REMARKS AROUND 50 LINES OF MATLAB: SHORT FINITE ELEMENT IMPLEMENTATION

JOCHEN ALBERTY, CARSTEN CARSTENSEN, STEFAN A. FUNKEN

ABSTRACT. A short Matlab implementation for P_1-Q_1 finite elements on triangles and parallelograms is provided for the numerical solution of elliptic problems with mixed boundary conditions on unstructured grids. According to the shortness of the programme and a given documentation, any adaptation from simple model examples to more complex problems can easily be performed. Numerical examples prove the flexibility of the Matlab tool.

1. INTRODUCTION

Unlike complex black-box commercial computer codes, this paper provides a simple and short open-box Matlab implementation of combined Courant's P_1 triangles and Q_1 elements on parallelograms for the numerical solutions of elliptic problems with mixed Dirichlet and Neumann boundary conditions. Based on four data files, arbitrary regular triangulations are determined. Instead of covering all kinds of possible problems in one code, the proposed tool aims to be simple, easy to understand and to modify. Therefore, only simple model examples are included to be adapted to whatever is needed. In further contributions we provide more complicated elements, a posteriori error estimators and flexible adaptive mesh-refining algorithms.

Compared to the latest Matlab toolbox [M], our approach is shorter, allows more elements, is easily adopted to modified problems like convection terms, and is open to easy modifications for basically any type of finite element.

The rest of the paper is organized as follows. As a model problem, the Laplace equation is described in Section 2. The discretization is sketched in a mathematical language in Section 3. The heart of this contribution is the data representation of the triangulation, the Dirichlet and Neumann boundary as the three functions specifying f, g, and u_D as described in Section 4 together with the discrete space. The main steps are the assembling procedure of the stiffness matrix in Section 5, the right-hand side in Section 6 and the incorporation of the Dirichlet boundary conditions in Section 7. A post-processing to preview the numerical solution is provided in Section 8. (The main program is given partly in these sections and in its total one page length in the Appendix A.) The applications follow in Section 9, 10, and 11 and illustrate the tool in a time-depending heat equation, in a non-linear and even in a three-dimensional example.

2. MODEL PROBLEM

The proposed Matlab-program employs the finite element method to calculate a numerical solution U which approximates the solution u to the two dimensional Laplace problem (P) with mixed boundary conditions: Let $\Omega \subset \mathbb{R}^2$ be a bounded Lipschitz domain with polygonal boundary Γ. On some closed subset Γ_D of the boundary with positive length, we assume Dirichlet conditions, while we have Neumann boundary conditions on the remaining part $\Gamma_N := \Gamma \setminus \Gamma_D$. Given $f \in L^2(\Omega)$, $u_D \in H^1(\Omega)$ and $g \in L^2(\Gamma_N)$, seek $u \in H^1(\Omega)$ with

$$-\Delta u = f \quad \text{in } \Omega, \tag{1}$$

$$u = u_D \quad \text{on } \Gamma_D, \tag{2}$$

$$\frac{\partial u}{\partial n} = g \quad \text{on } \Gamma_N. \tag{3}$$

According to the Lax-Milgram lemma, there always exists a weak solution to (1)-(3) which enjoys inner regularity (i.e. $u \in H^2_{loc}(\Omega)$), and envies regularity conditions owing to the smoothness of the boundary and the change of boundary conditions.

The inhomogeneous Dirichlet conditions (2) are incorporated through the decomposition $v = u - u_D$ so that $v = 0$ on Γ_D, i.e., $v \in H^1_D(\Omega) := \{w \in H^1(\Omega) |\ w = 0 \text{ on } \Gamma_D\}$.

Date: February 12, 1998.
1991 *Mathematics Subject Classification.* 68N15, 65N30, 65M60.
Key words and phrases. Matlab program.

. Then, the weak formulation of the boundary value problem (P) reads: Seek $v \in H^1_D(\Omega)$, such that

$$\int_\Omega \nabla v \cdot \nabla w \, dx = \int_\Omega f w \, dx + \int_{\Gamma_N} g w \, ds - \int_\Omega \nabla u_D \cdot \nabla w \, dx \quad (w \in H^1_D(\Omega)). \tag{4}$$

3. Galerkin Discretization of the Problem

For the implementation, problem (4) is discretized using the standard Galerkin–method, where $H^1(\Omega)$ and $H^1_D(\Omega)$ are replaced by finite dimensional subspaces S and $S_D = S \cap H^1_D$, respectively. Let $U_D \in S$ be a function that approximates u_D on Γ_D. (We define U_D as the nodal interpolant of u_D on Γ_D.) Then, the discretized problem (P_S) reads: Find $V \in S_D$ such that

$$\int_\Omega \nabla V \cdot \nabla W \, dx = \int_\Omega f W \, dx + \int_{\Gamma_N} g W \, ds - \int_\Omega \nabla U_D \cdot \nabla W \, dx \quad (W \in S_D). \tag{5}$$

Let $(\eta_1, \dots, \eta_N)$ be a basis of the finite dimensional space S, and let $(\eta_{i_1}, \dots, \eta_{i_M})$ be a basis of S_D, where $I = \{i_1, \dots, i_M\} \subseteq \{1, \dots, N\}$ is an index set of cardinality $M \leq N - 2$. Then, (5) is equivalent to

$$\int_\Omega \nabla V \cdot \nabla \eta_j \, dx = \int_\Omega f \eta_j \, dx + \int_{\Gamma_N} g \eta_j \, ds - \int_\Omega \nabla U_D \cdot \nabla \eta_j \, dx \quad (j \in I). \tag{6}$$

