Vectorized Matlab codes for linear two-dimensional elasticity

Jonas Koko
LIMOS, Université Blaise Pascal, CNRS UMR 6158, and ISIMA, Campus des Cézeaux, BP 10125, 63173 Aubière Cedex, France
E-mail: koko@isima.fr

Abstract. A vectorized Matlab implementation for the linear finite element is provided for the two-dimensional linear elasticity with mixed boundary conditions. Vectorization means that there is no loop over triangles. Numerical experiments show that our implementation is more efficient than the standard implementation with a loop over all triangles.

Keywords: Finite element method, elasticity, Matlab

1. Introduction

Matlab is nowadays a widely used tool in education, engineering and research and becomes a standard tool in many areas. But Matlab is a matrix language, that is, Matlab is designed for vector and matrix operations. For the best performance in large scale problems, one should take advantage of this.

We propose a Matlab implementation of the P_1 finite element for the numerical solutions of two-dimensional linear elasticity problems. Instead of mixing finite element types (e.g. [1,2], we propose a P_1-triangle vectorized code able to treat medium size meshes in “acceptable” time. Vectorization means that there is no loop over triangles nor nodes. Our implementation needs only Matlab basic distribution functions and can be easily modified and refined.

The paper is organized as follows. The model problem is described in Section 2, followed by a finite element discretization in Section 3. The data representation used in Matlab programs is given in Section 4. The heart of the paper is the assembling functions of the stiffness matrix in Section 5 and the right-hand side in Section 6. Numerical experiments are carried out in Section 7 where post-processing functions are given. Matlab programs used for numerical experiments are given in the appendix.

2. Model problem

We consider an elastic body which occupies, in its reference configuration, a bounded domain Ω in $\mathbb{R}^2$ with a boundary Γ. Let $\{\Gamma_D, \Gamma_N\}$ be a partition of Γ with Γ_N possibly empty. We assume Dirichlet conditions on Γ_D and Neumann conditions on Γ_N. Let $u = (u_1, u_2)$ be the two-dimensional displacement field of the elastic body. Under the small deformations assumption, constitutive equations are

$$\begin{aligned} \sigma_{ij}(u) &= 2\mu\epsilon_{ij}(u) + \lambda \,\mathrm{tr}\,(\epsilon(u))\mathbb{I}_2, \quad i,j = 1,2, \\ \epsilon(u) &= (\nabla u + \nabla u^T)/2, \end{aligned} \tag{1}$$

where λ and μ denote Lamé (positive) constants. These coefficients are related (in plane deformations) to the Young modulus E and the Poisson coefficient ν by

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}.$$

Given $f=(f_1,f_2)\in L^2(\Omega)$, $g=(g_1,g_2)\in L^2(\Gamma)$ and u_D, the problem studied in this paper can be formulated as follows

$$-\mathrm{div}\sigma(u) = f, \quad \text{in } \Omega, \tag{2}$$

$$\sigma(u)\cdot n = g, \quad \text{on } \Gamma_N, \tag{3}$$

$$u = u_D, \quad \text{on } \Gamma_D. \tag{4}$$

Let us introduce subspaces

$$V = \left\{ v \in H^1(\Omega) \ : \ v = 0 \text{ on } \Gamma_D \right\},$$

$$V^D = \left\{ v \in H^1(\Omega) \ : \ v = u_D \text{ on } \Gamma_D \right\}.$$

The variational formulation of Eqs (2)–(4) is

Find $u \in V^D$ such that

$$\int_\Omega \epsilon(u) : \mathbb{C}\epsilon(v)dx = \int_\Omega f\cdot v dx + \int_{\Gamma_N} g\cdot v ds, \quad \forall v \in V. \tag{5}$$

In Eq. (5), $\mathbb{C} = (\mathbb{C}_{ijkl})$ is the fourth-order elastic moduli tensor corresponding to Eq. (1), i.e.

$$\sigma_{ij}(u) = \sum_{k,l=1}^{2} \mathbb{C}_{ijkl}\epsilon_{ij}(u), \quad i,j=1,2,$$

where

$$\mathbb{C}_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}), \quad 1 \leqslant i,j,k,l \leqslant 2,$$

δ_{ij} being the Kronecker delta.

3. Finite element discretization

Let T_h be a regular (in the sense of Ciarlet [3]) triangulation of Ω. Spaces V and V^D are then replaced by their discrete approximations V_h and V_h^D defined by

$$V_h = \left\{ v_h \in \mathsf{C}^0(\bar\Omega);\ v_{|T} \in \mathbb{P}_1(T),\ \forall T \in \mathsf{T}_h;\ v_{h|\Gamma_D} = 0 \right\},$$

$$V_h^D = \left\{ v_h \in \mathsf{C}^0(\bar\Omega);\ v_{|T} \in \mathbb{P}_1(T),\ \forall T \in \mathsf{T}_h;\ v_{h|\Gamma_D} = u_D \right\},$$

where $\mathbb{P}_1(T)$ is the space of polynomials of degree less or equals to 1 on the triangle T. The discrete version of Eq. (5) is then

Find $u_h \in V_h^D$ such that

$$\int_\Omega \epsilon(u_h) : \mathbb{C}\epsilon(v_h)dx = \int_\Omega f\cdot v_h dx, \quad \forall v_h \in V_h. \tag{6}$$

Let $\{\phi^j\}$ be the system of piecewise global basis functions of V_h, i.e. for all $u_h = (u_{1h}, u_{2h}) \in V_h$

$$u_{\alpha h} = \sum_{j=1}^{N} \phi^j(x)u_\alpha^j, \quad \alpha = 1,2.$$

We set $U = (u_1^1\, u_2^1\, u_1^2\, u_2^2\, \ldots\, u_1^N\, u_2^N)$, where u_α^j are nodal values of u_h, i.e. $u_{\alpha h}(x_j) = u_\alpha^j$. Applying the standard Galerkin method to Eq. (6) yields

$$\sum_{j=1}^{N} \left[\int_\Omega \epsilon(\phi_i) : \mathbb{C}\epsilon(\phi_j) dx \right] U_j = \int_\Omega f \cdot \phi_i \, dx + \int_{\Gamma_N} g \cdot \phi_i \, ds, \quad i = 1, \ldots, N. \tag{7}$$

The stiffness matrix $A = (A_{ij})$ and the right-hand side $b = (b_i)$ are then given by

$$A_{ij} = \int_\Omega \epsilon(\phi_i) : \mathbb{C}\epsilon(\phi_j) dx,$$

$$b_i = \int_\Omega f \cdot \phi_i \, dx + \int_{\Gamma_N} g \cdot \phi_i \, ds.$$

The stiffness matrix A is sparse, symmetric and positive semi-definite before incorporating the boundary condition $u_{|\Gamma_D} = u_D$.

