Technische Universität Chemnitz

Sonderforschungsbereich 393

Numerische Simulation auf massiv parallelen Rechnern

Arnd Meyer

Programmer's Manual for Adaptive Finite Element Code SPC – PM 2Ad

Preprint SFB393/01-18

Preprint-Reihe des Chemnitzer SFB 393

SFB393/01-18 **Mai 2001**

Contents

Author's addresses:

Arnd Meyer
TU Chemnitz
Fakultät für Mathematik
D-09107 Chemnitz

`http://www.tu-chemnitz.de/sfb393/`

1 Overview

1.1 The Partial Differential Equations to be Approximated

The adaptiv finite element code is concerned to solve the following p.d.e.'s approximately

(A) potential / reaction–diffusion problem (`NDof=1`)

$$\begin{aligned} -div(A(\vec{x})\nabla u) + \gamma(\vec{x})u &= f(\vec{x}) \text{ in } \Omega \\ u &= g_D \text{ on } \Gamma_D \\ \mathfrak{n} \cdot A(\vec{x})\nabla u &= g_N \text{ on } \Gamma_N = \Omega \setminus \Gamma_D r \end{aligned} \tag{1.1}$$

here $f(\vec{x}),\ A(\vec{x}) = diag(\alpha_1(\vec{x}), \alpha_2(\vec{x})),\ \gamma(\vec{x}) \geq 0$ are given (constant in subdomains of $\Omega \subset \mathbb{R}^2$).

For the general weak formulation we define $B_\rho(\vec{x}) = \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix},\ \vec{x} = (x_1, x_2)^T$.

(B) The Lamé Equation of linear elasticity (`NDof=2`) with volume forces $\vec{f}(\vec{x})$ and given elasticity–modules $E(\vec{x})$ and Poissons ratio $\nu(\vec{x})$ (again constant in subdomains).

In both cases we use Cartesian (x, y)–coordinates or cylindrical (r, z)–coordinates, so the Lamé Equation is given in its weak formulation only:

For cylindrical coordinates we use the tensor basis $\vec{e}_1, \vec{e}_2, \vec{e}_3$. From $\vec{x} = \vec{x}(r, z, \varphi)$ we define

$$\vec{e}_1 = \frac{\partial}{\partial r}\vec{x},\ \vec{e}_2 \frac{\partial}{\partial z}\vec{x},\ \vec{e}_3 = \frac{1}{r}\frac{\partial}{\partial \varphi}\vec{x}, \quad \text{so}$$

$$\vec{x} = \begin{pmatrix} r\cos\varphi \\ r\sin\varphi \\ z \end{pmatrix}, \vec{e}_1 = \begin{pmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{pmatrix}, \vec{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \vec{e}_3 = \begin{pmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{pmatrix},$$

which means for the grad–operator

$$grad = \vec{e}_1 \frac{\partial}{\partial r} + \vec{e}_2 \frac{\partial}{\partial z} + \vec{e}_3 \frac{1}{r}\frac{\partial}{\partial \varphi}.$$

In case (A) we have scalar $u(\vec{x})$, which is $u(x, y)$ or $u(r, z)$ (no dependence on φ), hence

$$\begin{aligned} a(u, v) &= \int\limits_\Omega (B_\rho(\nabla)v)^T C B_\rho(\nabla)u \, d\Omega \\ \langle f, v \rangle &= \int\limits_\Omega f \cdot v \, d\Omega + \int\limits_{\Gamma_N} g_N \cdot v \, d\Gamma \end{aligned}$$

with $C = diag(\alpha_1, \alpha_2, \gamma)$,
where $(x_1, x_2) = (x, y)$ then $d\Omega = dx\, dy$
or $(x_1, x_2) = (r, z)$ then $d\Omega = r\, dr\, dz$.

