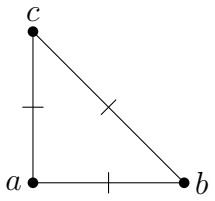
MATH 521

Assignment 5

February 28, 2018

Problem 1. Given three points $a, b, c \in \mathbb{R}^2$ that are not collinear (not all on one line) and that are sorted in anticlockwise order, we define

$$\begin{aligned} T &= \Delta(a,b,c) \\ P &= P_2(T) \\ L &= \Bigg\{ p \mapsto p(a), p \mapsto p(b), p \mapsto p(c), \\ &\qquad p \mapsto \frac{\partial p}{\partial n}\left(\frac{a+b}{2}\right), p \mapsto \frac{\partial p}{\partial n}\left(\frac{b+c}{2}\right), p \mapsto \frac{\partial p}{\partial n}\left(\frac{c+a}{2}\right)\Bigg\} \subset P* \end{aligned}$$



(a) Show that prescribed data for

$$p \mapsto p(a), p \mapsto p(b), p \mapsto p(c), p \mapsto \frac{\partial p}{\partial n}\left(\frac{a+b}{2}\right), p \mapsto \frac{\partial p}{\partial n}\left(\frac{b+c}{2}\right), p \mapsto \frac{\partial p}{\partial n}\left(\frac{c+a}{2}\right)$$

uniquely determines any $p \in P$. You don't have to show that such a p always exists.

Solution. A general second order polynomial in two-dimensions takes the form

$$p(x) = A_1 x_1^2 + A_2 x_1 x_2 + A_3 x_2^2 + A_4 x_1 + A_5 x_2 + A_6, \tag{1}$$

where $x = (x_1, x_2) \in \mathbb{R}^2$, as the space $P_2(T)$ is given by

$$P_2(T) = \text{span}\{1,\, x_1,\, x_2,\, x_1 x_2,\, x_1^2,\, x_2^2\}.$$

Since p has 6 unknown coefficients, 6 conditions on p are required in order to determine the coefficients A_i (assuming non-degeneracy of the prescribed conditions). The prescribed conditions are those in L, which we will show produces a unique polynomial p.

Suppose we take L to be the special case L_0 of all-zero conditions, where

$$p_0(a) = p_0(b) = p_0(c) = \frac{\partial p_0}{\partial n}\left(\frac{a+b}{2}\right) = \frac{\partial p_0}{\partial n}\left(\frac{b+c}{2}\right) = \frac{\partial p_0}{\partial n}\left(\frac{c+a}{2}\right) = 0. \tag{2}$$

Then, if we can show that the polynomial $p_0 \in P_2(T)$ which satisfies L_0 is exactly the zero solution $p_0 \equiv 0$, we will have shown uniqueness for the general case of p satisfying L due to the fact that we have linear degrees of freedom A_i.

We start by considering the normal derivative conditions. Since p_0 is second order in x_1 and x_2, it follows that the normal derivative $\frac{\partial p_0}{\partial n} = \nabla p_0 \cdot n$ is a linear function of x_1 and x_2, as

$$\nabla p_0 = \begin{bmatrix} 2A_1 x_1 + A_2 x_2 + A_4 \\ A_2 x_1 + 2A_3 x_2 + A_5 \end{bmatrix} \tag{3}$$

is a linear function in x_1 and x_2, and n is a constant vector.

Let us consider the tangential component of the gradient. Denote t the unit vector pointing from a to b. Then, by the fundamental theorem of vector calculus,

$$\begin{aligned}\int_{a\to b} \nabla p_0 \cdot d\vec{s} &= \int_{a\to b} \frac{\partial p_0}{\partial t} ds \\ &= p_0(b) - p_0(a) \\ &= 0.\end{aligned}$$

