R. Hiptmair
L. Scarabosio
E. Fonn

Spring Term 2013

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 5

Introduction. Problem 5.1 (theoretical) shows how the construction of a Finite Element space can easily fail if we are not careful. In Problem 5.2 and Problem 5.3 we get an introduction to using LehrFEM to solve two-dimensional problems. Some sample code is provided for the heat equation, which must be modified in two different ways. In Problem 5.2 we tweak the stiffness matrices, and in Problem 5.3 we change boundary conditions. Problem 5.4 and Problem 5.5 use parts of LehrFEM to solve more exotic problems, and these exercises will require some thought. In Problem 5.4 we solve for vector-valued functions, and in Problem 5.5 we implement a solution for the non-local boundary condition encountered in Problem 3.5.

Be aware that success in the final examination will require proficiency in implementing finite element methods in LehrFEM. The only way to acquire this proficiency is to develop LehrFEM codes by yourself again and again. Only reading the master solution is not enough!

Problem 5.1 "Difficulties" when Constructing Finite Element Spaces

[NPDE, Sect. 3.2] and, particular, [NPDE, Sect. 3.3.3] 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, Sect. 3.3.1], as depicted in Figure 5.1 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.

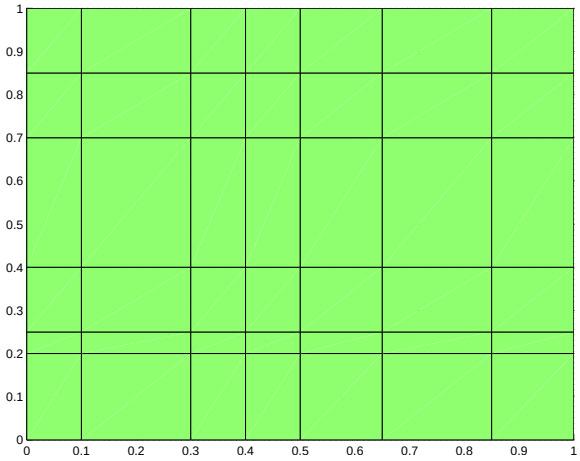


Figure 5.1: A tensor product mesh.

(5.1a) 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. 3.3.3]) on each element of $\mathcal{M}$, that are zero at the boundary. What is the dimension of W_N?

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

(5.1b) 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?

(5.1c) 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. 3.3.7]. What is the dimension of this V_N?

Problem 5.2 Heat Conduction with Mass Term and Dirichlet Boundary Conditions (Core problem)

This exercise deals with the variational formulation of a second-order elliptic boundary value problem (→ [NPDE, Sect. 2.8]), its approximate solution by means of Galerkin discretization based on linear Lagrangian finite elements on triangular meshes (→ [NPDE, Sect. 3.2]), and the implementation in the MATLAB finite element library LehrFEM, see [LehrFEM] and [NPDE, Sect. 3.5]. The initialization of the finite element Galerkin matrix ("assembly") in LehrFEM is explained in [NPDE, Sect. 3.5.3.1] and it may be useful to recapitulate this material. In order to understand the treatment of Dirichlet boundary conditions, please recall the contents of [NPDE, Sect. 3.5.5].

The stationary heat equation with a linear reaction term (zero-order term), reads

$$-\Delta u(\boldsymbol{x}) + \underbrace{c(\boldsymbol{x})u(\boldsymbol{x})}_{\text{reaction term}} = f(\boldsymbol{x}) \quad \text{in } \Omega. \tag{5.2.1}$$

where the reaction coefficient $c(\boldsymbol{x})$ is uniformly positive and bounded on Ω, *cf.* [NPDE, Eq. (2.5.4)]. In addition to this we impose the homogeneous Dirichlet boundary condition $u = 0$ on $\partial\Omega$. The computational domain $\Omega \in \mathbb{R}^2$ is a pentagon as shown in Figure 5.2.

