega \nu \left(|\nabla^\top u| \right) \nabla^\top u \nabla v \; dx = \int_\Omega f v \; dx$,
with $f =$ `DATMAT(8)`
(nonlinear magnetic field problem),

ELEM09 L $\int_\Omega e^\top(v) \, D \, e(u) \; dx = \int_\Omega v^\top f \; dx$,
with $e(u) = (\epsilon_{11}(u), \, \epsilon_{22}(u), \, 2\,\epsilon_{12}(u))^\top$, $\epsilon_{ij}(u) = (\partial u_i / \partial x_j + \partial u_j / \partial x_i)/2$

$$D = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \text{ for } \texttt{DATMAT(2)} = 1,$$

$$D = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix} \text{ for } \texttt{DATMAT(2)} = 2,$$

$E = \mathtt{DATMAT(1)}$, $\nu = \mathtt{DATMAT(3)}$, $f = (f_1, f_2)^\top = (\mathtt{DATMAT(7)}, \mathtt{DATMAT(8)})^\top$,
(plane linear elasticity problem).

The array $\mathtt{DATMAT(1 \cdots 8)}$ contains the values of the COMMON block GeomLoc. Therefore subroutine ELEM06 can be used for general linear problems in connection with data input via this block, as it is described in subsection 7.3.2.

For nonlinear magnetic field problems the material data (table of values: magnetic field strength H - induction B, and the space permeability) have to be pre-processed by the program MONSPI. The program MONSPI computes the coefficients for monotonicity preserving C^1 spline interpolation [7, 33]. The program MONSPI runs separately for each nonlinear material. The output of MONSPI is included into the Newton control (.nc) file (see 3.2.1) which is input of FEM⊗BEM .

Further versions of GSGEN will use several "boundary subroutines" RANDxx and DRANxx in dependence of the boundary material id IBC(2,j). Then the array MRIDX has to contain integer data (number of subroutine, number of data, pointer to data), and the array DATMR has to contain real data.

7.4 Methods of matrix storage

The library GSGEN uses two different methods of matrix storage, compressed row storage (CRS), and variable bandwidth storage (VBS). The VBS is prefered for the coarsest grid since it allows the efficient realization of the direct solver. The CRS is prefered for the finer grids since it requires minimal storage. For symmetric matrices, only the upper triangle is stored.

7.4.1 Compressed row storage

Only the nonzero elements of the right upper triangle are stored sequentially on a one-dimensional array A. The auxiliary vector IAL2 (also denoted as LAZ, length is number of nonzeros MEX) contains the column index of each element in A. The auxiliary vector IAL1 (also denoted as LAS, length is number of unknowns NEQ = NUMNP $*$ NDF) contains pointers to the last element of each row in the array A.

The storage method CRS is schematically described in figure 7.1.

7.4.2 Variable bandwidth storage

The elements of the upper triangle of a symmetric matrix are stored column-wise, each column beginning with its first (i.e., uppermost) element on a one-dimensional array A. In VBS, also zero elements below the first nonzero of a column are stored. The only auxiliary vector IAL1 (also LAS, length NEQ = NUMNP $*$ NDF) contains pointers to the last element of each column (i.e., the main diagonal element) in the array A.

The storage method VBS is schematically described in figure 7.2.

7.4.3 Arrays for solution and right-hand side

Approximate solutions and right-hand sides are stored as follows:

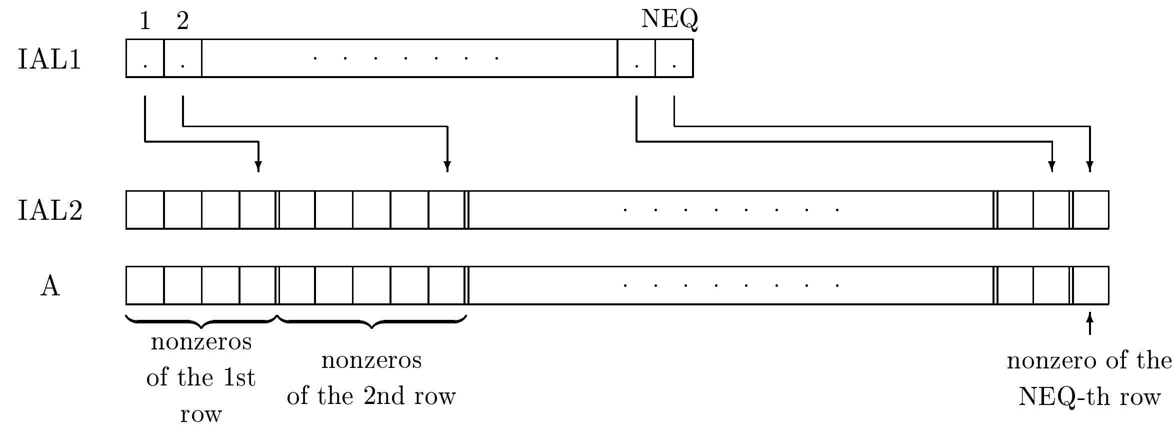

Figure 7.1: Scheme of CRS

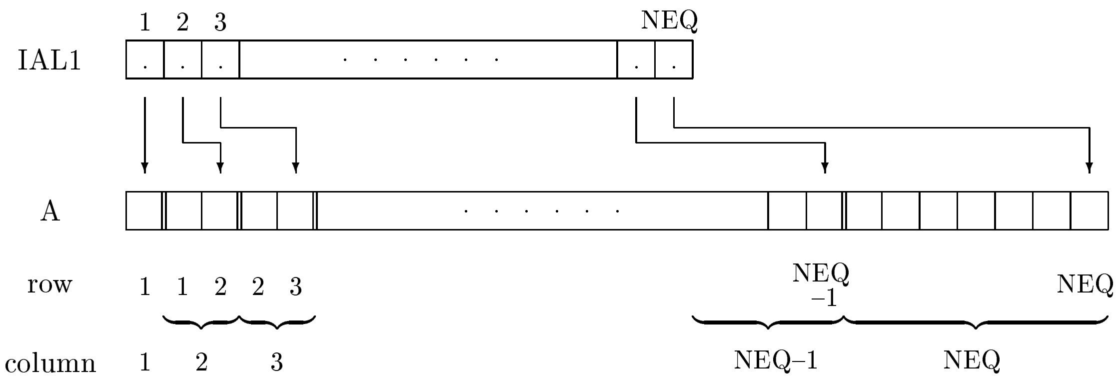

Figure 7.2: Scheme of VBS

- U – array of solution

U(1,j)	solution for the 1st degree of freedom in node j
U(2,j)	solution for the 2nd degree of freedom in node j
...	...
U(NDF,j)	solution for the last (NDF-th) degree of freedom in node j

j = 1(1)NUMNP

- F – array of right-hand side or defect

F(1,j)	right-hand side (resp. defect) for the 1st degree of freedom in node j
F(2,j)	right-hand side (resp. defect) for the 2nd degree of freedom in node j
...	...
F(NDF,j)	right-hand side (resp. defect) for the last (NDF-th) degree of freedom in node j

j = 1(1)NUMNP

7.5 Storage management

All the arrays describing the systems of equations are being stored on the REAL∗8 working array B. The INTEGER∗4 arrays IZEIG(64, NTR) and INZEI(100) contain both, information concerning the discretization itself, e.g. number of nodes and number of elements belonging to each grid, and the addresses of the arrays, describing the meshes, in B. The parameter LAEB stands for the length of B in double words and B(IBE) is the first free element of the array B, that is IBE is the address from which on B can be used at the present state of the execution of the program.
The table below shows the content of the array IZEIG concerning the discretization.

IZEIG$(1, j)$	NUMNP	number of nodes $(j = 1, \ldots, NTR)$
IZEIG$(5, j)$	NUMED	number of edges $(j = 1, \ldots, NTR)$
IZEIG$(6, j)$	NUMBED	number of edges on the boundary $(j = 1, \ldots, NTR)$
IZEIG$(8, j)$	NUMEL	number of elements $(j = 1, \ldots, NTR)$
IZEIG$(9, j)$	NUMMP	number of intermediate points $(j = 1, \ldots, NTR)$
IZEIG$(16, j)$	LX	address of the array X $(j = 1, \ldots, NTR)$
IZEIG$(17, j)$	LXM	address of the array XM $(j = 1, \ldots, NTR)$
IZEIG$(18, j)$	LIBC	address of the array IBC $(j = 1, \ldots, NTR)$
IZEIG$(19, j)$	LIECE	address of the array IECE $(j = 1, \ldots, NTR)$
IZEIG$(20, j)$	LIED	address of the array IED $(j = 1, \ldots, NTR)$
IZEIG$(24, j)$	LISONZ	address of the array ISONZ $(j = 2, \ldots, NTR)$
IZEIG$(25, j)$	LISONN	address of the array ISONN $(j = 2, \ldots, NTR)$
IZEIG$(33, j)$	IASS	method of matrix storage $(j = 2, \ldots, NTR)$

		= 1 : compressed row storage (CRS)
		= 2 : variable bandwidth storage (VBS)
		= 3 : CRS and VBS (matrix stored two times for different purposes)
IZEIG$(34, j)$	MEX	number of matrix elements stored $(j = 2, \ldots, NTR)$
IZEIG$(35, j)$	MEXV	number of matrix elements stored for additional VBS (i.e., in case of IZEIG$(33, j)$=3) $(j = 2, \ldots, NTR)$
IZEIG$(40, j)$	LA	address of the array A $(j = 2, \ldots, NTR)$
IZEIG$(41, j)$	LIAL1	address of the array IAL1 $(j = 2, \ldots, NTR)$
IZEIG$(42, j)$	LIAL2	address of the array IAL2 (for CRS only) $(j = 2, \ldots, NTR)$
IZEIG$(43, j)$	LU	address of the array U $(j = 2, \ldots, NTR)$
IZEIG$(44, j)$	LF	address of the array F $(j = 2, \ldots, NTR)$
IZEIG$(45, j)$	LUU	address of the array UU $(j = 2, \ldots, NTR)$
IZEIG$(46, j)$	LAV	address of the array A for additional VBS (i.e., in case of IZEIG$(33, j)$=3) $(j = 2, \ldots, NTR)$
IZEIG$(47, j)$	LIAVL	address of the array IAL1 of additional VBS (i.e., in case of IZEIG$(33, j)$=3) $(j = 2, \ldots, NTR)$

We demonstrate the use of the arrays IZEIG and INZEI by presenting a little program which prints the content of the array IED (array of edges) of the j-th level.

```
      SUBROUTINE UVW(J,B,IZEIG,INZEI)
      IMPLICIT NONE
      INTEGER*4  J,IZEIG,INZEI,NUMED,LIED
      REAL*8     B
      DIMENSION  B(1),IZEIG(64,1),INZEI(1)
C ***
      ... source code ...

      NUMED = IZEIG(5,J)
      LIED = IZEIG(20,J)
      CALL DRUCK(J,NUMED,B(LIED))

      ... source code ...

      RETURN
      END
C
C Subroutine for printing the content of IED on the j-th grid
C
      SUBROUTINE DRUCK(J,NUMED,IED)
      IMPLICIT NONE
      INTEGER*4  J,NUMED,IED,I,K
      DIMENSION  IED(5,1)
C ***
      WRITE(6,*) ' Output of the array of the edges of the ',j,'-th grid'
      WRITE(6,*)
      DO 1 I=1,NUMED
      WRITE(6,*) (IED(K,I),K=1,5)
    1 CONTINUE
```

```
RETURN
END
```

7.6 Calling the program GSGEN

Before starting the program GSGEN one has to ensure that the arrays IED, IECE, IBC, ICBN, X and XM (see section 5.4) have already been defined for the desired meshes. Furthermore, the parameters IZEIG(1, 1) to IZEIG(10, 1), IZEIG(16, 1) to IZEIG(21, 1), IZEIG(24,j), and IZEIG(25,j) for $j = 1, \ldots, NTR$ must have been defined before GSGENG is called.
The subroutine GSGENG calls GSGEN1 and the preparing routines GSVORB and GSGIVO if the generation of the systems for the grids $j = JG, \ldots, NTR$ is desired. There are called:

- GSVORB - common preparation for GSGEN1,
- GSGIVO - if $JG > 1$: preparation for the grids $1, \ldots, JG - 1$,
- GSGIVO and GSGEN1 - preparation and system generation for grids $JG, \ldots, NTR$,

The following parameters have to be defined in INZEI:

INZEI(1)	Type of problem = 1 : linear boundary value problem = 2 : nonlinear boundary value problem
INZEI(5)	maximum number of degrees of freedom per node
INZEI(12)	number of grid in the sequence of discretizations (NTR)
INZEI(84)	coarsest mesh for which a system is to generate (JG)

The program GSGENG is called as follows:

```
CALL GSGENG(IBE,LAEB,IZEIG,INZEI,B,IERR),
```

where

IBE	I*4	I	lowest address from which on the array B is available for GSGEN
		O	lowest address from which on the array B is available for other programs
LAEB	I*4	I	length of the working array B
IZEIG	I*4		array of pointers IZEIG(64, NTR) (see also section 5.4)
		I	information concerning all grids
		O	information concerning systems of equations
INZEI	I*4		information array INZEI(100)
		I	see table above

		O	unchanged input-data, INZEI(51) through INZEI(61) are used for internal parameters
B	R*8	I/O	working array B
IERR	I*4	O	error code = 0 : all system generation steps that had been demanded were performed = 2000...2999: memory too small, the error occured in the $[(IERR-2000)/100]$-th grid

The parameter marked by "I" are INPUT parameters whereas the parameters marked by "O" are OUTPUT parameters. I*4 and R*8 stand for data of the type INTEGER*4 and REAL*8, respectively.