Thus, the *average* value of the tangential component of the gradient, $\frac{\partial p_0}{\partial t}$, must be zero along the line segment. Now, since $\frac{\partial p_0}{\partial t}$ is linear, this means that $\frac{\partial p_0}{\partial t}$ must be zero at the midpoint:

$$\frac{\partial p_0}{\partial t}\left(\frac{a+b}{2}\right) = 0.$$

We are given that $\frac{\partial p_0}{\partial n}$ is zero at the midpoint as well, and so the full gradient ∇p_0 is zero at the midpoint. This argument holds for all three sides, and so we have that

$$\nabla p_0 = 0$$

at all three midpoints. Note that $\nabla p_0 = 0$ implies all directional derivatives are zero at that point, including partial derivatives $\frac{\partial p_0}{\partial x}$ and $\frac{\partial p_0}{\partial y}$.

The gradient ∇p_0 given by equation 3 shows that $\frac{\partial p_0}{\partial x}$ and $\frac{\partial p_0}{\partial y}$ are given by plane equations (linear equations in x_1 and x_2). Since planes are defined by their values at three points, and we have that $\frac{\partial p_0}{\partial x} = \frac{\partial p_0}{\partial y} = 0$ at all three edge midpoints, it must be that $\frac{\partial p_0}{\partial x} \equiv 0$ and $\frac{\partial p_0}{\partial y} \equiv 0$ on all of T, and thus $\nabla p_0 \equiv 0$ on all of T.

Finally, $\nabla p_0 \equiv 0$ implies that p_0 must be constant on T, and since p_0 takes the value 0 at all three corners, it must be that $p_0 \equiv 0$ identically.

Uniqueness of p for generic data L follows, as previously stated.

□

(b) Now let Ω^h be a domain with a regular triangulation T^h such that

$$\bar{\Omega}^h = \bigcup_{T \in T^h} T.$$

Is the space

$$V^h = \left\{ v^h \colon \bar{\Omega}^h \to \mathbb{R} \;\middle|\; v^h|_T \in P_2(T), v^h \text{ is continuous in all vertices}, \frac{\partial v^h}{\partial n} \text{ is continuous in all edge midpoints} \right\}$$

H^1-conforming, i.e. is $V^h \subset H^1(\Omega^h)$? *Hint:* Check if there may be any jumps of v^h across triangle edges.

Solution. We start by considering a generic triangle edge. We use the notation that this edge connects points a and b in the plane which belong to adjacent triangles T_1 and T_2, and we denote $v_1^h \in P_2(T_1)$ and $v_2^h \in P_2(T_2)$ the quadratic interpolating polynomials on the respective triangles, each taking the form 1.

Now, we know that v_1^h and v_2^h match at the end points, so the difference $e^h = v_1^h - v_2^h$ is zero at a and b. Additionally, since $\frac{\partial v_1^h}{\partial n} = \frac{\partial v_2^h}{\partial n}$ at the midpoint, $\frac{\partial e^h}{\partial n} = 0$ at the midpoint. Using the same argument as in Problem 1(a), due to the linearity of the gradient and the fact that e^h is zero at both endpoints, $\frac{\partial e^h}{\partial t} = 0$ at the midpoint and so $\nabla e^h = 0$ at the midpoint.

However, we have no guarantees for the function values away from the edges; there are too many degrees of freedom. Consider the following counterexample:

$$v_1^h = (x_1^2 - 1) - x_2 + 1$$
$$v_2^h = 2(x_1^2 - 1) + x_2 + 1$$

where T_1 has vertices $a_1 = (-1, 0)$, $b_1 = (1, 0)$, and $c_1 = (0, 1)$, and T_2 has vertices $a_2 = (-1, 0)$, $b_2 = (0, -1)$, and $c_2 = (1, 0)$. T_1 and T_2 are the top and bottom half, respectively, of the "diamond" with vertices at $(\pm 1, 0)$ and $(0, \pm 1)$, and are connected by their shared edge e which lies on the line $x_2 = 0$.

