Remarks around 50 lines of Matlab: short finite element implementation

Jochen Alberty, Carsten Carstensen and Stefan A. Funken

Mathematisches Seminar, Christian-Albrechts-Universität zu Kiel, Ludewig-Meyn-Str. 4, D-24098 Kiel, Germany

E-mail: {ja;cc;saf}@numerik.uni-kiel.de

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 program and the 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.

Keywords: Matlab program

AMS subject classification: 68N15, 65N30, 65M60

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 or five 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 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 [3], our approach is shorter, allows more elements, is easily adapted 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 organised as follows. As a model problem, the Laplace equation is described in section 2. The discretisation 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 procedures of the stiffness matrix in section 5 and of 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 appendix A.) The applications follow in sections 9–11 and illustrate the tool in a time-dependent heat equation, in a nonlinear and even in a 3-dimensional example.

Note that the given programs are written for Matlab 5. Though, in principle, it is possible to modify the package to run the program under Matlab 4, the changes are too many to state them here. The package in netlib provides both versions, for Unix-like and Windows machines.

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 $\Gamma_\mathrm{N} := \Gamma \setminus \Gamma_\mathrm{D}$. Given $f \in L^2(\Omega)$, $u_\mathrm{D} \in H^1(\Omega)$, and $g \in L^2(\Gamma_\mathrm{N})$, seek $u \in H^1(\Omega)$ with

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

$$u = u_\mathrm{D} \qquad \text{on } \Gamma_\mathrm{D}, \tag{2}$$

$$\frac{\partial u}{\partial n} = g \qquad \text{on } \Gamma_\mathrm{N}. \tag{3}$$

According to the Lax–Milgram lemma, there always exists a weak solution to (1)–(3) which enjoys inner regularity (i.e., $u \in H^2_\mathrm{loc}(\Omega)$), 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 $v = u - u_\mathrm{D}$ so that $v = 0$ on Γ_D, i.e.,

$$v \in H^1_\mathrm{D}(\Omega) := \bigl\{ w \in H^1(\Omega) \mid w = 0 \text{ on } \Gamma_\mathrm{D} \bigr\}.$$

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

$$\int_\Omega \nabla v \cdot \nabla w \,\mathrm{d}x = \int_\Omega f w \,\mathrm{d}x + \int_{\Gamma_\mathrm{N}} g w \,\mathrm{d}s - \int_\Omega \nabla u_D \cdot \nabla w \,\mathrm{d}x, \quad w \in H^1_\mathrm{D}(\Omega). \tag{4}$$

3. Galerkin discretisation of the problem

For the implementation, problem (4) is discretised using the standard Galerkin method, where $H^1(\Omega)$ and $H^1_\mathrm{D}(\Omega)$ are replaced by finite dimensional subspaces S and $S_\mathrm{D} = S \cap H^1_\mathrm{D}$, respectively. Let $U_\mathrm{D} \in S$ 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 discretised problem (P_S) reads: Find $V \in S_{\mathrm{D}}$ such that

$$\int_\Omega \nabla V \cdot \nabla W \,\mathrm{d}x = \int_\Omega fW \,\mathrm{d}x + \int_{\Gamma_{\mathrm{N}}} gW \,\mathrm{d}s - \int_\Omega \nabla U_{\mathrm{D}} \cdot \nabla W \,\mathrm{d}x, \quad W \in S_{\mathrm{D}}. \tag{5}$$

Let $(\eta_1, \dots, \eta_{\mathrm{N}})$ 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 $M \leqslant N - 2$. Then, (5) is equivalent to

$$\int_\Omega \nabla V \cdot \nabla \eta_j \,\mathrm{d}x = \int_\Omega f\eta_j \,\mathrm{d}x + \int_{\Gamma_{\mathrm{N}}} g\eta_j \,\mathrm{d}s - \int_\Omega \nabla U_{\mathrm{D}} \cdot \nabla \eta_j \,\mathrm{d}x, \quad j \in I. \tag{6}$$

Furthermore, let

$$V = \sum_{k\in I} x_k \eta_k \quad \text{and} \quad U_{\mathrm{D}} = \sum_{k=1}^{N} U_k \eta_k.$$