In practice, integrals in Eq. (6) are computed as sums of integrals over all triangles, using the fact that $\bar{\Omega} = \cup_{T \in \mathsf{T}_h} T$

$$\sum_{T \in \mathsf{T}_h} \int_T \epsilon(u_h) : \mathbb{C}\epsilon(v_h) dx = \sum_{T \in \mathsf{T}_h} \int_T f \cdot v_h dx + \sum_{E \subset \Gamma_N} \int_{\Gamma_N} g \cdot v_h \, ds, \quad \forall v_h \in V_h.$$

Let $\{\varphi_i\}$ be the linear basis functions of the triangle T or the edge E. If we set

$$A_{ij}^{(T)} = \int_T \epsilon(\varphi_i) : \mathbb{C}\epsilon(\varphi_j) dx \tag{8}$$

$$b_i^{(T)} = \int_T f \cdot \varphi_i \, dx, \tag{9}$$

$$b_i^{(E)} = \int_E g \cdot \varphi_i \, ds, \tag{10}$$

then assembling operations consist of

$$A_{ij} = \sum_{T \in \mathsf{T}_h} A_{ij}^{(T)}$$

$$b_i = \sum_{T \in \mathsf{T}_h} b_i^{(T)} + \sum_{E \subset \Gamma_N} b_i^{(E)},$$

4. Data representation of the triangulation

For the mesh, we adopt the data representation used in Matlab PDE Toolbox [6]. Nodes coordinates and triangle vertices are stored in two arrays `p(1:2,1:np)` and `t(1:3,1:nt)`, where `np` is the number of nodes and `nt` the number of triangles. The array `t` contains for each element the node numbers of the vertices numbered anti-clockwise. For the triangulation of Fig. 1, the nodes array `p` is (`np=9`)

```
0.0000  1.0000  1.0000  0.  0.5000  1.0000  0.5000  0.0000  0.5000
1.0000  1.0000  0.0000  0.  1.0000  0.5000  0.0000  0.5000  0.5000
```

and the elements array `t` is (`nt=8`)

```
    5     6     7     8     2     3     4     1
    1     2     3     4     5     6     7     8
    9     9     9     9     9     9     9     9
```

Neumann boundary nodes are provided by an array `ibcneum(1:2,1:nbcn)` containing the two node numbers which bound the corresponding edge on the boundary. Then, a sum over all edges E results in a loop over all entries of `ibcneum`. Dirichlet boundary conditions are provided by a list of nodes and a list of the corresponding prescribed boundary values.

Note that Matlab supports reading data from files in ASCII format (Matlab function `load`) and there exists good mesh generators written in Matlab, see e.g. [7].

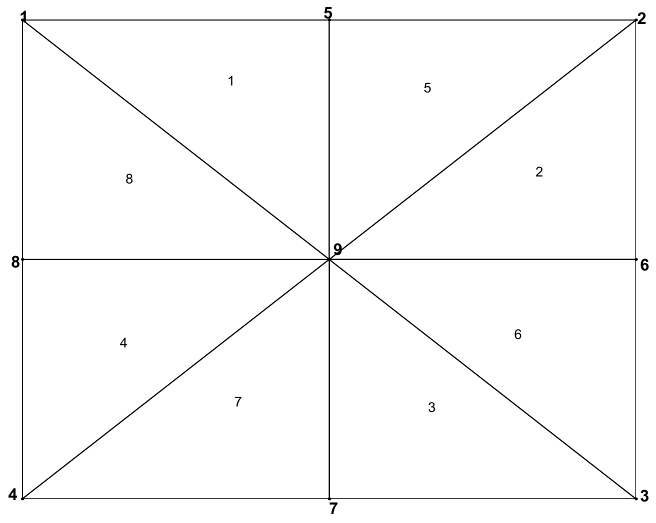


Fig. 1. Plot of a triangulation.

5. Assembling the stiffness matrix

As in [2,5] we adopt Voigt's representation of the linear strain tensor

$$\gamma(u) = \begin{bmatrix} \partial u_1/\partial x \\ \partial u_2/\partial y \\ \partial u_1/\partial y + \partial u_2/\partial x \end{bmatrix} = \begin{bmatrix} \epsilon_{11}(u) \\ \epsilon_{22}(u) \\ 2\epsilon_{12}(u) \end{bmatrix}.$$

Using this representation, the constitutive equation $\sigma(u_h) = \mathbb{C}\epsilon(u_h)$ becomes

$$\begin{bmatrix} \sigma_{11}(u_h) \\ \sigma_{22}(u_h) \\ \sigma_{12}(u_h) \end{bmatrix} = C\gamma(u_h), \tag{11}$$

where

$$C = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix}.$$

Let **u** $= (u_1\ u_2\ \cdots\ u_6)^t$ be the vector of nodal values of u_h on a triangle T. Elementary calculations provide, on a triangle T,

$$\gamma(u_h) = \frac{1}{2|T|} \begin{bmatrix} \varphi_{1,x} & 0 & \varphi_{2,x} & 0 & \varphi_{3,x} & 0 \\ 0 & \varphi_{1,y} & 0 & \varphi_{2,y} & 0 & \varphi_{3,y} \\ \varphi_{1,y} & \varphi_{1,x} & \varphi_{2,y} & \varphi_{2,x} & \varphi_{3,y} & \varphi_{3,x} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_6 \end{bmatrix} =: \frac{1}{2|T|} R\mathbf{u}, \tag{12}$$

where $|T|$ is the area of the triangle T. From Eqs (11) and (12), it follows that

$$\epsilon(v_h) : \mathbb{C}\epsilon(u_h) = \gamma(v_h)^t C\gamma(u_h) = \frac{1}{4|T|^2}\mathbf{v}^t R^t CR\mathbf{u}$$

The element stiffness matrix is therefore

$$A^{(T)} = \frac{1}{4|T|} R^t CR. \tag{13}$$

The element stiffness matrix Eq. (13) can be computed simultaneously for all indices using the fact that

$$\begin{bmatrix}\nabla\varphi_1^t\\ \nabla\varphi_2^t\\ \nabla\varphi_3^t\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{bmatrix}^{-1}\begin{bmatrix}0\,0\\ 1\,0\\ 0\,1\end{bmatrix} = \frac{1}{2|T|}\begin{bmatrix}y_2-y_3 & x_3-x_2\\ y_3-y_1 & x_1-x_3\\ y_1-y_2 & x_2-x_1\end{bmatrix}.$$

Matlab implementation given in [2,5] (in a more modular form) can be summarized by the Matlab Function 5. This implementation, directly derived from compiled languages as Fortran or C/C++, produces a very slow code in Matlab for large size meshes due to the presence of the loop `for`. Our aim is to remove this loop by reorganizing calculations.