In both cases

$$\nabla = (\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2})^T,$$

so (1.1) is equivalent to:
Find $u \in H^1(\Omega)$ with $u = g_D|_{\Gamma_D}$ and

$$a(u,v) = \langle f, v\rangle \quad \forall v \in H^1_0(\Omega) = \{v \in H^1(\Omega) \,:\, v = 0|_{\Gamma_D}\}.$$

The same is true for case (B), if we define

$$B_\rho(\vec{x}) = \begin{pmatrix} \rho & 0 \\ x_1 & 0 \\ 0 & x_2 \\ x_2 & x_1 \end{pmatrix} \text{ with } \rho = \begin{cases} 0 & \text{in } (x,y)\text{–coordinates} \\ \frac{1}{r} & \text{in } (r,z)\text{–coordinates.} \end{cases}$$

Then with the 3D–meaning: $\quad \vec{u}(\vec{x}) = u^1\vec{e}_1 + u^2\vec{e}_2 \quad$, we look for the 2–vector $u(\vec{x}) = \begin{pmatrix} u^1 \\ u^2 \end{pmatrix}$ (again independent of φ) and have

$$\underline{\epsilon} = B_\rho(\nabla)u$$

with the 4 nonzero components of the strain tensor

$$\underline{\epsilon} = (\epsilon_{33}, \epsilon_{11}, \epsilon_{22}, 2\epsilon_{12})^T.$$

In (x,y)–coordinates $\epsilon_{33} = 0$ else $\epsilon_{33} = u^1/r$ and $\epsilon_{ij} = \frac{1}{2}(\frac{\partial}{\partial x_i}u^j + \frac{\partial}{\partial x_j}u^i)$ for $i,j \le 2$.

Analogously the stress tensor has 4 nonzero components (w.r.t. the tensor basis $\vec{e}_i\vec{e}_j$), so $C\underline{\epsilon}$ are these stress components with

$$C = \mu\, diag(2,2,2,1) + \lambda \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}^T$$

where μ and λ are the Lamé coefficients

$$2\mu = \frac{E}{1+\nu}, \;\; \lambda = \mu \cdot \frac{\nu}{1-2\nu}.$$

<u>Summarizing the above:</u>

We look for $u \in (H^1(\Omega))^{\texttt{NDof}}$ with b.c. on Γ_D and

$$a(u,v) = \langle f, v\rangle \;\forall v \in (H^1_0(\Omega))^{\texttt{NDof}},$$

which is solved with adaptive Finite Element discretization, where

$$a(u,v) = \int_\Omega (B_\rho(\nabla)v)^T C (B_\rho(\nabla)u) d\Omega$$

and $B_\rho(\vec{x})$, C, $d\Omega$ defined above depending on cases (A) or (B) and the coordinates used.

1.2 Boundary Conditions

Let $\partial\Omega = \Gamma_D \cup \Gamma_N \quad \Gamma_D \cap \Gamma_N = \emptyset$.
We have Dirichlet type b.c.'s on Γ_D

(A): $u = g_D(x)$ along edge E
$g_D(x)$ can be a quadratic function prescribed by 3 values :
$g_D(N_A), g_D(N_E), g_D(N_M)$ when
N_A, N_E are endpoints of the edge E and
N_M is the midpoint of E;

(B): the same definition as (A), now $g_D(x)$ is a vector function.
Additionally we can prescribe a "slip–condition"
$\vec{u} \cdot \vec{n} = 0$ (normally to the edge–vector $\vec{E}$: $\quad \vec{E} \cdot \vec{n} = 0$)
or a contact condition:
$\vec{x} + \vec{u}(\vec{x})$ shall not penetrate an obstacle $\{\vec{p} + \lambda\vec{\mathfrak{t}} : \lambda \in \mathbb{R}\}$

$$\Longrightarrow \quad \vec{u} \cdot \vec{\mathfrak{n}} \geq (\vec{p} - \vec{x}) \cdot \vec{\mathfrak{n}}.$$

Here, $\quad \vec{\mathfrak{n}} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \vec{\mathfrak{t}}$, where $\vec{p}$ and $\vec{\mathfrak{t}}$ are given.

We have Neumann type b.c.'s on $\Gamma_N = \Gamma_{N,i} \cup \Gamma_{N,0}$.
The b.c.'s are prescribed on $\Gamma_{N,i}$ from imput file, on the remaining part $\Gamma_{N,0}$ homogeneous b.c.'s are taken automatically.