Then, 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

$$A_{jk} = \int_\Omega \nabla\eta_j \cdot \nabla\eta_k \,\mathrm{d}x \quad \text{and} \quad b_j = \int_\Omega f\eta_j \,\mathrm{d}x + \int_{\Gamma_{\mathrm{N}}} g\eta_j \,\mathrm{d}s - \sum_{k=1}^{N} U_k \int_\Omega \nabla\eta_j \cdot \nabla\eta_k \,\mathrm{d}x. \tag{8}$$

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

$$U = U_{\mathrm{D}} + V = \sum_{j=1}^{N} U_j \eta_j + \sum_{k\in I} x_k \eta_k.$$

4. Data-representation of the triangulation Ω

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

Regular triangulation in the sense of Ciarlet [2] 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 $\overline{\Gamma}_{\mathrm{N}}$ or to $\overline{\Gamma}_{\mathrm{D}}$.

Matlab supports reading data from files given in ascii format by .dat files. Figure 1 shows the mesh which is described by the following data. The file 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

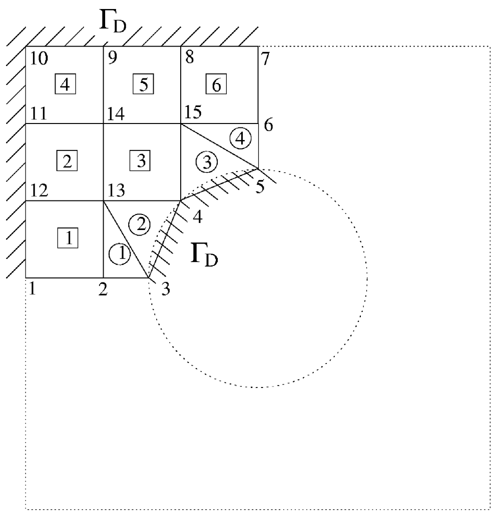

Figure 1. Example of a mesh.

`coordinates.dat` contains the coordinates of each node in the given mesh. Each row has the form

node # x-coordinate y-coordinate.

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 # node1 node2 node3.

Similarly, the data for the quadrilaterals are given in `elements4.dat`. Here, we use the format

element # 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 # node1 node2 resp. Dirichlet edge # node1 node2.

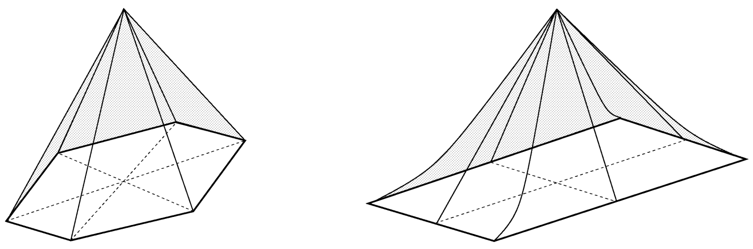

Figure 2. Hat functions.

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 chosen globally continuous and affine on each triangular element and bilinear isoparametric on each quadrilateral element. In figure 2 we display two typical hat functions η_j which are defined for every node (x_j, y_j) of the mesh by

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

The subspace $S_{\mathrm{D}} \subset S$ 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.

With these spaces S and S_{D} and their corresponding bases, the integrals in (8) can be calculated as a sum over all elements and a sum over all edges on Γ_{N}, i.e.,

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

$$b_j = \sum_{T\in\mathcal{T}} \int_T f\eta_j \,\mathrm{d}x + \sum_{E\subset\Gamma_{\mathrm{N}}} \int_E g\eta_j \,\mathrm{d}s - \sum_{k=1}^{N} U_k \sum_{T\in\mathcal{T}} \int_T \nabla\eta_j \cdot \nabla\eta_k \,\mathrm{d}x. \tag{10}$$

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 functions `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 moment's reflection reveals

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

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

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

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

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

Since we obtain similar formulae for three dimensions (cf. section 11), the following Matlab routine works simultaneously for $d = 2$ and $d = 3$:

```
function M = stima3(vertices)
d = size(vertices,2);
G = [ones(1,d+1);vertices'] \ [zeros(1,d);eye(d)];
M = det([ones(1,d+1);vertices']) * G * G' / 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

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

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

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

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

$$\begin{aligned} M_{jk} &:= \int_T \nabla\eta_j(x,y)\cdot\nabla\eta_k(x,y)\,\mathrm{d}(x,y) \\ &= \int_{(0,1)^2} \nabla\big(\varphi_j\circ\Phi_T^{-1}\big)\big(\Phi_T(\xi,\zeta)\big)\big(\nabla\big(\varphi_k\circ\Phi_T^{-1}\big)\big(\Phi_T(\xi,\zeta)\big)\big)^{\mathrm{T}}|\det D\Phi_T|\,\mathrm{d}(\xi,\zeta) \\ &= \det(D\Phi_T)\int_{(0,1)^2}\nabla\varphi_j(\xi,\zeta)\big((D\Phi_T)^{\mathrm{T}}D\Phi_T\big)^{-1}\big(\nabla\varphi_k(\xi,\zeta)\big)^{\mathrm{T}}\,\mathrm{d}(\xi,\zeta). \end{aligned}$$

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

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

where

$$\big((D\Phi_T)^{\mathrm{T}}D\Phi_T\big)^{-1} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