Furthermore, let $V = \sum_{k \in I} x_k \eta_k$ and $U_D = \sum_{k=1}^N U_k \eta_k$. Then, the equation (6) yields the linear system of equations

$$Ax = b. \tag{7}$$

The coefficient matrix $A = (A_{jk})_{j,k \in I} \in \mathbb{R}^{M \times M}$ and the right-hand side $b = (b_j)_{j \in I} \in \mathbb{R}^M$ are defined as

$$A_{jk} = \int_\Omega \nabla \eta_j \cdot \nabla \eta_k \, dx \quad \text{and} \quad b_j = \int_\Omega f \eta_j \, dx + \int_{\Gamma_N} g \eta_j \, ds - \sum_{k=1}^N U_k \int_\Omega \nabla \eta_j \cdot \nabla \eta_k \, dx. \tag{8}$$

The coefficient matrix is sparse, symmetric and positive definite, so (7) has exactly one solution $x \in \mathbb{R}^M$ which determines the Galerkin solution $U = U_D + V = \sum_{j=1}^N U_j \eta_j + \sum_{k \in I} x_k \eta_k$.

4. Data-Representation of the Triangulation Ω

Suppose the domain Ω has a polygonal boundary Γ, we can cover $\bar\Omega$ by a regular triangulation $\mathcal{T}$ of triangles and quadrilaterals, i.e. $\bar\Omega = \bigcup_{T \in \mathcal{T}} T$ and each T is either a closed triangle or a closed quadrilateral.

Regular triagulation in the sense of Ciarlet [Ci] means that the nodes $\mathcal{N}$ of the mesh lie on the vertices of the triangles or quadrilaterals, the elements of the triangulation do not overlap, no node lies on an edge of a triangle or quadrilateral, and each edge $E \subset \Gamma$ of an element $T \in \mathcal{T}$ belongs either to $\bar\Gamma_N$ or to $\bar\Gamma_D$.

Matlab supports reading data from files given in ascii format by .dat files. Fig. 1 shows the mesh which is described by the following data. The files `Coordinates.dat` contains the coordinates of each node in the given mesh. Each row has the form

node number x-coordinate y-coordinate.

Coordinates.dat

1	0	0
2	1	0
3	1.59	0
4	2	1
5	3	1.41
6	3	2
7	3	3
8	2	3
9	1	3
10	0	3
11	0	2
12	0	1
13	1	1
14	1	2
15	2	2

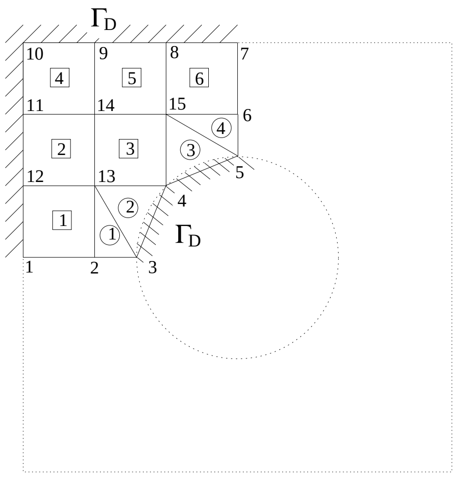

Figure 1. Example of a Mesh

In our code we allow subdivision of Ω into triangles and quadrilaterals. In both cases the nodes are numbered anti-clockwise. `Elements3.dat` contains for each triangle the node numbers of the vertices. Each row has the form

element number node1 node2 node3.

Similiarly, the data for the quadrilaterals are given in `Elements4.dat`. Here, we use the format

element number node1 node2 node3 node4.

Elements3.dat

1	2	3	13
2	3	4	13
3	4	5	15
4	5	6	15

Elements4.dat

1	1	2	13	12
2	12	13	14	11
3	13	4	15	14
4	11	14	9	10
5	14	15	8	9
6	15	6	7	8

`Neumann.dat` and `Dirichlet.dat` contain in each row the two node numbers which bound the corresponding edge on the boundary:

Neumann edge number node1 node2 resp. Dirichlet edge number node1 node2.

Neumann.dat

1	5	6
2	6	7
3	1	2
4	2	3

Dirichlet.dat

1	3	4
2	4	5
3	7	8
4	8	9
5	9	10
6	10	11
7	11	12
8	12	1

The spline spaces S and S_D are globally continous and affine on each triangular element and bilinear isoparametric on each quadrilateral element. In Fig. 2 we display the hat function η_j are defined for every node (x_j, y_j) of the mesh by

$$\eta_j(x_k, y_k) = \delta_{jk} \quad (j, k = 1, \dots, N) \,.$$

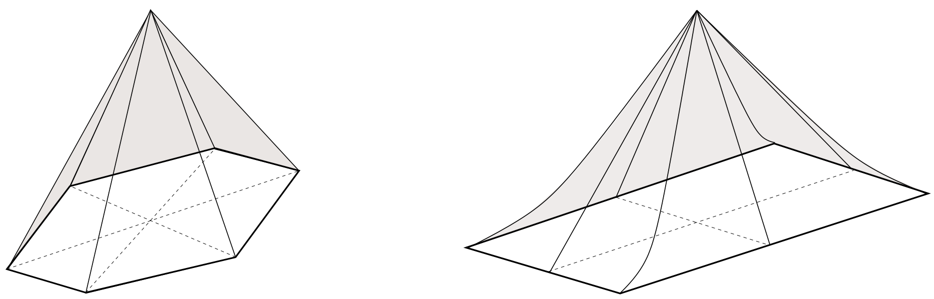

FIGURE 2. Hat Functions

The subspace $S_D \subset S$ is the spline space which is spanned by all those η_j for which (x_j, y_j) does not lie on Γ_D. Then U_D, defined as the nodal interpolant of u_D lies in S_D.