Function 1. Assembly of the stiffness matrix with the standard loop over triangles

```
%function A=elas2dmat1(p,t,Young,nu)
%----------------------------------------------------------------
% Two-dimensional linear elasticity
% Assembly of the stiffness matrix
%----------------------------------------------------------------
n=size(p,2); nt=size(t,2); nn=2*n;
% Lame constants
lam=Young*nu/((1+nu)*(1-2*nu)); mu=.5*Young/(1+nu);
% Constant matrices
C=mu*[2,0,0;0,2,0;0,0,1]+lam*[1,1,0;1,1,0;0,0,0];
Zh=[zeros(1,2);eye(2)];
% Loop over all triangles
A=sparse(nn,nn);
for ih=1:nt
    it=t(1:3,ih);
    itt=2*t([1,1,2,2,3,3],ih)-[1;0;1;0;1;0];
    xy=zeros(3,2); xy(:,1)=p(1,it)’; xy(:,2)=p(2,it)’;
    D=[1,1,1;xy’];
    gradphi=D\Zh;
    R=zeros(3,6);
    R([1,3],[1,3,5])=gradphi’;
    R([3,2],[2,4,6])=gradphi’;
    A(itt,itt)=A(itt,itt)+0.5*det(D)*R’*C*R;
end
```

Let us introduce the following notations

$$x_{ij} = x_i - x_j, \quad y_{ij} = y_i - y_j, \quad i,j = 1,2,3, \tag{14}$$

so that, from Eq. (12)

$$R = \begin{bmatrix} y_{23} & 0 & y_{31} & 0 & y_{12} & 0\\ 0 & x_{32} & 0 & x_{13} & 0 & x_{21}\\ x_{32} & y_{23} & x_{13} & y_{31} & x_{21} & y_{12}\end{bmatrix}.$$

If we rewrite **u** in the following non standard form $\mathbf{u} = (u_1\ u_3\ u_5\ u_2\ u_4\ u_6)^t$ then the element stiffness matrix can be rewritten as

$$A^{(T)} = \begin{bmatrix} A_{11}^{(T)} & A_{12}^{(T)}\\ A_{21}^{(T)} & A_{22}^{(T)}\end{bmatrix}$$

where $A_{11}^{(T)}$ and $A_{22}^{(T)}$ are symmetric and $A_{21}^{(T)} = (A_{21}^{(T)})^t$. After laborious (but elementary) calculations, we get

$$A_{11}^{(T)} = \begin{bmatrix} \widetilde{\lambda} y_{23}^2 + \mu x_{32}^2 & \widetilde{\lambda} y_{23} y_{31} + \mu x_{32} x_{13} & \widetilde{\lambda} y_{23} y_{12} + \mu x_{32} x_{21} \\ \widetilde{\lambda} y_{31} y_{23} + \mu x_{13} x_{32} & \widetilde{\lambda} y_{31}^2 + \mu x_{13}^2 & \widetilde{\lambda} y_{31} y_{12} + \mu x_{13} x_{21} \\ \widetilde{\lambda} y_{12} y_{23} + \mu x_{21} x_{32} & \widetilde{\lambda} y_{12} y_{31} + \mu x_{21} x_{13} & \widetilde{\lambda} y_{12}^2 + \mu x_{21}^2 \end{bmatrix}$$

$$A_{22}^{(T)} = \begin{bmatrix} \widetilde{\lambda} x_{32}^2 + \mu y_{23}^2 & \widetilde{\lambda} x_{32} x_{13} + \mu y_{23} y_{31} & \widetilde{\lambda} x_{32} x_{21} + \mu y_{23} y_{12} \\ \widetilde{\lambda} x_{13} x_{32} + \mu y_{31} y_{23} & \widetilde{\lambda} x_{13}^2 + \mu y_{31}^2 & \widetilde{\lambda} x_{13} x_{21} + \mu y_{31} y_{12} \\ \widetilde{\lambda} x_{21} x_{32} + \mu y_{12} y_{23} & \widetilde{\lambda} x_{21} x_{13} + \mu y_{12} y_{31} & \widetilde{\lambda} x_{21}^2 + \mu y_{12}^2 \end{bmatrix}$$

where $\widetilde{\lambda} = \lambda + 2\mu$; and

$$A_{12}^{(T)} = \begin{bmatrix} \lambda y_{23} x_{32} + \mu x_{32} y_{23} & \lambda y_{23} x_{13} + \mu x_{32} y_{31} & \lambda y_{23} x_{21} + \mu x_{32} y_{12} \\ \lambda y_{31} x_{32} + \mu x_{13} y_{23} & \lambda y_{31} x_{13} + \mu x_{13} y_{31} & \lambda y_{31} x_{21} + \mu x_{13} y_{12} \\ \lambda y_{12} x_{32} + \mu x_{21} y_{23} & \lambda y_{12} x_{13} + \mu x_{21} y_{31} & \lambda y_{12} x_{21} + \mu x_{21} y_{12} \end{bmatrix}.$$

Let us introduce the following vectors

$$\mathbf{x} = \begin{bmatrix} x_{32} \\ x_{13} \\ x_{21} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_{23} \\ y_{31} \\ y_{12} \end{bmatrix}.$$

One can verify that matrices $A_{11}^{(T)}$, $A_{22}^{(T)}$ and $A_{12}^{(T)}$ can be rewritten in a simple form, using **x** and **y**, as

$$A_{11}^{(T)} = (\lambda + 2\mu)\mathbf{y}\mathbf{y}^t + \mu\mathbf{x}\mathbf{x}^t$$

$$A_{22}^{(T)} = (\lambda + 2\mu)\mathbf{x}\mathbf{x}^t + \mu\mathbf{y}\mathbf{y}^t$$

$$A_{12}^{(T)} = \lambda\mathbf{y}\mathbf{x}^t + \mu\mathbf{x}\mathbf{y}^t$$

that is, for $1 \leqslant i, j \leqslant 3$

$$A_{11ij}^{(T)} = (\lambda + 2\mu)\mathbf{y}_i\mathbf{y}_j + \mu\mathbf{x}_i\mathbf{x}_j, \tag{15}$$

$$A_{22ij}^{(T)} = (\lambda + 2\mu)\mathbf{x}_i\mathbf{x}_j + \mu\mathbf{y}_i\mathbf{y}_j, \tag{16}$$

$$A_{12ij}^{(T)} = \lambda\mathbf{y}_i\mathbf{x}_j + \mu\mathbf{x}_i\mathbf{y}_j \tag{17}$$

With Matlab, $\mathbf{x}_i$ and $\mathbf{y}_i$ can be computed in a fast way for all triangles using vectorization. Then assembling the stiffness matrix reduces to two *constant* loops for computing $\mathbf{x}\mathbf{x}^t$, $\mathbf{y}\mathbf{y}^t$ and $\mathbf{x}\mathbf{y}^t$. We do not need to assemble separately $A_{11}^{(T)}$, $A_{22}^{(T)}$ and $A_{12}^{(T)}$. Sub matrices Eqs (15)–(17) are directly assembled in the global stiffness matrix, in its standard form, since we know their locations. Matlab vectorized implementation of the assembly of the stiffness matrix is presented in Function 2.