```
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 $\int_T f\eta_j\,\mathrm{d}x$ in (10) is approximated by

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

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. For the numerical example shown in figure 3 we used

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

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

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

```
% Neumann conditions
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
```

We here use the fact 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 corresponding stresses. For the numerical example `g.m` was

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

7. Incorporating Dirichlet conditions

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

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{12}^{\mathrm{T}} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} U \\ U_{\mathrm{D}} \end{pmatrix} = \begin{pmatrix} b \\ b_{\mathrm{D}} \end{pmatrix}, \tag{12}$$

with $U \in \mathbb{R}^M$, $U_\mathrm{D} \in \mathbb{R}^{N-M}$. Here, U are the values at the free nodes which are to be determined, 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

$$A_{11} \cdot U = b - A_{12} \cdot U_\mathrm{D}.$$

In fact, this is the formulation of (6) with $U_\mathrm{D} = 0$ at non-Dirichlet nodes.

In the second block of equations in (12) the unknown is b_D but since it is not of interest to us 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. For the numerical example `u_d.m` was

```
function DirichletBoundaryValue = u_d(x)
DirichletBoundaryValue =  zeros(size(x,1),1);
```

8. Computation and displaying the numerical solution

The rows of (7) corresponding to the first M 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 $\backslash$ installed in Matlab which gives the left inverse of a matrix.

```
FreeNodes=setdiff(1:size(coordinates,1),unique(dirichlet));
u(FreeNodes)=A(FreeNodes,FreeNodes)\b(FreeNodes);
```

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

A graphical representation of the solution is given by the function `show.m`.

```
function show(elements3,elements4,coordinates,u)
trisurf(elements3,coordinates(:,1),coordinates(:,2),u',...
        'facecolor','interp')
hold on
trisurf(elements4,coordinates(:,1),coordinates(:,2),u',...
        'facecolor','interp')
```