7.7 Tree structure of the program GSGENG

GSGENG

- ↪ GSVORB
- ↪ GSGIVO
  - ↪ VDCOPY
  - ↪ VICOPY
  - ↪ DIRVAT
  - ↪ DIRSET
    - ↪ VDCOPY
    - ↪ DRAN01
- ↪ GSGEN1
  - ↪ PRNKLZ
    - ↪ VICOPY
    - ↪ ASSMLN
  - ↪ PRNVBS
    - ↪ VICOPY
  - ↪ VDCOPY
  - ↪ GSELEM
    - ↪ VDCOPY
    - ↪ ELEM01
    - ↪ ELEM02
    - ↪ ELEM03
      - ↪ KOEFF
    - ↪ VDAXPY
    - ↪ ELEM04
    - ↪ ELEM05
    - ↪ ELEM06
    - ↪ ELEM07

$\hookrightarrow$ ELEM08
  $\hookrightarrow$ KOEFF
$\hookrightarrow$ ELEM09
$\hookrightarrow$ VDDIV
$\hookrightarrow$ CHRMATLE
$\hookrightarrow$ ADDST1
$\hookrightarrow$ ADDST2

$\hookrightarrow$ GSRAND
  $\hookrightarrow$ RAND01
  $\hookrightarrow$ ADDST1
  $\hookrightarrow$ ADDST2

$\hookrightarrow$ GROVBS
  $\hookrightarrow$ UKLVBS
    $\hookrightarrow$ VICOPY
    $\hookrightarrow$ VDCOPY

7.8 A short description of the routines

ADDST1	Assembling local (element or boundary) matrix entries into the global matrix (CRS).
ADDST2	Assembling local (element or boundary) matrix entries into the global matrix (VBS).
ASSMLN	Adapting the output of PRNKLZ to the CRS method being used.
CHRMATLE	Changing the coefficients in the array RMatle of the COMMON block GeomLoc for better preconditioning of the Jacobi matrix.
DIRSET	Generating IBCP from IED, setting Dirichlet boundary conditions in the solution array.
DIRVAT	Generating IBCP on the current grid from IBCP on the coarser grid.
DRAN01	Boundary subroutine (Dirichlet boundary condition).
ELEMxx	Element subroutines for linear and nonlinear problems, see 7.3.3, page 79.
ELEM06	Element subroutine for linear symmetric problems, input of all coefficients via RMatle.
GROVBS	Generating an additional VBS matrix from CRS matrix.
GSELEM	Generating the element part of matrix and right-hand side.
GSGEN1	Generating the system.
GSGENG	Generating the systems for certain grids.
GSGIVO	Preparing for system generation on one grid.
GSRAND	Generating the boundary part of matrix and right-hand side.
GSVORB	Setting addresses for system generation.

KOEFF	Computing the coefficient ν, and the derivative ν' depending of B by monotonicity preserving cubic spline interpolation.
MONSPI	Computing input data for monotonicity preserving cubic spline interpolation from given H – B – table. Running separately as a preprocessor.
PRNKLZ	Determining the structure of the finite element matrix for CRS
PRNVBS	Determining the structure of the finite element matrix for VBS
RAND01	Boundary subroutine.
UKLVBS	Ordering matrix elements from CRS into VBS.
VDAXPY	Vector operation $X = Y + \alpha * Z$ for REAL*8 arrays.
VDCOPY	Copy a REAL*8–type vector.
VDDIV	Vector division for REAL*8 arrays.
VICOPY	Copy an INTEGER*4–type vector.

Chapter 8

The Library NewtonSolv

8.1 Introduction

The library NEWTONSOLV contains FORTRAN routines that realize Newton's method for the solution of nonlinear systems of equations. Damped (relaxed) and non-damped Newton methods are possible, there are control routines to choose suitable relaxation parameters, and to decide whether to accept an approximate solution, or to refuse it. If the linear system is solved approximately, the accuracy of the solver is controlled by the Newton method, too.
The library NEWTONSOLV described in the following can be used on both, parallel computers with MIMD architecture and sequential computers. Using it on parallel machines requires the domain to be decomposed into subdomains, where each of them is handled by one processor.
Libraries for generating Jacobi matrices (e.g., GSGEN, cf. chapter 7) and for solving linear problems are assumed. For applying the "nested iteration" strategy in connection with multigrid methods, a hierarchical sequence of discretizations is required (as, e.g., generated by GIGEHI, cf. chapter 5). The library NEWTONSOLV makes use of the data structures of the other parts of the program FEM⊕BEM .
The library NEWTONSOLV is based on experience with solving nonlinear boundary value problems by the Full-Multigrid-Newton-Technique (FMGNT) [28, 29, 30, 32, 33]. This method has been implemented in the package FEMGP [48].

8.2 The Nested Newton Algorithm

The algorithm generalizes the FMGNT with respect to the linear problem solver. The algorithm connects Newton's method with the "nested iteration" principle (cf. [27]). Some results achieved with this algorithm are documented in [35, 34].
The following steps are necessary for solving (8.1) with $q = l$; the result will be the approximate solution $\mathbf{u}_l^\star$ with relative accuracy ε (nested iteration ε).

$$K_q \mathbf{u}_q = \mathbf{f}_q. \tag{8.1}$$

Algorithm NN (Nested Newton)
Step 0
Initialization of the grid number:

(0.)(P) $q := 1$.

Step 1
Set the initial solution for the grid q:

(1.1)(P) `IF` $q = 1$ `THEN` $\mathbf{u}_q^0 = \mathbf{0}$;

(1.2)(P) `IF` $q > 1$ `THEN` $\mathbf{u}_q^0 = \tilde{I}_{q-1}^q \mathbf{u}_{q-1}^\star$;
the initial solution is the interpolation of the best solution on the grid $q-1$.

(1.3)(P) Initialize the Newton iteration number
$j := 0$.

Step 2
Compute the Jacobi matrix and the defect vector

(2.)(P) $B_{q,0} = K_q'[\mathbf{u}_q^0]$;
$\mathbf{d}_q^1 = \mathbf{f}_q - K_q \mathbf{u}_q^0$

Step 3

(3.)(P) Choose a relaxation parameter τ_q^j with
$0 < \tau_q^j \leq 1$.

Step 4

(4.)(S) Solve the linear defect system

$$B_{q,j}\mathbf{w}_q^{j+1} = \mathbf{d}_q^{j+1}$$

with a parallel CG solver preconditioned by a DD method. Result is $\tilde{\mathbf{w}}_q^{j+1}$.

Step 5
Correct the solution:

(5.)(P) $\mathbf{u}_q^{j+1} = \mathbf{u}_q^j + \tau_q^j \tilde{\mathbf{w}}_q^{j+1}$.

Step 6
Control the convergence:
(The parameter c_τ with $c_\tau < 1$ is a priori chosen.)

(6.1)(P) Compute the new defect vector and the new Jacobi matrix
$\mathbf{d}_q^{j+2} = \mathbf{f}_q - K_q \mathbf{u}_q^{j+1}$;
$B_{q,j+1} = K_q'[\mathbf{u}_q^{j+1}]$.

(6.2)(C) Compute defect norms
$d_q^{j+1} = \|\mathbf{d}_q^{j+1}\|$;
$d_q^{j+2} = \|\mathbf{d}_q^{j+2}\|$;

(6.3)(P) `IF` $d_q^{j+2} \geq d_q^{j+1}$ `THEN` $\left(\begin{array}{l} \tau_q^j := \min\left\{ c_\tau \tau_q^j, \dfrac{\tau_q^j d_q^{j+1}}{d_q^{j+1} + d_q^{j+2}} \right\}; \\ \texttt{GOTO } \textit{Step 5} \end{array} \right)$;

(6.4)(P) `IF` $d_q^{j+2} \leq \epsilon d_q^1$ `THEN` $\left(\begin{array}{l} \mathbf{u}_q^\star := \mathbf{u}_q^{j+1}; \\ \texttt{IF } q < l \texttt{ THEN } (q := q+1; \texttt{ GOTO } \textit{Step 1}\ ); \\ \texttt{IF } q = l \texttt{ THEN EXIT}; \end{array} \right)$;

(6.5)(P) Perform a further Newton step:
$j := j + 1$;
`GOTO` *Step 3.*

If the algorithm PNN (Parallel Nested Newton) runs on a MIMD parallel computer, (P) indicates that the step is performed completely in parallel, i.e., independently on the processors. The note (C) indicates that global communication is necessary. The solver (S) includes parallel independent parts, communication between processors handling neighboured subdomains, and global communication.

8.3 Implementation and control parameters

In the present version, a conjugate gradient method with f.e.-b.e. domain decomposition preconditioner is implemented. A multigrid solver serves as local solver for interior subproblems on f.e. subdomains.
There are the integer control array NwCtrl(20) and the REAL*8 control array NwEps(20) for controlling the Newton method.
The input parameter "I" have to be initialized for each grid by the subroutine `InitNw`. The "standard" means a suitable choice recommended by the author.

8.3.1 INTEGER parameters

NwCtrl(1)	O	result of convergence test (defect test) =1 : convergence; next Newton iteration on that grid; =2 : no convergence, refuse iterate; new Newton iteration step on that grid; =3 : convergence, accept iterate; end iteration on that grid, go to the finer grid; =5 : no convergence, refuse iterate; end iteration on that grid, go to the finer grid;
NwCtrl(2)	O	=1 : last iterate UU accepted, add correction onto UU; =2 : last iterate UU refused, add correction to latest accepted iterate UUsave;
NwCtrl(3)	O	counter for refused iterates;
NwCtrl(4)	I	break choice parameter = 1 : break for norm of correction, =2 : break for norm of correction or of defect; standard: 1;
NwCtrl(5)	I	omitting the last matrix and defect computation =0 : no, =1 : yes, but not after one Newton iteration, =2 : yes, always if possible, standard: 0;
NwCtrl(6)	I	non-damped Newton method, =0 : no, =1 : yes, on grids $q \geq 2$ =2 : yes, always if possible, standard: 0;
NwCtrl(7)	I	maximal number of Newton iterations on the 1st (coarsest) grid, standard: 20;
NwCtrl(8)	I	maximal number of Newton iterations on the l-th (i.e., the finest) grid, standard: 10;

NwCtrl(9)	O	counter for Newton iterations;
NwCtrl(10)	I	maximal number of Newton iterations on the grids $2 \cdots l$ (i.e., the auxiliary grids), standard: 3;
NwCtrl(11)	O	result of convergence test (norm of correction) =1 : compute matrix and defect, then do defect test, =2 : convergence, accept iterate; end iteration on that grid, go to the finer grid;
NwCtrl(14)	I	generate new systems of equations for the coarse grids in the multigrid method =0 : no, =1 : yes, standard: 0;

The components 4, 5, 6, 7, 8, 10, and 14, are corresponding to the input parameters in the block NWCTRL of the (.nc) file, cf. 3.2.1.

8.3.2 REAL*8 parameters

NwEps(1)	I	relative accuracy ε with respect to the norm of defect, standard: 10^{-6};
NwEps(2)	I	relative accuracy ε with respect to the norm of correction, standard: 10^{-6};
NwEps(3)	O	initial norm of defect;
NwEps(4)	O	current norm of defect;
NwEps(5)	O	norm of defect of the latest accepted iterate;
NwEps(6)	O	energy norm of the solution of the linear system, i.e., the correction U
NwEps(7)	I	increasing factor for damping parameter; standard: 2.0;
NwEps(8)	I	reduction factor c_τ for damping parameter; standard: 0.25;
NwEps(9)	O	current damping parameter τ;
NwEps(10)	O	current relative norm of correction;
NwEps(11)	O	relative norm of correction of the latest accepted iterate;
NwEps(12)	O	energy norm first correction;
NwEps(13)	O	minimal energy norm of accepted corrections;
NwEps(14)	O	relative error δ of linear problem solver;
NwEps(15)	I	control constant for linear problem solver accuracy; standard: 4.0;
NwEps(16)	I	accuracy ε_{lin} demanded for the linear problem solver; standard (initially): 10^{-2};

The components 1, 2, 7, 8, 9, 15, and 16, are corresponding to the input parameters in the block NWEPS of the (.nc) file, cf. 3.2.1.

8.4 Calling NEWTONSOLV

For the Nested Newton algorithm, the subroutine is called in a loop over the grids:

```
      do 9 nLevel = jgg,ntrFull
      InZei(12) = nLevel
      if (nLevel.eq.jgg) then
        Call IniGlS(IBE,LAEB,IZeig,InZei,B,nLevel)
      else
        call IniNextS(IBE,LAEB,IZeig,InZei,B,nLevel)
      endif
      Call NewtonSolv(IBE,LAEB,IZeig,InZei,B,nLevel,cgEps)
 9    continue
```

where

nlevel	I∗4	I	number of current grid
jgg	I∗4	I	number of coarsest grids for Nested Newton algorithm
ntrfull	I∗4	I	number of finest grid for Nested Newton algorithm
cgEps	R∗8	I	initial value for relative accuracy of conjugate gradient solver
IBE	I∗4	I	lowest address from which on the array B is available for GSGEN
		O	lowest address from which on the array B is available for other programs
LAEB	I∗4	I	length of the working array B
IZEIG	I∗4		array of pointers IZEIG$(64, NTR)$ (see also section 5.4)
		I	information concerning all grids
		O	information concerning systems of equations
INZEI	I∗4		information array INZEI(100)
		I	see table above
		O	unchanged input-data, INZEI(51) through INZEI(61) are used for internal parameters
B	R∗8	I/O	working array B

The parameter marked by "I" are INPUT parameters whereas the parameters marked by "O" are OUTPUT parameters. I∗4 and R∗8 stand for data of the type INTEGER∗4 and REAL∗8, respectively.

8.5 Tree structure of the program NEWTONSOLV

NewtonSolv

- ↪ InitNw
- ↪ IniCoarse ∗
- ↪ NwInitDef
  - ↪ SKALAR ∗
  - ↪ DSQRT
- ↪ Ass_SKC4 ∗
- ↪ VDCOPY
- ↪ CG ∗
- ↪ NwAddCorr
  - ↪ SKALAR ∗
  - ↪ DSQRT
  - ↪ VDCOPY
  - ↪ VDAXPY
- ↪ NwTestCorr
  - ↪ SKALAR ∗
  - ↪ DSQRT
- ↪ NewtonGls
  - ↪ GSGEN1 ∗∗
  - ↪ MGPRGE ∗∗∗
- ↪ NwTestDef
  - ↪ VDCOPY
  - ↪ WADD ∗
  - ↪ SKALAR ∗
  - ↪ DSQRT

∗ subroutines from FEM⊗BEM for the solution of linear coupled f.e./b.e. problems by preconditioned cg method,
∗∗ subroutines from GSGEN, see chapter 7,
∗∗∗ subroutines from GIGEHI, see chapter 5.

8.6 A short description of the routines

Ass_SKC4	Assembling matrix entries for preconditioning on coupling boundaries.
CG	Calling the linear problem solver.
DSQRT	Square root.
GSGEN1	Generating the system of equations.
IniCoarse	Initializing the cross-point system, factorizing the cross-point matrix.
IniGls	Initializing the system of equations (calling GIGEHI and GSGENG).
IniNextS	Initializing the system of equations for the finer grid (calling GSGIVO and GSGEN1).
InitNw	Initializing the Newton control parameters.
MGPRFE	Interpolating the solution onto the finer grid.
MGPRGE	Restricting onto the coarser grid
NewtonGls	Generating the linear system of equations for all but the 1st Newton iteration.
NwAddCorr	Adding the correction U onto the approximate solution.
NwInitDef	Computing the initial defect of the nonlinear system.
NwTestCorr	Testing for convergence w.r.t. the norm of the correction U.
NwTestDef	Testing for convergence w.r.t. the norm of the defect F.
SKALAR	Computing a scalar product of two vectors. (Global communication subroutine).
VDAXPY	Vector operation $X = Y + \alpha * Z$ for REAL∗8 arrays.
VDCOPY	Copy a REAL∗8–type vector.
WADD	Adding a vector for all coupling boundaries. (Global communication subroutine).

