R. Hiptmair
L. Scarabosio
C. Urzua Torres

Spring Term 2015

# Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

## Homework Problem Sheet 6

### Problem 6.1 Linear Finite Element implementation for 2D reaction-diffusion

In [NPDE, Section 3.3] we have studied the algorithmic aspects related to the linear finite element Galerkin discretization of two-dimensional, second-order linear variational problems posed on the Sobolev space $H^1(\Omega)$. In [NPDE, Section 3.4], you have seen the extension to more general finite element subspaces of $H^1(\Omega)$. The present exercise is meant to make you more familiar with the techniques learned in class.

To this end, we consider the following Neumann problem on the unit square $\Omega = [0,1]^2$ with homogeneous Neumann data and reaction term (cf. [NPDE, Eq. (3.1.4)]):

$$u \in H^1(\Omega): \quad \underbrace{\int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v + u\, v \;\mathrm{d}\boldsymbol{x}}_{:=\mathsf{a}(u,v)} = \underbrace{\int_\Omega f v \;\mathrm{d}\boldsymbol{x}}_{:=\ell(v)} \quad \forall v \in H^1(\Omega). \tag{6.1.1}$$

We want to develop an efficient MATLAB code for the discretization of (6.1.1) on a triangular mesh using linear finite elements.

The mesh data structure contains the following fields, see also [NPDE, § 3.3.3]:

- `Mesh.Coordinates`: $N \times 2$ matrix, $i$-th row containing the coordinates of the $i$-th vertex, $i \in \{1, \dots, N\}$;
- `Mesh.Elements`: $M \times 3$-matrix, $j$-th row

Recall that for piecewise linear finite elements on triangular meshes the so-called local shape functions ($\rightarrow$ [NPDE, Def. 3.4.19]) agree with the barycentric coordinate functions $\lambda_1$, $\lambda_2$, and $\lambda_3$ of the triangles, see [NPDE, Fig. 84].

**(6.1a)** Implement the function

```
grad = gradbarycoords(Vertices)
```

which returns the values of the gradients of local shape functions (i.e. the barycentric coordinate functions) $\lambda_i(\boldsymbol{x})$, $i = 1, 2, 3$, in the vertices with coordinates contained in the $3 \times 2$-matrix `Vertices`. The output `grad` is a $2 \times 3$ matrix containing the gradients of the shape functions

evaluated at the vertices (the first column contains the gradient of $\lambda_1$, the second one the gradient of $\lambda_2$ and the last one the gradient of $\lambda_3$).

**Solution:** See Listing 6.1 for the code, see also [NPDE, Code 3.3.24].

Listing 6.1: Implementation for `gradbarycoords`

```
function G = gradbarycoords(Vertices)
% MATLAB function computing the gradients of barycentric coordinate functions
% on a triangle whose vertex positions are passed in the rows of a
% 3×2-matrix. The components of the gradients are returned in the
% columns of a 2×3-matrix.

% Solve for the coefficients of the barycentric coordinate functions, see \eqref{eq:lambdalse}
X = inv([ones(3,1),Vertices]);
G = X(2:3,:); % extract gradients
```

**(6.1b)** Implement the routine

`function Aloc = Elmat_Lapl_LFE(Vertices)`

to compute the element matrix associated to the bilinear form

$$\mathsf{a}_1(u,v) = \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x}, \quad u,v \in H^1(\Omega),$$

and linear Lagrangian finite elements.

Here, `Vertices` is a $3\times 2$-vector providing the coordinates of the element vertices. The function should return a $3 \times 3$ matrix `Aloc` containing the element matrix.

**Solution:** See Listing 6.2 for the code, see also [NPDE, Code 3.3.25].

Listing 6.2: Implementation for `Elmat_Lapl_LFE`

```
function Aloc = Elmat_Lapl_LFE(Vertices)
% Computation of element matrix for piecewise linear Lagrangian finite
% elements and a triangular elements, whose vertex positions are passed
% in the rows of the \texttt{Vertices} argument.

% Compute area of triangle
area = 0.5*det(Vertices(2:3,:) - kron([1;1],Vertices(1,:)));
% Compute gradients of barycentric coordinate functions,
% see \cref{mc:gradbarycoords}
G = gradbarycoords(Vertices);
```

```
% Compute inner products of gradients through matrix
   multiplication
Aloc = area*G'*G;
```

**(6.1c)** Implement the routine

```
function Aloc = Elmat_Mass_LFE(Vertices)
```

to compute the element matrix associated to the bilinear form

$$\mathsf{a}_2(u,v) = \int_\Omega u\,v\,\mathrm{d}\boldsymbol{x}, \quad u,v \in L^2(\Omega)\,,$$

and linear Lagrangian finite elements on triangular elements. The input and output arguments are the same as for `Elmat_Lapl_LFE`.

HINT: Compute the entries of the element matrix by analytic evaluation of the two-dimensional integrals. In order to avoid cumbersome computations, you may rely on the general formula from [NPDE, Lemma 3.6.61].

**Solution:** See Listing 6.3 for the code.

Listing 6.3: Implementation for `Elmat_Mass_LFE`

```
function Mloc = Elmat_Mass_LFE(Vertices)

%   Copyright 2005-2005 Patrick Meury
%   SAM - Seminar for Applied Mathematics
%   ETH-Zentrum
%   CH-8092 Zurich, Switzerland

% Compute area of triangle
  area = 0.5*det(Vertices(2:3,:) - kron([1;1],Vertices(1,:)));

  % Compute local mass matrix

  Mloc = area/12*[2 1 1; 1 2 1; 1 1 2];

return
```

**(6.1d)** Implement the routine

```
function Aloc = Elmat_LaplMass_LFE(Vertices)
```

to compute the element matrix associated to the bilinear form in (6.1.1) and linear Lagrangian finite elements.

The input and output arguments are the same as for `Elmat_Lapl_LFE`.

HINT: Combine the results from tasks (6.1b) and (6.1c).

**Solution:** See Listing 6.4 for the code.

Listing 6.4: Implementation for `Elmat_LaplMass_LFE`

```
function Aloc = Elmat_LaplMass_LFE(Vertices)

Aloc = Elmat_Lapl_LFE(Vertices) + Elmat_Mass_LFE(Vertices);
```

**(6.1e)** Implement the routine

`philoc = localLoadLFE(Vertices,FHandle)`

to compute the element vector `philoc` associated to the linear form in (6.1.1), for linear Lagrangian finite elements, see [NPDE, Section 3.3.6].

The input argument `Vertices` is a $3 \times 2$-matrix containing the element vertices, and `FHandle` is a function handle to the function $f$. You can assume that `FHandle` accepts as input $K \times 2$-matrices, for which each row $i = 1, \dots, K$, $K \in \mathbb{N}$, contains the coordinates of a point, and then it returns the values of $f$ in those points as a column vector of length $K$.