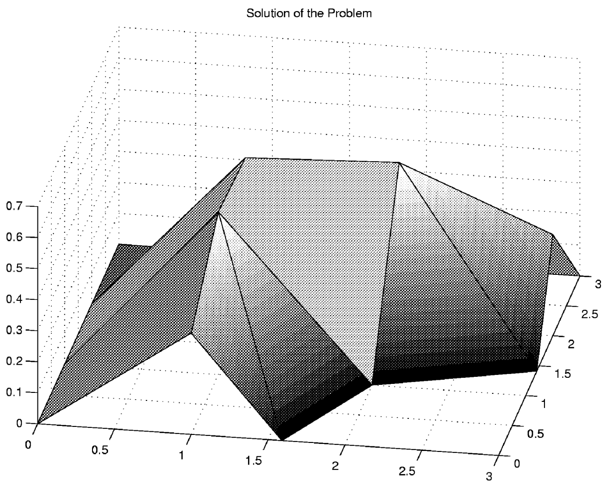

Figure 3. Solution for the Laplace problem.

```
hold off
view(10,40);
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`, respectively. The colour of the polygons is given by values of `U`. The additional parameters `,'facecolor','interp',` lead to an interpolated colouring. Figure 3 shows the solution for the mesh defined in section 4 and the data files `f.m`, `g.m`, and `u_d.m` given in sections 6 and 7.

Summarising sections 4–8, the main program, which is listed in appendix A, is structured as follows (the line references are according to the numbering in appendix A):

- Lines 3–10: Loading of the mesh geometry and initialisation.
- Lines 11–19: Assembly of the stiffness matrix in two loops, first over the triangular elements, then over the quadrilaterals.
- Lines 20–30: Incorporating the volume force in two loops, first over the triangular elements, then over the quadrilaterals.
- Lines 31–35: Incorporating the Neumann condition.
- Lines 36–39: Incorporating the Dirichlet condition.
- Lines 40–41: Solving the reduced linear system.
- Lines 42–43: Graphical representation of the numerical solution.

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 sized 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

$$\int_\Omega u_n v \,\mathrm{d}x + dt \int_\Omega \nabla u_n \cdot \nabla v \,\mathrm{d}x = dt\left(\int_\Omega f_n v \,\mathrm{d}x + \int_{\Gamma_\mathrm{N}} g_n v \,\mathrm{d}x\right) + \int_\Omega u_{n-1} v \,\mathrm{d}x$$

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

$$(dtA + B)U_n = dtb + BU_{n-1}.$$

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

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

For triangular, piecewise affine elements we obtain

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

Appendix B shows the modified code for the heat equation. The numerical example was again based on the domain in figure 1, this time with $f \equiv 0$ and $u_\mathrm{D} = 1$ on the outer boundary. The value on the (inner) circle is still $u_\mathrm{D} = 0$. On the Neumann boundary we still have $g \equiv 0$. Figure 4 displays the solution the given code produced for four different times $T = 0.1$, 0.2, 0.5 and $T = 1$. (Variable `T` in line 10 of the main program.)

The main program, listed in appendix B, is structured as follows (the line references are according to the numbering in appendix B):

- Lines 3–11: Loading of the mesh geometry and initialisation.
- Lines 12–16: Assembly of the stiffness matrix A in a loop over all triangular elements.
- Lines 17–20: Assembly of the mass matrix B in a loop over all triangular elements.
- Lines 21–22: Defining initial condition of the discrete `U`.

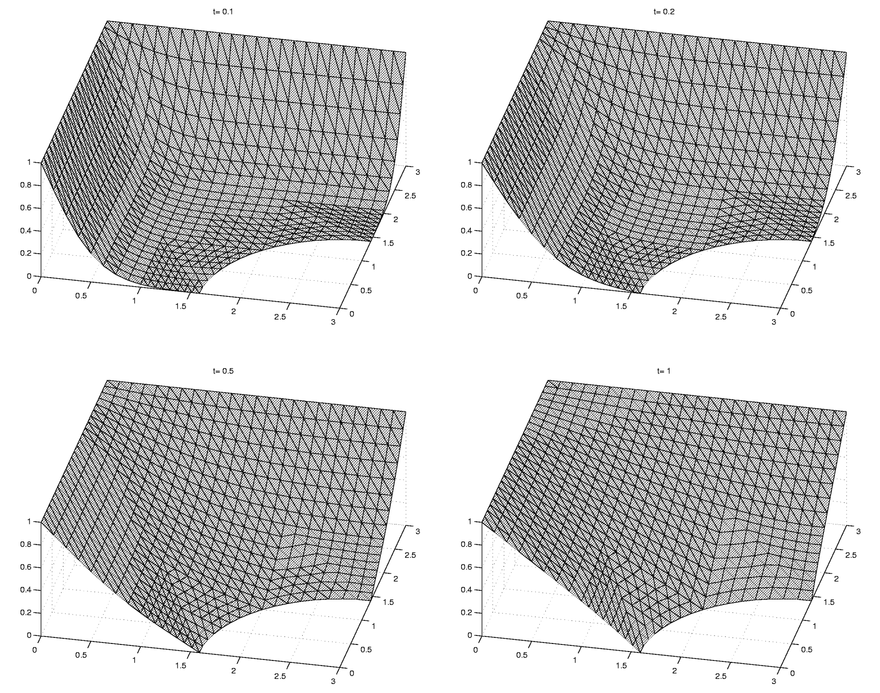

Figure 4. Solution for the heat equation.

- Lines 23–48: Loop over the time-steps. In particular:
- Line 25: Clearing the vector of the right-hand side.
- Lines 26–31: Incorporating the volume force at time step n.
- Lines 32–37: Incorporating the Neumann condition at time step n.
- Lines 38–39: Incorporating the solution at the previous time step $n-1$.
- Lines 40–43: Incorporating the Dirichlet condition at time step n.
- Lines 44–47: Solving the reduced linear system for the solution at time step n.
- Lines 49–50: Graphical representation of the numerical solution at the final time step.

10. A nonlinear problem

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

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

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

$$J(u,v) := \int_\Omega \varepsilon\nabla u\cdot\nabla v\,\mathrm{d}x - \int_\Omega \big(u-u^3\big)v\,\mathrm{d}x = 0, \quad v\in H_0^1(\Omega), \tag{15}$$

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

$$\min \int_\Omega \left[\frac{\varepsilon}{2} |\nabla u|^2 + \frac{1}{4}\bigl(u^2 - 1\bigr)^2 \right] \mathrm{d}x! \tag{16}$$