(A): $\vec{\mathfrak{n}} \cdot (A(x)\nabla u) = g_N(x)$ on E ,
again as for Dirichlet b.c.'s $g_N(x)$ is a quadratic function on E from $g_N(N_A), g_N(N_E)$ and $g_N(N_M)$

(B): analogously, tractions $\vec{g}_N(x)$ on E are prescribed

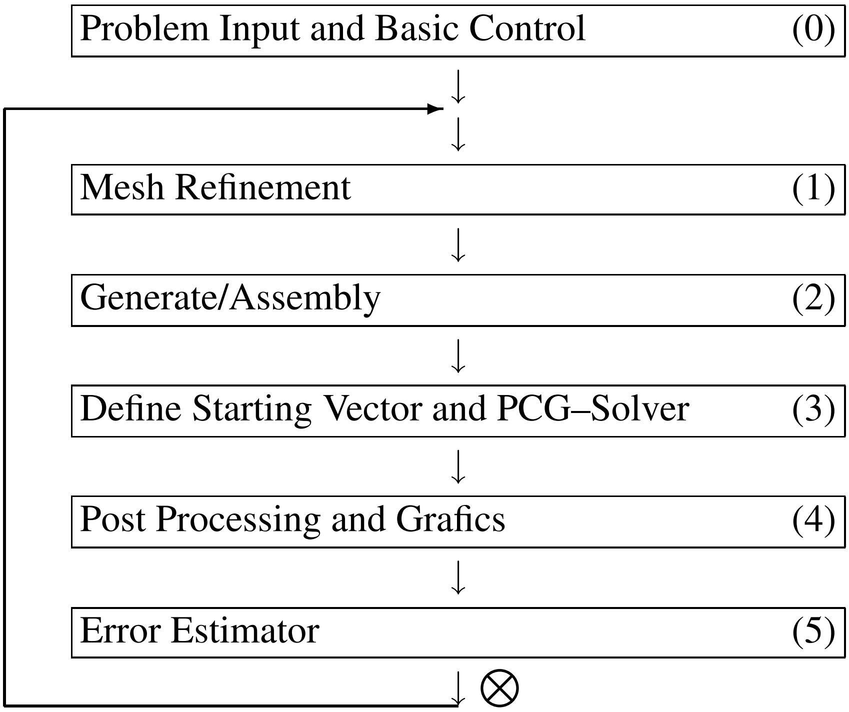
$$B_0(\vec{\mathfrak{n}})^T (CB_\rho(\nabla)u) = \vec{g}_N(x)$$

$\vec{g}_N(x)$ possibly quadratic on E.

In both cases $\vec{\mathfrak{n}}$ is outer normal on Ω ($\vec{E} \perp \vec{n}$).

1.3 General Structure of the Adaptive Algorithm

The general run overview is given in the following flow chart, so we are going to present the six main steps (0) to (5) in the following chapters.



At the position ⊗ a user control can enforce

an additional solve without refining the mesh	(input: `0`),
a total subdividion of each actual element	(input: `t`),
or stop this calculation for a new example	(input: `n`).

2 The Data Structures

For efficient implementation of adaptive F.E. work in 2D we need the following 3 fixed data structures for holding:

- Material informations
- Dirichlet type b.c.'s
- Neumann type b.c.'s

and 3 in work growing data structures for defining

- the nodes together with solution values
- the edges E together with pointers to son–edges and edge infos
- the elements together with infos on material, element nodes, element matrices and right hand sides.

The data structures for implementing multi level solvers (the hierarchical grid information) are contained within the edge tree, so no additional data structur is required.

2.1 Fixed Structures

The following data structures are read from the input file and occupy a small fixed area in the memory.

2.1.1 Material–Information (`NMat, MMax, RMat`)

A 1D-Array `RMAT` with `NMat` material infos for maximally `NMat` subdomains with constant material. Each material info consists of 1 INTEGER value `kMat = RMat(1,.)` and `NMax` REAL–Values. `kMax` $\leq$ `NMax` is the valid number of material infos for this subdomain. Currently we have `NMax` = 4, with the meaning

	case (A)	case (B)
$\mathtt{RMat}(2,\cdot) =$	α_1	E
$\mathtt{RMat}(3,\cdot) =$	α_2	ν
$\mathtt{RMat}(4,\cdot) =$	γ	f_1
$\mathtt{RMat}(5,\cdot) =$	f	f_2

($\alpha_i, \gamma, E, \nu, f_i$ see Chapter 1)

2.1.2 Dirichlet Type Boundary Conditions

1D–Array `DirB` with `NDir` structures of the folowing type:
One Dirichlet information are two INTEGERs E0, `DofMask`
and 3 REAL–Values for each degree of freedom, so (2+3 * `NDof`) words.