Now, it is clear that on $v_1^h(\pm 1, 0) = v_2^h(\pm 1, 0) = 1$, i.e. they are equal at the endpoints of e. Additionally, since the normal direction to e is $-x_2$ on T_1 and $+x_2$ on T_2, we have that $\frac{\partial v_1^h}{\partial n}(0, 0) = \frac{\partial v_2^h}{\partial n}(0, 0) = 1$, i.e. the normal derivatives are continuous at the midpoints. However, clearly $v_1^h \neq v_2^h$ anywhere else on e (where $x_2 = 0$).

In conclusion, V^h **is not H^1-conforming**, due to the fact that jumps of v^h across triangle edges is indeed possible in general.

□

Problem 2. We will now complete our finite-element solver for the linear elasticity problem

$$\begin{aligned} -c\Delta u + au &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega \end{aligned} \tag{4}$$

(a) Remove lines 1-10 from `discretiseLinearElasticity.m` and uncomment the sections of code that are currently commented out. Complete the missing commands, including the subfunction `assembleStiffness`. Also inspect the `assembleLoad` subfunction.

Solution. The finished function `discretiseLinearElasticity.m` and subfunction `assembleStiffness` is included in AppendixA. □

(b) Write a script `hw6.m` which

- Solves the linear elasticity problem on Ω^h, which you may choose from kiwi.mat, maple.mat, pi.mat, ubc.mat. You may also select your own data for $f(x_1, x_2)$, $g(x_1, x_2)$, a and c.

  Hint: You have to set `GammaD = @(x1,x2) true(size(x1))`. For debugging, you might want to use `video10.mat` and check the sparsity patterns of the various matrices.

- Calculates the L^2, H^1, and B energy norms of the solution, where B is the bilinear form corresponding to the elliptic operator.

- Creates undistorted plots of the mesh, the force f, and the solution u^h.

Solution. The script `hw6.m` is included in Appendix B.

I chose to use the `maple.mat` dataset. The mesh and sparsity patterns of the mass and stiffness matrices are plotted below. For the forcing function, I used a constant forcing $f \equiv 1$ with fixed Dirichlet boundaries $g \equiv 0$, as it is the most intuitive to tell if the solution looks right or not (should bend more in the flat regions than narrow regions/near edges).

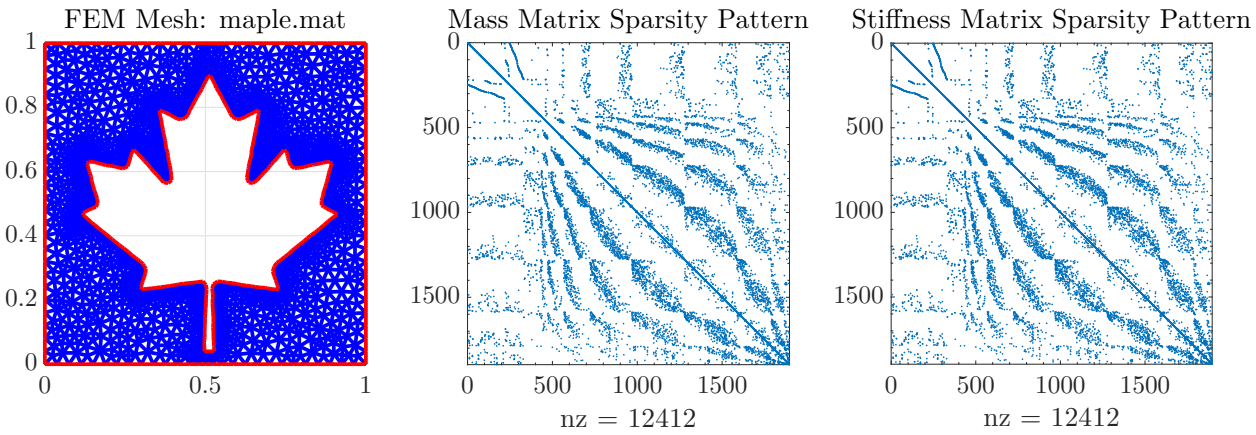


Figure 1: Mesh and sparsity patterns of the mass and stiffness matrices for the `maple.mat` data set.