Function 2. Assembly of the stiffness matrix with a vectorized code

```
function K=elas2dmat2(p,t,Young,nu)
%-------------------------------------------------------------
% Two-dimensional finite element method for the Lame Problem
% Assembly of the stiffness matrix
%-------------------------------------------------------------
n=size(p,2); nn=2*n;
%
% Lame constants
```

```
lam=Young*nu/((1+nu)*(1-2*nu)); mu=.5*Young/(1+nu);
%
% area of triangles
it1=t(1,:); it2=t(2,:); it3=t(3,:);
x21=(p(1,it2)-p(1,it1))'; x32=(p(1,it3)-p(1,it2))'; x31=(p(1,it3)-p(1,it1))';
y21=(p(2,it2)-p(2,it1))'; y32=(p(2,it3)-p(2,it2))'; y31=(p(2,it3)-p(2,it1))';
ar=.5*(x21.*y31-x31.*y21);
muh=mu./(4*ar); lamh=lam./(4*ar); lamuh=(lam+2*mu)./(4*ar);
clear it1 it2 it3
%
x=[ x32 -x31  x21]; y=[-y32  y31 -y21];
it1=2*t'-1; it2=2*t';
% Assembly
K=sparse(nn,nn);
for i=1:3
for j=1:3
  K=K+sparse(it1(:,i),it2(:,j),lamh.*y(:,i).*x(:,j)+muh.*x(:,i).*y(:,j),nn,nn);
  K=K+sparse(it2(:,j),it1(:,i),lamh.*y(:,i).*x(:,j)+muh.*x(:,i).*y(:,j),nn,nn);
  K=K+sparse(it1(:,i),it1(:,j),lamuh.*y(:,i).*y(:,j)+muh.*x(:,i).*x(:,j),nn,nn);
  K=K+sparse(it2(:,i),it2(:,j),lamuh.*x(:,i).*x(:,j)+muh.*y(:,i).*y(:,j),nn,nn);
end
end
```

6. Assembling the right-hand side

We assume that the volume forces $f = (f_1, f_2)$ are provided at mesh nodes. The integral Eq. (10) is approximated as follows

$$\int_T f \cdot \phi_i dx \approx \frac{1}{6}\det\begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} f(x_c, y_c), \quad j = \mathrm{mod}(i-1, 2) + 1$$

where (x_c, y_c) is the center of mass of the triangle T. With the assumption on f, $f(x_c, y_c) = (f_1(x_c, y_c), f_2(x_c, y_c))$ with

$$f_j(x_s, y_s) = (f_j(x_1, y_1) + f_j(x_2, y_2) + f_j(x_3, y_3))/3, \quad j = 1, 2, \tag{18}$$

where $\{(x_i, y_i)\}_{i=1,3}$ are vertices of the triangle T. Using the notation convention Eq. (14), we have

$$\begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} = x_{21}y_{31} - x_{31}y_{12}. \tag{19}$$

We can compute Eqs (18) and (19) over all triangles using vectorization techniques, and assemble the result with the Matlab function `sparse`. Matlab implementation of the assembly of the volume forces is presented in Function 3.

Function 3. Assembly of the right-hand side: body forces

```
function f=elas2drhs1(p,t,f1,f2)
%
%-------------------------------------
% Two-dimensional linear elasticity
% Assembly of the right-hand side 1: body forces
%-------------------------------------
%
n=size(p,2); nn=2*n;
% triangle vertices
```

```
it1=t(1,:); it2=t(2,:); it3=t(3,:);
% edge vectors
a21=p(:,it2)-p(:,it1); a31=p(:,it3)-p(:,it1);
% area of triangles
area=abs(a21(1,:).*a31(2,:)-a21(2,:).*a31(1,:))/2;
% assembly
f1h=(f1(it1)+f1(it2)+f1(it3)).*area'/9;
f2h=(f2(it1)+f2(it2)+f2(it3)).*area'/9;
ff1=sparse(it1,1,f1h,n,1)+sparse(it2,1,f1h,n,1)+sparse(it3,1,f1h,n,1);
ff2=sparse(it2,1,f2h,n,1)+sparse(it2,1,f2h,n,1)+sparse(it3,1,f2h,n,1);

% right-hand side
f=zeros(nn,1); f(1:2:nn)=full(ff1); f(2:2:nn)=full(ff2);
```

Integrals involving Neumann conditions are approximated using the value of $g = (g_1, g_2)$ at the center (x_c, y_c) of the edge E

$$\int_E g \cdot \phi_i dx \approx \frac{1}{2}|E|g(x_c, y_c).$$

As for volume forces, $|E|$ and $g_j(x_c, y_c)$ are computed over all triangles using vectorization techniques and assembled using Matlab function `sparse`, Function 4.

Function 4. Assembly of the right-hand side: Neumann boundary conditions

```
function g=elas2drhs2(p,ineum,g1,g2)
%
%--------------------------------------------------
% Two-dimensional linear elasticity
% Assembly of the right-hand side 2: Neumann condition
%--------------------------------------------------
%
n=size(p,2); nn=2*n;
%
ie1=ineum(1,:); ie2=ineum(2,:);
ibcn=union(ie1,ie2);
ne=length(ibcn);
% edge lengths
exy=p(:,ie2)-p(:,ie1);
le=sqrt(sum(exy.^2,1));
% g1,g2 at the center of mass
g1h=(g1(ie1)+g1(ie2)).*le/4;
g2h=(g2(ie1)+g2(ie2)).*le/4;
% assembly
gg1=sparse(ne,1);
gg1=sparse(ie1,1,g1h,ne,1)+sparse(ie2,1,g1h,ne,1);
gg2=sparse(ne,1);
gg2=sparse(ie1,1,g2h,ne,1)+sparse(ie2,1,g2h,ne,1);
% right-hand side
g=zeros(nn,1);
g(2*ibcn-1)=full(gg1);
g(2*ibcn)  =full(gg2);
```

7. Numerical experiments

In elasticity, it is usual to display undeformed and deformed meshes. In the graphical representation of the deformed mesh, a magnification factor is used for the displacement. The Matlab Function 5 displays the deformed mesh with an additional function `sf` issued from post-processing. It is directly derived from the Matlab function `Show` given in [2]. The additional function can be stresses, strains, potential energy, etc. evaluated at mesh nodes.