Chapter 9

The Library OstBEM2d

9.1 Introduction

We consider the boundary integral equations related to a selfadjoint and elliptic differential operator $L(\cdot)$ of second order in the form

$$\frac{1}{2}\begin{pmatrix} u \\ t \end{pmatrix} = (-1)^j \begin{pmatrix} -K & V \\ D & K' \end{pmatrix},$$

where $j = 0$ holds for a problem in an interior domain Ω and $j = 1$ in the case of an exterior domain. The boundary integral operators are defined in the usual way as

$$\begin{aligned}
(Vt)(x) &= \int_\Gamma U^*(x,y)\, t(y)\, ds_y, \\
(Ku)(x) &= \int_\Gamma \gamma_1(y) U^*(x,y)\, u(y)\, ds_y, \\
(K't)(x) &= \int_\Gamma \gamma_1(x) U^*(x,y)\, t(y)\, ds_y, \\
(Du)(x) &= -\gamma_1(x) \int_\Gamma \gamma_1(y) U^*(x,y)\, u(y)\, ds_y
\end{aligned}$$

for $x \in \Gamma$. Here we suppose, that a fundamental solution $U^*(x,y)$ of the underlying differential operator is given, and $\gamma_1(\cdot)$ denotes the corresponding boundary stress operator. In the case of the Laplacian, we have

$$U^*(x,y) = -\frac{1}{2\pi} \log|x-y|$$

and

$$\gamma_1 u(x) = \frac{\partial u(x)}{\partial n(x)},$$

when $n(x)$ is the normalized outer normal vector for $x \in \Gamma$. For the system of differential equations of linear elasticity the fundamental solution is given by the Kelvin solution

$$U^*_{k\ell} = \frac{\lambda + 3\mu}{4\pi\mu(\lambda+2\mu)} \left[-\delta_{k\ell} \log|x-y| + \frac{\lambda+\mu}{\lambda+3\mu} \frac{(x_k - y_k)(x_\ell - y_\ell)}{|x-y|^2} \right]$$

for $k, \ell = 1, 2$ and the boundary stress operator can be described by

$$\gamma_1 u(x) = \sigma_{ij}(u) n_j(x)$$

using the stress tensor σ_{ij}.
Via the discretization of the boundary Γ we define both, the trial and test functions required for the Galerkin method. Let the spaces of these functions be defined by

$$V_h = \operatorname{span}\{\varphi_k^\nu\} \subset H^{-1/2}(\Gamma), \quad W_h = \operatorname{span}\{\varphi_k^\mu\} \subset H^{1/2}(\Gamma)$$

consisting of smoothest B–splines $\varphi_k^\mu(\cdot)$ of piecewise polynomial degree μ. Here we choose $\mu = 1$ in general, and $\nu = 0$ or $\nu = 1$. For $\nu = 1$ we may define the piecewise linear trial functions only with respect to the edges, especailly, they may be discontinuous in the corners. Let N_C be the number of corners, then we define the basis of V_h by

$$V_h = \operatorname{span}\left\{\varphi_k^1(\cdot)\right\}_{k=1}^{N} \cup \left\{\varphi_{k_\ell}^{1-}(\cdot)\right\}_{\ell=1}^{N_C},$$

where $\varphi_{k_\ell}^{1-}(\cdot)$ denotes the linear decreasing form function on the element next to the corner with the index k_ℓ.
In the Galerkin approximation procedure we have to compute the Galerkin weights

$$\begin{aligned}
V_h[\ell,k] &= \langle V\varphi_k^\nu, \varphi_\ell^\nu\rangle_{L^2(\Gamma)},\\
K_h[\ell,i] &= \langle K\varphi_i^\mu, \varphi_\ell^\nu\rangle_{L^2(\Gamma)},\\
M_h[\ell,i] &= \langle \varphi_i^\mu, \varphi_\ell^\nu\rangle_{L^2(\Gamma)},\\
D_h[j,i] &= \langle D\varphi_i^\mu, \varphi_j^\mu\rangle_{L^2(\Gamma)}.
\end{aligned}$$

Note, that in the case of the Laplacian we have the relation [5]

$$\langle Du, v\rangle_{L^2(\Gamma)} = \int_\Gamma\int_\Gamma U^*(x,y)\,\frac{du(x)}{ds_x}\,\frac{dv(y)}{ds_y}\,ds_x ds_y$$

and analogously in linear elasticity [46]

$$\langle Du, v\rangle_{L^2(\Gamma)} = -\sum_{k,\ell=1}^{2}\int_\Gamma\int_\Gamma G_{k\ell}(x,y)\,\frac{du(x)}{ds_x}\,\frac{dv(y)}{ds_y}\,ds_x ds_y$$

with

$$G_{k\ell} = \frac{\mu(\lambda+\mu)}{\pi(\lambda+2\mu)}\left[\delta_{k\ell}\log|x-y| - \frac{(x_k-y_k)(x_\ell-y_\ell)}{|x-y|^2}\right].$$

Hence we do not have to discretize the hypersingular integral operator D directly, the computation can be replaced by the discretization of the single layer potential with trial functions of degree less than one.
For the Galerkin discretization of the boundary integral operators, we follow here [51] for a full analytic discretization scheme, if the boundary Γ is given as a polygon.

9.2 Geometric preliminaries

We suppose that the boundary Γ is given by N points $\{x_k\}_{k=1}^N$ in mathematically positive sense, with $x_{N+1} = x_1$ the boundary element Γ_k may be described by

$$\Gamma_k = \{x = x_k + t\cdot r_k, t \in [0,h_k]\}$$

with

$$h_k = |x_{k+1} - x_k|, \quad r_k = \frac{1}{h_k}(x_{k+1} - x_k).$$

With respect to the parametrization of Γ_k we define the piecewise constant trial function as

$$\varphi_k^0(x(t)) = \begin{cases} 1 & \text{for } x \in \Gamma_k, \\ 0 & \text{elsewhere,} \end{cases}$$

and the piecewise linear form functions

$$\varphi_k^{1-}(x(t)) = \begin{cases} \frac{h_k - t}{h_k} & \text{for } x \in \Gamma_k \\ 0 & \text{elsewhere} \end{cases}$$

and

$$\varphi_k^{1+}(x(t)) = \begin{cases} \frac{t}{h_k} & \text{for } x \in \Gamma_k \\ 0 & \text{elsewhere.} \end{cases}$$

Then we have

$$\varphi_k^1(x) = \varphi_{k-1}^{1+}(x) + \varphi_k^{1-}(x)\,.$$

For the analytic integration we distinguish three cases, where we introduce local coordinate systems. For the local representations of two boundary elements Γ_k and Γ_ℓ we consider the system of linear equations

$$x_k + t \cdot r_k = x_\ell + \tau \cdot r_\ell\,.$$

If this system is uniquely solvable, we will denote it as general case, where the origin of the local coordinate system will be the common point of the two lines described by the elements, in any other case the origin will be x_k. If there exists no solution, the boundary elements are parallel with a positive distance. In the last case, there are infinite solutions, since Γ_k and Γ_ℓ are in one line, which includes the singular case $\Gamma_k = \Gamma_\ell$. Depending on the situation one can integrate all Galerkin weights with respect to the local coordinates, these procedures are described in detail in [51].

9.3 The subroutine MATRIX2D

This subroutine generates for a polygonal boundary given by N nodes the Galerkin matrices for all boundary integral operators of the direct approach for the differential operators of the Laplacian and for problems in linear elasticity. For the trial functions for the potential/displacements one has to use piecewise linear functions, while for the flux/stresses one may use piecewise constant or piecewise linear trial functions.

CALL MATRIX2D(N,X,KE,KD,KH,KM,KPH,NU,MU,D,P,MAT1,MAT2,
ZKE,ZKD,ZKH,ZKM,ZKP,ZV,ZVT,ZK,ZKT,ZM,ZMT,ZD,NCL,CIND)

N	(Integer)	$\longrightarrow$	number of elements/nodes on the boundary
X(*)	(REAL*8)	$\longrightarrow$	array of boundary nodes
			x_k^1 =X[2*k-1], x_k^2 =X[2*k], k=1,...,N+1
KE(*)	(REAL*8)	$\longleftarrow$	Galerkin matrix for the single layer potential

KD(*)	(REAL*8)	$\longleftarrow$	Galerkin matrix for the double layer potental
KH(*)	(REAL*8)	$\longleftarrow$	Galerkin matrix of the hypersingular integral operator
KM(*)	(REAL*8)	$\longleftarrow$	Mass matrix
KPH(*)	(REAL*8)	$\longleftarrow$	array of element length's
NU	(INTEGER)	$\longrightarrow$	degree of trial functions for the flux/stresses (NU=0,1)
MU	(INTEGER)	$\longrightarrow$	degree of trial functions for the potential/displacements (MU=1)
D	(INTEGER)	$\longrightarrow$	D=2 for Galerkin discretization
P	(INTEGER)	$\longrightarrow$	P=1 for Laplace equation, P=2 for linear elasticity
MAT1	(REAL*8)	$\longrightarrow$	material parameter, for P=2 MAT1 = λ
MAT2	(REAL*8)	$\longrightarrow$	for P=2 material parameter μ
ZKE	(INTEGER)	$\longrightarrow$	if ZKE=1 KE will be computed
ZKD	(INTEGER)	$\longrightarrow$	if ZKD=1 KD will be computed
ZKH	(INTEGER)	$\longrightarrow$	if ZKH=1 KH will be computed
ZKM	(INTEGER)	$\longrightarrow$	if ZKM=1 KM will be computed
ZKP	(INTEGER)	$\longrightarrow$	if ZKP=1 KPH will be computed
ZV	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with V_h
ZVT	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with $V_h^\top$
ZK	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with K_h
ZKT	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with $K_h^\top$
ZM	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with M_h
ZMT	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with $M_h^\top$
ZD	(INTEGER)	$\longleftarrow$	pointer for matrix multiplication with D_h
NCL	(INTEGER)	$\longrightarrow$	number of local coarse grid nodes (corners)
CIND(*)	(INTEGER)	$\longrightarrow$	array of corner indicees

Note, that the Galerkin matrices of the single layer potential and of the hypersingular integral operator are symmetric, while the Galerkin matrix of the double layer potential is nonsymmetric. The mass matrix is banded and sparse, so it is sufficient to store only two diagonals in general.

9.4 Matrix representation

9.4.1 Potential problem

Single layer potential

- Galerkin
  - conforming trial functions ($\nu = 0, 1$)
    $K_E \in I\!R^{N\times N}$ symmetric
    $\Rightarrow$ row oriented storage

$$K_E \leftrightarrow \begin{pmatrix} 1 & \dots & \dots & N \\ & N+1 & \dots & 2N-1 \\ & & \dots & \vdots \\ & & & N(N+1)/2 \end{pmatrix}$$

  - nonconforming trial functions $\nu = 1\pm$
    $K_E \in I\!R^{(N+N_C)\times(N+N_C)}$ symmetric
    $\Rightarrow$ row oriented block storage
    $N1 = N(N+1)/2,$

$N2 = N1 + NN_C,$
$N3 = N2 + N_C(N_C+1)/2$

$$K_E \leftrightarrow \left(\begin{array}{cccc|cccc}
1 & \dots & \dots & N & N1+1 & & \dots & N1+N_C \\
 & N+1 & \dots & 2N-1 & \vdots & & & \vdots \\
 & & \dots & \vdots & \vdots & & & \vdots \\
 & & & N1 & N1+(N-1)N_C+1 & & \dots & N2 \\ \hline
 & & & & N2+1 & \dots & \dots & N2+N_C \\
 & & & & & \dots & \dots & \\
 & & & & & & \dots & \vdots \\
 & & & & & & & N3
\end{array}\right)$$

Double layer potential

- Galerkin ($\mu = 1$)
  - conforming trial functions $\nu = 0, 1$)
    $K_D \in I\!R^{N\times N}$ nonsymmetric
    $\Rightarrow$ row oriented storage

$$K_D \leftrightarrow \begin{pmatrix}
1 & \dots & \dots & N \\
N+1 & \dots & \dots & 2N \\
\vdots & & & \vdots \\
(N-1)N+1 & \dots & \dots & N^2
\end{pmatrix}$$

  - nonconforming linear test functions ($\nu = 1\pm$)
    $K_D \in I\!R^{(N+N_C)\times N}$ nonsymmetric
    $\Rightarrow$ nonsymmetric block storage

$$K_D \leftrightarrow \left(\begin{array}{cccc}
1 & \dots & \dots & N \\
N+1 & \dots & \dots & 2N \\
\vdots & & & \vdots \\
(N-1)N+1 & \dots & \dots & N^2 \\ \hline
N^2+1 & & \dots & N^2+N \\
\vdots & & & \vdots \\
N^2+(N_C-1)N+1 & & \dots & N^2+NN_C
\end{array}\right)$$

Hypersingular operator

- Galerkin
  - conforming trial functions ($\mu = 1$)
    $K_H \in I\!R^{N\times N}$ symmetric
    $\Rightarrow$ row oriented storage

$$K_H \leftrightarrow \begin{pmatrix}
1 & \dots & \dots & N \\
 & N+1 & \dots & 2N-1 \\
 & & \dots & \vdots \\
 & & & N(N+1)/2
\end{pmatrix}$$

Mass matrix

- Galerkin ($\mu = 1$)

  - $\nu = 0$
    $\Rightarrow$ nonsymmetric matrix

$$K_M \leftrightarrow \begin{pmatrix} 1 & N+1 & & \\ & 2 & N+2 & \\ & & \vdots & \\ 2N & & & N \end{pmatrix}$$

  - $\nu = 1, \mu = 1$
    $\Rightarrow$ symmetric matrix

$$K_M \leftrightarrow \begin{pmatrix} 1 & N+1 & & \\ & 2 & N+2 & \\ & & \vdots & \\ 2N & & & N \end{pmatrix}$$

9.4.2 Linear elasticity

Single layer potential

- Galerkin

  - conforming trial functions ($\nu = 0, 1$)
    $K_E \in I\!R^{2N \times 2N}$ symmetric
    $\Rightarrow$ row oriented storage

$$K_E \leftrightarrow \begin{pmatrix} 1 & \dots & \dots & 2N \\ & 2N+1 & \dots & 4N-1 \\ & & \dots & \vdots \\ & & & 2N(2N+1)/2 \end{pmatrix}$$

Double layer potential

- Galerkin ($\mu = 1$)

  - conforming test functions ($\nu = 0, 1$)
    $K_D \in I\!R^{2N \times 2N}$ nonsymmetric block matrix
    $\Rightarrow$ row oriented storage

$$K_D \leftrightarrow \begin{pmatrix} K_D^1 & K_D^3 \\ K_D^4 & K_D^2 \end{pmatrix}$$

with the starting indeces

$$K_D^i[1] \;=\; (i-1) \cdot N^2 + 1 \qquad (i = 1, \dots, 4)$$

of the nonsymmertic blocks K_D^i.