E0	Pointer to (INPUT)–coarse–edge E for which this b.c. is defined
`DofMask`	bitwise information if this b.c. is valid for each d.o.f. (Bit $2^o = 1 \Longleftrightarrow$ valid for first d.o.f. Bit $2^o = 0 \Longrightarrow$ first 3 REAL–Values unused).

The 3 REAL values for each d.o.f. are $g(E_A), g(E_E), g(E_M)$ resp.

So, for `NDof` $\in \{1, 2, \}$ we have `DofMask` $\in \{0, 1, 2, 3\}$, then other values of `DofMask` can define additional Dirichlet b.c.'s.

`DofMask` = 4:	Slip–condition for this edge (REAL–values unused)
`DofMask` = 8:	contact–condition (see 1.1) the six REAL–values are $(p_1, t_1$, unused p_2, t_2 , unused)

Use in FORTRAN: `DirB(2+3*NDof, NDir)`

2.1.3 Neumann Type Boundary Conditions

Same structur as Dirichlet type b.c.'s : a 1D–array of Neumann b.c. infos `NeumW`
One Neumann b.c. info consists of :
E0, `DofMask` and (3 REAL–values g_1, g_2, g_3 for each d.o.f.)
defining $g(x)$ (see 1.1) as quadratic function on the edge E0 from the 3 values
$g_1 = g(N_A), g_2 = g(N_E), g_3 = g(N_M)$.

Use in FORTRAN: `NeumW(2+3*NDof, NNeum)`

2.2 Growing Data Structures

2.2.1 Nodes

The Structur X is an 1 dimensional array of nodal informations. One nodal information contains 2+`NDof` REAL values:

$$(x_1, x_2, u_1(\vec{x}), \cdots, u_{\texttt{NDof}}(\vec{x})),$$

x_1, x_2 the coordinates of this node ($x_1 = x$ coordinate or $x_1 = r$–coordinate resp.)
$u_i(\vec{x})$ values of the FE solution at $\vec{x}$ for the actual mesh.

2.2.2 Edges

The structur `Edg` plays a central role for both mesh refinement and hierarchical solvers. It is a 1D–array of `Nedg` edge–structures. Each egde–structure contains basically 5 INTEGER–infos. If the BPX–solver is required, we have to add 5 INTEGER–infos addidionally.

The basic edge–structure contains:

$$(N_M, N_A, N_E, K1, K2)$$

with:	N_M mid node of this edge N_A start node of this edge N_E end node this edge
K1 ≤ 0	then this edge is not subdivided (leave of Edge–Tree) then K2 stores edge information
else	K1 / K2 are numbers of son–edges

Edge information contains geometry and b.c.'s:

Geometry–Typ = `(K2 .AND. GeoByte)`
Neumann–b.c.–number = `(K2 .AND. NeumByte) / NeumDiv`
Dirichlet–b.c.–number = `(K2 .AND. DirByte) / DirDiv.`

Here (x `.AND.` y) means bit by bit `AND` operation, so `GeoByte`/`NeumByte`/`DirByte` are masks for picking out this subinformation from K2. Currently they are defined as:

`GeoByte = 7`	(so 3 Bits for Geometry)
`NeumByte = 8 * 1023`	(so 10 Bits for Neumann b.c.)
`DirByte = 1024 * 8 * 1023`	(so 10 Bits for Dirichlet b.c.)
	(hence `NeumDiv = 8, DirDiv = 8*1024`)

All these fixed constants are set in `'adapmesh.inc'` and are changeable befor recompiling.

Currently 2 kinds of edge geometries are used:

Geometry–typ = 0	straight edge
Geometry–typ = 1	circular arc

2.2.3 Elements

The structure `Elem` contains all element data on the actual mesh. It is a 1D–array of `Nel` Element–Structures. Each element–structure contains the following informations:

1. E1, ..., E4 — the 3(4) edges of this element (triangle has E4=0)
2. `ElMat` — the material number
3. N1, ..., N9 — the (maximally) 9 nodes of this element (triangles have $N7 = N8 = N9 = 0$, quadrilaterals have 8 nodes, but the 9th is temporarily used in the linear case)
4. The REAL*8–Element–matrix, the upper triangle of this (6×6) or (8×8)–matrix is stored
5. The REAL*8–element right hand side

(From the fact of having double precision values mixed in this structure, the total number of INTEGERs in the beginning has to be even.)