Since $f$ is given in procedural form, the entries of the element vectors can be computed only approximately by means of numerical quadrature, *cf.* [NPDE, § 3.3.44]. Use *composite edge midpoint quadrature rule* that, for a triangle $K$ with vertices $\boldsymbol{a}^1$, $\boldsymbol{a}^2$, $\boldsymbol{a}^3$, and edge midpoints $\boldsymbol{m}^1 := \frac{1}{2}(\boldsymbol{a}^2 + \boldsymbol{a}^3)$, $\boldsymbol{m}^2 := \frac{1}{2}(\boldsymbol{a}^1 + \boldsymbol{a}^3)$, $\boldsymbol{m}^3 := \frac{1}{2}(\boldsymbol{a}^2 + \boldsymbol{a}^1)$, reads

$$\int_K \varphi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{|K|}{3}\big(\varphi(\boldsymbol{m}^1) + (\varphi(\boldsymbol{m}^2) + (\varphi(\boldsymbol{m}^3)\big) \,. \tag{6.1.2}$$

HINT: See [NPDE, Code 3.3.47] for a code performing the same task using the 2D trapezoidal quadrature rule [NPDE, Eq. (3.3.45)].

**Solution:** See Listing 6.5 for the code, see also [NPDE, Code 3.3.47].

Listing 6.5: Implementation for `localLoadLFE`

```
function philoc = localLoadLFE(Vertices,FHandle)

% Compute area of triangle
area = 0.5*det(Vertices(2:3,:) - kron([1;1],Vertices(1,:)));
% Evaluate source function for vertex location
philoc = zeros(3,1);

philoc(1) = FHandle(sum(Vertices([1
   2],:),1)/2)/2+FHandle(sum(Vertices([1 3],:),1)/2)/2;
philoc(2) = FHandle(sum(Vertices([1
   2],:),1)/2)/2+FHandle(sum(Vertices([2 3],:),1)/2)/2;
philoc(3) = FHandle(sum(Vertices([1
   3],:),1)/2)/2+FHandle(sum(Vertices([2 3],:),1)/2)/2;

% Scale with 1/3·area of triangle
philoc = philoc*area/3.0;
```

**(6.1f)** Implement an efficient MATLAB function

```
A = assemMat_LFE(Mesh,getElementMatrix)
```

that assembles the Galerkin matrix `A` associated to the bilinear form in (6.1.1), for linear Lagrangian finite elements. This routine receives in input the mesh data structure `Mesh` (as described at the beginning of the problem) and a function handle `getElementMatrix` to a function that expects a $3 \times 2$-array of vertex coordinates and returns a $3 \times 3$ element matrix.

HINT: Use the MATLAB's sparse matrix data format to store `A`. Remember the discussion in class about the efficient way of filling a sparse matrix.

**Solution:** See Listing 6.6 for the code, see also [NPDE, Code 3.3.33]

Listing 6.6: Implementation for `assemMat_LFE`

```
function A = assemMat_LFE(Mesh,getElementMatrix)
% Efficient assembly of global Galerkin matrix for piecewise linear
% Lagrangian finite elements on a triangular mesh without special
% treatment of boundaries and/or interfaces.

M = size(Mesh.Elements,1); % Obtain number of elements/cells
% Preallocate index and value vectors for the initialization
% of the sparse Galerkin matrix
I = zeros(9*M,1); J = zeros(9*M,1); A = zeros(9*M,1);

% Loop over elements and add local contributions
loc = 1:9; % Moving index into the vectors \texttt{I}, \texttt{J}, and \texttt{A}
for i = 1:M
  % Get local→global index mapping array for current element
  dofh = Mesh.Elements(i,:);
  % Extract vertices of current element
  Vertices = Mesh.Coordinates(dofh,:);
  % Compute element contributions
  Aloc = getElementMatrix(Vertices);
  % Insert contributions into temporary vectors.
  I(loc) = dofh([1 2 3 1 2 3 1 2 3]);
  J(loc) = dofh([1 1 1 2 2 2 3 3 3]);
  A(loc) = Aloc(:);
  % Advance indices into temporary vectors
  loc = loc+9;
end

A = sparse(I,J,A);
```

**(6.1g)** Implement the function

```
phi = assemLoad_LFE(Mesh,getElementVector,FHandle)
```

to assemble the right-hand side vector `phi` given the mesh structure `Mesh`, a handle to a function `getElementVector` expecting a $3 \times 2$ array of vertex coordinates as input and returning an element load vector as a column vector of size 3, and a handle `FHandle` to the function $f$.

HINT: The procedure is similar to the one for `assemMat_LFE`.

**Solution:** See Listing 6.7 for the code.

Listing 6.7: Implementation for `assemLoad_LFE`

```
function phi = assemLoad_LFE(Mesh,getElementVector,FHandle)
% Element oriented assembly of right hand side vector for Galerkin finite
% element discretization with piecewise linear Lagrangian finite elements.

N = size(Mesh.Coordinates,1); % get no. of vertices
M = size(Mesh.Elements,1);    % get no. of elements
phi = zeros(N,1);  % Preallocate memory

% Main assembly loop over cells of the mesh
for i = 1:M
  % Extract vertices of current element
  dofh = Mesh.Elements(i,:);
  Vertices = Mesh.Coordinates(dofh,:);
  % Compute element right hand side vector
  philoc = getElementVector(Vertices,FHandle);
  % Add contributions to global load vector
  phi(dofh) = phi(dofh) + philoc;
end
```

**(6.1h)** Implement the function

```
err = L2Err_LFE(Mesh,U,UHandle)
```

to compute the error $\|u - u_h\|_{L^2(\Omega)}$, where $u$ is the exact solution to (6.1.1), passed in the function handle `UHandle`, and $u_h$ is the discrete solution, passed through the coefficient vector `U` with respect to the nodal basis of $\mathcal{S}_1^0(\mathcal{M})$. The argument `Mesh` contains the mesh data structure.

To compute the integrals, use the 2D trapezoidal quadrature rule, see [NPDE, Eq. (3.3.45)].

**Solution:** See Listing 6.8 for the code.

Listing 6.8: Implementation for `L2Err_LFE`

```
function err = L2Err_LFE(Mesh,U,UHandle)
% L2ERR_LFE Discretization error in L2 norm for linear finite elements
% using 2D trapezoidal quadrature rule.
```

```

%   Copyright 2005-2005 Patrick Meury
%   SAM - Seminar for Applied Mathematics
%   ETH-Zentrum
%   CH-8092 Zurich, Switzerland

  nElements = size(Mesh.Elements,1);

  % Compute discretization error

  err = 0;
  for i = 1:nElements

    % Extract vertex numbers
    dofh = Mesh.Elements(i,:);

    % Extract vertex coordinates
    Vertices = Mesh.Coordinates(dofh,:);

    % Compute area of triangle
    area = 0.5*det(Vertices(2:3,:) -
       kron([1;1],Vertices(1,:)));

    % Evaluate solutions

    u_EX = UHandle(Vertices);
    u_FE = U(dofh);

    % Compute error on current element

    err = err+sum((u_EX-u_FE).^2,1)*area/3;

  end
  err = sqrt(err);

return
```

**(6.1i)** Implement the function

```
err = H1SErr_LFE(Mesh,U,gradUHandle)
```

to compute the error $|u - u_h|_{H^1(\Omega)}$, where $u$ is the exact solution to (6.1.1), for which the gradient is passed in the function handle `gradUHandle` (that returns a column vector), and $u_h$ is the discrete solution, passed through the coefficient vector `U`. Assume that, given a $K \times 2$-matrix of point coordinates, $K \in \mathbb{N}$, the function `gradUHandle` returns the value of $\mathbf{grad}\, u$ in these points in a $2 \times K$-matrix. The input argument `Mesh` contains the mesh data structure.