Hypersingular operator

- Galerkin
  - conforming trial function ($\mu = 1$)
    $K_H \in I\!R^{2N\times 2N}$ symmetric matrix
    $\Rightarrow$ row oriented storage

$$K_H \leftrightarrow \begin{pmatrix} 1 & \dots & \dots & 2N \\ & 2N+1 & \dots & 4N-1 \\ & & \dots & \vdots \\ & & & 2N(2N+1)/2 \end{pmatrix}$$

Mass matrix

- Galerkin ($\mu = 1$)
  storage of the matrices as in the scalar case (potential problem):

$$K_M^E = \begin{pmatrix} K_M^P & 0 \\ 0 & K_M^P \end{pmatrix}$$

  - $\nu = 0$
    $\Rightarrow$ nonsymmetric block–diagonal matrix
  - $\nu = \mu = 1$
    $\Rightarrow$ symmetric block–diagonal–matrix

9.5 Postprocessing

To compute the solution of the given boundary value problem inside the domain one can use the representation formula

$$u(x) = \int\limits_\Gamma U^*(x,y)\, t(y)\, ds_y - \int\limits_\Gamma \gamma_1(y) U^*(x,y)\, u(y)\, ds_y\ .$$

Inserting the approximations t_h and u_h of the given or previously computed Cauchy data, we get for $x \in \Omega$

$$u_h(x) = \sum_{k=1}^{N} t_k \int\limits_\Gamma U^*(x,y)\, \varphi_k^\nu(y)\, ds_x - \sum_{k=1}^{M} u_k \int\limits_\Gamma \gamma_1(y) U^*(x,y)\, \varphi_k^\mu(x)\, ds_x,$$

i.e. one has to compute the collocation weights of the single and double layer potentials, respectively. These routines are included in the software package **ostbem2d**.

CALL INNER_POINT(N,NN,NU,MU,P,X1,X2,DD,ND,X,CIND,U)

N	(INTEGER)	$\longrightarrow$	number of elements/nodes on the boundary
NN	(INTEGER)	$\longrightarrow$	number of corners, if nonconform linera trials are used
NU	(INTEGER)	$\longrightarrow$	degree of trial functions for t
MU	(INTEGER)	$\longrightarrow$	degree of trial functions for u
P	(INTEGER)	$\longrightarrow$	P=1 for Laplace equation, P=2 for linear elasticity
X1,X2	(REAL*8)	$\longrightarrow$	cartesian coordinates of the inner point
DD(*)	(REAL*8)	$\longrightarrow$	coefficent vector for t
ND(*)	(REAL*8)	$\longrightarrow$	coefficient vector for u
X(*)	(REAL*8)	$\longrightarrow$	array of boundary nodes
CIND(*)	(INTEGER)	$\longrightarrow$	array of corner indicees
U(*)	(REAL*8)	$\longleftarrow$	solution in (X1,X2)

9.6 Overview of the library

9.6.1 The tree-structure

OstBEM2d

$\hookrightarrow$ MATRIX2D

- $\hookrightarrow$ LOCCOORDGAL2D
- $\hookrightarrow$ SINGLEPOTGAL2D
- $\hookrightarrow$ DOUBLEPOTGAL2D
- $\hookrightarrow$ SINGLEELASTGAL2D
- $\hookrightarrow$ DOUBLEELASTGAL2D

$\hookrightarrow$ INNERPOINT

- $\hookrightarrow$ LOCCOORDCOL2D
- $\hookrightarrow$ SINGLEPOTCOL2D
- $\hookrightarrow$ DOUBLEPOTCOL2D
- $\hookrightarrow$ SINGLEELASTCOL2D
- $\hookrightarrow$ DOUBLEELASTCOL2D

Note, that the subroutine Matrix2D includes the subroutines to compute all collocation weights too.

9.6.2 The routines of OstBEM2d

Matrix2D	Computation of the BE-matrices $K_\Lambda, K_{C\Lambda}$ and K_C and of the mass matrix M_h for the Laplacian and for elasticity problems (fully analytically with $\nu = 0,1; \mu = 1$).
LocCoordGal2D	Destination of the local coordinate system described by two boundary elements in the Galerkin case and computation of the local parameters.
SinglePotGal2D	Computation of all Galerkin weights for all form functions on two boundary elements for the single layer potential in the case of the Laplacian.
DoublePotGal2D	Computation of all Galerkin weights for all form functions on two boundary elements for the double layer potential in the case of the Laplacian.
SingleElastGal2D	Computation of all Galerkin weights for all form functions on two boundary elements for the single layer potential in the case of linear elasticity.
DoubleElastGal2D	Computation of all Galerkin weights for all form functions on two boundary elements for the double layer potential in the case of linear elasticity.
InnerPoint	Computes an approximate value of the solution in an inner point of the given boundary value problem by means of the representation formula, if all Cauchy data on the boundary are given. This routine is based on the computation of corresponding collocation weights.
LocCoordCol2D	Destination of the paramter of a given collocation node with respect to a specified boundary element.
SinglePotCol2D	Computation of all collocation weights for all form functions on the boundary element for the single layer potential in the case of the Laplacian.
DoublePotCol2D	Computation of all collocation weights for all form functions on the boundary element for the double layer potential in the case of the Laplacian.
SingleElastCol2D	Computation of all collocation weights for all form functions on the boundary element for the single layer potential in the case of linear elasticity.
DoubleElastCol2D	Computation of all collocation weights for all form functions on the boundary element for the double layer potentail in the case of linear elasticity.

Chapter 10

The Libraries CoBEM and GeBEM

10.1 Introduction

Boundary and finite element methods are widely used for solving partial differential equations numerically. Both methods do have their advantages and weaknesses. So nonlinear, inhomogeneous problems can be solved using finite element methods whereas exterior problems can only be formulated properly by using integral equations. Because we want to meet the advantages of both methods we are going to couple the two approaches via domain decomposition techniques. An algorithm for solving coupled FE/BE-equations has been presented in [44].
The libraries *GeBEM* and *CoBEM* contain the routines which are required for the implementation of the boundary element method into the parallel FEM-code. The following tasks are included:

- discretization of the boundary
- mapping between FE/BE-data formats
- realization of matrix-by-vector operations
- realization of the preconditioner C_Λ for the single layer potential
- postprocessing
- realization of the Schur-complement preconditioner mgD.

The routines for generating the discrete operators (BE–matrices) are containes in the library OstBEM2d, see chapter 9.
The most important routines will be described in detail in this documentation. The arrows ' $\longrightarrow$ ' and ' $\longleftarrow$ ' occurring in the description mark the input data and output data, respectively.
In order to start any library function the geometrical data contained in the common blocks geomloc.inc and geomglo.inc as well as the corresponding data on the working array must have been defined already and one has to ensure that the edges are numbered anti-clockwise along the boundary according to their geometrical position. The routines allow an implementation into a nested multilevel algorithm as it is applied to nonlinear problems.
The remainder of this chapter is organized as follows. In section 10.2 we describe the principle of storage management being used and the most important auxiliary routines. The latter involve the discretization of the boundary and the conversion of data formats. Section 10.3 contains practical aspects concerning the boundary element method. That is, the continuous

operators and the routines generating and manipulating the discrete operators (matrices) are explained. Finally, an overview of the library is given in section 10.5. Theoretical aspects concerning the BEM can be found in chapter 9.

10.2 Auxiliary routines

10.2.1 Storage management

Within the program FEM⊗BEM a dynamic storage allocation technique is being used. For this purpose a huge working array B has to be supplied. Within the subroutines, no major storage allocation takes place. Instead, all vectors, matrices, auxiliary arrays etc. are written temporarily or permanently onto that working array B. The global variable IBE contains the lowest free address of B at the present state of the execution. All permanent data contained in B are marked by pointers that are contained in the Integer*4 array IZEIG(.,.). The following addresses are related to BEM data.

IZEIG(1,j)	number of elements on the boundary,
IZEIG(16,j)	address of the array XX (coordinates of the nodes on the boundary),
IZEIG(43,j)	address of u (vector of the solution),
IZEIG(44,j)	address of F, vector of the right hand side,
IZEIG(51,j)	address of the matrix matrix K_Λ,
IZEIG(52,j)	address of the matrix $K_{\Lambda C}$,
IZEIG(53,j)	address of the matrix K_C,
IZEIG(54,j)	address of the Cholesky-decomposition of the mass matrix,
IZEIG(55,j)	address of the mass matrix M,
IZEIG(56,j)	address of the preconditioning matrix C_Λ,
IZEIG(57,j)	address of the pointer IDM,
IZEIG(58,j)	address of the array PZUG,
IZEIG(59,j)	address of the array DZUG,
IZEIG(60,j)	address of the array LINT.

The 2nd component of IZEIG is the number of the level so that multilevel algorithms can be used. Additionally two control parameters can be defined in the .nc file (chapter 3):

INZEI(70): Type of the b.e. trial functions (3-bitwise)
- bit 1–3 (for tractions):
  - 0 : piecewise constant tractions
  - 1 : piecewise linear tractions
- bit 4–6 (for displacement):
  - 0 : piecewise constant displacement
  - 1 : piecewise linear displacement, conforming
  - 2 : piecewise constant displacement, nonconforming.

INZEI(86): Type of the b.e. local preconditioning method.

0 : Integral operator (linear) or multigrid (constant tractions)
1 : scaled circle circulant
2 : integral operator (linear tractions)
3 : multigrid (constant tractions)
4 : direct inversion (for test purposes only)
5 : linear Toeplitz
6 : algebraic Toeplitz
13 : linear circulant.

Additionally we use the address

INZEI(113) address of OrEdg,

where *OrEdg* itself is described below.

10.2.2 Data formats

Both, the discretization and the ordering of the elements on the boundary do have an influence on the properties of the boundary integral operators. That is why we assume that the elements on the boundary are numbered such that the boundary is passed through anti-clockwise. That is, geometrically neighboured elements obtain subsequent numbers. This format is called *BE-format* and is being used within the routines of the BEM libraries. On the other hand the *FE-format,* where the numbering starts with the crosspoints followed by the pure coupling points and finally the inner points, is being used within all FEM-routines.
In order to allow data to be transferred between different processes the format of the data being exchanged must be the same. Thus, we require routines for converting data concerning the boundary from one format into the other. These routines are named *F2BLev* and *B2FLev* and can be called as follows.

Call F2BLev(DIM,BEVEC,FEVEC,OrEdg,IE,NUMEDGES,NDF)

DIM	(Integer)	$\longrightarrow$	number of elements on the boundary,
FEVEC(*)	(Real*8)	$\longrightarrow$	vector in FE-format,
BEVEC(*)	(Real*8)	$\longleftarrow$	vector in BE-format,
OrEdg (*)	(Integer)	$\longrightarrow$	array describing the orientation of the edges,
IE(5,*)	(Integer)	$\longrightarrow$	array describing the edges,
NUMEDGES	(Integer)	$\longrightarrow$	number of edges,
NDF	(Integer)	$\longrightarrow$	number of degrees of freedom per node.

F2BLev converts data given in FE-format into data in BE-format. The inverse problem, the conversion of the BE-format into the FE-format, can be realized by the routine *B2FLev.*

Call B2FLev(DIM,FEVEC,BEVEC,OrEdg,IE,NUMEDGES,NDF)

DIM	(Integer)	$\longrightarrow$	number of elements on the boundary,
FEVEC(*)	(Real*8)	$\longleftarrow$	vector in FE-format,

BEVEC(*)	(Real*8)	$\longrightarrow$	vector in BE-format,
OrEdg (*)	(Integer)	$\longrightarrow$	array describing the orientation of the edges,
IE(5,*)	(Integer)	$\longrightarrow$	array describing the edges,
NUMEDGES	(Integer)	$\longrightarrow$	number of edges,
NDF	(Integer)	$\longrightarrow$	number of degrees of freedom per node.

Note, in the FE-format the numbering along some edge starts at that cross-point which has the lower global number. That is why the array $OrEdg$ has been introduced. It is defined as follows:

$$OrEdg\,(i) = \begin{cases} +1, & \text{if the orientation of the ith edge is such that it fits into} \\ & \text{the sequence of edges giving a closed boundary,} \\ -1, & \text{if the ith edge has to be re-oriented to fit into the sequence.} \end{cases}$$

10.2.3 Discretization

In opposite to finite element methods the boundary element method requires the discretization of the boundary only. The coarsest mesh along the boundary is defined uniquely by the data given in the input file (see section 3.1.2). Every subsequent mesh is obtained by halving the elements on the edges. This procedure guarantees the compatibility between the subdomains. We assume that the edges, which define the subdomains, and all corresponding data are being ordered such that the boundary of the subdomain is passed through anti-clockwise. By calling the routine *DISBOUND* the boundary of a subdomain can be discretized.

Call Disbound(Level,Radius,Kreis,BFeld,Bmax)

Level	(Integer)	$\longrightarrow$	number of the grid to be generated (coarsest grid = 1),
Radius	(Real*8)	$\longleftarrow$	Ratio of the domain if it is a circle,
Kreis	(Logical)	$\longleftarrow$	true, iff the domain is a Bcircle as defined in section 10.1,
BFeld(*)	(Real*8)	$\longrightarrow$	working and auxiliary array,
Bmax	(Integer)	$\longrightarrow$	available length of the array BFeld.

The data shown below will be generated. They are written onto the working array B or they are defined in common blocks.

IBE	(Integer)	$\longleftarrow$	lowest free address of $BFeld$ (mgfem.inc),
IE(5,*)	(Integer)	$\longleftarrow$	array describing the edges (geomloc.inc),
XX(2,*)	(Real*8)	$\longleftarrow$	array of coordinates in BE-format (BFeld),
DZUG(*)	(Integer)	$\longleftarrow$	describes the discretization, see below (BFeld),

The array *DZUG* is of dimension NV. It contains the numbers of the vertices ordered anti-clockwise along the boundary:

DZUG(i) = number of $i-th$ vertex in the local anti-clockwise BE–numbering, $i = 1, \ldots, NV$.

The routine *DisEdge* is the basis for discretizing the boundary. Here, a straight line given by its starting and end point is being discretized.