In the triangular case, the element matrix and right hand side occupy `6·NDof·(6·NDof+1)/2 + 6·NDof` REAL*8 values, so at the end we can store 4 REAL*8 entries of the transformation matrix T (see 5.).

3 Problem Read – Step (0)

The first step of the algorithm is related to the definition of the problem:
(A) with 1 d.o.f. or (B) with 2 d.o.f. and reading the coarse mesh with triangles or quadrilaterals. (For this reason, an input file is read from local directories:
'mesh31' : $(\triangle, A)$
'mesh32' : $(\triangle, B)$
'mesh41' : $(\square, A)$
'mesh42' : $(\square, B)$
for this 4 cases). Additionally we start with basic control inputs on the elements (linear or quadratic) and the refinement strategy, which is in the triangular case :
"g" Bänsch–green
"r" red–green
"h" (standard) red with hanging nodes.
In the quadrilateral case hanging nodes are always used. We can choose a maximal aspect ratio R for the quadrilaterals (subdivide each quadrilateral into 2 half parts whenever its local aspect ratio is larger R, so $R = 2.0$ produces near quadratic shapes, $R < 2$ is impossible). The strategie "anisotropic" is an exception for future use.

4 Mesh Refinement – Step (1)

Due to input informations on the elements and/or edges and due to local properties of each element (aspect ratio $> R$, some subdivided edges) the elements are subdivided into 2 or 4 smaller elements by changing the three data structures (more nodes/edges/elements). This is repeated until no more reason for subdividing remains.

Strategies for Triangles:
A "red" subdivision produces 4 equal subtriangles, each input edge is subdivided and 3 interior edges are generated new. The routine `DivRed` does this work.
The subdivision takes place, if 2 or more edges of this element are just subdivided.
A "green" subdivision produces 2 subtriangles whereas exactly the first edge of this element is subdivided (this is the so called subdivision edge of this element).
The new son–elements have the remaining two father edges as new subdivision edge (strategy due to Bänsch [Ban91]). The routine `DivGreen` subdivides a triangle if at least one divided edge occurs (but first the subdivision edge is taken in any case).

From combining both strategies we obtaine our 3 variants:

"Bänsch–green":	only green subdivisions
"red–green":	First all possible elements red, then the rest green, (which is repeated)
"hanging nodes":	only red subdivision, if necessary, triangles with one subdivided edge may remain (these have hanging nodes).

Remark to only red subdivisions (with hanging nodes):
The routine `DivRed` guarantees the same edge ordering in son–elements as in their father element. More precisely, let $\vec{e}_1, \vec{e}_2, \vec{e}_3$ the 3 edge vectors of the given element to be subdivided belonging to input edge numbers $E1, E2, E3$ then the edge numbers E_s1, E_s2, E_s3 of each son–element correspond to $\pm\frac{1}{2}\vec{e}_1, \pm\frac{1}{2}\vec{e}_2, \pm\frac{1}{2}\vec{e}_3$. So, knowing the transformation matrix T from the master–element to the father–element (note: $T^T = (\vec{e}_2, \vec{e}_3)$), the same transformation holds for each son–element to be $\pm\frac{1}{2}T$ (when no curved edge occurs). This is used for stable calculations in the "hanging nodes" case (see next Chapter).

Indicating hanging nodes

The routine `DivEdg` for subdividing an edge into 2 halfs produces for both son–edges the information $K1 = -2$. Each `DivRed` adds 1 to $K1$ of each of the outer edges. So, at the end we have $K1 = 0$ for each regular non–divided edge and $K1 = -1$ for an edge with its midpoint as hanging node:

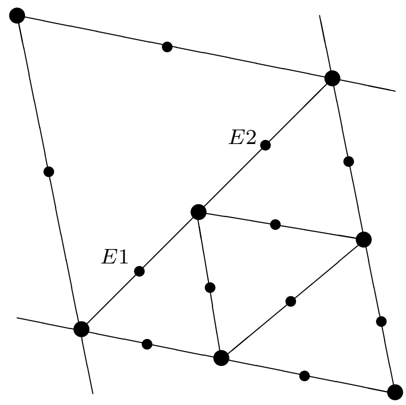


Figure 1, here $E1, E2$ have their $K1 = -1$

The same is done in the quadrilateral case (routine `Div4Eck`).