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

$$DJ\bigl(u^n, v; u^n - u^{n+1}\bigr) = J\bigl(u^n, v\bigr), \quad v \in H_0^1(\Omega), \tag{17}$$

where

$$DJ(u, v; w) = \int_\Omega \varepsilon \nabla v \cdot \nabla w \, \mathrm{d}x - \int_\Omega \bigl(vw - 3vu^2 w\bigr) \, \mathrm{d}x. \tag{18}$$

The integrals in $J(U, V)$ and $DJ(U, V; W)$ are again calculated as a sum over all elements. The resulting local integrals can be calculated analytically and are implemented in `localj.m`, respectively `localdj.m`, as given in appendix C. The actual Matlab code again only needs little modifications, shown in appendix C. Essentially, one has to initialise the code (with a start vector that fulfils the Dirichlet boundary condition (lines 9 and 10)), to add a loop (lines 12 and 45), to update the new Newton approximation (line 41), and to supply a stopping criterion in case of convergence (lines 42–44).

It is known that the solutions are not unique. Indeed, for any local minimiser u, $-u$ is also a minimiser 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 separate large regions, where u is almost constant ± 1.

In the finite dimensional problem, different initial values u^0 may lead to different numerical approximations. Figure 5 displays two possible solutions found for two different starting values after about 20–30 iteration steps. The figure on the left is achieved with the starting values as chosen in the program provided in appendix C. Changing the statement in line 9 in appendix C to `U = sign(coordinates(:,1));` leads to the figure on the right.

The main program, which is listed in appendix C, is structured as follows (the line references are according to the numbering in appendix C):

- Lines 3–7: Loading of the mesh geometry and initialisation.
- Lines 8–10: Setting the starting vector U for the iteration scheme, incorporating the Dirichlet condition on the solution.
- Lines 11–45: Loop for the Newton–Raphson iteration. It ends after a maximum of 50 iteration steps (set in line 12) or in case of convergence (lines 42–44).
- Lines 13–18: Assembly of the matrix of the derivative of the functional J evaluated at the current iteration step U.

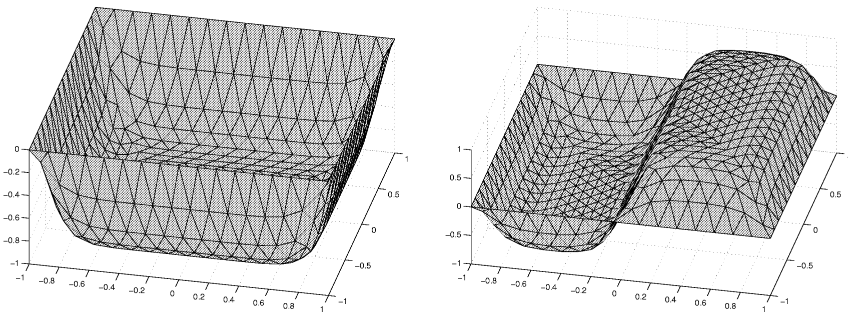

Figure 5. Solutions for the non-linear equation.

- Lines 19–24: Assembly of the vector of the functional J evaluated at the current iteration step U.
- Lines 25–30: Incorporating the volume force.
- Lines 31–35: Incorporating the Neumann condition.
- Lines 36–38: Incorporating the homogeneous Dirichlet conditions of the update vector W.
- Lines 39–40: Solving the reduced linear system for the update vector W.
- Line 41: Updating U.
- Lines 42–44: Breaking out of the loop if the update vector W is sufficiently small (its norm being smaller than 10^{-10}).
- Lines 46–47: Graphical representation of the final iterate.

11. Three-dimensional problems

With a few modifications, the Matlab code for linear 2-dimensional problems discussed in sections 5–8 can be extended to 3-dimensional problems. Tetraeders are used as finite elements. The basis functions are defined corresponding to those in two dimensions, e.g., for a tetraeder element T let (x_j, y_j, z_j) $(j = 1, \dots, 4)$ be the vertices and η_j the corresponding basis functions, i.e.,

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

Each of the `*.dat` files gets 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 `elements3.dat` now reads

$$j \quad k \quad \ell \quad m \quad n,$$

where k, ℓ, m, n are the numbers of vertices $P_k, \dots, P_n$ of the jth element. `elements4.dat` is not used for 3-dimensional problems. The sequence of nodes is organised 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, cancellation of lines 5, 16–19 and 26–30 and substituting 22–24, 33–34, 43 by the following lines gives a short and flexible tool for solving scalar, linear 3-dimensional problems:

```
b(elements3(j,:)) = b(elements3(j,:)) + ...
  det([1,1,1,1;coordinates(elements3(j,:),:)']) * ...
  f(sum(coordinates(elements3(j,:),:))/4) / 24;

b(neumann(j,:)) = b(neumann(j,:)) + ...
  norm(cross(coordinates(neumann(j,3),:) - ...
  coordinates(neumann(j,1),:),coordinates(neumann(j,2),:) - ...
  coordinates(neumann(j,1),:))) ...
  * g(sum(coordinates(neumann(j,:),:))/3)/6;

showsurface([dirichlet;neumann],coordinates,full(u));
```

The graphical representation for 3-dimensional problems can be done by a shortened version of `show.m` from section 8.

```
function showsurface(surface,coordinates,u)
trisurf(surface,coordinates(:,1),coordinates(:,2),...
        coordinates(:,3),u', 'facecolor','interp')
axis off
view(160,-30)
```

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

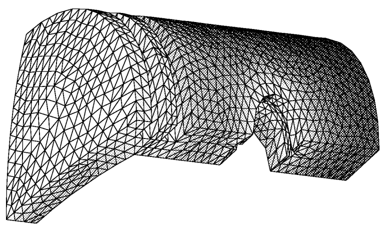

Figure 6. Temperature distribution of a piston.

The main program, which is listed in appendix D, is structured as follows (the line references are according to the numbering in appendix D):

- Lines 3–9: Loading of the mesh geometry and initialisation.
- Lines 11–14: Assembly of the stiffness matrix A in a loop over all tetraeders.
- Lines 16-20: Incorporating the volume force in a loop over all tetraeders.
- Lines 22–27: Incorporating the Neumann condition.
- Lines 29–31: Incorporating the Dirichlet condition.
- Line 33: Solving the reduced linear system.
- Line 35: Graphical representation of the numerical solution.

Appendix

A. The complete Matlab code for the 2-dimensional Laplace problem

The following program can be found in the package, under the path `acf/fem2d`. It is called `fem2d.m`. The other files under that path are the fixed functions `stima3.m`, `stima4.m`, and `show.m` as well as the functions and data files that describe the discretisation and the data of the problem, namely `coordinates.dat`, `elements3.dat`, `elements4.dat`, `dirichlet.dat`, `neumann.dat`, `f.m`, `g.m`, and `u_d.m`. Those problem-describing files must be adapted by the user for other geometries, discretisations, and/or data.

```
% FEM2D   two-dimensional finite element method for Laplacian.
% Initialisation
load coordinates.dat; coordinates(:,1)=[];
eval('load elements3.dat; elements3(:,1)=[];','elements3=[];');
eval('load elements4.dat; elements4(:,1)=[];','elements4=[];');
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),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
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
% Dirichlet conditions
u = sparse(size(coordinates,1),1);
u(unique(dirichlet)) = u_d(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));
```

B. Matlab code for the heat equation

The following program can be found in the package, under the path `acf/fem2d_heat`. It is called `fem2d_heat.m`. The other files under that path are the fixed functions `stima3.m` and `show.m` as well as the functions and data files that describe the discretisation and the data of the problem, namely `coordinates.dat`, `elements3.dat`, `dirichlet.dat`, `neumann.dat`, `f.m`, `g.m`, and `u_d.m`. Those problem-describing files must be adapted by the user for other geometries, discretisations, and/or data.

```
%FEM2D_HEAT finite element method for two-dimensional 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
  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
  % 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)));
```

C. Matlab code for the nonlinear problem

The following program can be found in the package, under the path `acf/fem2d_nonlinear`. It is called `fem2d_nonlinear.m`. The other files under that path are the fixed function `show.m` as well as the functions and data files that describe the functional J, its derivative DJ, the discretisation and the data of the problem, namely, `localj.m`, `localdj.m`, `coordinates.dat`, `elements3.dat`, `dirichlet.dat`, `f.m`, and `u_d.m`. Those problem-describing files must be adapted by the user for other nonlinear problems, geometries, discretisations, and/or data (including possibly adding appropriate files `neumann.dat` and `g.m`).