With these spaces S and S_D and their corresponding basis, the integrals in (8) can be calculated as a sum over all elements and a sum over all edges on Γ_N, i.e.,

$$A_{jk} = \sum_{T\in\mathcal{T}} \int_T \nabla\eta_j \cdot \nabla\eta_k \, dx \,, \tag{9}$$

$$b_j = \sum_{T\in\mathcal{T}} \int_T f\eta_j \, dx + \sum_{E\subset\Gamma_N} \int_E g\eta_j \, ds - \sum_{k=1}^{N} U_k \int_\Omega \nabla\eta_j \cdot \nabla\eta_k \, dx \,. \tag{10}$$

5. Assembling the Stiffness Matrix

The local stiffness matrix is determined by the coordinates of the vertices of the corresponding element and is calculated in the function `STIMA3.m` and `STIMA4.m`.

For a triangular element T let (x_1, y_1), (x_2, y_2) and (x_3, y_3) be the vertices and η_1, η_2 and η_3 the corresponding basis functions in S, i.e.

$$\eta_j(x_k, y_k) = \delta_{jk}, \quad j, k = 1, 2, 3.$$

A moments reflection reveals

$$\eta_j(x,y) = \det \begin{pmatrix} 1 & x & y \\ 1 & x_{j+1} & y_{j+1} \\ 1 & x_{j+2} & y_{j+2} \end{pmatrix} / \det \begin{pmatrix} 1 & x_j & y_j \\ 1 & x_{j+1} & y_{j+1} \\ 1 & x_{j+2} & y_{j+2} \end{pmatrix}, , \tag{11}$$

whence

$$\nabla \eta_j(x,y) = \frac{1}{2|T|} \begin{pmatrix} y_{j+1} - y_{j+2} \\ x_{j+2} - x_{j+1} \end{pmatrix}.$$

Here the indices are to be understood modulo 3, and $|T|$ is the area of T, i.e.,

$$2\,|T| = \det \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}.$$

The resulting entry of the stiffness matrix is

$$M_{jk} = \int_T \nabla \eta_j (\nabla \eta_k)^T \, dx = \frac{|T|}{(2|T|)^2} \left(y_{j+1} - y_{j+2}, \; x_{j+2} - x_{j+1} \right) \begin{pmatrix} y_{k+1} - y_{k+2} \\ x_{k+2} - x_{k+1} \end{pmatrix}$$

with indices modulo 3. This is written simultaneously for all indices as

$$M = \frac{|T|}{2} \cdot G\,G^T \quad \text{with} \quad G := \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Since we obtain similar formulae for three dimensions, the following matlab routine works simultaneously verbatim for $d = 2$ and $d = 3$.

```
function M = STIMA3(vertices)
d = size(vertices,2);
D_eta = [ones(1,d+1);vertices'] \ [zeros(1,d);eye(d)];
M = det([ones(1,d+1);vertices']) * D_eta * D_eta' / prod(1:d);
```

For a quadrilateral element T let $(x_1, y_1), \ldots, (x_4, y_4)$ denote the vertices with the corresponding hat functions $\eta_1, \ldots, \eta_4$. Since T is a parallelogram, there is an affine mapping

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{pmatrix} \begin{pmatrix} \xi \\ \zeta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix},$$

which maps $[0,1]^2$ onto T. Then $\phi_j(x,y) = \varphi_j(\Phi_T^{-1}(\xi, \zeta))$ where shape functions

$$\varphi_1(\xi,\eta) := (1-\xi)(1-\eta), \qquad \varphi_2(\xi,\eta) := \xi(1-\eta),$$
$$\varphi_3(\xi,\eta) := \xi\eta, \qquad \varphi_4(\xi,\eta) := (1-\xi)\eta.$$

From the substitution law it follows for the integrals of (9) that

$$\begin{aligned} M_{jk} &:= \int_T \nabla\varphi_j(x,y) \cdot \nabla\varphi_k(x,y) \, d(x,y) \\ &= \int_{(0,1)^2} \nabla\left(\phi_j \circ \Phi_T^{-1}\right)\left(\Phi(\xi,\eta)\right) \Big(\nabla\left(\phi_k \circ \Phi_T^{-1}\right)\left(\Phi(\xi,\eta)\right)\Big)^T \left|\det D\Phi(\xi,\eta)\right| d(\xi,\eta) \\ &= \det(A) \int_{(0,1)^2} \nabla\phi_j(\xi,\eta) \left(D\Phi(\xi,\eta)^T D\Phi(\xi,\eta)\right)^{-1} \left(\nabla\phi_k(\xi,\eta)\right)^T d(\xi,\eta). \end{aligned}$$

Solving these integrals the local stiffness matrix for a quadrilateral element results in

$$M = \frac{\det(A)}{6} \begin{pmatrix} 3b + 2(a+c) & -2a + c & -3b - (a+c) & a - 2c \\ -2a + c & -3b + 2(a+c) & a - 2c & 3b - (a+c) \\ -3b - (a+c) & a - 2c & 3b + 2(a+c) & -2a + c \\ a - 2c & 3b - (a+c) & -2a + c & -3b + 2(a+c) \end{pmatrix}.$$

```
function M = STIMA4(vertices)
D_Phi = [vertices(2,:)-vertices(1,:); vertices(4,:)-vertices(1,:)]';
B = inv(D_Phi'*D_Phi);
C1 = [2,-2;-2,2]*B(1,1)+[3,0;0,-3]*B(1,2)+[2,1;1,2]*B(2,2);
C2 = [-1,1;1,-1]*B(1,1)+[-3,0;0,3]*B(1,2)+[-1,-2;-2,-1]*B(2,2);
M = det(D_Phi) * [C1 C2; C2 C1] / 6;
```