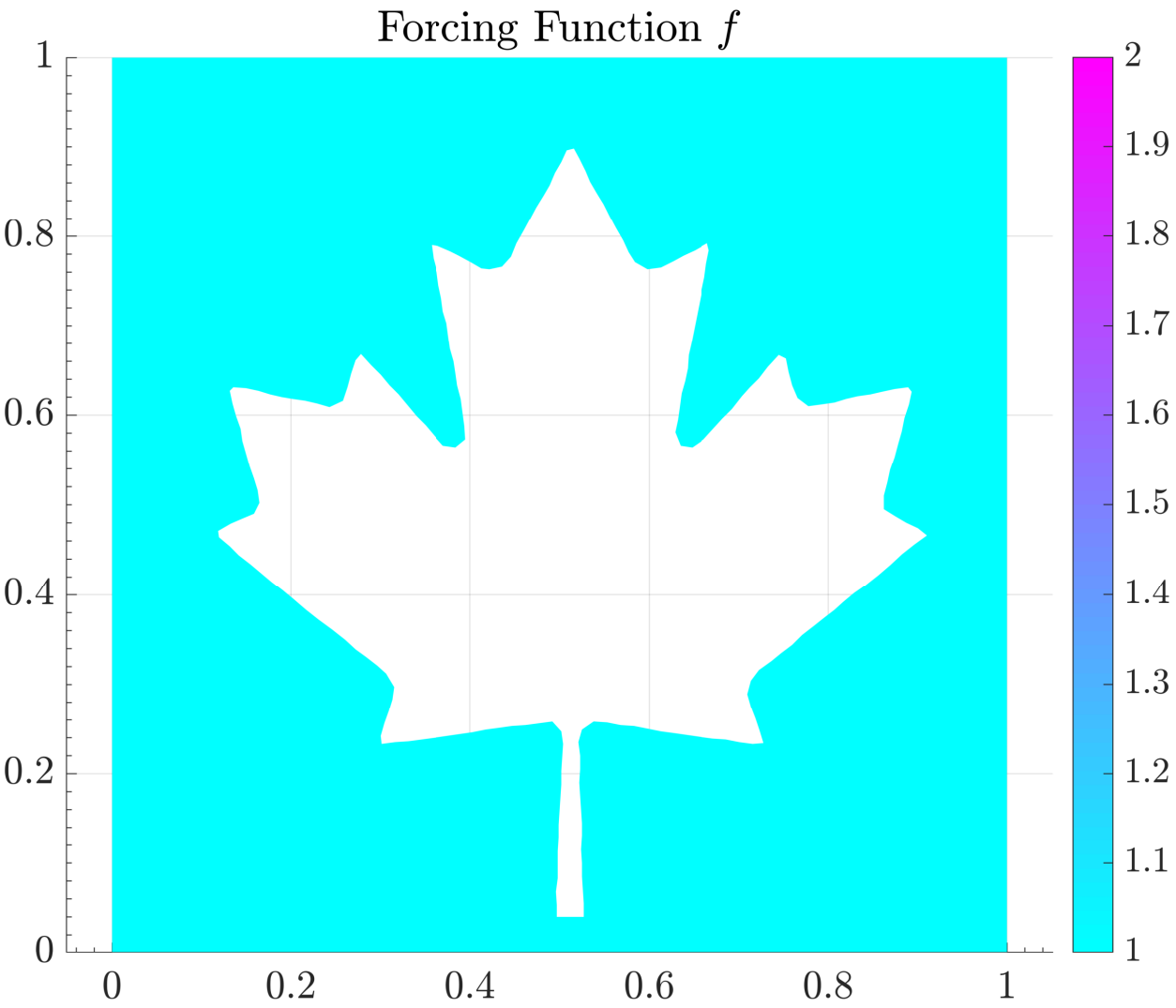


Figure 2: Plot of the constant forcing function $f \equiv 1$ on the `maple.mat` finite elements mesh.