To compute the integrals, again rely on the 2D trapezoidal quadrature rule, see [NPDE, Eq. (3.3.45)].

**Solution:** See Listing 6.9 for the code.

Listing 6.9: Implementation for `H1SErr_LFE`

```
function err = H1SErr_LFE(Mesh,U,GradUHandle)
% H1SERR_LFE Discretization error in H1 semi-norm for linear finite
% elements using the 2D trapezoidal quadrature rule.

%   Copyright 2005-2005 Patrick Meury & Kah Ling Sia
%   SAM - Seminar for Applied Mathematics
%   ETH-Zentrum
%   CH-8092 Zurich, Switzerland

  nElements = size(Mesh.Elements,1);

  % Compute discretization error

  err = 0;
  for i= 1:nElements

    % Extract vertex numbers

    dofh = Mesh.Elements(i,:);

    % Extract vertex coordinates
    Vertices = Mesh.Coordinates(dofh,:);

    % Compute area of triangle
    area = 0.5*det(Vertices(2:3,:) - kron([1;1],Vertices(1,:)));

    % Evaluate solutions

    gradu_EX = GradUHandle(Vertices);
    gradbarc = gradbarycoords(Vertices);
    gradu_FE = U(dofh(1))*gradbarc(:,1)+U(dofh(2))*gradbarc(:,2)+U(dofh(3))*gradba
    gradu_FE = repmat(gradu_FE,1,3); % the gradient is the same in all the 3 vertices

    % Compute error on the current element

    err = err + sum(sum((gradu_EX-gradu_FE).^2,1),2)*area/3;

  end
  err = sqrt(err);
```

```
return
```

**(6.1j)** Implement a function

```
[U,L2err,H1serr] = mainNeumann(Mesh)
```

that, given in input a mesh data structure `Mesh`, computes the discrete solution $u_h$ to (6.1.1) in the case that the exact solution is $u(\boldsymbol{x}) = \cos(2\pi x_1)\cos(2\pi x_2)$, plots the mesh and $u_h$. The function returns the coefficient vector `U` of $u_h$, the $L^2$-norm and the $H^1$-seminorm of the discretization error.

Create a plot of the discrete solution using the mesh `Square.mat` provided in the handout to be downloaded from the course webpage.

HINT: Given the exact solution, you can use (6.1.1) to obtain the right-hand side $f$.

HINT: To plot the mesh you can use the MATLAB function `triplot`, and to plot the solution you can use the function `trisurf`.

HINT: To load the mesh use the MATLAB function `load`.

HINT: Using the mesh given in the handout, the $L^2$-norm error should be around $0.0020$ and the $H^1$-seminorm error around $0.6627$.

**Solution:** See Listing 6.10 for the code and Figure 6.1 for the plot.

Listing 6.10: Implementation for `mainNeumann`

```
function [U,L2err,H1serr] = mainNeumann(Mesh)

FHandle = @(x)
   (8*pi^2+1)*cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));
UHandle = @(x) cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));
GradUHandle = @(x)
   -2*pi*[(sin(2*pi*x(:,1)).*cos(2*pi*x(:,2)))';
   (cos(2*pi*x(:,1)).*sin(2*pi*x(:,2)))'];

% Plot mesh
figure(1)
triplot(Mesh.Elements(:,:),Mesh.Coordinates(:,1),Mesh.Coordinates(:,2))

% Assemble stiffness matrix and load vector
A = assemMat_LFE(Mesh,@Elmat_LaplMass_LFE);          %
   Stiffness matrix in sparse format
L = assemLoad_LFE(Mesh,@localLoadLFE,FHandle);

% No Dirichlet boundary data => no modification of rhs

% Solve the linear system
U = A\L;
```

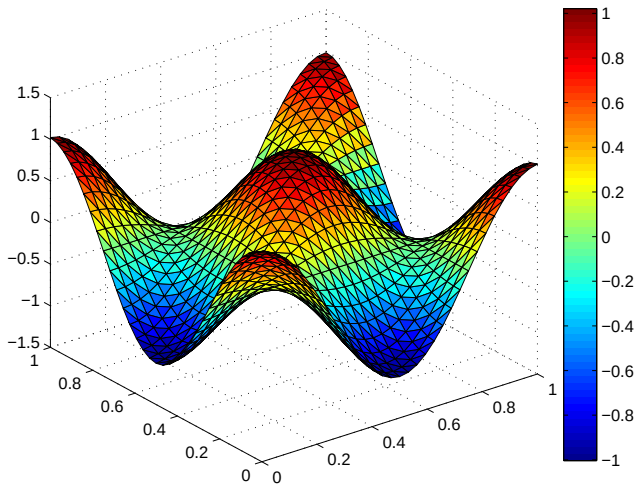


Figure 6.1: Solution plot for subproblem (6.1j).

```
% Plot solution
figure(2)
trisurf(Mesh.Elements(:,:),Mesh.Coordinates(:,1),Mesh.Coordinates(:,2),U)
colorbar

% Compute the errors
L2err = L2Err_LFE(Mesh,U,UHandle);
H1serr = H1SErr_LFE(Mesh,U,GradUHandle);
```

Listing 6.11: Testcalls for Problem 6.1

```
Vertices = [0 0; 1 0; 0 1];
FHandle = @(x) x(:,1).*x(:,2);

Mesh = load(['Square.mat']);

fprintf('\n##gradbarycoords')
gradbarycoords_ref(Vertices)

fprintf('\n##Elmat_Lapl_LFE')
Elmat_Lapl_LFE_ref(Vertices)

fprintf('\n##Elmat_Mass_LFE')
Elmat_Mass_LFE_ref(Vertices)

fprintf('\n##Elmat_LaplMass_LFE')
Elmat_LaplMass_LFE_ref(Vertices)

fprintf('\n##localLoadLFE')
localLoadLFE_ref(Vertices,FHandle)
```

```

fprintf('\n##assemMat_LFE')
A = assemMat_LFE_ref(Mesh,@Elmat_LaplMass_LFE);
A(1:10,1:10)

fprintf('\n##assemLoad_LFE')
L = assemLoad_LFE_ref(Mesh,@localLoadLFE,FHandle);
L(1:10)
```

Listing 6.12: Output for Testcalls for Problem 6.1

```
testcall

##gradbarycoords
ans =

    -1     1     0
    -1     0     1

##Elmat_Lapl_LFE
ans =

    1.0000   -0.5000   -0.5000
   -0.5000    0.5000         0
   -0.5000         0    0.5000

##Elmat_Mass_LFE
ans =

    0.0833    0.0417    0.0417
    0.0417    0.0833    0.0417
    0.0417    0.0417    0.0833

##Elmat_LaplMass_LFE
ans =

    1.0833   -0.4583   -0.4583
   -0.4583    0.5833    0.0417
   -0.4583    0.0417    0.5833

##localLoadLFE
ans =

         0
    0.0208
    0.0208

##assemMat_LFE
ans =

```

```
   (1,1)        1.0001
   (2,2)        1.0002
   (3,3)        1.0001
   (4,4)        1.0002
   (5,5)        2.0002
   (6,6)        2.0002
   (7,7)        4.0005
   (8,8)        2.0002
   (9,9)        2.0002
  (10,10)       2.0002

##assemLoad_LFE
ans =

   1.0e-03 *

         0
    0.0038
    0.1602
    0.0038
    0.0025
    0.0025
    0.2441
    0.2441
    0.2441
    0.0012
```

## Problem 6.2 Rigidity of Piecewise Polynomial Continuous Functions

[NPDE, Section 3.3] and, particular, [NPDE, Section 3.5] probably created the impression that the construction of a viable finite element space is straightforward: one starts from a mesh, fixes a piecewise polynomial space and, finally, finds suitable locally supported basis functions. However, at each stage this procedure can fail, which is strikingly demonstrated in this problem.