**Call DisEdge(x1,y1,x2,y2,x3,y3,Type,
Kind,Nint,r1,r2,X,HFeld,Hmax)**

x1,y1	(Real*8)	$\longrightarrow$	starting point of the edge,
x2,y2	(Real*8)	$\longrightarrow$	end point of the edge,
x3,y3	(Real*8)	$\longrightarrow$	coordinates of the midpoint of the edge being required in order to define an arc of a circle uniquely,
Type	(Integer)	$\longrightarrow$	kind of the edge (straight line, arc of a circle),
Kind	(Integer)	$\longrightarrow$	kind of discretization of the edge (see section 3.1),
Nint	(Integer)	$\longrightarrow$	number of elements to be generated on the edge,
r1	(Real*8)	$\longrightarrow$	ratio between longest and shortest element,
r2	(Real*8)	$\longrightarrow$	additional ratio, not being used,
X(2,*)	(Real*8)	$\longleftarrow$	coordinates of the inner nodes on the edge,
HFeld(*)	(Real*8)	$\longrightarrow$	auxiliary array,
Hmax	(Integer)	$\longrightarrow$	available length of HFeld.

10.2.4 The multilevel case

In nonlinear applications a nested iteration is desired, that is the grid and the matrices have to be computed for an arbitrary level between two limits MinLev and MaxLev. Additionally the interpolation u_h on the finer grid of the coarse grid solution u_H is used as a starting guess.
The routine *LEVUP* performs the setup which is necessary for this process.

Call LevUP(MinLev,MaxEW,BETime,IBE,LAEB,IZEIG,INZEI,BFeld)

MinLev	(Integer)	$\longrightarrow$	number of the coarsest grid,
MaxEW	(Integer)	$\longrightarrow$	maximum dimension for circular domains,
BETime	(Real*4)	$\longleftarrow$	time required for the generation of the BE-matrices,
IBE	(Integer)	$\longleftarrow$	first free address in BFeld,
LAEB	(Integer)	$\longrightarrow$	available length of BFeld,.
IZeig(150,*)	(Integer)	$\longleftarrow$	pointer in BFeld,
INZEI(*)	(Integer)	$\longleftarrow$	parameter and pointer in BFeld,
BFeld(*)	(Real*8)	$\longleftarrow$	working array,

10.3 BEM-related routines

For the basic underlying equations related to the boundary element method we refer to section 9.1. In this section we present the basic BEM routines. For this purpose we introduce the word Bcircle. A domain is called Bcircle if it is a circle discretized uniformly into 2^l elements.

10.3.1 Generating matrices

If the domain is a Bcircle we obtain circulant matrices which are represented internally by the vector of their eigenvalues. The main work in computing these eigenvalues is the computation of infinite sums which are independent of the geometry. That is why these sums can be computed a-priori. The file *sums.dat* contains the values which have to be read once at the beginning of the program in the case that there is at least one Bcircle. Additionally this idea is welled suited for multilevel algorithms. In this case only the sums for the finest grid have to be known since the coarse grid values are a subset of the fine grid values.

Thus, if there is at least one Bcircle or if a circulant preconditioner is required the following routine has to be called.

Call DefEW(Level,Dim,Kreis,VK,BFeld,Bmax).

Level	(Integer)	$\longrightarrow$	number of the present level,
Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
Kreis	(Logical)	$\longrightarrow$	true, iff the domain is a Bcircle,
VK	(Integer)	$\longrightarrow$	kind of preconditioning (=1 for pc with circulant),
BFeld(*)	(Real*8)	$\longleftarrow$	working array,
Bmax	(Integer)	$\longleftarrow$	available length of BFeld (updated),

DefEW reads or computes the values of the indefinite sums for the finest grid, allocates memory on the working array and stores the values. Pre-defined eigenvalues are stored in the files *sum1024.dat* and *sums.dat.*
In general the matrices are fully populated, where K_Λ and K_C are symmetric and therefore, in opposite to $K_{C\Lambda}$, only the upper triangle has to be computed and stored. The mass matrix is being stored as a vector of the length $(3 * Dim - 3)$, where the following elements are generated:

$$m_{11}, \ldots, m_{1Dim}, m_{22}, m_{23}, \ldots, m_{ii}, m_{ii+1}, \ldots, m_{Dim-1Dim-1}, m_{Dim-1Dim}, m_{DimDim}.$$

The routine *IniGlsB* allocates memory for the matrices and generates the matrices, the right hand side, the preconditioner and implements the boundary conditions.

**Call IniGlsB(Radius,Level,Dim,NDF,Kreis,VK,MaxEW,MaxLev,
DZUG,XX,BFeld,Bmax,BETime)**

Radius	(Real*8)	$\longrightarrow$	Ratio, if domain is a Bcircle,
Level	(Integer)	$\longrightarrow$	number of the level,
Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
NDF	(Integer)	$\longrightarrow$	number of degrees of freedom per node,
Kreis	(Logical)	$\longrightarrow$	true, iff domain is a Bcircle,
VK	(Integer)	$\longrightarrow$	kind of preconditioning (=1 for pc with circulant),
MaxEW	(Integer)	$\longrightarrow$	number of elements on the finest grid,
MaxLev	(Integer)	$\longrightarrow$	number of the finest grid,
DZUG(*)	(Integer)	$\longrightarrow$	describes the discretization,
XX(2,*)	(Real*8)	$\longrightarrow$	coordinates of the nodes on the boundary (BE-format),
BFeld(*)	(Real*8)	$\longleftarrow$	working array,
Bmax	(Integer)	$\longleftarrow$	available length of BFeld (updated),
BETime	(Real*4)	$\longleftarrow$	time required for generating the BE-matrices.

Additionally the vector of the right hand side is defined to be zero and it is manipulated according to the boundary conditions.
The matrices are now computed fully analytically. The trial functions are chosen depending on the value INZEI(70) (see above).
The following routine contains only the generation, that is no boundary conditions will be implemented. The memory for the matrices has to be allocated before *GenMa* is called.

**Call GenMa(Radius,RMatle,MaxEW,MaxLev,Kreis,PZUG,DZUG,
XX,KE,KD,KH,KM,IDM,LINT,BFeld,IZEIG,INZEI,HFeld,Hmax)**

Radius	(Real*8)	$\longrightarrow$	Ratio, if domain is a Bcircle,
RMatle(*)	(Real*8)	$\longrightarrow$	material-data,
MaxEW	(Integer)	$\longrightarrow$	number of elements on the finest grid,
MaxLev	(Integer)	$\longrightarrow$	number of the finest grid,
Kreis	(Logical)	$\longrightarrow$	true, iff domain is a Bcircle,
DZUG(*)	(Integer)	$\longrightarrow$	describes the discretization,
XX(2,*)	(Real*8)	$\longrightarrow$	coordinates of the nodes on the boundary (BE-format),
KE(*)	(Real*8)	$\longleftarrow$	matrix K_Λ of the single layer potential,
KD(*)	(Real*8)	$\longleftarrow$	matrix $K_{\Lambda C}$ of the double layer potential,
KH(*)	(Real*8)	$\longleftarrow$	matrix K_C of the hyper-singular integral operator,
KM(*)	(Real*8)	$\longleftarrow$	Galerkin-mass matrix,
IDM(*)	(Integer)	$\longleftarrow$	pointer to the last element of each row of KM,
LINT(*)	(Real*8)	$\longleftarrow$	array of the sizes of the elements on the boundary,
BFeld(*)	(Real*8)	$\longrightarrow$	working array,
IZeig(150,*)	(Integer)	$\longleftarrow$	pointer in BFeld,
INZEI(*)	(Integer)	$\longleftarrow$	parameter and pointer in BFeld,
HFeld(*)	(Real*8)	$\longleftarrow$	auxiliary array,
Hmax	(Integer)	$\longleftarrow$	available length of HFeld.

Genma calls the routines from *OstBEM2d.* Then, the matrices are copied and they are finally stored as upper triangular matrices to allow the implementation of boundary conditions. The latter are implemented by setting rows to zero, except the element on the main diagonal which is set to 1. The right hand side is changed according to these manipulations.
The mass matrix multiplied by 0.5 is added to the matrix of the double layer potential.

10.3.2 Matrix-by-vector operations

A single matrix-times-vector operation can be performed by calling *MatVec.*

Call MatVec(Dim,Mat,Vec,ResV,Zeig,HFeld,Hmax)

Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
Mat(*)	(Real*8)	$\longrightarrow$	matrix or vector of eigenvalues,
Vec(*)	(Real*8)	$\longrightarrow$	input vector,
ResV(*)	(Real*8)	$\longleftarrow$	resulting vector,
Zeig	(Integer)	$\longrightarrow$	pointer to the multiplication routine (0=circulant matrix, 1=symmetric matrix, 2=non-symmetric matrix, 3=transposed of a matrix, 4=mass matrix),
HFeld(*)	(Real*8)	$\longleftarrow$	auxiliary array,
Hmax	(Integer)	$\longleftarrow$	available length of HFeld.

10.3.3 Postprocessing

The routine *PostB* allows to compute the BE–solution (displacement and stresses) at interior (exterior) points which lie on a straight line or a circle.

Call PostB(Dim,XXB,UB, Insid,Loes,Anz,XX,UF,HFeld,Hmax)

Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
XXB(2,*)	(Real*8)	$\longrightarrow$	coordinates of the nodes on the boundary (BE-format),
UB(*)	(Real*8)	$\longrightarrow$	BEM-solution (dimension 2*Dim, components $1,\ldots,Dim$ = values of u, $Dim+1,\ldots,2*Dim$ = values of λ),
Insid	(Logical)	$\longrightarrow$	true, if the interior problem is to be solved (alternatively the exterior problem is solved),
Loes	(Logical)	$\longrightarrow$	true, if values of u are to be computed (alternatively the gradient of u is computed),
Anz	(Integer)	$\longrightarrow$	number of points at which values are to be computed,
XX(2,*)	(Real*8)	$\longrightarrow$	array of the coordinates of those points,
UF(*)	(Real*8)	$\longleftarrow$	array of the solution (if Loes=true) or
UF(2,*)	(Real*8)	$\longleftarrow$	array of the gradients,
HFeld(*)	(Real*8)	$\longleftarrow$	auxiliary array,
Hmax	(Integer)	$\longleftarrow$	available length of HFeld.

10.4 The preconditioner C_Λ

A crucial point in the coupling algorithm is the choice of the preconditioning operator C_Λ which must be spectrally equivalent to K_Λ and its inverse action must be cheap. That is the relation

$$\alpha_1\, C_\Lambda \le K_\Lambda \le \alpha_2\, C_\Lambda \tag{10.1}$$

with the additional condition $\alpha_1 > 1$ has to be fulfilled.
Depending on the value IZEIG(86) the following choices for C_Λ have been implemented:

0 : Integral operator (linear) or multigrid (constant tractions)
1 : scaled circle circulant
2 : integral operator (linear tractions)
3 : multigrid (constant tractions)
4 : direct inversion (for test purposes only)
5 : linear Toeplitz
6 : algebraic Toeplitz
13 : linear circulant.

We refer to [4, 52, 53] for a detailed description of these operators.
The call for applying one of the preconditioners is as follows.

Call PreSing(Dim,NUMEDGES,NDF,Lambda,KE,Cl,KM,IDM,Vec1,Vec2, Vec3,Vec4,ZE,VK,Mat,LINT,IE,HFeld,Hmax)

Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
NUMEDGES	(Integer)	$\longrightarrow$	number of edges,
NDF	(Integer)	$\longrightarrow$	number of degrees of freedom per node,

Lambda(2)	(Real*8)	$\longrightarrow$	min/max eigenvalue of $C_\Lambda^{-1}K_\Lambda$,
KE(*)	(Real*8)	$\longrightarrow$	matrix K_Λ of the single layer potential,
Cl(*)	(Real*8)	$\longrightarrow$	matrix C_Λ,
KM(*)	(Real*8)	$\longrightarrow$	Galerkin-mass matrix,
IDM(*)	(Integer)	$\longrightarrow$	pointer to the last element of each row of KM,
VecI(*)	(Real*8)	$\longrightarrow$	auxiliary vectors, $I = 1, 2, 3, 4$,
ZE	(Integer)	$\longrightarrow$	=1 if domain is a Bcircle, =0 otherwise,
VK	(Integer)	$\longrightarrow$	kind of preconditioning (=1 for pc with circulant),
Mat	(Real*8)	$\longrightarrow$	material dependent coefficient (=Rmatle(1)),
LINT(*)	(Real*8)	$\longleftarrow$	array of the sizes of the elements on the boundary,
IE(5,*)	(Integer)	$\longrightarrow$	array describing the edges,
HFeld(*)	(Real*8)	$\longrightarrow$	working array,
Hmax	(Integer)	$\longrightarrow$	available length of HFeld.

In order to guarantee $\alpha_1 > 1$ in (10.1) the routine *SpekCG,* which scales the operator C_Λ appropriately by guessing the smallest eigenvalue of $C_\Lambda^{-1}K_\Lambda$ [49], has to be called.

Call SpekCG(Dim,NUMEDGES,NDF,Lambda,KE,Cl,KM,IDM,Vec1,Vec2, Vec3,Vec4,ZE,VK,Mat,LINT,IE,HFeld,Hmax)

Dim	(Integer)	$\longrightarrow$	number of elements on the boundary,
NUMEDGES	(Integer)	$\longrightarrow$	number of edges,
NDF	(Integer)	$\longrightarrow$	number of degrees of freedom per node,
Lambda(2)	(Real*8)	$\longrightarrow$	min/max eigenvalue of $C_\Lambda^{-1}K_\Lambda$,
KE(*)	(Real*8)	$\longrightarrow$	matrix K_Λ of the single layer potential,
Cl(*)	(Real*8)	$\longrightarrow$	matrix C_Λ,
KM(*)	(Real*8)	$\longrightarrow$	Galerkin-mass matrix,
IDM(*)	(Integer)	$\longrightarrow$	pointer to the last element of each row of KM,
VecI(*)	(Real*8)	$\longrightarrow$	auxiliary vectors, $I = 1, 2, 3, 4$,
ZE	(Integer)	$\longrightarrow$	=1 if domain is a Bcircle, =0 otherwise,
VK	(Integer)	$\longrightarrow$	kind of preconditioning (=1 for pc with circulant),
Mat	(Real*8)	$\longrightarrow$	material dependent coefficient (=Rmatle(1)),
LINT(*)	(Real*8)	$\longleftarrow$	array of the sizes of the elements on the boundary,
IE(5,*)	(Integer)	$\longrightarrow$	array describing the edges,
HFeld(*)	(Real*8)	$\longrightarrow$	working array,
Hmax	(Integer)	$\longrightarrow$	available length of HFeld.