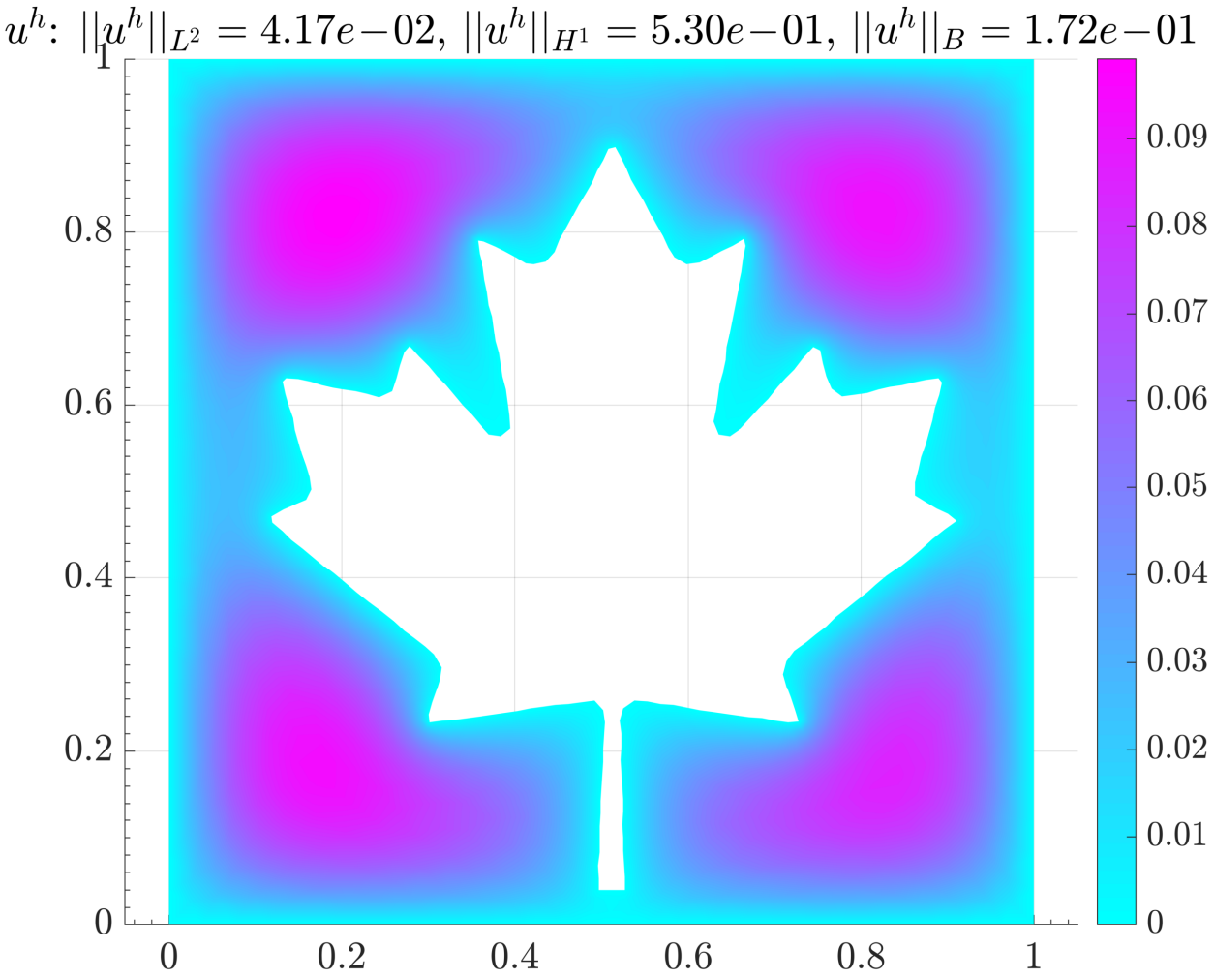


Figure 3: Plot of the solution function u^h corresponding to the constant forcing function $f \equiv 1$ on the `maple.mat` finite elements mesh, with relevant norms computed.