Let $\mathcal{M} = \{K\}$ be a tensor product mesh, see [NPDE, Section 3.4.1], as depicted in Figure 6.2 with $N_x, N_y$ grid lines in $x$- and $y$-direction, respectively. All cells (elements) are rectangles, and there are $N = N_x N_y$ vertices in the mesh.

**(6.2a)** Define the function space

$$W_N = \left\{ v \in H_0^1(\Omega) \,\middle|\, v|_K \in \mathcal{P}_1(\mathbb{R}^2),\, \forall K \in \mathcal{M} \right\},$$

of piecewise linear functions (see [NPDE, Def. 3.4.8]) on each element of $\mathcal{M}$, that are zero at the boundary. What is the dimension of $W_N$?

HINT: Remember from [NPDE, § 3.3.8] that an (affine) linear function $\mathbb{R}^2 \mapsto \mathbb{R}$ is already fixed by prescribing values in three non-collinear points.

**Solution:** The dimension is actually zero. Consider a corner element. Since the function must be zero on the boundary, it must be zero in three of four vertices of this element. Because a linear function is fixed by its values in three non-collinear points, it must be the zero function, so it is

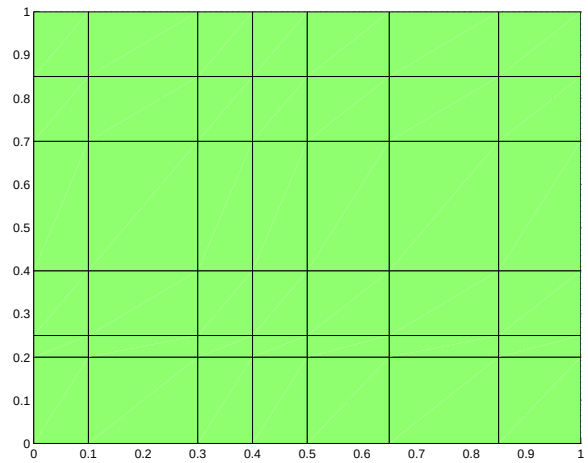


Figure 6.2: A tensor product mesh.

also zero on the last vertex. In this way, one can reason that the function must be zero on *all* vertices, and so the only function in $W_N$ is the zero function.

**(6.2b)** Define the function space

$$V_N = \left\{ v \in H^1(\Omega) \,\middle|\, v|_K \in \mathcal{P}_1(\mathbb{R}^2) \,\forall K \in \mathcal{M} \right\},$$

of piecewise linear functions on each element of $\mathcal{M}$. What is the dimension of $V_N$?

**Solution:** In the same way as before, we see that the values of all vertices will be given if we only prescribe the values on two non-parallel edges of the boundary. Then we can iterate through the elements in the right order, using three vertices with known values to get the value of the fourth vertex. There are $N_x + N_y - 1$ vertices we can define to begin with, and so this must be the dimension of $V_N$.

**(6.2c)** Define the function space

$$V_N = \left\{ v \in H^1(\Omega) \,\middle|\, v|_K \in \mathcal{Q}_1(\mathbb{R}^2) \,\forall K \in \mathcal{M} \right\},$$

of piecewise bi-linear functions on each element of $\mathcal{M}$, see [NPDE, Def. 3.4.13]. What is the dimension of this $V_N$?

**Solution:** This is the Lagrangian finite element space introduced in [NPDE, Section 3.5.2] and [NPDE, Ex. 3.5.8] and its dimension agrees with the total number of vertices of the mesh, which serve as interpolation points in this case.

**(6.2d)** If we abandon nice "confoming" finite element meshes and even admit "hanging nodes", additional difficulties loom. To appreciate this, now consider the non-conforming triangular mesh $\mathcal{M}$ of $\Omega = ]0,1[^2$ in Figure 6.3. There, the hanging nodes are located on the midpoints of the edges of the other triangle.

Determine the dimension of the space

$$W_N = \left\{ v \in C^0(\overline{\Omega}) \,\middle|\, v|_K \in \mathcal{P}_1(\mathbb{R}^2) \,\forall K \in \mathcal{M}, v|_{\partial\Omega} = 0 \right\},$$

and describe a basis of locally supported functions.

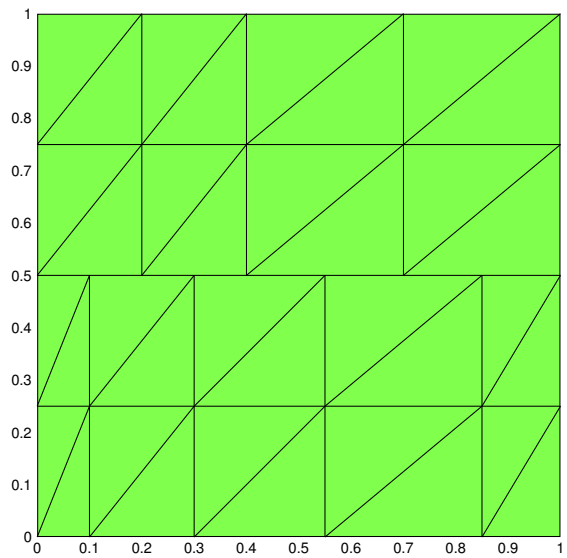


Figure 6.3: Non-conforming triangular mesh

**Solution:** As we have zero boundary conditions, we only have degrees of freedom in the internal nodes. However, one must also neglect the dofs in the hanging nodes, since in these vertices we cannot define a basis function which is continuous in the domain and a linear polynomial when restricted to each element. Recall that a function $p \in \mathcal{P}_1$ is uniquely determined by the value at 3 non-aligned points, so the hanging nodes located at the midpoints pose a contradiction.

In this concrete example, this leads us to 7 basis functions, as marked with yellow dots in Figure 6.4.

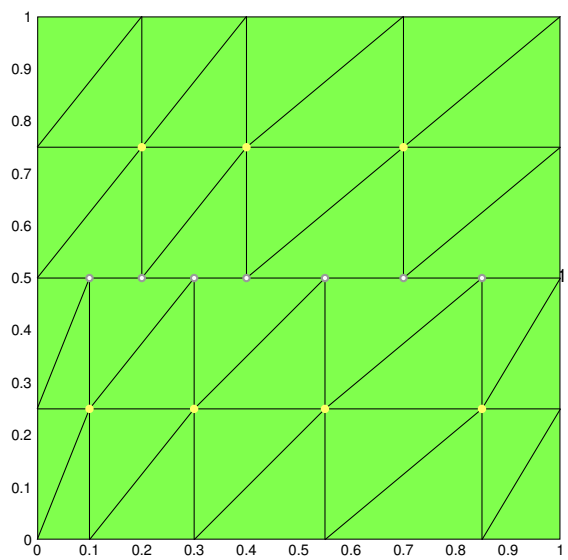


Figure 6.4: Non-conforming triangular mesh.