5 Generate/Assembly – Step (2)

This step defines the stiffness matrix and right hand side for approximating the p.d.e. on the actual mesh. The main control routine ist `Assem`. We store each element matrix / r.h.s. in the date structure `Elem`, so only the new elements of this actual mesh have to be considered for calling the subroutine `Element`, where this calculation is done.

In the case of linear elements, we consider each 6–node triangle as the macroelement of 4 linear subtriangles, so the `(6 NDof)` $\times$ `(6 NDof)` element matrix is assembled of its submatrices in the routine `MacroElem`. In the quadrilateral case, the same is true, but the 9th (interior) node has to be generated before and is temporarily used. Its `NDof` unknowns are eliminated on the element level, defining a `(8 Ndof)`$\times$`(8 NDof)` elemement matrix.

The element routine `Element` produces the stiffness matrix

$$A_{el} = \int\limits_{el} \breve{B}^T C \breve{B} d\Omega$$

with

$$\breve{B} = \left[B_\rho(\nabla)N_1 \vdots \ldots \vdots B_\rho(\nabla)N_n \right].$$

Here n is the number of nodes per element (so n = 3,4,6 or 8), N_i the i–th form function on the element and $B_\rho(x)$ the matrix defined in 1.1 . The integral is done with Gaussian quadrature, so the (3$\times$`NGauss`) input array `Gauss` is reqired for defining the Gaussian points $\hat{g}_j \in \mathbb{R}^2$ on the master element together with its quadrature weights ω_j = `Gauss(3,j)`.

This routine performs the following steps for each Gaussion point $\hat{g}$:

1. Define $\mathcal{N} = (N_1(\hat{g}), \ldots, N_n(\hat{g}))$
   and $\hat{\nabla}\mathcal{N} = (\hat{\nabla}N_1, \ldots, \hat{\nabla}N_n)|_{\hat{g}}$
   with $\hat{\nabla}$ the gradient w.r.t. the master coordinates
   (routine `DoMasterVal`)

2. Let X the (input–)matrix with the nodes $x^{(1)}, \ldots, x^{(n)}$ of this element as rows ($X^T = (x^{(1)} \vdots \ldots \vdots x^{(n)})$) , then $T = (\hat{\nabla}\mathcal{N}) \cdot X$ is the Jacobian of the transformation from master element to actual element.

   <u>Remark</u> to stabilizing this calculation for red subdivision of triangles:
   If we have triangular elements with straight sides, then T is exactly defined from the 3 vertices $x^{(1)}, x^{(2)}$ and $x^{(3)}$ according to the first three node numbers N_1, N_2, N_3 of this element

   $$T = (x^{(2)} - x^{(1)} \vdots x^{(3)} - x^{(1)})^T.$$

   For very fine elements (after L subdivisions of a coarse element) L binary digits of $x^{(1)}, x^{(2)}$ and $x^{(3)}$ coincide. Hence T is unstable calculated by the formular above and the process will break down. One way out is found, if only “red” subdivisions are used. Then T is stable calculated on coarse elements and during the mesh refinement T is reproduced in a stable manner:

   $$T_{son} := \pm\frac{1}{2}T_{father}.$$

   So, the switch `Isw` controls if this takes place or not:

`Isw`> 0 : T is given (input)
`Isw`< 0 : T is to be returned for future use
else T is calculated temporalily (so for "green" or curved triangles).