□

(c) What problem do you solve numerically when you set `GammaD = @(x1,x2) false(size(x1))`? Analyse the code to infer its weak formulation.

Solution. Setting `GammaD = @(x1,x2) false(size(x1))` would result in no Dirichlet boundary points prescribed with the boundary data $g(x_1, x_2)$, which result in all boundary points being free.

In terms of code, these means that the matrix free projection matrix `Pf` would be the identify matrix. The reduced linear system defined by `A` and `b` would become the full system and would be equivalent to

```
A = c*Kbar + a*Mbar;
b = fbar;
```

The weak form in it's discretized form is therefore given by

$$\sum_{i=1}^{N} \left(c \int_\Omega \nabla \phi_i^h \cdot \nabla \phi_j^h \, \mathrm{d}x + a \int_\Omega \phi_i^h \phi_j^h \, \mathrm{d}x \right) u_j^h = \int_\Omega f \phi_i^h \, \mathrm{d}x,$$

where *all* values u_j^h are free. The non-discretized form is then given by

$$c \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x + a \int_\Omega uv \, \mathrm{d}x = \int_\Omega fv \, \mathrm{d}x. \tag{5}$$

□

Your Learning Progress What is the one most important thing that you have learnt in this assignment?

That it is very non-trivial to determine properties of finite elements! Especially with regards to uniqueness and existence proofs.

Any new discoveries or achievements towards the objectives of your course?

I'm starting to like finite element methods a lot more! Quite finicky, but worthwhile.

What is the most substantial new insight that you have gained from this course this week? Any *aha moment*?

Only that it is not at all obvious whether or not a finite element will be a valid one or not without doing the analysis.