6. Assembling the right-hand side

The volume forces are used for assembling the right hand side. Using the value of f in the centre of gravity (x_S, y_S) of T the integral $\int_T f\eta_j\,dx$ in (10) is approximated by

$$\int_T f\eta_j\,dx \approx 1/k_T \,\det\begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} f(x_S, y_S),$$

where $k_T = 6$ if T is a triangle and $k_T = 4$ if T is a parallelogram.

```
% Volume Forces
for j = 1:size(Elements3,1)
  b(Elements3(j,:)) = b(Elements3(j,:)) + ...
        det([1 1 1; Coordinates(Elements3(j,:),:)']) * ...
        f(sum(Coordinates(Elements3(j,:),:))/3)/6;
end
for j = 1:size(Elements4,1)
   b(Elements4(j,:)) = b(Elements4(j,:)) + ...
        det([1 1 1; Coordinates(Elements4(j,1:3),:)']) * ...
        f(sum(Coordinates(Elements4(j,:),:))/4)/4;
end
```

The values of f are given by the function `f.m` which depends on the problem. The function is called with the coordinates of points in Ω and it returns the volume forces at these locations.

```
function VolumeForce = f(u);
VolumeForce = ones(size(u,1),1);
```

Likewise, the Neumann conditions contribute to the right hand side. The integral $\int_E g\eta_j\,ds$ in (10) is approximated using the value of g in the centre (x_M, y_M) of E with length $|E|$by

$$\int_E g\eta_j\,ds \approx \frac{|E|}{2} g(x_M, y_M)\,.$$

```
% Neumann conditions
if ~isempty(Neumann)
  for j = 1 : size(Neumann,1)
   b(Neumann(j,:)) = b(Neumann(j,:)) + norm(Coordinates(Neumann(j,1),:)- ...
   Coordinates(Neumann(j,2),:)) * g(sum(Coordinates(Neumann(j,:),:))/2)/2;
  end
end
```

It is used here that in Matlab the size of an empty matrix is set equal to zero and that a loop of 1 through 0 is totally omitted. In that way, the question of the existence of Neumann boundary data is to be renounced.

The values of g are given by the function `g.m` which again depends on the problem. The function is called with the coordinates of points on Γ_N and returns the coresseponding stresses.

```
function Stress = g(u)
Stress = zeros(size(u,1),1);
```

7. Incooperating Dirichlet conditions

With a suitable numbering of the nodes the system of linear equations resulting from the construction described in the previous section without incooperating Dirichlet conditions can be written as follows

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} \cdot \begin{pmatrix} \vec{U} \\ \vec{U}_D \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{b}_D \end{pmatrix}, \tag{12}$$

with $\vec{U} \in \mathbb{R}^M$, $\vec{U}_D \in \mathbb{R}^{N-M}$. Here $\vec{U}$ are the values at the free nodes which are to be determined, $\vec{U}_D$ are the values at the nodes which are on the Dirichlet boundary and thus are known a priori. Hence, the first block of equations can be rewritten as

$$A_{11} \cdot \vec{U} = \vec{b} - A_{12} \cdot \vec{U}_D.$$

In fact, this is the formulation of (6) with $U_D = 0$ at non-Dirichlet nodes.

In the second block of equations in (12) the unknown is $\vec{b}_D$ but since it is not of our interest it is omitted in the following.

```
% Dirichlet conditions
u = sparse(size(Coordinates,1),1);
u(unique(Dirichlet)) = u_D(Coordinates(unique(Dirichlet),:));
b = b - A * u;
```

The values u_D at the nodes on Γ_D are given by the function `u_D.m` which depends on the problem. The function is called with the coordinates of points in Γ_D and returns the values at the corresponding locations.

```
function DirichletBoundaryValue = u_D(u)
DirichletBoundaryValue =  zeros(size(u,1),1);
```

8. Computation and Displaying the numerical solution

The rows of (7) corresponding to the rows of (12) form a reduced system of equations with a symmetric, positive definite coefficient matrix A_{11}. It is obtained from the original system of equations by taking the rows and columns corresponding to the free nodes of the problem. The restriction can be achieved in Matlab through proper indexing.

The system of equations is solved by the binary operator \ installed in Matlab which gives the left inverse of a matrix.

```
u(FreeNodes)=A(FreeNodes,FreeNodes)\b(FreeNodes);
```

Matlab makes use of the properties of a symmetric, positive definit and sparse matrix for solving the system of equations efficiently.

The graphical representation of the solution is given by the function `SHOW.m`.

```
function SHOW(Elements3,Elements4,Coordinates,u)
hold off
trisurf(Elements3,Coordinates(:,1),Coordinates(:,2),u')
hold on
trisurf(Elements4,Coordinates(:,1),Coordinates(:,2),u')
view(-67.5,30);
title('Solution of the Problem')
```

Here, the Matlab-routine `trisurf(ELEMENTS,X,Y,U)` is used to draw triangulations for equal types of elements. Every row of the matrix `ELEMENTS` determines one polygon where the x-, y-, and z-coordinate of each corner of this polygon is given by the corresponding entry in `X`, `Y` and `U`. The colour of the polygons is given by values of `U`.