Function 5. Visualization function

```
function elas2dshow(p,t,u,magnify,sf)
%
%--------------------------------------
% Two-dimensional linear elasticity
% Visualization
%--------------------------------------
%
np=size(p,2);
uu=reshape(u,2,np)';
pu=p(1:2,:)'+magnify*uu;
colormap(1-gray)
trisurf(t(1:3,:)',pu(:,1),pu(:,2),zeros(np,1),sf,'facecolor','interp')
view(2)
```

Function 6. Matlab function for computing the shear energy density `Sed` and the Von Mises effective stress `Vms`

```
function [Sed,Vms]=elas2dsvms(p,t,uu,E,nu)
%------------------------------------------
% Two-dimensional linear elasticity
% Sed  Shear energy density
% Vms  Von Mises effective stress
%-----------------------------------------
n=size(p,2); nn=2*n;

% Lame constant
lam=E*nu/((1+nu)*(1-2*nu));  mu=E/(2*(1+nu));
% area of traingles
it1=t(1,:); it2=t(2,:); it3=t(3,:);
x21=(p(1,it2)-p(1,it1))'; x32=(p(1,it3)-p(1,it2))'; x31=(p(1,it3)-p(1,it1))';
y21=(p(2,it2)-p(2,it1))'; y32=(p(2,it3)-p(2,it2))'; y31=(p(2,it3)-p(2,it1))';
ar=.5*(x21.*y31-x31.*y21);
% gradient of scalar basis functions
phi1=[-y32./(2*ar)  x32./(2*ar)];
phi2=[ y31./(2*ar) -x31./(2*ar)];
phi3=[-y21./(2*ar)  x21./(2*ar)];
% displacements
u=uu(1:2:end); v=uu(2:2:end);
uh=[u(it1) u(it2) u(it3)]; vh=[v(it1) v(it2) v(it3)];
% strains
e11=uh(:,1).*phi1(:,1)+uh(:,2).*phi2(:,1)+uh(:,3).*phi3(:,1);
e22=vh(:,1).*phi1(:,2)+vh(:,2).*phi2(:,2)+vh(:,3).*phi3(:,2);
e12=uh(:,1).*phi1(:,2)+uh(:,2).*phi2(:,2)+uh(:,3).*phi3(:,2)...
   +vh(:,1).*phi1(:,1)+vh(:,2).*phi2(:,1)+vh(:,3).*phi3(:,1);
clear uh vh
% stresses
```

```
sig11=(lam+2*mu)*e11+lam*e22; sig22=lam*e11+(lam+2*mu)*e22; sig12=mu*e12;
clear e11 e22 e12
% area of patches
arp=full(sparse(it1,1,ar,n,1)+sparse(it2,1,ar,n,1)+sparse(it3,1,ar,n,1));
% mean value of stresses on patches
sm1=ar.*sig11; sm2=ar.*sig22; sm12=ar.*sig12;
s1=full(sparse(it1,1,sm1,n,1)+sparse(it2,1,sm1,n,1)+sparse(it3,1,sm1,n,1));
s2=full(sparse(it1,1,sm2,n,1)+sparse(it2,1,sm2,n,1)+sparse(it3,1,sm2,n,1));
s12=full(sparse(it1,1,sm12,n,1)+sparse(it2,1,sm12,n,1)+sparse(it3,1,sm12,n,1));
s1=s1./arp; s2=s2./arp; s12=s12./arp;
% Shear energy density
Sed=((.5+mu*mu/(6*(mu+lam)^2))*(s1+s2).^2+2*(s12.^2-s1.*s2))/(4*mu);
% Von Mises effective stress
delta=sqrt((s1-s2).^2+4*s12.^2); sp1=s1+s2+delta; sp2=s1+s2-delta;
Vms=sqrt(sp1.^2+sp2.^2-sp1.*sp2);
```

In our numerical experiments, the additional function used in the Function 5 is either the Von Mises effective stress or the shear energy density $|\mathrm{dev}\sigma_h|^2/(4\mu)$, where

$$|\mathrm{dev}\sigma_h|^2 = \left(\frac{1}{2} + \frac{\mu^2}{6(\mu+\lambda)}\right)(\sigma_{h11}+\sigma_{h22})^2 + 2(\sigma_{h12}^2 - \sigma_{h11}\sigma_{h22}).$$

The stress tensor σ_h is computed at every node as the mean value of the stress on the corresponding patch. Matlab Function 6 calculates approximate shear energy density and Von Mises effective stress.

In all examples, Dirichlet boundary conditions are taken into account by using large spring constants (i.e. penalization).

7.1. L-shape

The L-shape test problem is a common benchmark problem for which the exact solution is known. The domain Ω is described by the polygon

$$(-1,-1),(0,-2),(2,0),(0,2),(-1,-1),(0,0).$$

The exact solution is known in polar coordinates (r,θ),

$$u_r(r,\theta) = \frac{1}{2\mu} r^\alpha \left[(c_2-\alpha-1)c_1\cos((\alpha-1)\theta) - (\alpha+1)\cos((\alpha+1)\theta)\right], \tag{20}$$

$$u_\theta(r,\theta) = \frac{1}{2\mu} r^\alpha \left[(\alpha+1)\sin((\alpha+1)\theta) + (c_2+\alpha-1)c_1\sin((\alpha-1)\theta)\right]. \tag{21}$$

The exponent α is the solution of the equation

$$\alpha\sin(2\omega) + \sin(2\omega\alpha) = 0$$

with $\omega = 3\pi/4$ and

$$c_1 = -\frac{\cos((\alpha+1)\omega)}{\cos((\alpha-1)\omega)}, \qquad c_2 = 2\frac{\lambda+2\mu}{\lambda+\mu}.$$

The displacement field Eqs (20)–(21) solves Eqs (2)–(4) with $f=0$ and $u_D=(u_r,u_\theta)$ on $\Gamma=\Gamma_D$. Numerical experiments are carried out with Young's modulus $E=100000$ and Poisson's coefficient $\nu=0.3$. The magnitude of the gradient $|\nabla u|$ of the exact solution Eqs (20)–(21) has a singularity at the re-entrant corner $(0,0)$. This singularity, despite being very local, is a significant source of error.

To know the percentage of computing effort the assembling functions take, we have run the Matlab program given in Appendix (without the four last lines) with the Matlab command `profile` which records informations about execution time, number of calls, parent functions, child functions, code line hit count, etc. Figures 2–3

Table 1
Percentage of computing effort taken by `elas2dmat1`

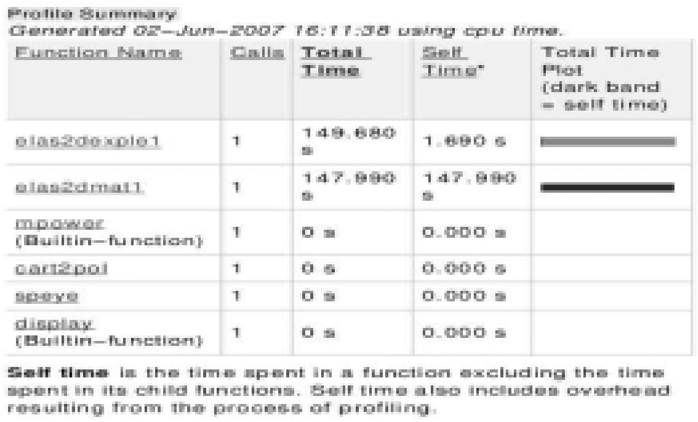
Mesh triangles	100	400	1600	6400	25600	102400
Percentage of CPU	57.1%	76.2%	90.5%	95.8%	98.9%	99.7%