3. Solve for $\nabla\mathcal{N} = T^{-1}\hat{\nabla}\mathcal{N}$.

4. In case (A) we have to add

$$\begin{pmatrix} \nabla\mathcal{N} \\ \mathcal{N} \end{pmatrix}^T C \begin{pmatrix} \nabla\mathcal{N} \\ \mathcal{N} \end{pmatrix} \cdot \omega_j \cdot \det T \cdot r$$

to A_{el} $(r \equiv 1$ in (x, y)–coordinates)
otherwise build $\breve{B}$ from $\nabla\mathcal{N}$ and $\mathcal{N}$ and add

$$\breve{B}^T C \breve{B}\ \omega_j \cdot \det T \cdot r.$$

6 The Solver – Step (3)

Let actually `Nn` nodes belong to the mesh, then we have $n =$ `Nn*NDof` as the dimension of the linear system

$$K\underline{u} = \underline{b}$$

to solve. Here, the true number of unknowns is smaller due to some given Dirichlet type b.c.'s or due to the fact of having hanging nodes. From the refinement strategie each entry of the vector $\underline{u}$ has a value (interpolation of father values at new son nodes). So at first, values on Γ_D are to be corrected to given true values (routine `STAVE`), which produces the PCG starting vector $\underline{u}^{(0)}$. In case of contact b.c.'s an additional correction onto the non–penetrating condition is done.
Now, we have to solve the linear system

$$P^T K\underline{u} = P^T\underline{b}$$

with $\underline{b}$ the assembly of all element right hand sides (practically done in `Assem`) and K the assembly of all element matrices (only theoretical , the element matrices on the structure `Elem` are input for the solver, $diagK$ is assembled in `Assem` for preconditioning).

Here P is a projector onto the subspace of admissible unknowns, i.e.

$$\underline{u} = \underline{u}^{(0)} + \underline{u}^{(c)}$$

where $\underline{u}^{(0)}$ is the fixed starting vector and $\underline{u}^{(c)} \in imP$ the allowed correction in the image space of this projector. For Dirichlet–type b.c.'s at the node $x^{(j)}$ we have $P = diag(P_1, \ldots, P_{\mathtt{Nn}})$ with (`NDof`$\times$`NDof`)–blocks P_i and:

$P_j = \mathbb{O}$	if $u(x^{(j)})$ prescribed
$P_j = I - \vec{\mathrm{n}}\vec{\mathrm{n}}^T = \vec{\mathrm{t}}\vec{\mathrm{t}}^T$	if $\vec{u} \cdot \vec{\mathrm{n}} = 0$ (slip conditions or non–penetration condition , $\vec{\mathrm{t}}$ is the edge vector)
$P_j = I$	otherwise.

For existing hanging nodes, the projector P is described in [Mey99] and has in the case of linear elements a matrix part

$$P = \begin{pmatrix} \ddots & & & & \\ & I & & & \\ & & I & & \\ & \frac{1}{2}I & \frac{1}{2}I & \mathbb{O} & \\ & & & & \ddots \end{pmatrix} \begin{array}{l} \\ \text{father node 1} \\ \text{father node 2} \\ \text{son node} \\ \\ \end{array}$$

With this definition, we use the PCG solver (running in the subspace imP) with the preconditioner $PC^{-1}P^T$

$$C^{-1} = Q \begin{pmatrix} C_0^{-1} & \\ & J \end{pmatrix} Q^T .$$

Here Q is the basis–transformation of nodal into hierarchical f.e. functions. The multiply with Q and Q^T is carried out by the routines `Hismul / Hstmul` within the preconditioner using the edge–informations. C_0 is the coarse grid stiffness matrix (formed in the first step on coarsest mesh) and J the diagonal of K (except on coarse grid nodes), which is assembled together with $\underline{b}$ in step (2).

Note that two reasons lead to ratherly small numbers of iteration: We choose a relative stopping criterion of 10^{-2} only and the good starting vector coincides to an error vector of only "high frequency" type.

7 Post Processing – Step (4)

This step is not necessary for the adaptiv work, but is worthy for graphical output to the user. The routine `EfromU` calculates gradients resp. strains or stresses on element level which is required as nodal information for the following graphical output routine `gebgraph` (see [Pes94a, Pes94b] for this graphics library used from `gebgraph`).

The output of `EfromU` is temporarily stored on REAL*8–vectors of length `Nn` (as nodal information), which contains:

for case (A): the values of $\frac{\partial u}{\partial x_1}$ and $\frac{\partial u}{\partial x_2}$

for case (B): the values of $\underline{\epsilon}$ are available, but for brevity only
$||\underline{\epsilon}||^2 = \epsilon_{33}^2 + \epsilon_{11}^2 + \epsilon_{22}^2 + \epsilon_{12}^2$ is returned.