Appendix A

The `discretiseLinearElasticity.m` function:

```
function [A,b,uD,Kbar,Mbar,Pf,PD] = discretiseLinearElasticity(c,a,f,g,GammaD,msh)
%DISCRETISELINEARELASTICITY(c,a,f,g,GammaD,msh) Applies linear finite
% elements on the triangulation defined in the structure msh to discretise
% the equation
%   -c*Lap(u) + a*u = f
% with Dirichlet boundary conditions u = g on GammaD.
%   Input:
%       c: a positive real number
%       a: a positive real number
%       f: a function handle for the inhomogeneity f(x1,x2)
%       g: a function handle for the boundary values g(x1,x2)
%       GammaD: a function handle to a function of x1 and x2, which returns
%       true, if Dirichlet conditions are imposed at the boundary point
%       (x1,x2) and false otherwise
%       msh: a mesh structure with fields P, E, T
%   Output:
%       A: a sparse array corresponding to the discrete elliptic operator
%       b: a column vector for the discretised right hand side and boundary
%       terms
%       uD: a column vector with the prescribed values of the solution on
%       the Dirichlet boundary
%       Kbar: the sparse stiffness matrix, including all boundary points
%       Mbar: the sparse mass matrix, including all boundary points
%       Pf: a sparse matrix that projects onto all free nodes
%       PD: a sparse matrix that projects onto all Dirichlet nodes

% ------------------------------------------------------------------------
% COMPUTE EXTRA MESH DATA
% ------------------------------------------------------------------------

% edge vectors
msh.D32 = msh.P(:,msh.T(3,:)) - msh.P(:,msh.T(2,:));
msh.D13 = msh.P(:,msh.T(1,:)) - msh.P(:,msh.T(3,:));
msh.D21 = msh.P(:,msh.T(2,:)) - msh.P(:,msh.T(1,:));

% row vector of triangle areas = [|T_1|, |T_2|, |T_3|, ..., |T_nt|]
msh.A = (msh.D13(1,:).*msh.D21(2,:) - msh.D21(1,:).*msh.D13(2,:))./2;

% number of points
msh.np = size(msh.P,2); % includes boundary points

% number of triangles
msh.nt = size(msh.T,2);

% ------------------------------------------------------------------------
% ASSEMBLE THE DISCRETE SYSTEM
% ------------------------------------------------------------------------
```

```
% complete system including all boundary points
Kbar = assembleStiffness(msh);
Mbar = assembleMass(msh);
fbar = assembleLoad(f,msh);

% ------------------------------------------------------------------------
% ELIMINATE DIRICHLET BOUNDARY CONDITIONS
% ------------------------------------------------------------------------

[Pf,PD,uD] = eliminateDirichletBC(g,GammaD,msh);

% contributions from points with prescribed Dirichlet data
kD = Pf*Kbar*PD'*uD;
mD = Pf*fbar - kD;

% reduced system for the free points only
K = Pf*Kbar*Pf';
M = Pf*Mbar*Pf';
A = c*K + a*M;
b = kD + mD;

end

function Mbar = assembleMass(msh)
%ASSEMBLEMASS Assembles the complete mass matrix including all boundary
% points, discretised with linear finite elements

% need all pairs points, including pairing with themselves:
pointRows = [1,2,3];
pointsAndNeighboursRows = [1,2,3, 2,3,1, 3,1,2];

% Assembly II and JJ arrays such that (II(idx), JJ(idx)) pairs contain all
% pairs of triangle points with themselves and their neighbours
%   NOTE: There will certainly be repeated points, as triangles share
%         vertices. However this is accounted for in the sparse constructor
%         as it adds all repeated indices together, i.e. all contributions
%         from all element mass matrices
II = msh.T(pointsAndNeighboursRows, :);
JJ = repmat( msh.T(pointRows, :), 3, 1);

% First row will be pairs of points with themselves, so value is 2/12*|T|
% 2nd/3rd rows are pairs of points with other points, so value is 1/12*|T|
mij = repmat( [ (2/12) * msh.A
                (1/12) * msh.A
                (1/12) * msh.A ], 3, 1);

% complete mass matrix
Mbar = sparse(II,JJ,mij,msh.np,msh.np);

end

function Kbar = assembleStiffness(msh)
%ASSEMBLESTIFFNESS Assembles the complete stiffness matrix including all
% boundary points, discretised with linear finite elements
```

```
% arrays of row/column indices: [9 x nT] arrays (note semicolons)
II = [msh.T(1,:); msh.T(1,:); msh.T(1,:);
      msh.T(2,:); msh.T(2,:); msh.T(2,:);
      msh.T(3,:); msh.T(3,:); msh.T(3,:)]; % row indices

JJ = [msh.T(1,:); msh.T(2,:); msh.T(3,:);
      msh.T(1,:); msh.T(2,:); msh.T(3,:);
      msh.T(1,:); msh.T(2,:); msh.T(3,:)]; % column indices

% entries of the element stiffness matrices (before division by areas):
% [9 x nT] arrays (note semicolons)
kij = [ dot(msh.D32,msh.D32,1); dot(msh.D32,msh.D13,1); dot(msh.D32,msh.D21,1);
        dot(msh.D13,msh.D32,1); dot(msh.D13,msh.D13,1); dot(msh.D13,msh.D21,1);
        dot(msh.D21,msh.D32,1); dot(msh.D21,msh.D13,1); dot(msh.D21,msh.D21) ];

% computationally efficient division by 4*|T|
kij = bsxfun( @rdivide, kij, 4*msh.A );

% complete stiffness matrix
Kbar = sparse(II,JJ,kij,msh.np,msh.np);

end

function qf = assembleLoad(f,msh)
%ASSEMBLELOAD Assembles the complete load vector including all boundary
%points, discretised with linear finite elements

% triangle midpoints
xm = (msh.P(:,msh.T(1,:)) + msh.P(:,msh.T(2,:)) + msh.P(:,msh.T(3,:)))./3;
fm = f(xm(1,:).',xm(2,:).'); % f(xm)

% quadrature operator (midpoint rule) on one element
II  = msh.T(1:3,:); % row indices
JJ  = repmat(1:msh.nt,3,1); % column indices
qij = repmat(msh.A./3,3,1); % midpoint rule on each triangle

% full quadrature operator
Q = sparse(II,JJ,qij,msh.np,msh.nt);

% complete load vector
qf = Q*fm;

end

function [Pf,PD,uD] = eliminateDirichletBC(g,GammaD,msh)
%ELIMINATEDIRICHLETBC Calculates the vector uD = g(x1,x2) for all boundary
%points (x1,x2) on the Dirichlet section GammaD of the boundary. Also
%assembles the projection matrices Pf and PD.

% indices of all boundary points
indGamma = unique(msh.E(1:2,:));

% coordinates of all boundary points
```

```
Gamma = msh.P(:,indGamma);

% points on that part of the boundary where Dirichlet conditions are
% imposed
indD = indGamma(GammaD(Gamma(1,:),Gamma(2,:))); % indices of Dirichlet points
nD = length(indD); % number of Dirichlet points

% all other points of the mesh
indf = setdiff(1:msh.np,indD); % indices of free points
nf = msh.np - nD; % number of free points

% projection onto the Dirichlet points
PD = sparse(1:nD,indD,ones(1,nD),nD,msh.np);

% projection onto the free points
Pf = sparse(1:nf,indf,ones(1,nf),nf,msh.np);

% boundary values of u on the Dirichlet points
uD = g(msh.P(1,indD).',msh.P(2,indD).');

end
```