Table 2
Performances of assembling functions `elas2dmat1` and `elas2dmat2`

Mesh	CPU time (in Sec.)	
triangles/nodes	`elas2dmat1`	`elas2dmat2`
100/66	0.04	0.02
400/231	0.16	0.03
1600/861	0.86	0.11
6400/3321	7.95	0.48
25600/13041	148.02	2.13
102400/51681	3394.51	8.95

Table 3
Percentage of computing effort taken by `elas2dmat2`

Mesh triangles	100	400	1600	6400	25600	102400
Percentage of CPU	40.0%	42.0%	50.0%	57.0%	57.0%	49.7%



Profile Summary
Generated 02–Jun–2007 16:11:38 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
elas2dexple1	1	149.680 s	1.690 s	▬
elas2dmat1	1	147.990 s	147.990 s	▬
mpower (Builtin–function)	1	0 s	0.000 s	
cart2pol	1	0 s	0.000 s	
speye	1	0 s	0.000 s	
display (Builtin–function)	1	0 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Fig. 2. Matlab `profile` command report with `elas2dmat1` assembling function using a mesh with 13041 nodes.

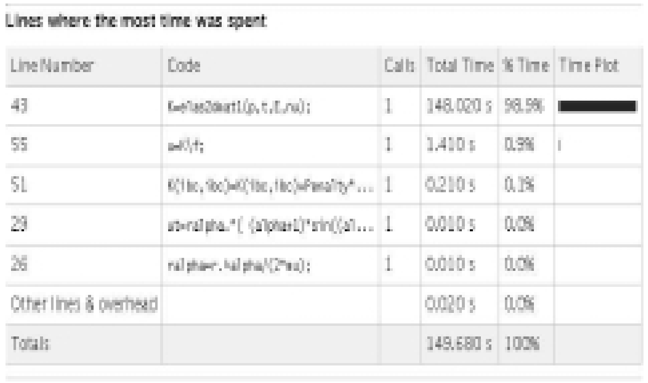
show informations recorded with a mesh with 13041 nodes. It appears that the assembling function `elas2dmat1` takes about 98% of CPU time. Note that the time given in Table 3 for the assembling operations is the time taken by intruction `K=elas2dmat1(p,t,E,nu)` while the time given in Table 2 is only the time *spent* in the assembling function (without calling and returning operations). Table 1 shows clearly that the assembling function `elas2dmat1` is the bottleneck of the program.

We now compare the performances of Matlab functions `elas2dmat1` and `elas2dmat2` for assembling the stiffness matrix of the L-shape problem. To this end, various meshes of the L-shape are generated and we use Matlab command `profile` to compute the elapsed time. Table 2 shows the performances of assembling Functions 1–2. One can notice that the saving of computational time, with the vectorized function `elas2dmat2`, is considerable. Table 3 shows that the finite element program with `elas2dmat2` is more balanced compared to `elas2dmat1`.

We report in Table 4 the H^1 and L^2 distances between the exact solution Eqs (20)–(21) and the approximate solution using the assembling function `elas2dmat2`. The distances are computed using a 13-point Gaussian quadrature. One can notice that $\|u - u_h\|_{L^2(\Omega)} \to 0$ and $\|u - u_h\|_{H^1(\Omega)} \to 0$ has the mesh size h goes to zero but the convergence rates are lower than theoretical ones (2 for the L^2-error and 1 for the H^2-error for elliptic problems,

Table 4
L^2 and H^1 errors and convergence rates

Mesh nodes	$\|u - u_h\|_{L^2(\Omega)}$	*Rate*	$\|u - u_h\|_{H^1(\Omega)}$	*Rate*
66	1.7843×10^{-6}		1.7372×10^{-5}	
		1.31		*0.52*
231	7.1572×10^{-7}		1.2058×10^{-5}	
		1.31		*0.53*
861	2.8760×10^{-7}		8.3279×10^{-6}	
		1.30		*0.53*
3321	1.1624×10^{-7}		5.7377×10^{-6}	
		1.30		*0.54*
13041	4.7203×10^{-8}		3.9449×10^{-6}	
		1.29		*0.54*
51681	1.9220×10^{-8}		2.7089×10^{-6}	



Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
43	K=elas2dmat1(p,t,E,nu);	1	148.020 s	98.9%	▇▇▇▇
55	u=K\f;	1	1.410 s	0.9%	▏
51	K(ibc,ibc)=K(ibc,ibc)+Penalty*...	1	0.210 s	0.1%	
29	ut=ralpha.*((alpha+1)*sin((al...	1	0.010 s	0.0%	
26	ralpha=r.^alpha/(2*mu);	1	0.010 s	0.0%	
Other lines & overhead			0.020 s	0.0%	
Totals			149.680 s	100%	

Fig. 3. Matlab `profile` command report details with `elas2dmat1` assembling function using a mesh with 13041 nodes.

see e.g. [3,4]). The reason is that the exact solution u does not belong to $H^2(\Omega)$ so that the error estimate theorems do not hold.

Figure 4 shows the deformed mesh (231 nodes and 400 triangles) of the L-shape with a displacement field multiplied by a factor 3000. The grey tones visualize the approximate shear energy density and show the singularity at $(0, 0)$.

7.2. *A membrane problem*

We consider a two-dimensional membrane Ω described by the polygon

$$(0, -5),\ (35, -2),\ (35, 2),\ (0, 5)$$

with $E = 30000$ and $\nu = 0.4$. The membrane is clamped at $x = 0$ and subjected to a a volume force $f = (0, -0.75)$ and shearing load $g = (0, 10)$ at $x = 35$. Figures 5–6 show initial (undeformed) and deformed configurations of the membrane, for a mesh with 995 nodes (i.e. 1990 degrees of freedom). The magnification factor, for the displacement field, is 20. The grey tones visualize the approximate Von Mises effective stress. The deformed configuration shows peak stresses at corners $(0, -5)$ and $(0, 5)$ as expected.

8. Conclusion

We have demonstrated that, for solving isotropic linear elasticity problems in Matlab with the finite element method, the vectorized code is much more efficient than a standard implementation with a loop over triangles.

Further work is underway to derive vectorized codes for the three-dimensional linear elasticity using tetrahedral elements. The main difficulty is to invert analytically, with *easy to implement formulas*, the matrix

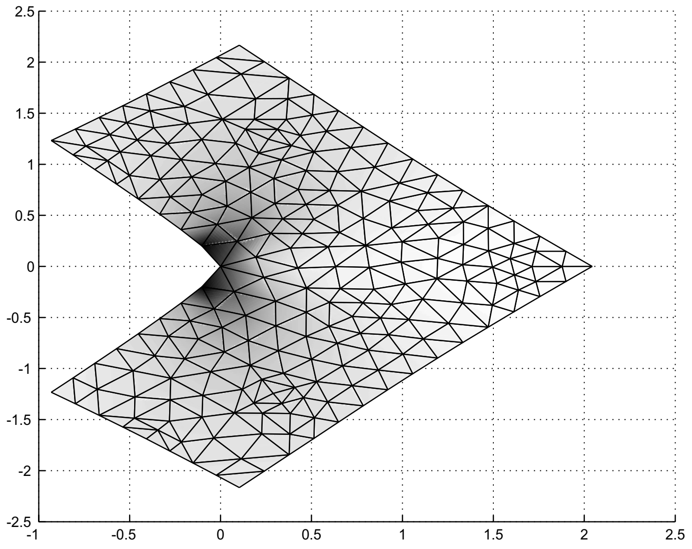


Fig. 4. Deformed mesh for the L-shape problem.