10.5 Overview of the libraries

10.5.1 The tree-structure

GeBEM & CoBEM

- $\hookrightarrow$ F2BLev
- $\hookrightarrow$ B2FLev
- $\hookrightarrow$ DefEW
    - $\hookrightarrow$ Lesen
    - $\hookrightarrow$ Sum
- $\hookrightarrow$ LevUP

- ↪ DefBCB
- ↪ DIBCP
- ↪ Disbound
- ↪ IniGlsB
- ↪ ProlU
- ↪ BEMalS

↪ Disbound

- ↪ TestCircle
  - ↪ Test2k
- ↪ Disedge
  - ↪ DetMR
  - ↪ Angle
  - ↪ Dis1D
- ↪ Aufweiten
- ↪ InitPDZUG

↪ IniGlsB

- ↪ GenMa
  - ↪ CirMa
    - ↪ RelPos
  - ↪ ZirkEW
  - ↪ Matrix2d (OstBEM2d)
- ↪ CirVK
- ↪ SpekCG
  - ↪ MatVec
  - ↪ PreSing
- ↪ RbdB
  - ↪ LinInt

↪ CGBEM

- ↪ BEMalS
  - ↪ MatVec
    - ↪ ZirVec
    - ↪ FFTm
    - ↪ Mirror
    - ↪ Block
    - ↪ BlockT
    - ↪ Blqs
- ↪ BDCirc
- ↪ CGResB
  - ↪ MatVec

        ↪ PreSing

↪ PostB

    ↪ Dis1C
    ↪ InnerPoint (OstBEM2d)

↪ GenHyp (mgD)

    ↪ IniFEM
        ↪ BEMa20f
    ↪ RestKc
    ↪ SetDirB

↪ MGHyp (libbound.a)

    ↪ Matvec

10.5.2 The routines of GeBEM

Aufweiten	Generate a finer grid by halving the current discretization.
BDCirc	Special routine to be called during the cg-iteration if C_Λ has been eliminated.
CirMa	Read sums for the circulant matrices from the array belonging to the finest grid.
CirVK	Read sums for C_Λ from the array belonging to the finest grid.
DefBCB	Define the array BCB for BEM-domains.
DefEW	Compute or read the sums for the eigenvalues of the circulant matrices for the finest grid.
DetMR	Compute the midpoint and the ratio of a circle given by three points.
DIBCP	Initialize IBCP for BEM-domains.
Dis1D	Discretization of a 1D-interval.
Disbound	Call routines for discretizing the boundary.
Disedge	Discretization of an edge via discretizing an 1d-interval.
GenHyp	Generate the operators which are necessary for the mgD-Schur-complement preconditioner.
GenMa	Start generation of the matrices.
HierIE	Initialze the list of IE on B for all levels.
IniFEM	To be called for FEM-domains during GenHyp. Generates the BE-matrices.
IniGlsB	Initialize matrices and call the generation routines.
InitPDZUG	Initialize BE-data structures describing the boundary.
Lesen	Read sums from the file sums.dat.
LevUP	Initialize matrices and discretization parameters for a fixed level.
LinInt	Linear interpolation of values given at the nodes on the boundary.
PostB	BEM-Postprocessing, computation of u and λ at discrete points.
RbdB	Implementation of boundary conditions into the BE-system.
RelPos	Computation of the absolute position of a value within the file sums.dat.
RestKc	Compute the lower-grid operators from the fine-grid operators.
SetDirB	Manipulate the operators according to given Dirichlet conditions.
Sum	Compute sums for circulant matrices.
Test2k	Test if a number is a power of 2.
TestCircle	Test, if domain is a Bcircle.
ZirkEW	Compute eigenvalues of the circulant matrices for Bcircles.

10.5.3 The routines of CoBEM

Angle	Compute the angle between two lines with a given intersection.
BE2FE	Convert a vector given in BE-format into a vector in FE-format.
BEMalS	Multiplication BE-(block)-matrix times vector.

Block	Multiplication (quadratic) matrix times vector.
BlockT	Multiplication (quadratic) matrix transposed times vector.
BLQS	Multiplication symmetric matrix, given by the upper triangle times vector.
BResC	Perform parts of one cg-step.
CGResB	Compute the residuum within the cg-algorithm. This routine has been designed such that a synchronisation between FE and BE domains is achieved.
Dis1C	Discretization of a 1D-interval (circle or straight line) to be used for the post-processing.
FE2BE	Convert a vector given in FE-format into a vector in BE-format.
MatVec	Start the appropriate matrix times vector routine.
Mirror	Mirror the bits of an integer.
OutNorm	Compute the vector of the outward normal.
PreSing	Perform the operation C_Λ^{-1}.
ProlU	Prolongate a vector to a finer grid.
SpekCG	Estimation of the maximum/minimum eigenvalue of $B^{-1}A$.
ZirVec	Matrix times vector operation for circulant matrices.

Appendix A

The Preprocessor ADDPre

In this chapter, we give a short overview of the preprocessor ADDPre. A complete description and numerical examples can be found in [11].

A.1 Preprocessing

Let Ω be a bounded domain and ν denotes a piecewise constant coefficient function which defines an a-priori decomposition of the domain Ω by

$$\bar{\Omega} = \bigcup_{j=1}^{P_{mat}} \bar{\hat{\Omega}}_j, \quad \text{with} \quad \hat{\Omega}_i \cap \hat{\Omega}_j = \emptyset \quad \forall i \neq j, \quad \text{and} \quad \nu(x) = \nu_j \quad \text{for} \quad x \in \hat{\Omega}_j. \tag{A.1}$$

Having in mind to use a multiprocessor computer to solve DD-equations the most natural idea is to use one processor per subdomain. In our case we assume that the number of processors (p) to be used is (much) larger than the number of a-priori given material-defined domains, i.e., we have $p > P_{mat}$. Thus, we wish to decompose the domains $\hat{\Omega}$ further, which leads finally to

$$\bar{\Omega} = \bigcup_{i \in \mathcal{I}} \bar{\Omega}_i, \quad \text{with} \quad \bar{\hat{\Omega}}_j = \bigcup_{i \in \mathcal{I}_j} \bar{\Omega}_i \quad \forall j = 1, \ldots, P_{mat} \tag{A.2}$$

where the sets of indices are given by

$$\mathcal{I}_j \subset \mathcal{I} := \{1, \ldots, p\}, \quad \bigcup_{j=1}^{P_{mat}} \mathcal{I}_j = \mathcal{I}, \quad \mathcal{I}_j \cap \mathcal{I}_k = \emptyset \quad \forall j \neq k,$$

i.e., the subdomains $\hat{\Omega}_j$ determined by the materials may be decomposed further (see, e.g., [24]). We assume that there exist open balls $B_{\underline{r}_i}$ and $B_{\overline{r}_i}$ $(i \in \mathcal{I})$ with positive radii $\underline{r}_i$ and $\overline{r}_i$, such that

$$B_{\underline{r}_i} \subset \Omega_i \subset B_{\overline{r}_i} \quad \text{and} \quad 0 < \underline{c} \leq \overline{r}_i / \underline{r}_i \leq \overline{c} \quad \forall i \in \mathcal{I} \tag{A.3}$$

with fixed (i-independent) constants $\underline{c}$ and $\overline{c}$.
The task of decomposing Ω into p subdomains starting with a fixed discretization of the interfaces $\Gamma_M := \cup_{j=1}^{P_{mat}} \partial\hat{\Omega}_j$ aiming at a load-balanced decomposition is not trivial. Actually we seek such a decomposition for an a-priori defined discretization of the boundary which minimizes the imbalance between the processors, i.e.,

$$\max_{i=1,\ldots,p} \{\beta_i \; N_i\} \,/ \min_{i=1,\ldots,p} \{\beta_i \; N_i\} \longrightarrow \text{Minimum} \tag{A.4}$$

where N_i $(i = 1, \ldots, p)$ is the number of local elements arising after the decomposition and β_i is a piecewise constant (over $\bar{\hat{\Omega}}$) weight function representing the work per element..
In the following we describe the data formats, the file types, the tools and, finally, the preprocessor ADDPre.

A.2 The Codes

The preprocessor ADDPre uses three main tools which are AdapMesh, Decomp and Tri2DD. Now, we will describe these tools including the main ideas they are based on.

A.2.1 AdapMesh

AdapMesh converts dd-data into tri-data. Indeed, the code applies an improved version of the mesh generator ParMesh [9] to each of the given subdomains in parallel. The resulting global mesh is conforming along the interfaces since the discretization at these interfaces has been predescribed.
Additionally, it is possible to adapt the refinement to singularities, i.e., one has to change the refinement parameter which are given in the input dd-file before starting the mesh generator. For this purpose, a possibly sequential coarse grid calculation could give rise to the right choice of the new refinement parameter.

A.2.2 Decomp

The program Decomp decomposes a domain Ω containing P_{mat} single-material domains $\widehat{\Omega}_k$ into an arbitrary number of subdomains $p \geq P_{mat}$ so that in analogy to (A.2)

$$\overline{\Omega} = \bigcup_{k=1}^{P_{mat}} \overline{\widehat{\Omega}_k} = \bigcup_{k=1}^{P_{mat}} \bigcup_{i=1}^{p_k} \overline{\Omega}_i = \bigcup_{i=1}^{p} \overline{\Omega}_i \qquad , \quad p \equiv \sum_{k=1}^{P_{mat}} p_k$$

is hold, where p_k denotes the number of subdomains in which a single-material domain $\widehat{\Omega}_k$ will be decomposed and $\overline{\widehat{\Omega}_k} = \bigcup\limits_{i=1}^{p_k} \overline{\Omega}_i$ is fulfilled. So each subdomain Ω_i belongs to exactly one material. In the context above the term material is not just restricted to different material constants but is also extended to different type of differential equations describing the local problem or different techniques for solving.
The decomposition of the domain Ω cannot be done by pure geometric information of the domain Ω. For the sake of decomposition we need a triangular element grid and decompose this grid. This grid will be read in by the program as a *.tri file. Each element has to include the number of the material it belongs to.
Let denote the number of elements in the whole domain Ω by N, the number of elements in the single-material domains $\widehat{\Omega}_k$ by $\widehat{N}_k$ and the number of elements in the subdomains Ω_i by N_i we can rewrite the relations between the domains into relations of the numbers of elements

$$N = \sum_{k=1}^{P_{mat}} \widehat{N}_k = \sum_{k=1}^{P_{mat}} \sum_{i=1}^{p_k} N_i = \sum_{i=1}^{p_k} N_i \qquad , \quad p \equiv \sum_{k=1}^{P_{mat}} p_k \ .$$

For explaining the technique used for decomposing the grid we first assume that we have just one material in the whole domain, e.g. $\overline{\Omega} = \overline{\widehat{\Omega}}$ and $P_{mat} = 1$. In the case of a decomposition into $p = 2^s$, $s \in \mathbf{N}$ subdomains the usual recursive spectral bisection method (rsb) is available [50]. For distributing the grid into an arbitrary number of subdomains we adapted an idea of Clemens Brand [2]. The aim of all those decompositions is to distribute the elements in such a way that each subdomain possesses the same (or nearly the same) number of elements ($N_i \approx N/p$).
In the case of $P_{mat} > 1$ we handle each material domain $\widehat{\Omega}_k$ ($k=\overline{1,P_{mat}}$) separately as described above. Now the difficulty is to determine a priori the number of subdomains p_k per material domain. The proper calculation for a distribution into p subdomain is done by the following **algorithm:**

1. Initialization: $\overline{p} := P_{mat}$, $p_k := 1$, $N_k := \widehat{N}_k \qquad \forall k = \overline{1, P_{mat}}$
2. WHILE $(\overline{p} < p)$ DO   Find $j : N_j = \max\limits_{i=1,\dots,\overline{p}} N_i$
   Set $\overline{p} := \overline{p} + 1$, $N_{\overline{p}} := N_j/2$, $N_j := N_j/2$

   DONE

A load balanced distribution should also take into account that different material domains may yield to different numerical efforts. So a weighted distribution of the elements is necessary which fulfills the condition

$$\max_{i=\overline{1,p}}\{\beta_i N_i\} \,/\, \min_{i=\overline{1,p}}\{\beta_i N_i\} \quad \longrightarrow \quad \text{Minimum} \quad .$$

The weights β_i denote the average arithmetical work per element for solving the local problem in a subdomain Ω_i. That means that f.e. a nonlinear behavior of the material constants in that subdomain Ω_i will be reflected by a larger β_i. To include this weighting condition in the algorithm above all N_i have to be declared as REAL and the third initialization statement has to be changed into $N_k := \beta_k \widehat{N}_k$.

As a result each triangle obtains the number of the subdomain it belongs to. This information will be written as a *.tri file.

REMARK A.2.1 The decomposition above is element based. Because of the fact that most of the numerical algorithms solving large systems of equations (resulting from partial differential equations) are node based the decomposition above does not reflect the numerical load balancing correctly but it works sufficiently good. Additionally no load balancing concerning communication is taken into account but one great advantage of the rsb is the minimization of the subdomain boundaries. In some practical problems with a rather strange geometry of the material domains the communication balance is a priori very poor.

A.2.3 Tri2DD

Tri2DD converts tri-data into dd-data. Hereby, interfaces between different materials and the refinement information at these interfaces are fully maintained. On the other hand interfaces within one material are to be smoothed to get "simple" polygons as subdomains. The aim is to obtain smooth interfaces between subdomains belonging to the same material and to minimize the global number of crosspoints without destroying the load balance generated by Decomp.

The **algorithm** can be described as follows.

1. Find, for each subdoamin, the local sequence of boundary edges. Define these edges as 'free'.
2. Find primary crosspoints (cp).
   a) cp defined in the fb–file.
   b) points which are the intersection of at least three subdomains.
3. DO for all subdomains

   WHILE there are 'free' (local) boundary edges

   a) Find a local cp P_1 which is a vertex of a 'free' boundary edge. If there is no such cp then define some vertex P_1 of a 'free' edge as cp.
   b) Start at P_1, go along the 'free' boundary edges UNTIL for two subsequent points P_2, P_3
      i) P_2 is a cp
      ii) $\overline{P_1P_2}$ would be accepted as new geometrical edge, whereas $\overline{P_1P_3}$ would not be accepted as new geometrical edge. Then define P_2 as new cp.
   c) Define $\overline{P_1P_2}$ as new geometrical edge and set the corresponding boundary edges to 'busy'. Define refinement parameter for the new geometrical edge according to the boundary edges.

   END WHILE

   END DO
4. Renumber new crosspoints.

This algorithm is finite since in each iteration at least one 'free' boundary edge is set to 'busy'. An open question is how to decide whether a virtual edge $\overline{P_1P_2}$ is to be accepted as new geometrical edge or not. For this purpose we distinguish between two cases:

C1: The boundary edges describe an interface between two different materials.

C2: The boundary edges describe an interface between two subdomains which belong to the same material.

The following criteria are applied:

C1: If the new edge describes exactly the interface it is allowed.

C2: The new edge must intersect all polygons defined by the boundary points. If this is not the case, or if it intersects a third subdoamin it is not allowed.

The refinement parameter are determined such that they reflect the behaviour of the boundary edges (length and grading) as good as possible.

A.3 The Preprocessor ADDPre

The complete **algorithm** can now be formulated as follows.