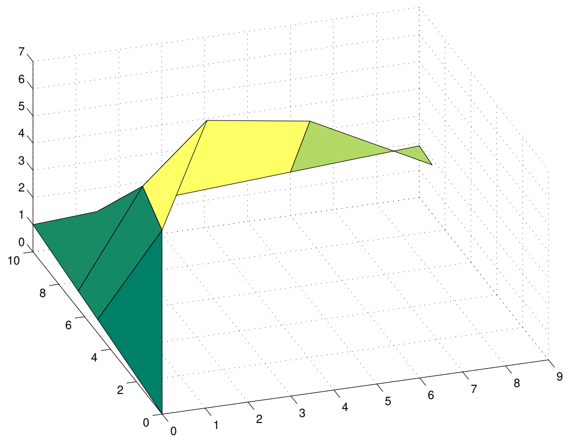

Figure 3. Solution for the Examples listed above

9. The Heat equation

For numerical simulations of the heat equation,

$$\partial u/\partial t = \Delta u + f \text{ in } \Omega \times [0, T],$$

with an implicit Euler scheme in time, we split the time interval $[0, T]$ into N equally size subintervals of size $dt = T/N$ which leads to the equation

$$(\mathrm{id} - dt\,\Delta)u_n = dt\, f_n + u_{n-1}\,, \tag{13}$$

where $f_n = f(x, t_n)$ and u_n is the time discrete approximation of u at time $t_n = n\,dt$. The weak form of (13) is

$$\int_\Omega u_n v\, dx + dt \int_\Omega \nabla u_n \cdot \nabla v\, dx = dt \left(\int_\Omega f_n v\, dx + \int_{\Gamma_N} g_n v\, dx \right) + \int_\Omega u_{n-1} v\, dx$$

with $g_n = g(x, t_n)$ and notation as in Section 2. For each time step, this equation is solved using finite elements which leads to the linear system

$$(dt\, A + B)U_n = b + BU_{n-1}\,.$$

The stiffness matrix A and right-hand side b are as before (see (8)). The mass matrix B results from the terms $\int_\Omega u_j v\, dx$, i.e.

$$B_{jk} = \sum_{T \in \mathcal{T}} \int_T \eta_j \eta_k\, dx.$$

For piecewise affine elements we obtain

$$\int_T \eta_j \eta_k\, dx = \frac{1}{24} \det \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

Appendix B shows the modified code for the heat equation. The numerical example was again based on the domain in Fig. 1, this time with $f \equiv 0$ and $u_D = 1$ on the outer boundary. The value on the (inner) circle is still $u_D = 0$. Fig. 4 displays the solution the given code produced for four different times $t = 0.1$, 0.2, 0.5 and $t = 1$.

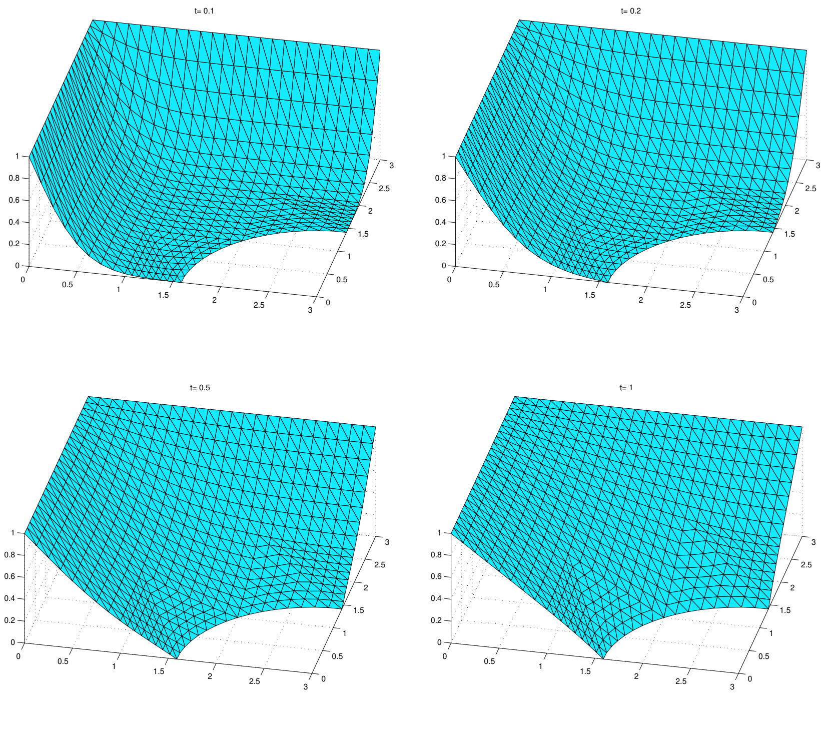

Figure 4. Solution for the heat equation

10. A non-linear problem

As a simple application to non-convex variational problems, we consider the Ginzburg-Landau equation

$$\varepsilon \Delta u = u^3 - u \qquad \text{in } \Omega, \qquad u = 0 \qquad \text{on } \Gamma \tag{14}$$

for $\varepsilon = 1/100$. Its weak formulation, i.e.

$$F(u,v) := \int_\Omega \varepsilon \nabla u \cdot \nabla v \, dx - \int_\Omega (u - u^3) v \, dx = 0 \qquad (v \in H_0^1(\Omega)), \tag{15}$$

can also be regarded as the necessary condition for the minimizer in the variational problem

$$\min \quad \int_\Omega \left[\frac{\varepsilon}{2} |\nabla u|^2 + \frac{1}{4}(u^2 - 1)^2 \right] dx! \tag{16}$$