Correspondingly the graphical routine draws

for (A): u and $\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}$

for (B): u_1, u_2 and $||\underline{\epsilon}||^2$.

8 Error Estimator – Step (5)

This is the most important step for the basic control of the adaptiv meshing. Edge oriented error estimators η_E are calculated from one loop over all elements. Then we take

$$\eta = \max_{\forall E} \eta_E$$

and mark each edge E for subdivision, if

$$\eta_E^2 \geq \tau_{Tol} \cdot \eta^2$$

(The starting tolerance τ_{Tol} is `Threshold`).

When the number of marked edges is too low (not at least 10% of all edges), τ_{Tol} is decreased by $\tau_{Tol} := \tau_{Tol} \cdot$`ThreshScale` and the above test is repeated (currently `Threshold`=0.8, `ThreshScale`=0.5).

The calculation of η_E follows the basic ideas of residual error estimators. An important part of such an error estimator on the element F is the value (see [Kun97, Kun00a, Kun01a, Kun01c])

$$\sum_{E \subset \partial F} h_E \| \, [\sigma \cdot \mathfrak{n}] \, \|_{L_2(E)}^2 .$$

Here, $[.]$ denotes the jump of a function over the element boundaries. In case (A), σ is the conormal derivative $A(x)\nabla u$. For case (B) we have stresses and $\sigma \cdot \mathfrak{n}$ are the boundary tractions.

So, our edge based error estimator is defined as

$$\eta_E^2 = h_E \| \, [\sigma \cdot \mathfrak{n}] \, \|_{L_2(E)}^2 ,$$

which is simplified in using midpoint rule for integration over E, hence

$$\eta_E^2 = h_E \cdot |E| \cdot \|[\sigma \cdot \mathfrak{n}]\|^2|_{\vec{x}_M}$$

with $\vec{x}_M$ the midnode of the edge E (from $M = N_M$ of the edge information).
For isotropic elements we have $|E| \sim h_E$ and $|E| \cdot \mathfrak{n} = \vec{E}^{\perp}$ (orthogonal to edge vector). From this reason the routine `JumpEL` adds the value(s) $(\sigma \cdot \vec{E}^{\perp})|_{\vec{x}_M}$ to the Mth component of a REAL*8 vector, which represents each of the edges. The sign of $\vec{E}^{\perp}$ is chosen to be outwards of the actual element, so the jump arises automatically. We obtain η_E^2 by squaring the results after the element loop.

References

[Mey99] A. Meyer. Projected PCGM for Handling Hanging Nodes in Adaptive Finite Element Procedures.
Preprint SFB393/99-25, TU Chemnitz, 1999.

[Kun97] G. Kunert. Error estimation for anisotropic tetrahedral and triangular finite element meshes.
Preprint SFB393/97_16, TU Chemnitz, 1997.

[Kun00a] G. Kunert. An a posteriori residual error estimator for the finite element method on anisotropic tetrahedral meshes.
Numer. Math., 86(3):283–303, 2000.

[Kun01a] G. Kunert. A local problem error estimator for anisotropic tetrahedral finite element meshes.
SIAM J. Numer. Anal., 2001.

[Kun01c] G. Kunert. A posteriori l_2 error estimation on anisotropic tetrahedral finite element meshes.
IMA J. Numer. Anal., 2001.

[KV00] G. Kunert und R. Verfürth. Edge residuals dominate a posteriori error estimates for linear finite element methods on anisotropic triangular and tetrahedral meshes.
Numer. Math., 86(2):283–303, 2000.

[Ban91] E. Bänsch. Local mesh refinement in 2 and 3 dimensions.
IMPACT of Computing in Science and Engineering 3, 181–191, 1991.

[Pes94a] M. Pester. On-line visualization in parallel computations.
Preprint SFB393/94-23, TU Chemnitz, 1994.
in: W. Borchers et.al. (Eds.), Visualization Methods in High Performance Computing and Flow Simulation, Int. Science Publ., 1996, pp. 91-98.

[Pes94b] M. Pester. Grafik-Ausgabe vom Parallelrechner für 2D-Gebiete.
Preprint SFB393/94-24, TU Chemnitz, 1994.