**(6.2e)** What is the dimension of the space obtained from $W_N$ by dropping the boundary condition $v|_{\partial\Omega} = 0$. Also in this case describe a basis and specify the supports of the basis functions.

**Solution:** For the reasons stated above, we neglect the dofs in the hanging nodes (hollow gray circles in Figure 6.4) . Thus, in this particular setting we have 24 "tent" basis functions.

## Problem 6.3 Convection Bi-linear Form

Hitherto, in class we have exclusively studied (linear) variational problems with symmetric bilinear forms, which are connected with quadratic minimization problems, as explained in [NPDE, Section 2.2.3]. Yet, many PDE models have variational formulations that involve non-symmetric bilinear forms. A simple representative will be examined in this problem. We will practise multi-dimensional integration by parts from [NPDE, Section 2.5.1] and also some local computations connected with Galerkin discretization by means of linear finite elements, see [NPDE, Section 3.3.5].

Let $\Omega \subset \mathbb{R}^2$ be a bounded polygonal domain. We define the *convection bilinear form* as

$$\mathsf{a}(u,v) = \int_\Omega (\mathbf{b}(\mathbf{x}) \cdot \mathbf{grad}\, u(\mathbf{x}))v(\mathbf{x})\,\mathrm{d}\mathbf{x}, \qquad u \in H^1(\Omega),\ v \in L^2(\Omega),$$

where $\mathbf{b} : \Omega \to \mathbb{R}^2$ is a vector field, with each component in $H^1(\Omega)$.

**(6.3a)** Show that for $u, v \in H^1_0(\Omega)$

$$\mathsf{a}(u,v) = -\int_\Omega u(\mathbf{x})\,\mathrm{div}(\mathbf{b}(\mathbf{x})v(\mathbf{x}))\,\mathrm{d}\mathbf{x}.$$

HINT: Use Green's formula [NPDE, Thm. 2.5.9]

**Solution:** First notice

$$\mathsf{a}(u,v) = \int_\Omega (\mathbf{b}(\mathbf{x}) \cdot \mathbf{grad}\, u(\mathbf{x}))v(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_\Omega (v(\mathbf{x})\mathbf{b}(\mathbf{x})) \cdot \mathbf{grad}\, u(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

Then, using Green's formula we get

$$\mathsf{a}(u,v) = -\int_\Omega \mathrm{div}(v(\mathbf{x})\mathbf{b}(\mathbf{x}))u(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega} (\mathbf{b}(\mathbf{x}) \cdot \mathbf{n})u(\mathbf{x})v(\mathbf{x}),$$

as the boundary term is zero when $u, v \in H^1_0(\Omega)$, we complete our proof.

**(6.3b)** Show that, if $\mathrm{div}\,\mathbf{b}(\mathbf{x}) = 0$, then

$$\mathsf{a}(u,u) = 0, \quad \forall u \in H^1_0(\Omega).$$

HINT: Use the general product rule [NPDE, Lemma 2.5.4].

**Solution:** Consider the formula obtained in the previous subproblem:

$$\mathsf{a}(u,v) = -\int_\Omega \mathrm{div}(v(\mathbf{x})\mathbf{b}(\mathbf{x}))u(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

and the general product rule

$$\mathrm{div}(\mathbf{b}(\mathbf{x})v(\mathbf{x})) = v(\mathbf{x})\,\mathrm{div}\,\mathbf{b}(x) + \mathbf{b}(\mathbf{x}) \cdot \mathbf{grad}\, v(\mathbf{x}).$$

Combining these two we get

$$\mathsf{a}(u,v) = -\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x} - \int_\Omega (\mathbf{b}(\mathbf{x})\cdot\mathbf{grad}\,v(\mathbf{x}))u(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

In particular,

$$\mathsf{a}(u,u) = -\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})u(\mathbf{x})\,\mathrm{d}\mathbf{x} - \mathsf{a}(u,u),$$

from where

$$\mathsf{a}(u,u) = -\frac{1}{2}\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})u(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

which becomes zero when $\operatorname{div}\mathbf{b}(\mathbf{x}) = 0$.

**(6.3c)** Show that, if $\operatorname{div}\mathbf{b}(\mathbf{x}) = 0$ and $\mathbf{b}(\mathbf{x})\cdot\mathbf{n} = 0$ on $\partial\Omega$, then

$$\mathsf{a}(u,u) = 0, \quad \forall u \in H^1(\Omega).$$

**Solution:** Taking the cue of subproblem (6.3a), we use Green's formula to obtain

$$\mathsf{a}(u,v) = -\int_\Omega \operatorname{div}(v(\mathbf{x})\mathbf{b}(\mathbf{x}))u(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}(\mathbf{b}(\mathbf{x})\cdot\mathbf{n})u(\mathbf{x})v(\mathbf{x}),$$

In addition, by the general product rule we can rewrite it as

$$\mathsf{a}(u,v) = -\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x} - \mathsf{a}(v,u) + \int_{\partial\Omega}(\mathbf{b}(\mathbf{x})\cdot\mathbf{n})u(\mathbf{x})v(\mathbf{x}).$$

Therefore

$$\mathsf{a}(u,u) = \frac{1}{2}\left(-\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}(\mathbf{b}(\mathbf{x})\cdot\mathbf{n})u(\mathbf{x})v(\mathbf{x})\right),$$

and we get $\mathsf{a}(u,u) = 0, \quad \forall u \in H^1(\Omega)$, if $\operatorname{div}\mathbf{b}(\mathbf{x}) = 0$ and $\mathbf{b}(\mathbf{x})\cdot\mathbf{n} = 0$ on $\partial\Omega$.

**(6.3d)** Show that

$$\mathsf{a}(u,u) > 0, \quad \forall u \in H_0^1(\Omega),$$

if $-\operatorname{div}\mathbf{b}(\mathbf{x})$ is uniformly positive ( see [NPDE, Def. 2.2.15]).

**Solution:** From our computations in subproblem (6.3b), we know

$$\mathsf{a}(u,u) = -\frac{1}{2}\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})u(\mathbf{x})u(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

If $-\operatorname{div}\mathbf{b}(\mathbf{x})$ is uniformly positive, then $\exists 0 < \gamma^- \le \gamma^+ < \infty : \gamma^- \le -\operatorname{div}\mathbf{b}(\mathbf{x}) \le \gamma^+$ for almost all $\mathbf{x} \in \Omega$. Consequently, the integral satisfies

$$\mathsf{a}(u,u) = -\frac{1}{2}\int_\Omega \operatorname{div}\mathbf{b}(\mathbf{x})|u(\mathbf{x})|^2\,\mathrm{d}\mathbf{x} \ge \gamma^- \int_\Omega |u(\mathbf{x})|^2\,\mathrm{d}\mathbf{x} > 0.$$

From now on assume that the vector field is constant on $\Omega$: $\mathbf{b}(x) := \mathbf{b}$, $\forall x \in \Omega$.