We aim to solve (15) with Newton-Raphson's method. Starting with some u^0, in each iteration step we compute $u^n - u^{n+1} \in H_0^1(\Omega)$ satisfying

$$DF(u^n, v; u^n - u^{n+1}) = F(u^n, v) \qquad (v \in H_0^1(\Omega)), \tag{17}$$

where

$$DF(u, v; w) = \int_\Omega \varepsilon \nabla v \cdot \nabla w \, dx - \int_\Omega (vw - 3vu^2 w) \, dx. \tag{18}$$

The integrals in $F(U,V)$ and $DF(U,V;W)$ can be calculated analytically and the actual Matlab code again only needs little modifications, shown in Appendix C. Essentially, one has to initialize the code (with a random start vector that fullfills the Dirichlet boundary condition (lines 7 and 8)), to add a loop (lines 9 and 48), to update the new Newton approximation (line 44), and to supply a stopping criteria in case of convergence (lines 45–47). Lines 20–24 represent (15) in the discrete space.

It is known that the solutions are not unique. Indeed, for any local minimizer u, $-u$ is also a minimizer and 0 solves the problem as well. The constant function $u = \pm 1$ leads to zero energy, but violates the continuity or the boundary conditions. Hence, boundary or internal layers are observed which seperate large regions where u is almost constant ± 1.

In the finite dimensional problem, different initial values u^0 may lead to different numerical approximations. Fig. 5 displays two possible solutions found for two different starting values after about 20-30 iteration steps.

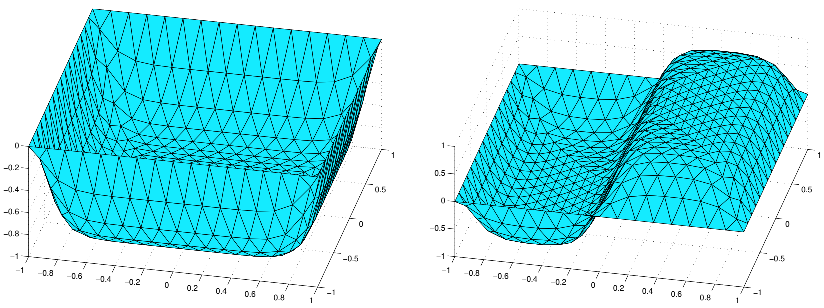

Figure 5. Solutions for the non-linear equation

11. Three-dimensional problems

With a few modifications the matlab code for linear two-dimensional problems discussed in Sections 5–8 can be extented to three-dimensional problems. Tetraeder are used as finite elements. The basis functions are defined corresponding to two dimensions, e.g., for a tetraeder element T let (x_j, y_j, z_j) $(j = 1, \ldots, 4)$ be the vertices and η_j the corresponding basis functions, i.e.

$$\eta_j(x_k, y_k, z_k) = \delta_{jk} \quad j, k = 1, \ldots, 4\,.$$

Each of the `*.dat` files get an additional entry per row. In `Coordinates.dat` it is the z-component of each node $P_j = (x_j, y_j, z_j)$. A typical entry in `Coordinates.dat` reads now

j k ℓ m n.

where k, ℓ, m, n are the numbers of vertices $P_k, \ldots, P_n$ of the jth element. The sequence of nodes is organized such that the right-hand side of

$$6|T| = \det \begin{pmatrix} 1 & 1 & 1 & 1 \\ x_k & x_\ell & x_m & x_n \\ y_k & y_\ell & y_m & y_n \\ z_k & z_\ell & z_m & z_n \end{pmatrix}$$

is positive. The numbering of surface elements defined in `Neumann.dat` and `Dirichlet.dat` is done with mathematical positive orientation viewing from outside Ω onto the surface.

Using the matlab code in Appendix A, cancelation of lines 5 and 16–19 and substituting 22–24, 34–35, 45 by the following lines gives a short and flexibel tool for solving scalar, linear three-dimensional problems.

```
22*  b(Elements3(j,:)) = b(Elements3(j,:)) + ...
  *     det([1,1,1,1;Coordinates(Elements3(j,:),:)']) * ...
24*     f(sum(Coordinates(Elements3(j,:),:))/4) / 24;

34*  b(Neumann(j,:)) = b(Neumann(j,:)) + ...
  *     norm(cross(Coordinates(Neumann(j,3),:)-Coordinates(Neumann(j,1),:), ...
  *                 Coordinates(Neumann(j,2),:)-Coordinates(Neumann(j,1),:))) ...
35*                * g(sum(Koordinates(Neumann(j,:),:))/3)/6;

45*  ShowSurface([Dirichlet;Neumann],Coordinates,full(u));
```

The graphical representation for three-dimensional problems can be done by a shortened version of `SHOW` in Section 8

```
function ShowSurface(Surface,Coordinates,u)
trisurf(Surface,Coordinates(:,1),Coordinates(:,2),Coordinates(:,3),u')
view(130,90)
```

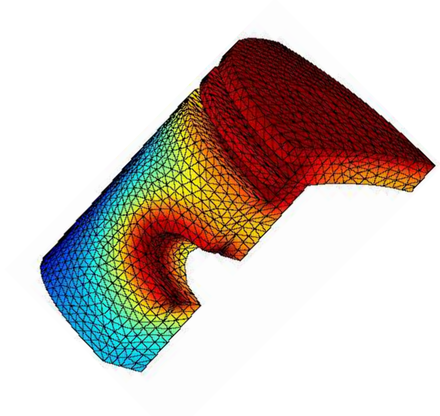

FIGURE 6. Temperature distribution of a piston

The temperature distribution of a simplified piston is presented in Fig. 6. Calculation of the temperature distribution with 3728 nodes and 15111 elements (including the graphical output) takes a few minutes on a workstation.