```
% FEM2D_NONLINEAR  finite element method for two-dimensional
% nonlinear 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));
% Initial value
U = -ones(size(coordinates,1),1);
U(unique(dirichlet)) = u_d(coordinates(unique(dirichlet),:));
% Newton-Raphson iteration
for i=1:50
  % Assembly of DJ(U)
  A = sparse(size(coordinates,1),size(coordinates,1));
  for j = 1:size(elements3,1)
    A(elements3(j,:),elements3(j,:)) = A(elements3(j,:), ...
        elements3(j,:)) ...
        + localdj(coordinates(elements3(j,:),:),U(elements3(j,:)));
  end
  % Assembly of J(U)
  b = sparse(size(coordinates,1),1);
  for j = 1:size(elements3,1);
    b(elements3(j,:)) = b(elements3(j,:)) ...
        + localj(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
  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
  % Dirichlet conditions
  W = zeros(size(coordinates,1),1);
  W(unique(dirichlet)) = 0;
  % Solving one Newton step
  W(FreeNodes) = A(FreeNodes,FreeNodes)\b(FreeNodes);
  U = U - W;
  if norm(W) < 10^(-10)
    break
  end
end
% graphic representation
show(elements3,[],coordinates,full(U));
```

```
function b = localj(vertices,U)
Eps = 1/100;
G = [ones(1,3);vertices'] \ [zeros(1,2);eye(2)];
Area = det([ones(1,3);vertices']) / 2;
b=Area*((Eps*G*G'-[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 = localdj(vertices,U)
Eps = 1/100;
G = [ones(1,3);vertices'] \ [zeros(1,2);eye(2)];
Area = det([ones(1,3);vertices']) / 2;
M = Area*(Eps*G*G'-[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);
```

D. Matlab code for the 3-dimensional problem

The following program can be found in the package, under the path `acf/fem3d`. It is called `fem3d.m`. The other files under that path are the fixed functions `stima3.m`, and `showsurface.m` as well as the functions and data files that describe the discretisation and the data of the problem, namely `coordinates.dat`, `elements3.dat`, `dirichlet.dat`, `neumann.dat`, `f.m`, `g.m`, and `u_d.m`. Those problem-describing files must be adapted by the user for other geometries, discretisations, and/or data.

```
% FEM3D  three-dimensional finite element method for Laplacian.
% 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),1);
% Assembly
for j = 1:size(elements3,1)
```

```
  A(elements3(j,:),elements3(j,:)) = A(elements3(j,:), ...
    elements3(j,:)) + stima3(coordinates(elements3(j,:),:));
end
% Volume Forces
for j = 1:size(elements3,1)
  b(elements3(j,:)) = b(elements3(j,:)) + ...
  det([1,1,1,1;coordinates(elements3(j,:),:)’]) ...
  * f(sum(coordinates(elements3(j,:),:))/4) / 24;
end
% Neumann conditions
for j = 1 : size(neumann,1)
  b(neumann(j,:)) = b(neumann(j,:)) + ...
  norm(cross(coordinates(neumann(j,3),:)- ...
  coordinates(neumann(j,1),:), ...
  coordinates(neumann(j,2),:)-coordinates(neumann(j,1),:))) ...
  * g(sum(coordinates(neumann(j,:),:))/3)/6;
end
% Dirichlet conditions
u = sparse(size(coordinates,1),1);
u(unique(dirichlet)) = u_d(coordinates(unique(dirichlet),:));
b = b - A * u;
% Computation of the solution
u(FreeNodes) = A(FreeNodes,FreeNodes) \ b(FreeNodes);
% Graphic representation
showsurface([dirichlet;neumann],coordinates,full(u));
```

References

[1] S.C. Brenner and L.R. Scott, *The Mathematical Theory of Finite Element Methods*, Texts in Applied Mathematics, Vol. 15 (Springer, New York, 1994).

[2] P.G. Ciarlet, *The Finite Element Method for Elliptic Problems* (North-Holland, Amsterdam, 1978).

[3] L. Langemyr et al., *Partial Differential Equation Toolbox User’s Guide* (The Math Works, Inc. 1995).

[4] H.R. Schwarz, *Methode der Finiten Elemente* (Teubner, Stuttgart, 1991).