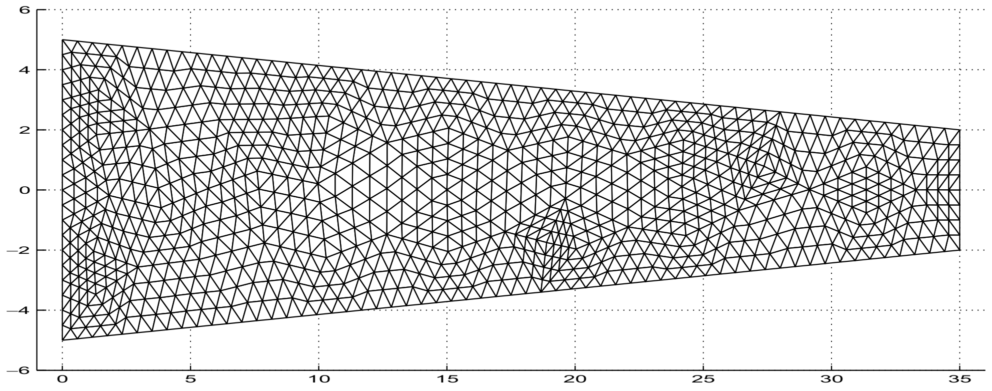


Fig. 5. Elastic Membrane: Initial configuration.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}$$

where $\{(x_i, y_i, z_i)\}_{i=1,\ldots,4}$ are the vertices of a tetrahedron. Indeed, if $\{\varphi_i\}_i$ are the linear basis functions on a tetrahedron, their gradients are given by (see e.g. [2])

$$\begin{bmatrix} \nabla\varphi_1^t \\ \nabla\varphi_2^t \\ \nabla\varphi_3^t \\ \nabla\varphi_4^t \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

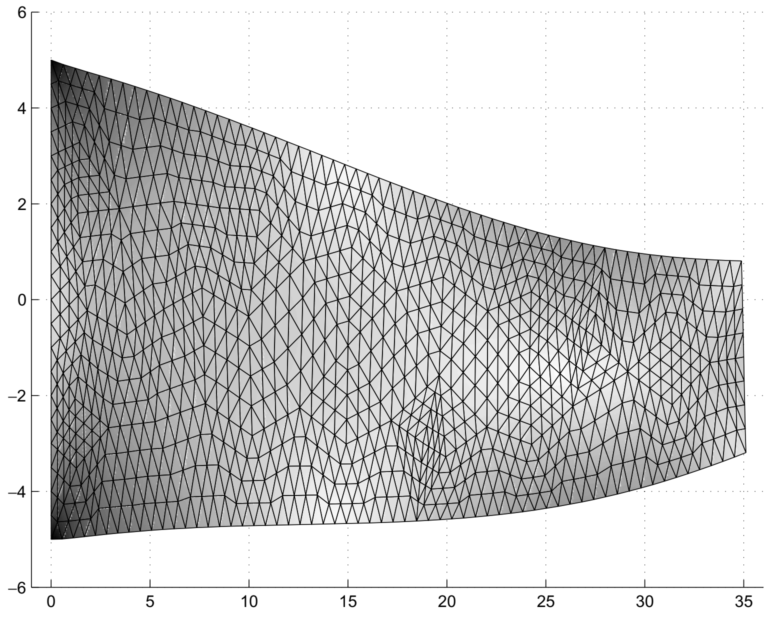


Fig. 6. Elastic membrane: deformed configuration.

Appendix

Main program for the L-shape problem

```
%------ Two-Dimensional linear elasticity
%       Example 1: L-Shape problem
%---------------------------------------
% Elastic constants
E=1e5; nu=0.3;
lam=E*nu/((1+nu)*(1-2*nu)); mu=E/(2*(1+nu));
%
Penalty=10^20;
% Mesh
load lshape231 p t ibcd
n=size(p,2); nn=2*n;
% Exact solution
ue=zeros(nn,1);
[th,r]=cart2pol(p(1,:),p(2,:));
alpha=0.544483737; omg=3*pi/4; ralpha=r.^alpha/(2*mu);
C2=2*(lam+2*mu)/(lam+mu); C1=-cos((alpha+1)*omg)/cos((alpha-1)*omg);
ur=ralpha.*(-(alpha+1)*cos((alpha+1)*th)+(C2-alpha-1)*C1*cos((alpha-1)*th));
ut=ralpha.*( (alpha+1)*sin((alpha+1)*th)+(C2+alpha-1)*C1*sin((alpha-1)*th));
ue(1:2:end)=ur.*cos(th)-ut.*sin(th);
ue(2:2:end)=ur.*sin(th)+ut.*cos(th);
% Boundary conditions
nnb=2*length(ibcd);
ibc=zeros(nnb,1); ibc(1:2:end)=2*ibcd-1; ibc(2:2:end)=2*ibcd;
ubc=zeros(nnb,1); ubc(1:2:end)=ue(2*ibcd-1); ubc(2:2:end)=ue(2*ibcd);
% Assembly of the Stiffness matrix
K=elas2dmat1(p,t,E,nu);
% Right-hand side
```

```
f=zeros(nn,1);
% Penalization of Stiffness matrix and right-hand sides
K(ibc,ibc)=K(ibc,ibc)+Penalty*speye(nnb);
f(ibc)=Penalty*ubc;
% Solution by Gaussian elimination
u=K\f;
% Shear energy density & Von Mises effective stress
[Sh,Vms]=elas2dsvms(p,t,u,E,nu);
% Show the deformed mesh and shear energy density
elas2dshow(p,t,u,3000,Sh)
```