Appendix A. The complete Matlab code

```
1 % 2_D-FEM for Laplace-operator
2 % Initialisation
3  load Coordinates.dat; Coordinates(:,1)=[];
4  load Elements3.dat; Elements3(:,1)=[];
5  load Elements4.dat; Elements4(:,1)=[];
6  eval('load Neumann.dat; Neumann(:,1) = [];','Neumann=[];');
7  load Dirichlet.dat; Dirichlet(:,1) = [];
8  FreeNodes=setdiff(1:size(Coordinates,1),unique(Dirichlet));
9  A = sparse(size(Coordinates,1),size(Coordinates,1));
```

```
 b = sparse(size(Coordinates,1),1);
% Assembly
 for j = 1:size(Elements3,1)
   A(Elements3(j,:),Elements3(j,:)) = A(Elements3(j,:),Elements3(j,:)) ...
     + STIMA3(Coordinates(Elements3(j,:),:));
 end
 for j = 1:size(Elements4,1)
   A(Elements4(j,:),Elements4(j,:)) = A(Elements4(j,:),Elements4(j,:)) ...
     + STIMA4(Coordinates(Elements4(j,:),:));
 end
% Volume Forces
 for j = 1:size(Elements3,1)
   b(Elements3(j,:)) = b(Elements3(j,:)) + ...
         det([1,1,1; Coordinates(Elements3(j,:),:)']) * ...
         f(sum(Coordinates(Elements3(j,:),:))/3)/6;
 end
 for j = 1:size(Elements4,1)
   b(Elements4(j,:)) = b(Elements4(j,:)) + ...
         det([1,1,1; Coordinates(Elements4(j,1:3),:)']) * ...
         f(sum(Coordinates(Elements4(j,:),:))/4)/4;
 end
% Neumann conditions
 if ~isempty(Neumann)
   for j = 1 : size(Neumann,1)
     b(Neumann(j,:))=b(Neumann(j,:)) + norm(Coordinates(Neumann(j,1),:)- ...
     Coordinates(Neumann(j,2),:)) * g(sum(Coordinates(Neumann(j,:),:))/2)/2;
   end
 end
% Dirichlet conditions
 u = sparse(size(Coordinates,1),1);
 u(unique(Dirichlet)) = u_0(Coordinates(unique(Dirichlet),:));
 b = b - A * u;
% Computation of the solution
 u(FreeNodes) = A(FreeNodes,FreeNodes) \ b(FreeNodes);
% graphic representation
 SHOW(Elements3,Elements4,Coordinates,full(u));
```

Appendix B. Matlab code for the heat equation

```
% 2_D-FEM for Heat-Equation
% Initialisation
 load Coordinates.dat; Coordinates(:,1)=[];
 load Elements3.dat; Elements3(:,1)=[];
 eval('load Neumann.dat; Neumann(:,1) = [];','Neumann=[];');
 load Dirichlet.dat; Dirichlet(:,1) = [];
 FreeNodes=setdiff(1:size(Coordinates,1),unique(Dirichlet));
 A = sparse(size(Coordinates,1),size(Coordinates,1));
 B = sparse(size(Coordinates,1),size(Coordinates,1));
 T = 1; dt = 0.01; N = T/dt;
 U = zeros(size(Coordinates,1),N+1);
% Assembly
 for j = 1:size(Elements3,1)
   A(Elements3(j,:),Elements3(j,:)) = A(Elements3(j,:),Elements3(j,:)) ...
     + STIMA3(Coordinates(Elements3(j,:),:));
 end
 for j = 1:size(Elements3,1)
   B(Elements3(j,:),Elements3(j,:)) = B(Elements3(j,:),Elements3(j,:)) ...
 + det([1,1,1;Coordinates(Elements3(j,:),:)'])*[2,1,1,;1,2,1;1,1,2]/24;
 end
```

```
% Initial Condition
 U(:,1) = zeros(size(Coordinates,1),1);
% time steps
 for n = 2:N+1
   b = sparse(size(Coordinates,1),1);
% Volume Forces
   for j = 1:size(Elements3,1)
     b(Elements3(j,:)) = b(Elements3(j,:)) + ...
     det([1,1,1; Coordinates(Elements3(j,:),:)']) * ...
     dt*f(sum(Coordinates(Elements3(j,:),:))/3,n*dt)/6;
   end
  % Neumann conditions
   if ~isempty(Neumann)
     for j = 1 : size(Neumann,1)
 b(Neumann(j,:)) = b(Neumann(j,:)) + ...
    norm(Coordinates(Neumann(j,1),:)-Coordinates(Neumann(j,2),:))* ...
        dt*g(sum(Coordinates(Neumann(j,:),:))/2,n*dt)/2;
     end
   end
% previous timestep
   b = b + B * U(:,n-1);
% Dirichlet conditions
   u = sparse(size(Coordinates,1),1);
   u(unique(Dirichlet)) = u_D(Coordinates(unique(Dirichlet),:),n*dt);
   b = b - (dt * A + B) * u;
% Computation of the solution
   u(FreeNodes) = (dt*A(FreeNodes,FreeNodes)+ ...
                 B(FreeNodes,FreeNodes))\b(FreeNodes);
   U(:,n) = u;
 end
% graphic representation
 SHOW(Elements3,[],Coordinates,full(U(:,N+1)));
```