Appendix B

The `hw6.m` script:

```
%% clear workspace
clear all
close all force

%% set params
c = 0.1;
a = 1;
f = @(x1,x2) ones(size(x1)); % uniform forcing
g = @(x1,x2) zeros(size(x1)); % fixed boundaries
GammaD = @(x1,x2) true(size(x1));
% GammaD = @(x1,x2) false(size(x1));

%% choose a mesh
% fname = 'kiwi.mat';
fname = 'maple.mat';
% fname = 'pi.mat';
% fname = 'ubc.mat';
% fname = 'video10.mat';
msh = load(fname);

%% get discrete linear elasticity operators
[A,b,uD,Kbar,Mbar,Pf,PD] = discretiseLinearElasticity(c,a,f,g,GammaD,msh);

%% solve system
uN = A\b;
uh = Pf'*uN + PD'*uD;

%% calculate various norms
L2normsq = uh'*Mbar*uh;
L2gradsq = uh'*Kbar*uh;

L2norm = sqrt(L2normsq);
H1norm = sqrt(L2normsq + L2gradsq);
Bnorm = sqrt(c*L2gradsq + a*L2normsq);

%% undistorted plots of mesh, force function f, and solution uh
figure, subplot(3,1,1);
pdeplot(msh.P,msh.E,msh.T);
axis equal
xlim([min(msh.P(1,:)), max(msh.P(1,:))]);
ylim([min(msh.P(2,:)), max(msh.P(2,:))]);
title(sprintf('FEM Mesh: %s',fname))

subplot(3,1,2);
spy(Mbar);
title('Mass Matrix Sparsity Pattern');

subplot(3,1,3);
```

```
spy(Kbar);
title('Stiffness Matrix Sparsity Pattern');

%% undistorted plots of force function f and solution uh
fh = f(msh.P(1,:),msh.P(2,:)).';
figure, pdeplot(msh.P,msh.E,msh.T,'xydata',fh);
axis equal, title('Forcing Function $f$');

figure, pdeplot(msh.P,msh.E,msh.T,'xydata',uh);
axis equal
NUM2STR = @(x) strrep(sprintf('%.2e',x),'e-','e\!-\!');
titlestr = ['$u^h$: ', ...
    '$||u^h||_{L^2} = ', NUM2STR(L2norm), '$, ' ...
    '$||u^h||_{H^1} = ', NUM2STR(H1norm), '$, ' ...
    '$||u^h||_{B} = ', NUM2STR(Bnorm), '$' ];
title(titlestr);
```