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 λ_1, λ_2, and λ_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 3×2-matrix `Vertices`. The output `grad` is a 2×3 matrix containing the gradients of the shape functions evaluated at the vertices (the first column contains the gradient of λ_1, the second one the gradient of λ_2 and the last one the gradient of λ_3).

(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 3×2-vector providing the coordinates of the element vertices. The function should return a 3×3 matrix `Aloc` containing the element matrix.

(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].

(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).

(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 3×2-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 $i = 1, \ldots, K$, $K \in \mathbb{N}$, 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)].

(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 3×2-array of vertex coordinates and returns a 3×3 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.

(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 3×2 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`.

(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)].

(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)].

(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.

Listing 6.1: 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 6.2: 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 6.1 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.

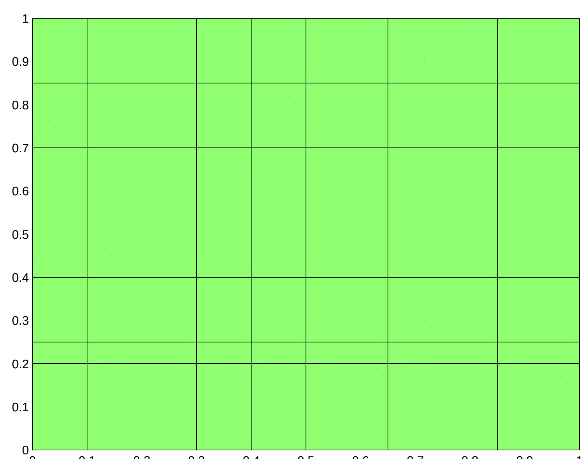

Figure 6.1: A tensor product mesh.

(6.2a) Define the function space

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

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.

(6.2b) Define the function space

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

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

(6.2c) Define the function space

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

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?

(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 6.2. There, the hanging nodes are located on the midpoints of the edges of the other triangle.

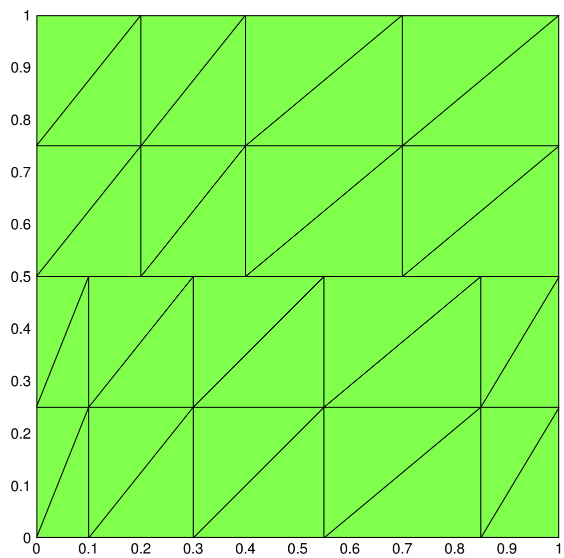

Figure 6.2: Non-conforming triangular mesh

Determine the dimension of the space

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

and describe a basis of locally supported functions.

(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.

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 $u, v \in H_0^1(\Omega)$

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

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

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

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

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

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

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

(6.3d) Show that

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

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

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

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 6.3: 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 6.4: 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
```

(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).

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

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

HINT: See [NPDE, Rem. 3.5.16].

(6.4b) Compute the 4×4 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 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 Φ in this case will simply be a scaling.

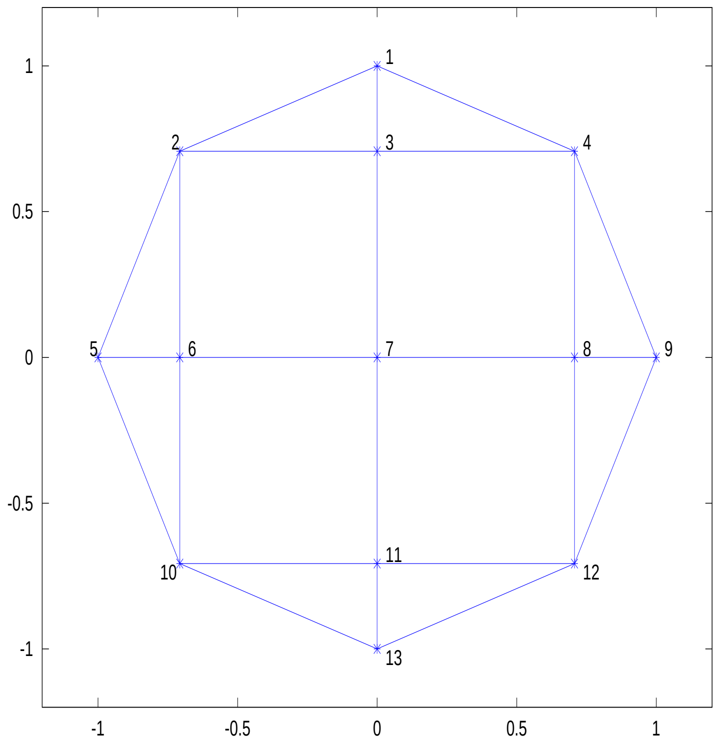

Figure 6.3: A hybrid mesh of triangles and quadrilaterals.

(6.4c) Compute the 3×3 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).

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

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

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

(6.4e) What is the full 13×13 Galerkin matrix for the numbering of nodes given in Figure 6.3?

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.

(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.

(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 6.3?

(6.4h) Write $\mathbf{A}_Q$ for 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 6.3 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?

Published on March 25.

To be submitted on April 1.

References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equations".SVN revision # 74741.

[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 1, 2015