Appendix C. Matlab code for the non-linear problem

```
% Initialisation
 load Coordinates.dat; Coordinates(:,1)=[];
 load Elements3.dat; Elements3(:,1)=[];
 eval('load Neumann.dat; Neumann(:,1) = [];','Neumann=[];');
 load Dirichlet.dat; Dirichlet(:,1) = [];
 FreeNodes=setdiff(1:size(Coordinates,1),unique(Dirichlet));
 U = rand(size(Coordinates,1),1);
 U(unique(Dirichlet)) = u_D(Coordinates(unique(Dirichlet),:));
 for i=1:50
  % Assembly of DF(u^n)
   A = sparse(size(Coordinates,1),size(Coordinates,1));
   for j = 1:size(Elements3,1)
     A(Elements3(j,:),Elements3(j,:)) = A(Elements3(j,:),Elements3(j,:)) ...
    + localDF(Coordinates(Elements3(j,:),:),U(Elements3(j,:)));
   end
  % Assembly of F(U^n)
   b = sparse(size(Coordinates,1),1);
   for j = 1:size(Elements3,1)
     b(Elements3(j,:)) = b(Elements3(j,:)) ...
   + localF(Coordinates(Elements3(j,:),:),U(Elements3(j,:)));
   end
  % Volume Forces
   for j = 1:size(Elements3,1)
    b(Elements3(j,:)) = b(Elements3(j,:)) - ...
```

```
    det([1 1 1; Coordinates(Elements3(j,:),:)']) * ...
    f(sum(Coordinates(Elements3(j,:),:))/3)/6;
   end
  % Neumann conditions
   if ~isempty(Neumann)
     for j = 1 : size(Neumann,1)
       b(Neumann(j,:))=b(Neumann(j,:)) - norm(Coordinates(Neumann(j,1),:)- ...
       Coordinates(Neumann(j,2),:))*g(sum(Coordinates(Neumann(j,:),:))/2)/2;
     end
   end
  % Dirichlet conditions
   V = zeros(size(Coordinates,1),1);
   V(unique(Dirichlet)) = 0;

  % Solving one Newton step
   V(FreeNodes) = A(FreeNodes,FreeNodes)\b(FreeNodes);
   U = U - V;
   if norm(V) < 10^(-10)
     break
   end
 end
% graphic representation
 SHOW(Elements3,[],Coordinates,full(U));
```

```
function b = localF(vertices,U)
Eps = 1/100;
D_eta = [ones(1,3);vertices'] \ [zeros(1,2);eye(2)];
Area = det([ones(1,3);vertices']) / 2;
b=Area*((Eps*D_eta*D_eta'-[2,1,1;1,2,1;1,1,2]/12)*U+ ...
    [4*U(1)^3+ U(2)^3+U(3)^3+3*U(1)^2*(U(2)+U(3))+2*U(1) ...
     *(U(2)^2+U(3)^2)+U(2)*U(3)*(U(2)+U(3))+2*U(1)*U(2)*U(3);
     4*U(2)^3+ U(1)^3+U(3)^3+3*U(2)^2*(U(1)+U(3))+2*U(2) ...
     *(U(1)^2+U(3)^2)+U(1)*U(3)*(U(1)+U(3))+2*U(1)*U(2)*U(3);
     4*U(3)^3+ U(2)^3+U(1)^3+3*U(3)^2*(U(2)+U(1))+2*U(3) ...
     *(U(2)^2+U(1)^2)+U(2)*U(1)*(U(2)+U(1))+2*U(1)*U(2)*U(3)]/60);
```

```
function M = localDF(vertices,U)
Eps = 1/100;
D_eta = [ones(1,3);vertices'] \ [zeros(1,2);eye(2)];
Area = det([ones(1,3);vertices']) / 2;
M = Area*(Eps*D_eta*D_eta'-[2,1,1;1,2,1;1,1,2]/12 + ...
    [12*U(1)^2+2*(U(2)^2+U(3)^2+U(2)*U(3))+6*U(1)*(U(2)+U(3)),...
     3*(U(1)^2+U(2)^2)+U(3)^2+4*U(1)*U(2)+2*U(3)*(U(1)+U(2)),...
     3*(U(1)^2+U(3)^2)+U(2)^2+4*U(1)*U(3)+2*U(2)*(U(1)+U(3));
     3*(U(1)^2+U(2)^2)+U(3)^2+4*U(1)*U(2)+2*U(3)*(U(1)+U(2)),...
     12*U(2)^2+2*(U(1)^2+U(3)^2+U(1)*U(3))+6*U(2)*(U(1)+U(3)),...
     3*(U(2)^2+U(3)^2)+U(1)^2+4*U(2)*U(3)+2*U(1)*(U(2)+U(3));
     3*(U(1)^2+U(3)^2)+U(2)^2+4*U(1)*U(3)+2*U(2)*(U(1)+U(3)),...
     3*(U(2)^2+U(3)^2)+U(1)^2+4*U(2)*U(3)+2*U(1)*(U(2)+U(3)),...
     12*U(3)^2+2*(U(1)^2+U(2)^2+U(1)*U(2))+6*U(3)*(U(1)+U(2))]/60);
```

References

[BS] S.C. Brenner, L.R. Scott: *The mathematical theory of finite element methods.* Texts in Applied Mathematics **15** Springer New York 1994.

[Ci] P.G. Ciarlet: *The Finite Element Method for Elliptic Problems.* North-Holland 1978.

[M] L. Langemyr et. al.: *Partial Differential Equation Toolbox User's Guide.* The Math Works, Inc. 1995.

[S] H.R. Schwarz: *Methode der Finiten Elemente.* Teubner 1991.

Mathematisches Seminar, Christian-Albrechts-Universität zu Kiel Ludewig-Meyn-Str. 4, D-24098 Kiel, FRG.
E-mail address: {ja,cc,saf}@numerik.uni-kiel.de