0. Create (manually) a dd-description of the problem and state a-priori refinement information on the boundaries (see Fig. A.1a).
1. Use AdapMesh to convert the dd-data into tri-data. (see Fig. A.1b).
2. Use Decomp to decompose the domain, i.e., to assign each triangle to one of the subdomains (see Fig. A.1c).
3. Use Tri2DD to convert the tri-data back into dd-data (see Fig. A.1d).
4. Stop or, if desired, modify the refinement information in the dd-file and goto (1.).

Starting off with any of the data formats the preprocessor has to maintain the original information throughout all stages ending up with a dd-file which describes finally the decomposition of our domain into subdomains according to the a-priori given refinement information.

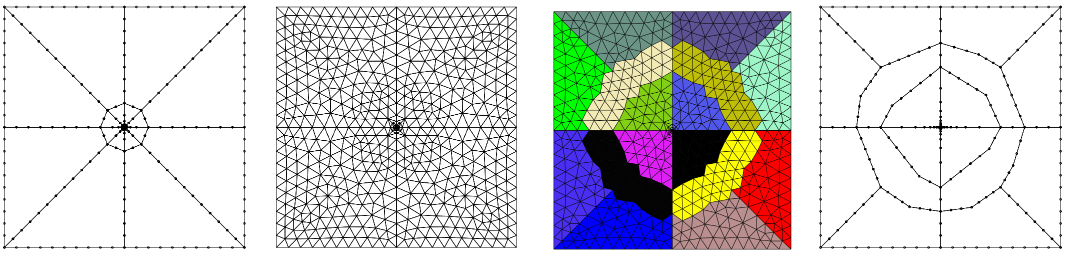

Figure A.1: The 4 stages (a–d from left to right) of preprocessing.

The preprocessor performs a quasi-static decomposition. It is possible to restart the preprocessing optionally in step (4). This may be necessary if the refinement information (grading or density of the mesh) have been modified, e.g., according to the bahaviour of the solution obtained from a coarse grid calculation. The a-posteriori change of the refinement information, i.e., of the discretization, destroys in general the load-balance, that is the present decomposition is no longer a solution to (A.4).

REMARK A.3.1 The preprocessor allows us to start off with any data format and to decompose Ω into any number $p \geq P_{mat}$ of subdomains. The only data which are fixed

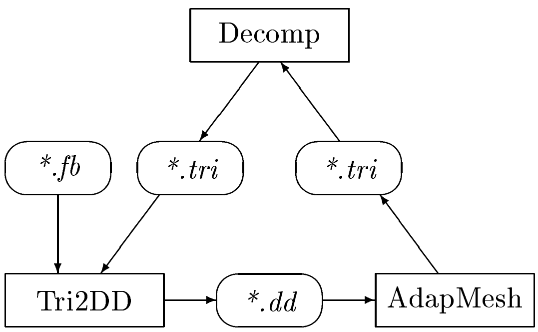

Figure A.2: The preprocessor can be started with any file. The output is the dd–file at the bottom.

throughout the preprocessing are the definition of Γ_M and the a-priori given refinement information on Γ_M !

REMARK A.3.2 Since the rsb requires a full 2d-mesh it is always, i.e. also for DD–BEM, necessary to generate a full 2d mesh ! At a first glance this may seem disadvantageous, especially if one already starts with a dd-description of the problem. On the other hand a slightly improved decomposition may give major improvements if time-dependent or optimization problems involving repeated calls of the linear solver are to be solved.

Appendix B

New experimental components

This section introduces some useful components that have joint the code recently. Because of the lack of time they are not fully checked so that some bugs may occur. Please refer any bugs connected with these new components to *ghaase@numa.uni-linz.ac.at* .

B.1 A more flexible domain handling

The data structure supported by the standard input file and the common blocks described in chapter 3.1 was fixed on the restrictive condition that one subdomain consists of exactly one face. To overcome this bottleneck we extend the common blocks *GeomGlo* and *GeomLoc* (see the files *geomglo.inc* and *geomloc.inc*). Now we are in the position to handle subdomains including more than one face.

Faces in one subdomain		allowed ?
One face.	Ω_i	Yes
All faces are separate.	$\bigcap_{i=1}^{nFaces} \Omega_i = \emptyset$	Yes
Couples of faces are connected via edges.	$\forall i \neq j : \Omega_i \bigcap \Omega_j = \emptyset$ or edge	Yes
At least two faces intersect just in a number of separate points.	$\exists i \, \exists j : \Omega_i \bigcap \Omega_j =$ point	No

Additionally, the faces may posses different material classes so that one subdomain may contain regions of different material properties. In that case the interface line between these regions belongs to the interior of the subdomain.
So, besides the old standard input file (section 3.1.2) a slightly changed standard-II input file can be read. Both files contain the same blocks (with different use of the data) excluding the block MAPPING in the old file and the block MAPPING2 in the new one. These different code words are the sign for the program which kind of file is to be read.

B.2 New facilities of the user net generator

We improved the net generator PARMESH (see [8]) working on each FE-subdomain in the following senses

- All floating point calculation are done in Double Precision.
- Some additional tests were implemented to ensure a regular triangularization in the case of non convex domains.

- Tests were added including the neighborhood of a vertex to find the best vertex of the boundary contour in the sense of an average edge length and in the sense of a good partitioning of the angle. This strategy may result in a splitting of the domain in two closed domains so that a recursive implementation became necessary.
- The net generator handles domains containing several faces preserving the DD-data structures.
- A second net smoothing algorithm was implemented improving the aspect ratio of the triangles. It results in much better triangularizations.

To distinguish from the old version we named the library *libpghd.a* . The second net smoothing algorithm is included in *libutilmg.a(netsmooth.f)* .

B.3 Postprocessing for the elasticity problem

The program uses the 2D-graphical postprocessing routines in *libGraph.a* developed by M. Pester and Th. Hommel [17]. For the sake of visualizing the quantities of the elasticity like the displacement $(u,v)^T$, the strain $(\varepsilon_x, \varepsilon_y, \varepsilon_{xy})^T$ and the stress $(\sigma_x, \sigma_y, \sigma_{xy})^T$ (the principal stress and main stress respectively) we just have to calculate it and adapt the postprocessing. The following formulae are used.

The strain

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ .5\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \end{pmatrix} \tag{B.1}$$

The stress

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{pmatrix} = \begin{pmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & 2\mu \end{pmatrix} \cdot \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{pmatrix} \tag{B.2}$$

The principal stress

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} = \begin{pmatrix} .5(\sigma_x + \sigma_y) + \sqrt{.25(\sigma_x + \sigma_y)^2 + \sigma_{xy}^2} \\ .5(\sigma_x + \sigma_y) - \sqrt{.25(\sigma_x + \sigma_y)^2 + \sigma_{xy}^2} \end{pmatrix} \tag{B.3}$$

The main stress

$$\sigma_m = .5\,(\sigma_1 - \sigma_2) \tag{B.4}$$

B.4 The parallelized global multigrid method

As an alternative to the PCG algorithm with DD preconditioning, a parallelized global multigrid method has been implemented on the basis of the DD data structure. It applies nonstandard Gauss-Seidel smoothers with minimal communication effort as well as a Schur complement coarse grid solver. The method is described in [40, 38]. Therein, first promising results for linear and nonlinear problems are documented, too.

Appendix C

The arrays IZEIG and INZEI

IZEIG(1,j)	NUMNP	number of nodes
IZEIG(2,j)	NUMCRP	number of cross points
IZEIG(3,j)	NUMCP	number of coupling points
IZEIG(4,j)	NUMINP	number of inner nodes
IZEIG(5,j)	NUMED	number of edges
IZEIG(6,j)	NUMBED	number of boundary edges
IZEIG(7,j)	NUMCBP	number of coupling edges
IZEIG(8,j)	NUMEL	number of elements
IZEIG(9,j)	NUMMP	number of intermediate nodes
IZEIG(10,j)	NUMWK	number of nodes with the interpolation/restriction weight $\neq 1/2$ inside the multigrid–algorithm
IZEIG(16,j)	LX	address of the array X
IZEIG(17,j)	LXM	address of the array XM
IZEIG(18,j)	LIBC	address of the array IBC
IZEIG(19,j)	LIECE	address of the array IECE
IZEIG(20,j)	LIED	address of the array IED
IZEIG(21,j)	LICBN	address of the array ICBN
IZEIG(22,j)	LIEDM	address of the array IEDM
IZEIG(23,j)	LICF	address of the array ICF
IZEIG(24,j)	LISONZ	address of the array ISONZ
IZEIG(25,j)	LISONN	address of the array ISONN
IZEIG(26,j)	LWK	address of the array WK
IZEIG(27,j)	LIBCP	address of the array IBCP
IZEIG(33,j)	IASS	storage of the matrix $= 1:$ CRS - storage $= 2:$ VBS - storage $= 3:$ CRS - storage of A, IAL1, IAL2, and additional VBS - storage of AV, IAV

		on the coarsest grid
IZEIG(34,j)	MEX	number elements of A being stored
IZEIG(35,j)	MEXV	number of matrix elements of AV being stored
IZEIG(40,j)	LA	address of the array A
IZEIG(41,j)	LIAL1	address of the array IAL1
IZEIG(42,j)	LIAL2	address of the array IAL2 (only for CRS-storage)
IZEIG(43,j)	LU	address of the array U
IZEIG(44,j)	LF	address of the array F
IZEIG(45,j)	LUU	address of the array UU (nonlinear problems)
IZEIG(46,j)	LAV	address of the array AV for additional VBS-storage
IZEIG(47,j)	LIAVL	address of the array IAVL for additional VBS-storage
IZEIG(50,j)		address of the cg-vector ResKl
IZEIG(51,j)		address matrix K_Λ
IZEIG(52,j)		address of the matrix $K_{\Lambda C}$
IZEIG(53,j)		address of the matrix K_C
IZEIG(54,j)		address of the Cholesky-decomposition of the mass matrix
IZEIG(55,j)		address of the mass matrix M
IZEIG(56,j)		address of the preconditioning matrix C_Λ
IZEIG(57,j)		address of the pointer IDM
IZEIG(58,j)		address of the array PZUG
IZEIG(59,j)		address of the array DZUG
IZEIG(60,j)		address of the array LINT
IZEIG(61,j)		address of original r.h.s.
IZEIG(62,j)	LIE	pointer at IE(level j) in B (level dep.).

INZE1(1)	IPROB	= 1 linear problem = 2 nonlinear problem
INZEI(2)	NEN1	maximum number of nodes per edge
INZEI(3)	NEN2	maximum number of nodes per element
INZEI(4)	NDF	number of degrees of freedom per node
INZEI(5)	NNDF	= NDF for thermal problems, elasticity problems = number of degrees of freedom per node for thermal problems + number of degrees of freedom per node for the elasticity problem for solving coupled mechanic–thermal problems
INZEI(6)	NINT1	number of the quadrature formula (1-dimensional)
INZEI(7)	NQP1	number of quadrature points
INZEI(8)	NINT2M	number of the quadrature formula for evaluating the integrals during the generation of the element stiffness matrices
INZEI(9)	NQP2M	number of quadrature points for formula NINT2M

INZEI(10)	NINT2R	number of the quadrature formula for evaluating the integrals during the generation of the element last vector
INZEI(11)	NQP2R	number of quadrature points for formula NINT2R
INZEI(12)	NTR	number of grids
INZEI(13)	IVERFT	number of the refinement criterion (maximum of the values INZEI(41), ..., INZEI(39+NTR))
INZEI(21)	NUNPM	estimation for the number of nodes in the new grid which is to be generated
INZEI(22)	NUNCRPM	number of cross points
INZEI(23)	NUCPM	estimation for the number of coupling nodes in the new grid which is to be generated
INZEI(24)	NUINPM	estimation for the number of inner nodes in the new grid which is to be generated
INZEI(25)	NUEDM	estimation for the number of edges in the new grid which is to be generated
INZEI(26)	NUBEDM	estimation for the number of boundary nodes in the new grid which is to be generated
INZEI(27)	NUINPM	number of coupling edges in the new grid which is to be generated
INZEI(28)	NUELM	estimation for the number of elements in the new grid which is to be generated
INZEI(29)	NUMPM	estimation for the number of intermediate nodes in the new grid which is to be generated
INZEI(30)	NUWKM	estimation for the number of nodes with the weight $\neq$ 1/2 (for interpolation/restriction) in the new grid which is to be generated
INZEI(31)	IISNN	length of the array ISONN of the finest grid
INZEI(41) –		array IVFN: choice of the refinement criteria for the generation of the grids $q = 2, 3, \ldots, l$
INZEI(50)		= 1: quartering the triangles
INZEI(51)	MAX	maximum number of edges starting at a some node
INZEI(52)	LUL	starting address of the array UL (array for the local solution) within the working array B
INZEI(53)	LUL	starting address of the array S (array for the local system matrix) within the working array B
INZEI(54)	LUL	starting address of the array P (array for the local r.h.s.) within the working array B
INZEI(55)	LMAIDX	starting address of the array MAIDX (material - index array, length $3 * NMAIDX$) within the working array B
INZEI(56)	LDATMA	starting address of the array DATMA (material - data) within the working array B
INZEI(57)	NMAIDX	maximum of the material-id

INZEI(58)	LMRIDX	starting address of the array MRIDX (boundary - index array, length $3 * NMRIDX$) within the working array B
INZEI(59)	LDATMR	starting address of the array DATMR (boundary - data array) within the working array B
INZEI(60)	NMRIDX	maximum of the boundary-id
INZEI(61)	LUUL	starting address of the array UUL (array of the local solution of the nonlinear problem) within the working array B
INZEI(63)	JGG	globally coarsest grid
INZEI(64)	NTRFULL	finest grid for nested iteration
INZEI(70)		type of the BE-matrix generation method (0: numerically, ≥ 3: analytically within the given bandwidth)
INZEI(71)	LITYP1	starting address of the array ITYP1 (array for choosing the method of pre-smoothing) within the working array B
INZEI(72)	LNUE1	starting address of the array NUE1 (number of pre-smoothing steps on each grid) within the working array B
INZEI(73)	LPAR1	starting address of the array PAR1 (parameter of the pre-smoothing) within the working array B
INZEI(74)	LITYP2	starting address of the array ITYP2 (array for choosing the method of post-smoothing) within the working array B
INZEI(75)	LNUE2	starting address of the array NUE2 (number of post-smoothing steps on each grid) within the working array B
INZEI(76)	LPAR2	starting address of the array PAR2 (parameter of the post-smoothing) within the working array B
INZEI(77)	LMART	starting address of the array MART (kind of the multigrid cycle) within the working array B
INZEI(78)	LITMG	starting address of the array ITMG (number of iteration steps of the multigrid method on each grid within the full–mg–algorithm) within the working array B
INZEI(79)	LITCG	starting address of the array ITCG (number of iteration steps of the PCCG-method on each grid within the full–mg–algorithm) within the working array B
INZEI(80)	LLZYK	starting address of the array LZYK (number of iteration steps of the mg-algorithm or the PCCG-algorithm that have been performed on each grid within the full–mg–algorithm) within the working array B
INZEI(81)	LLEVEL	starting address of the array LEVEL (auxiliary array in the mg-algorithm) within the working array B