We perform a Finite Element Galerkin discretization of the linear variational problem: Seek $u \in H_0^1(\Omega)$ such that

$$\mathsf{a}(u,v) = \ell(v), \quad \forall v \in L^2(\Omega),$$

on a triangular mesh $\mathcal{M}$ and based on the discrete trial and test space $\mathcal{S}_{1,0}^0(\mathcal{M})$ (linear finite elements as [NPDE, Section 3.3]). The nodal basis of "tent functions" as introduced in [NPDE, Section 3.3.3] is used throughout.

**(6.3e)** Write a C++ function

```
template <class Coord_t, class Vector2d, class Matrix>
  void locMatConvect( Coord_t const & a1, Coord_t const & a2,
                      Coord_t const & a3, Vector2D const & b,
                      Matrix & elmat)
```

that computes the element matrix for $\mathsf{a}(\cdot,\cdot)$ on a triangle $K$ with vertices $\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3$, whose coordinates are passed in as `a1, a2, a3`. The argument `b` supplies the vector $\mathbf{b}$.

Objects of type `Coord_t` and `Vector2D` represent vectors with 2 components and must allow component access via `[0]` and `[1]`.

`Matrix` objects provide the following methods and types

- `value_t`
- `index_t`
- `rows()`
- `cols()`
- `value_t operator( index_t, index_t) const` to access the matrix values.
- `value_t & operator( index_t, index_t)` to assign the matrix values.

the `elmat` instance passed as argument can be assumed to have the right size.

A C++ template file is available in the lecture's webpage as guidance for implementation.

**Remark:** Note that essential conditions don't matter at the level of element matrices.

HINT: Revising [NPDE, Section 3.3.5] might be useful, particularly to compute the gradients.

Listing 6.13: Testcall for subproblem (6.3e) (fragment from main file).

```
    // test call:
    // initialize vertices and b vector
    coord_t a1(0,1), a2(2,1), a3(1,3);
    vector_t b(2); b.setOnes();
    // initialize local matrix and call locMatConvect
    matrix_t local(3,3);
    locMatConvect(a1, a2, a3, b, local);
    // print the obtained matrix
    std::cout << "local matrix for element with vertices : ("
```

```
            << a1.transpose() << ") , (" << a2.transpose() << ") , ("
            << a3.transpose() << ") : \n \n" << local << std::endl;
```

Listing 6.14: Output for Testcalls for subproblem (6.3e)

```
local matrix for element with vertices : (0 1) , (2 1) , (1 3) :

    -0.5 0.166667 0.333333
    -0.5 0.166667 0.333333
    -0.5 0.166667 0.333333
```

**Solution:** From [NPDE, Section 3.3.5], we know the local matrix is given by

$$\mathsf{a}_K(b_N^j, b_N^i) = \int_K (\mathbf{b} \cdot \mathbf{grad}\, b_{N|K}^j) b_{N|K}^i \,\mathrm{d}\mathbf{x}.$$

Since the gradient is constant for linear finite elements, this reduces to:

$$\mathsf{a}_K(b_N^j, b_N^i) = (\mathbf{b} \cdot \mathbf{grad}\, b_{N|K}^j) \int_K b_{N|K}^i \,\mathrm{d}\mathbf{x} = (\mathbf{b} \cdot \mathbf{grad}\, b_{N|K}^j) \frac{|K|}{3}.$$

Using the formula for the gradients, we notice the element's area is cancelled and the implementation follows as in the listing Listing 6.15.

Listing 6.15: Implementation for `locMatConvect`

```
#include <stdexcept>
#include <cassert>
#include <cstdlib>
#include <math.h>
#include <iostream>
// Eigen headers
#include <Eigen/Dense>

using namespace std;
using vector_t = Eigen::VectorXd;
using coord_t  = Eigen::Vector2d;
using matrix_t = Eigen::MatrixXd;

template <class Coord_t, class Vector2D, class Matrix>
void locMatConvect( Coord_t const& a1, Coord_t const& a2,
                    Coord_t const& a3, Vector2D const& b,
                    Matrix & elmat){
    // Compute the gradients (considering the are will cancel)
    Matrix grad(2,3);
    grad(0,0) = (a2[1] - a3[1])/2.0;
    grad(1,0) = (a3[0] - a2[0])/2.0;
    grad(0,1) = (a3[1] - a1[1])/2.0;
    grad(1,1) = (a1[0] - a3[0])/2.0;
    grad(0,2) = (a1[1] - a2[1])/2.0;
    grad(1,2) = (a2[0] - a1[0])/2.0;

```

```
    // Fill the matrix
    for(int i = 0; i<3; i++)
      for( int j = 0; j<3; j++)
        elmat(j,i) = (b[0]*grad(0,i)+b[1]*grad(1,i))/3.0;
}

int main(int argc, char *argv[]){
    // initialize vertices and b vector
    coord_t a1(0,1), a2(2,1), a3(1,3);
    vector_t b(2); b.setOnes();
    // initialize local matrix and call locMatConvect
    matrix_t local(3,3);
    locMatConvect(a1, a2, a3, b, local);
    // print the obtained matrix
    std::cout << "local matrix for element with vertices : ("
          << a1.transpose() << ") , (" << a2.transpose() << ") , ("
          << a3.transpose() << ") : \n \n" << local << std::endl;

    return 0;
}
```

**(6.3f)** Show that the Galerkin matrix is skew-symmetric.

HINT: A square matrix $\mathbf{A}$ is skew-symmetric, if $\mathbf{A}^T = -\mathbf{A}$. Also recall the computations of subproblem (6.3a).

**Solution:** In subproblem subproblem (6.3a) we found

$$\mathsf{a}(u,v) = -\int_\Omega u(\mathbf{x})\,\mathrm{div}(\mathbf{b}v(\mathbf{x}))\,\mathrm{d}\mathbf{x},$$

which boils down to

$$\mathsf{a}(u,v) = -\int_\Omega u(\mathbf{x})(\mathbf{b}\cdot\mathbf{grad}\,v(\mathbf{x}))\,\mathrm{d}\mathbf{x} = -\mathsf{a}(v,u).$$

Therefore, the corresponding Galerkin matrix is given by

$$(\mathbf{A})_{ij} = \int_\Omega (\mathbf{b}\cdot\mathbf{grad}\,b_N^j)b_N^i\;\mathrm{d}\mathbf{x} = -\int_\Omega (\mathbf{b}\cdot\mathbf{grad}\,b_N^i)b_N^j\;\mathrm{d}\mathbf{x} = -(\mathbf{A})_{ji},$$

proving that the Galerkin matrix is skew-symmetric.

## Problem 6.4 Hybrid-Mesh Galerkin Matrices and Right-Hand Side Vectors

In [NPDE, Rem. 3.5.16] we saw that both linear and bilinear Lagrangian finite elements can be easily blended on a 2D hybrid mesh comprising both quadrilaterals and triangles. In this exercise we study the details of such a finite element method with focus on local computations and assembly.