Main program for the membrane problem

```
%------ Two-Dimensional linear elasticity
%       Example 2: Elastic membrane
%----------------------------------------
% Elastic constants
E=30000; nu=0.4;
%
Penalty=10ˆ15;
% Mesh
load exple2mesh995 p t ibcd ibcneum
n=size(p,2); nn=2*n;
% Boundary conditions
nnb=2*length(ibcd);
ibc=zeros(nnb,1); ibc(1:2:end)=2*ibcd-1; ibc(2:2:end)=2*ibcd;
% Assembly of the Stiffness matrix
K=elas2dmat2(p,t,E,nu);
% Right-hand side: body forces
f1=zeros(n,1); f2=-.75*ones(n,1);
f=elas2drhs1(p,t,f1,f2);
% Right-hand side: Neumann forces
ibcn1=ibcneum(1,:); ibcn2=ibcneum(2,:); ibcn=union(ibcn1,ibcn2);
g1=zeros(n,1); g2=zeros(n,1); g2(ibcn)=10;
g=elas2drhs2(p,ibcneum,g1,g2);
clear ibcn ibcn1 ibcn2
% Penalization of Stiffness matrix and right-hand sides
K(ibc,ibc)=K(ibc,ibc)+Penalty*speye(nnb);
b=f+g; b(ibc)=0;
% Solution by Gaussian elimination
u=K\b;
% Shear energy density &  Von Mises effective stress
[Sh,Vms]=elas2dsvms(p,t,u,E,nu);
%
% Show the deformed mesh and shear energy density
elas2dshow(p,t,u,20,Vms)
```

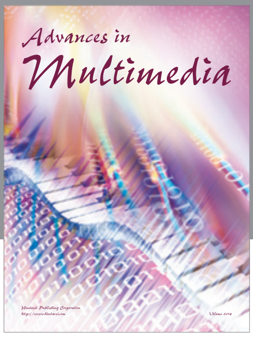
References

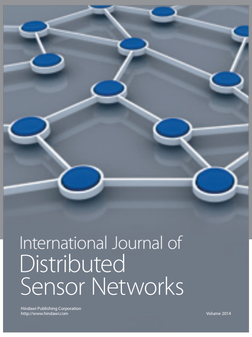
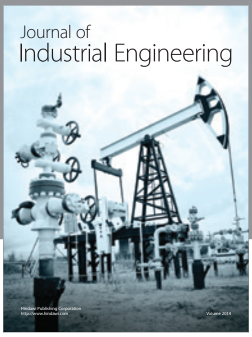
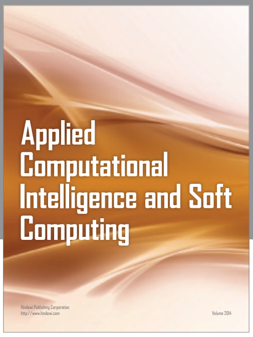
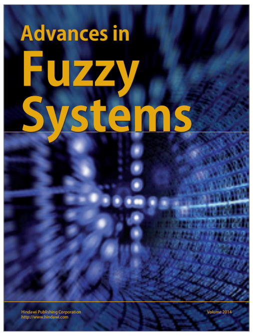
[1] J. Alberty, C. Carstensen and S.A. Funken, Remarks around 50 lines of matlab: short finite element implementation, *Numer Algorithms* **20** (1999), 117–137.

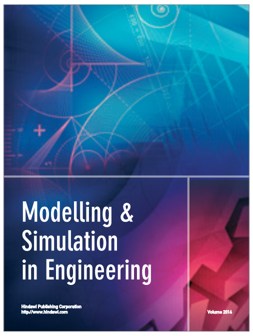
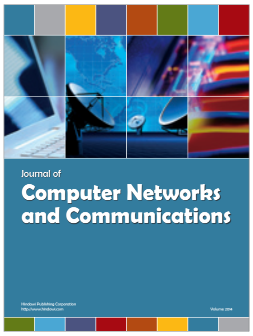
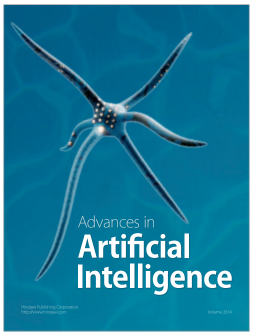
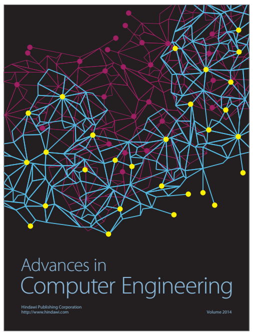
[2] J. Alberty, C. Carstensen, S.A. Funken and R. Klose, Matlab implementation of the finite element method in elasticity, *Computing* **69** (2002), 239–263.

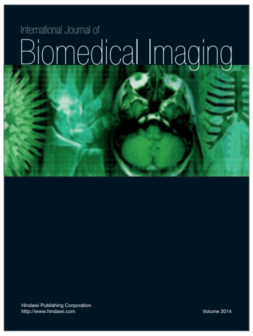
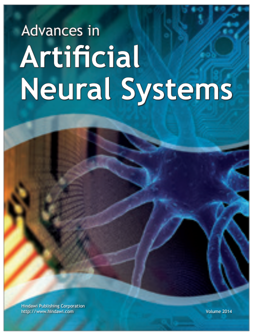
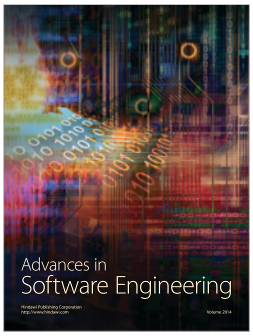
[3] P.G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1979.

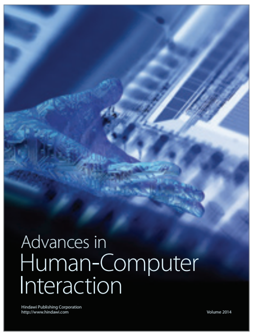
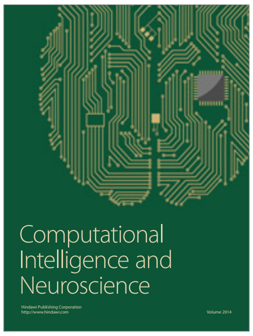
[4] P.-G. Ciarlet, Basic error estimates for elliptic problems, in: *Finite Element Methods* (*Part 1*), P.-G. Ciarlert and J.-L. Lions, eds, volume II of Handbook of Numerical Analysis, North-Holland, Amsterdam, 1991, pp. 23–343.
[5] Y.W. Kwon and H. Bang, *The Finite Element Method Using MATLAB*, CRC Press, New York, 2000.
[6] L. Langemyr, A. Nordmark, M. Ringh, A. Ruhe, Oppelstrup and M. Doro-Bantu, *Partial Differential Equations Toolbox User's Guide*, The Math Works, Inc., 1995.
[7] P.-O. Persson and G. Strang, A simple mesh generator in Matlab, *SIAM Rev* **42** (2004), 329–345.


Advances in
Multimedia

The Scientific World Journal

International Journal of
Distributed Sensor Networks

Journal of
Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in
Fuzzy Systems


Modelling & Simulation in Engineering

Journal of
Computer Networks and Communications

Advances in
Artificial Intelligence

Advances in
Computer Engineering

International Journal of
Computer Games Technology

International Journal of
Biomedical Imaging

Advances in
Artificial Neural Systems

Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of
Reconfigurable Computing

Journal of
Electrical and Computer Engineering