INZEI(82)	IABBR	terminating code of the multigrid-/PCCG-algorithm
INZEI(83)	ICC	pointer, whether the Cholesky-decomposition of the system matrix on the coarsest grid has been performed already (=1) or not (=0)
INZEI(84)	JG	number of the grid which is to be used as coarsest grid within the mg-algorithm
INZEI(86)		pointer to type of the local BE-preconditioning which is to be used (1: force the use of a circulant matrix, 0,2: force the $M^{-T}DM^{-1}$-method, 4: elimination of C_Λ)
INZEI(101)	LevMax	number of hierarchical levels (level dep.)
INZEI(102)	LISP	pointer at ISP in B (level dep.)
INZEI(103)	LISV	pointer at ISV in B (level dep.)
INZEI(104)	LDNEIGH	pointer at DNEIGH in B (level dep.)
INZEI(105)	LSKC4	pointer at SKC4 in B (level dep.)
INZEI(106)	LL2G	pointer at NrVerL2G in B
INZEI(107)	KindDisk	kind of discretization in B
INZEI(108)	LRM	pointer at rMatle in B (non linearity !)
INZEI(109)	LJE	pointer at JE in B (level dep.)
INZEI(110)	nCrP	global number of cross points
INZEI(111)	LIDV	pointer at IDV in B
INZEI(112)	LSKV	pointer at SKV in B
INZEI(113)	LOrEdg	pointer at OrEdg in B
INZEI(114)	BEMPc	describes the BEM domain and the kind of preconditioning
INZEI(115)	LMGCW	pointer at weights in B for MG-CC preconditioning
INZEI(116)	GMGPCG	use global multigrid as preconditioner
INZEI(117)	MAXCG	max. number of cg-iterations
INZEI(120)	NumMat	number of material classes in the domain
INZEI(121)	DimMat	number of parameters per material (= 8)
INZEI(122)	MaxNrEdg	max. number of edges per face describing the domain
INZEI(123)	NumSub	number of faces (in 2D subdomains) describing the domain on one processor
INZEI(124)	LNRD	pointer at NrDEdge in B
INZEI(125)	LIFA	pointer at IFacel in B
INZEI(126)	LIMA	pointer at IMatle in B.

Appendix D

Programmers and their addresses

- Ulrich LANGER
  Johannes Kepler University Linz, Institute of Mathematics
  Altenbergerstr. 69, A-4040 Linz
  Phone : +43(0732) 2468 9167
  e-mail : ulanger@numa.uni-linz.ac.at
  Coordinator of the project FEM$\otimes$BEM .

- Gundolf HAASE
  Johannes Kepler University Linz, Institute of Mathematics
  Altenbergerstr. 69, A-4040 Linz
  Phone : +43(0732) 2468 9164
  e-mail : ghaase@numa.uni-linz.ac.at
  Responsible for general libraries and for the parallel implementation.

- Bodo HEISE
  Johannes Kepler University Linz, Institute of Mathematics
  Altenbergerstr. 69, A-4040 Linz
  Phone : +43(0732) 2468 9157
  e-mail : heise@numa.uni-linz.ac.at
  Programmer of GSGen, NewtonSolv.

- Michael JUNG
  Technical University Chemnitz–Zwickau, Faculty of Mathematics
  PSF 964, 09009 Chemnitz
  Telefon : +49(0371) 531 2163
  e-mail : dr.michael.jung@mathematik.tu-chemnitz.de
  Programmer of GiGeHi, MGSolv.

- Michael KUHN
  Johannes Kepler University Linz, Institute of Mathematics
  Altenbergerstr. 69, A-4040 Linz
  Phone : +43(0732) 2468 9157
  e-mail : kuhn@numa.uni-linz.ac.at
  Programmer of GeBEM & CoBEM, ADDPre.

- Olaf STEINBACH
  University of Stuttgart, Mathematical Institute A
  Pfaffenwaldring 57, D-70569 Stuttgart

Phone : +49(0711)685 5528
e-mail : steinbach@mathematik.uni-stuttgart.de
Programmer of OstBEM2D.

Papers written by the authors and additional information can be accessed via ftp and WWW:

- ftp://ftp.numa.uni-linz.ac.at and http://www.numa.uni-linz.ac.at

Bibliography

[1] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I – IV. *Mathematics of Computation*, 1986, 1987, 1988, 1989. 47, 103–134, 49, 1–16, 51, 415–430, 53, 1–24.

[2] C. Brand. A modification of the rsb. Personal communication at a worshop in Graz, Sept. 1995.

[3] F. Bräuer. Grafische Ergebnisaufbereitung auf Parallelrechnern. Jahresarbeit, TU Chemnitz-Zwickau, FB Mathematik, 1993.

[4] C. Carstensen, M. Kuhn, and U. Langer. Fast parallel solvers for symmetric boundary element domain decomposition equations. Report 50x, Institute of Mathematics, Johannes Kepler University Linz, 1996.

[5] M. Costabel, V. J. Ervin, and E. P. Stephan. Experimental convergence rates for various couplings of boundary and finite elements. Preprint 1193, TH Darmstadt, 1988.

[6] M. Costable. Symmetric methods for the coupling of finite elements and boundary elements. In C. A. Brebbia, W. L. Wendland, and G. Kuhn, editors, *Boundary Elements IX*, pages 411–420. Springer–Verlag, 1987.

[7] F. N. Fritsch and R. E. Carlson. Monotone piecewise cubic interpolation. *SIAM J. Numer. Anal.*, 15:235–246, 1980.

[8] G. Globisch. PARMESH - a parallel mesh generator. Preprint SPC 93_03, Technische Universität Chemnitz, Fachbereich Mathematik, 1993.

[9] G. Globisch. PARMESH - a parallel mesh generator. *Parallel Computing*, 21(3):509–524, 1995.

[10] J. Goodman, V. R. Kohn, and L. Reyna. Numerical study of a relaxed variational problem from optimal design. *Computer Methods in Applied Math. and Engineering*, 57:107–127, 1986.

[11] M. Goppold, G. Haase, B. Heise, and M. Kuhn. Preprocessing in BE/FE domain decomposition methods. Technical Report 96-2, Institute of Mathematics, Johannes Kepler University Linz, 1996.

[12] G. Haase. Die nichtüberlappende Gebietszerlegungsmethode zur Parallelisierung und Vorkonditionierung der CG-Verfahren: Algorithmen und numerische Vergleiche. Report, IWR Heidelberg, Heidelberg, 1992.

[13] G. Haase. Die nichtüberlappende Gebietszerlegungsmethode zur Parallelisierung und Vorkonditionierung des CG-Verfahrens. Report 92-10, IWR Heidelberg, 1992.

[14] G. Haase, B. Heise, M. Jung, and M. Kuhn. FEM⊗BEM - a parallel solver for linear and nonlinear coupled FE/BE-equations. DFG-Schwerpunkt "Randelementmethoden", Report 94-16, University of Stuttgart, 1994.

[15] G. Haase, B. Heise, M. Kuhn, and U. Langer. Adaptive domain decomposition methods for finite and boundary element equations. Technical Report 95-2, Universität Linz, Institut für Mathematik, Arbeitsgruppe Numerische Mathematik und Optimierung, 1995.

[16] G. Haase, T. Hommel, A. Meyer, and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. Preprint SPC 93_01, TU Chemnitz–Zwickau, 1993.

[17] G. Haase, T. Hommel, A. Meyer, and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. Preprint, Faculty for Mathematics SPC 95_20, TU Chemnitz-Zwickau, 1995. 3rd edn., June 1995.

[18] G. Haase and U. Langer. On the use of multigrid preconditioners in the domain decomposition method. In W. Hackbusch, editor, *Parallel Algorithms for PDEs*, pages 101–110, Braunschweig, 1990. Vieweg. Proc. of the 6th GAMM–Seminar, Kiel, 1990.

[19] G. Haase and U. Langer. The non-overlapping domain decomposition multiplicative Schwarz method. *International Journal of Computer Mathematics*, 44:223–242, 1992.

[20] G. Haase, U. Langer, and A. Meyer. A new approach to the Dirichlet domain decomposition method. In S. Hengst, editor, *Fifth Multigrid Seminar, Eberswalde 1990*, pages 1–59, Berlin, 1990. Karl–Weierstrass–Institut. Report R–MATH–09/90.

[21] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet decomposition method. part I,II. *Computing*, 47:137–167, 1991.

[22] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part I: An algebraic approach. Part II: Applications to 2nd-order elliptic boundary value problems. *Computing*, 47:137–151 (Part I), 153–167 (Part II), 1991.

[23] G. Haase, U. Langer, and A. Meyer. Domain decomposition methods with inexact subdomain solvers. *Journal of Numerical Linear Algebra with Applications*, 1:27–41, 1991.

[24] G. Haase, U. Langer, and A. Meyer. Domain decomposition preconditioners with inexact subdomain solvers. *J. of Num. Lin. Alg. with Appl.*, 1:27–42, 1992. (Preprint 192, TU Chemnitz, 1991).

[25] G. Haase, U. Langer, and A. Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In G. Bader, R. Rannacher, and G. Wittum, editors, *Numerische Algorithmen auf Transputer–Systemen*, pages 80–116. Teubner–Verlag Stuttgart, 1993. Proceedings of the GAMM–Seminar Heidelberg, 1991.

[26] G. Haase, U. Langer, A. Meyer, and S. V. Nepomnyaschikh. Hierarchical extension operators and local multigrid methods in domain decomposition preconditioners. *East-West Journal of Numerical Mathematics*, 2(3):173–193, 1994.

[27] W. Hackbusch. *Multi–Grid methods and applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer–Verlag, Berlin, 1985.

[28] B. Heise. Multigrid–Newton methods for the calculation or electromagnetic fields. In G. Telschow, editor, *Third Multigrid Seminar, Biesenthal 1988*, pages 53–73, Berlin, 1989. Karl–Weierstrass–Institut. Report R–MATH–03/89.

[29] B. Heise. Mehrgitter–Newton–Verfahren zur Berechnung nichtlinearer magnetischer Felder. Wissenschaftliche Schriftenreihe 4/1991, Technische Universität Chemnitz, 1991.

[30] B. Heise. *Mehrgitter–Newton–Verfahren zur Lösung elektromagnetischer Feldgleichungen.* PhD thesis, Technische Universität Chemnitz, 1991.

[31] B. Heise. Nonlinear field calculations with multigrid-Newton methods. *IMPACT of Computing in Science and Engineering*, 5:75–110, 1993.

[32] B. Heise. Sensitivity analysis for nonlinear magnetic field simulation. In H. Bandemer, editor, *Modelling Uncertain Data*, volume 68 of *Mathematical Research*, pages 40–45, Berlin, 1993. Akademie Verlag. Proc. of GAMM–Workshop, Bergakademie Freiberg, March 21–24, 1992.

[33] B. Heise. Analysis of a fully discrete finite element method for a nonlinear magnetic field problem. *SIAM J. Numer. Anal.*, 31(3):745–759, 1994.

[34] B. Heise. Nonlinear magnetic field simulation with FE/BE domain decomposition methods on MIMD parallel computers. DFG-Schwerpunkt "Randelementmethoden", Report 94-19, University Stuttgart, 1994.

[35] B. Heise. Nonlinear simulation of electromagnetic fields with domain decomposition methods on MIMD parallel computers. *Journal of Computational and Applied Mathematics*, 63:373–381, 1995.

[36] B. Heise. Parallel solvers for coupled FEM-BEM equations with applications to nonlinear magnetic field problems. In W. Hackbusch and G. Wittum, editors, *Numerical Treatment of Coupled Systems*, volume 51 of *Notes on Numerical Fluid Mechanics*, Braunschweig, Wiesbaden, 1995. Vieweg. Proc. of the 11th GAMM–Seminar, Kiel, January 20 to 22, 1995.

[37] B. Heise. Nonlinear field simulation with FE domain decomposition methods on massively parallel computers. *Surveys on Mathematics for Industry*, 1996. To appear.

[38] B. Heise and M. Jung. Comparison of parallel solvers for nonlinear elliptic problems based on domain decomposition ideas. Report 494, Johannes Kepler University Linz, Institute of Mathematics, 1995. Submitted for publication.

[39] B. Heise and M. Kuhn. Parallel solvers for linear and nonlinear exterior magnetic field problems based upon coupled FE/BE formulations. *Computing*, 56(3):237–258, 1996.

[40] M. Jung. Parallelization of multigrid methods based on domain decomposition ideas. Preprint SPC 95_27, Technical University Chemnitz–Zwickau, Faculty for Mathematics, 1995.

[41] B. Kawohl, J. Stara, and G. Wittum. Analysis and numerical studies of a problem of shape design. *Archive of Rational Mechanics and Analysis*, 114:349–363, 1991.

[42] N. Kikuchi. *Finite Element Methods in Mechanics.* Cambridge University Press, 1986.

[43] M. Kuhn. A parallel solver for exterior problems based on a coupled FE/BE-model. DFG-Schwerpunkt "Randelementmethoden", Report 94-15, University Stuttgart, 1994.

[44] U. Langer. Parallel iterative solution of symmetric coupled fe/be-equations via domain decomposition. *Contemporary Mathematics*, 8:335–344, 1994.

[45] S. F. McCormick. Multigrid methods for variational problems: General Theory for the V–cycle. *SIAM Journal of Numerical Analysis*, 22(4):634–643, 1985.

[46] J. C. Nedelec. Integral equations with non integrable kernels. *J. Int. Eqns. Oper. Theor.*, 5:562–572, 1982.

[47] J. T. Oden and J. N. Reddy. *An introduction to the mathematical theory of finite elements.* John Wiley & Sons, New York, 1976.

[48] W. Queck. FEMGP – Finite-Element-Multi-Grid-Package – Programmdokumentation und Nutzerinformation. Report, Technische Universität Chemnitz-Zwickau, Fachbereich Mathematik, 1993.

[49] A. A. Samarskij and E. S. Nikolaev. Metody rešenija setočnych uravnenija. *Nauka*, 1978. In Russian.

[50] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Comput. System in Eng.*, 2:135–148, 1991.

[51] O. Steinbach. Galerkin– und Kollokationsdiskretisierungen für Randintegralgleichungen in 2D. Preprint 96–5, Universität Stuttgart, 1996.

[52] O. Steinbach. Verallgemeinerte Gebietszerlegungsmethode für Randintegralgleichungen mit Anwendungen. Dissertation, Universität Stuttgart, 1996. In Vorbereitung.

[53] O. Steinbach and W. L. Wendland. Efficient preconditioners for boundary element methods and their use in domain decomposition. DFG-Schwerpunkt "Randelementmethoden", Report 95-19, University of Stuttgart, 1995.