Figure 6.5 displays a hybrid mesh $\mathcal{M}$ consisting of 13 vertices, 8 triangular elements and 4 quadrilateral elements. The coordinates of some of the vertices are

$$\boldsymbol{a}^7 = (0,0), \quad \boldsymbol{a}^1 = (0,1), \quad \boldsymbol{a}^4 = (1,1)/\sqrt{2}, \quad \boldsymbol{a}^3 = (0,1)/\sqrt{2}.$$

The coordinates of the rest follow from symmetry.

In this problem we will compute the Galerkin matrix for (bi-)linear Lagrangian finite elements [NPDE, Section 3.5] on such a mesh for the bilinear form asscociated with $-\Delta$

$$\mathsf{a}(u,v) = \int_\Omega \operatorname{grad} u(\mathbf{x}) \cdot \operatorname{grad} v(\mathbf{x})\,\mathrm{d}\mathbf{x}, \quad u,v \in H^1(\Omega), \tag{6.4.1}$$

and the right-hand side vector arising from the linear form

$$\ell(v) = \int_\Omega f(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x}, \tag{6.4.2}$$

with $f \in C^0(\Omega)$.

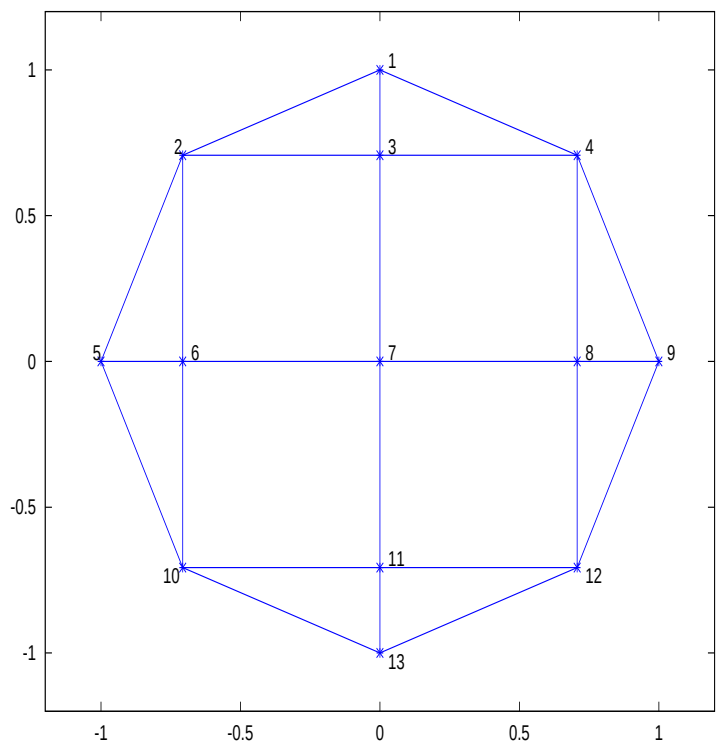


Figure 6.5: A hybrid mesh of triangles and quadrilaterals.

**(6.4a)** What is the dimension of the finite element space $\mathcal{S}_1^0(\mathcal{M})$?

HINT: See [NPDE, Rem. 3.5.16].

**Solution:** The dimension is $13$. There is one basis function for each node.

**(6.4b)** Compute the $4 \times 4$ element Galerkin matrix for one of the squares using the standard bilinear local shape functions from [NPDE, Eq. (3.5.10)]

HINT: All the square elements are equal, and they have side lengths $1/\sqrt{2}$. Number the nodes either clockwise or counterclockwise around the square (due to symmetry, any such numbering should yield the same matrix). There are two ways to compute their corresponding element matrices and you may choose either of them:

1. direct evaluation of the localized bilinear form $\mathsf{a}_K$ for pairs of local shape functions. Note that their gradients are not constant this time.

2. computation of the Galerkin matrix on the unit square, and subsequent transformation. See [NPDE, Eq. (3.5.10)] for the basis functions on the unit square. [NPDE, Section 3.7.3] explains transformation techniques. Your transformation $\Phi$ in this case will simply be a scaling.

**Solution:** Using the transformation technique we get

$$\mathbf{A}^K_{i,j} = \int_{\widehat{K}} \operatorname{grad} \widehat{b}^i(\widehat{\mathbf{x}}) \cdot \operatorname{grad} \widehat{b}^j(\widehat{\mathbf{x}}) \, \mathrm{d}\widehat{\mathbf{x}}.$$

Due to symmetry, we only have to evaluate this integral for three different choices of $i, j$. Note that with the standard node numbering from [NPDE, Eq. (3.5.10)] we get

$$\begin{aligned}
\operatorname{grad} \widehat{b}^1(\widehat{x}_1, \widehat{x}_2) &= (\widehat{x}_2 - 1, \widehat{x}_1 - 1) \\
\operatorname{grad} \widehat{b}^2(\widehat{x}_1, \widehat{x}_2) &= (1 - \widehat{x}_2, -\widehat{x}_1) \\
\operatorname{grad} \widehat{b}^3(\widehat{x}_1, \widehat{x}_2) &= (\widehat{x}_2, \widehat{x}_1) \\
\operatorname{grad} \widehat{b}^4(\widehat{x}_1, \widehat{x}_2) &= (-\widehat{x}_2, 1 - \widehat{x}_1).
\end{aligned}$$

The computation is

$$\begin{aligned}
\mathbf{A}^K_{i,i} = \mathbf{A}^K_{3,3} &= \int_{\widehat{K}} \left(\widehat{x}_1^2 + \widehat{x}_2^2\right) \mathrm{d}\widehat{\mathbf{x}} = 2 \int_0^1 \widehat{x}_1^2 \, \mathrm{d}\widehat{x}_1 = \frac{2}{3}, \\
\mathbf{A}^K_{i,i\pm 2} = \mathbf{A}^K_{1,3} &= \int_{\widehat{K}} \left(\widehat{x}_1(1 - \widehat{x}_1) + \widehat{x}_2(1 - \widehat{x}_2)\right) \mathrm{d}\widehat{\mathbf{x}} \\
&= 2 \int_0^1 \widehat{x}_1(1 - \widehat{x}_1) \, \mathrm{d}\widehat{x}_1 = -\frac{1}{3}, \\
\mathbf{A}^K_{i,i\pm 1} = \mathbf{A}^K_{i,i\pm 3} = \mathbf{A}^K_{1,4} &= -\int_{\widehat{K}} \left((\widehat{x}_1 - 1)^2 + \widehat{x}_2(\widehat{x}_2 - 1)\right) \mathrm{d}\widehat{\mathbf{x}} \\
&= -\int_0^1 (\widehat{x}_1 - 1)^2 \, \mathrm{d}\widehat{x}_1 - \int_0^1 \widehat{x}_2(\widehat{x}_2 - 1) \, \mathrm{d}\widehat{x}_2 = -\frac{1}{6}.
\end{aligned}$$

So in the end we get

$$\mathbf{A}^K = \frac{1}{6} \begin{pmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{pmatrix}.$$

**(6.4c)** Compute the $3 \times 3$ element Galerkin matrix for the triangle with vertices $1, 2, 3$ using the standard linear local shape functions (barycentric coordinate functions, see )

HINT: The triangle has side lengths $1/\sqrt{2}$, $1 - 1/\sqrt{2}$ and $\sqrt{2 - \sqrt{2}}$. Check out [NPDE, Eq. (3.3.21)]. Use the local node numbering inherited from the global one (i.e. vertex $1$ is number $1$, and so on).

**Solution:** The cotangents are

$$\begin{aligned}\cot\omega_1 &= \sqrt{2}-1,\\ \cot\omega_2 &= \frac{1}{\sqrt{2}-1},\\ \cot\omega_3 &= 0,\end{aligned}$$

giving

$$\mathbf{A}^K = \frac{1}{\sqrt{2}-1}\begin{pmatrix} 1 & & -1 \\ & (\sqrt{2}-1)^2 & -(\sqrt{2}-1)^2 \\ -1 & -(\sqrt{2}-1)^2 & (\sqrt{2}-1)^2+1 \end{pmatrix}.$$

**(6.4d)** Compute the element right-hand side vector for a quadrilateral cell. For this, use the quadrature formula

$$\int_K f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{|K|}{4}\sum_{i=1}^{4} f(\mathbf{a}^i), \tag{6.4.3}$$

where $\mathbf{a}^i$ are the vertices of the square $K$.

**Solution:** Note that the area of the quadrilaterals are $1/2$. The quadrature rule should then give

$$\mathbf{L}_i^K = \frac{1}{8} f(\mathbf{a}^i),$$

since the basis functions are only supported on one vertex each.

**(6.4e)** What is the full $13\times 13$ Galerkin matrix for the numbering of nodes given in Figure 6.5?

HINT: Do an assembly "by hand" (see [NPDE, Section 3.6.3]). For each pair of neighboring vertices $i, j$, walk through the elements shared by $i$ and $j$, find the local element contribution from subproblems (6.4b) or (6.4c) and sum them up.

**Solution:** Define $p = \sqrt{2}/6$ and $q = \sqrt{2}-1$. Then,

$$\mathbf{A} = \begin{pmatrix}
\frac{2}{q} & & -\frac{2}{q} & & & & & & & & & & \\
 & 4p+2q & -p-q & & & -p-q & -2p & & & & & & \\
-\frac{2}{q} & -p-q & 8p+2q+\frac{2}{q} & -p-q & & -2p & -2p & -2p & & & & & \\
 & & -p-q & 4p+2q & & & -2p & -p-q & & & & & \\
 & & & & \frac{2}{q} & -\frac{2}{q} & & & & & & & \\
 & -p-q & -2p & & -\frac{2}{q} & 8p+2q+\frac{2}{q} & -2p & & -p-q & -2p & & & \\
 & -2p & -2p & -2p & & -2p & 16p & -2p & -2p & -2p & -2p & & \\
 & & -2p & -p-q & & & -2p & 8p+2q+\frac{2}{q} & -\frac{2}{q} & & -2p & -p-q & \\
 & & & & & & & -\frac{2}{q} & \frac{2}{q} & & & & \\
 & & & & & -p-q & -2p & & 4p+2q & -p-q & & & \\
 & & & & & -2p & -2p & -2p & -p-q & 8p+2q+\frac{2}{q} & -p-q & -\frac{2}{q} & \\
 & & & & & & -2p & -p-q & & -p-q & 4p+2q & & \\
 & & & & & & & & & -\frac{2}{q} & & \frac{2}{q} &
\end{pmatrix}$$

**(6.4f)** Compute the full right-hand side vector using the local contributions found in subproblem (6.4d). For the local contributions from the triangles, you can use the corresponding quadrature rule there,

$$\int_K f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{|K|}{3}\sum_{i=1}^{3} f(\mathbf{a}^i),$$

with $\mathbf{a}^i$ the vertices of the triangle.

**Solution:** The quadrature rule gives the following formula for the triangle contributions:

$$\mathbf{L}_i^K = \frac{r}{8} f(\mathbf{a}^i),$$

where $r = 2(\sqrt{2}-1)/3$. That should give the following right-hand side vector:

$$\begin{aligned}
\mathbf{L}_1 = \mathbf{L}_5 = \mathbf{L}_9 = \mathbf{L}_{13} &= 2rf(\mathbf{a}^i)/8,\\
\mathbf{L}_2 = \mathbf{L}_4 = \mathbf{L}_{10} = \mathbf{L}_{12} &= (2r+1)f(\mathbf{a}^i)/8,\\
\mathbf{L}_3 = \mathbf{L}_6 = \mathbf{L}_8 = \mathbf{L}_{11} &= (2r+2)f(\mathbf{a}^i)/8,\\
\mathbf{L}_7 &= 4f(\mathbf{a}^7)/8,
\end{aligned}$$

where in each case, $i$ will have to be replaced with the relevant node number.

**(6.4g)** [NPDE, Rem. 3.5.18] discusses the choice of interpolation nodes and, thus, implicitly, the choice of global shape functions, for quadratic Lagrangian finite elements on hybrid meshes. What is the dimension of $\mathcal{S}_2^0(\mathcal{M})$, if $\mathcal{M}$ is the hybrid mesh display in Figure 6.5?

**Solution:** Let $N_V, N_E, N_T, N_Q$ the number of vertices, edges, triangles and quadrilaterals respectively. Then the dimension of $\mathcal{S}_2^0(\mathcal{M})$ is $N_V + N_E + N_Q$ = 41.

**(6.4h)** Consider the Galerkin matrix $\mathbf{A}_Q$ for a general linear second-order elliptic Neumann boundary value problem when the space $\mathcal{S}_2^0(\mathcal{M})$ of quadratic Lagrangian finite elements on the hybrid mesh from Figure 6.5 is used as a trial and test space. Give a sharp bound on the number $\mathrm{nnz}(\mathbf{A}_Q)$ of non-zero entries of $\mathbf{A}_Q$.

HINT: In light of the supports of global shape functions, which pairs of them can interact in the bilinear form?

**Solution:** For this particular setting, we notice there are 4 possible situations for vertix nodes, 4 for edges nodes and just one for midpoint nodes. Adding the nodes which interact with each other and substracting the overlaps, one obtains 473 non-zero entries.

In light of the supports of global shape functions, we can extend this to a general hybrid mesh. First we take into account that we have $N_V + N_E + N_Q$ diagonal entries. For each edge, we have 3x2 entries, since each edge connects 3 couples of nodes. Each triangle supports 3 vertex/opposite-edge and 3 edge/edge interactions, contributing with 6x2 entries. Each quadrilateral supports 8 vertex/midpoint, 8 vertex/opposite-edge, 4 consecutive-edge and 4 opposite-vertex interactions, therefore 24x2 entries in total. Finally, adding all these quantities, we get $N_V + 7N_E + 12N_T + 49N_Q$ non-zero entries. In particular, for the hybrid mesh from Figure 6.5 this is 473.

Published on March 25.

To be submitted on April 1.

# References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equa-

tions".SVN revision # 75265.

[1] M. Struwe. Analysis für Informatiker. Lecture notes, ETH Zürich, 2009. https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145.

[NCSE] Lecture Slides for the course "Numerical Methods for CSE".

Last modified on April 23, 2015