Listing 5.1: MATLAB script `main_LFE_no_reac.m`

```
% MATLAB script main_LFE_no_reac.m

% Solving a simple 2nd-order elliptic Dirichlet problem by means of
% linear finite elements on a triangular mesh.
F_HANDLE = @(x,varargin) 1; % Right-hand side function f(x)
GD_HANDLE = @(x,varargin) 0; % Dirichlet data g(x)
```

```
7
8  % Load vertex coordinates and vertex indices for triangles from
9  % file, see [NPDE, Ex.  3.5.1]
10 Mesh = load_Mesh('Coord_Polygon_6.dat', 'Elem_Polygon_6.dat');
11
12 % Add element flags.  This is not used.
13 Mesh.ElemFlag = ones(size(Mesh.Elements,1),1);
14
15 % Initialization of the Edges and Vert2Edge records of Mesh, see
16 % [LehrFEM, Sect. 1.1] and [NPDE, Sect. 3.5.2], in particular
17 % [NPDE, Ex. 3.5.8].
18 Mesh = add_Edges(Mesh);
19
20 % Extract indices of edges located on the boundary, that is, edges
21 % that abut only a single triangle.  The edge indices refer to the
22 % position of the edge in the Mesh.Edges record of the Mesh data
23 % structure, see [LehrFEM, Sect. 1.1] for details.
24 Loc = get_BdEdges(Mesh);
25
26 % Initialize array flagging boundary edges.
27 Mesh.BdFlags = zeros(size(Mesh.Edges,1), 1);
28 Mesh.BdFlags(Loc) = -1; %
29
30 % Assemble Galerkin matrix and load vector, see [LehrFEM, Ch. 5]
31 % and [NPDE, Sect. 3.5.3].  Initialization of sparse Galerkin
     matrix;
32 % the second argument is a handle to a MATLAB function providing the
33 % element matrix, see [NPDE, Def. 3.5.13].  Note that A will be the
34 % Galerkin matrix for the full space S_1^0(M) also taking into account
35 % basis functions on the boundary!
36 A = assemMat_LFE(Mesh, @STIMA_Lapl_LFE); %
37
38 % Build right-hand side vector φ using a 7-point quadrature rule
39 % of order 6, see [NPDE, Rem. 3.5.45] and [LehrFEM, Ch. 3].
     F_HANDLE
40 % passes a handle to the source function f(x).  Basis functions
41 % associated with vertices on ∂Ω are taken into account.
42 L = assemLoad_LFE(Mesh,P7O6(), F_HANDLE);
43
44 % Set components of μ (↔ U) for basis functions located on ∂Ω to
45 % the corresponding value of the Dirichlet data g (passed through
46 % G_HANDLE. Also returns a vector FreeDofs of indices of vertices
47 % located in the interior of Ω.  This functions expects a valid
48 % BdFlags field in Mesh.  The argument '-1' matches the flag for
49 % boundary edges set in Line 28.
50 [U,FreeDofs] = assemDir_LFE(Mesh, -1, GD_HANDLE);
51
52 % Modify right-hand side φ in order to take into account Dirichlet
```

```
% data.  This is explained in [NPDE, Sect. 3.5.5].  Note that the
    length
% of both L and U agrees with the total number of vertices in the
% mesh.
L = L - A*U;

% Solve the linear system using MATLAB's \-operator.  Only the
% coefficients for basis functions belonging to interior vertices
% are involved.
U(FreeDofs) = A(FreeDofs,FreeDofs)\L(FreeDofs);

% Plot solution, see [LehrFEM, Sect. 7.1].
plot_LFE(U,Mesh); colorbar; %

% Clear memory
clear all;
```

The MATLAB script `main_LFE_no_reac.m`, see Listing 5.1, uses LehrFEM to solve (5.2.1) with $c \equiv 0$, that is, the reaction term is missing, and $f \equiv 1$. In this problem, we will extend the code so that the full homogeneous Dirichlet problem for (5.2.1) is solved approximately using the 2D linear finite elements introduced in [NPDE, Sect. 3.2].

(5.2a) Make sure that you master the material of [NPDE, Sect. 3.2].

(5.2b) Study [NPDE, Sect. 2.5] and try to explain why the term $c(\boldsymbol{x})$ is called a "reaction term".

HINT: As you may remember from secondary school, high temperatures hasten most chemical reactions. Assume an endothermic reaction.

(5.2c) Derive the variational formulation of homogeneous Dirichlet problem for (5.2.1). Do not forget to specify the trial and test spaces.

HINT: As in [NPDE, Sect. 2.8] rely Green's first formula [NPDE, Thm. 2.4.7] and use Sobolev spaces.

(5.2d) [NPDE, Lem. 3.5.29] gives a general closed form formula for the integration of products of barycentric coordinate functions ($\rightarrow$ [NPDE, Fig. 75]) over a simplex in arbitrary dimensions. Write down what this formula yields for $d = 2$ and all products of barycentric coordinate functions contained in $\mathcal{P}_2(\mathbb{R}^2)$.

(5.2e) Now assume that $c(\boldsymbol{x}) = \text{const}$ and that we use linear finite elements on a triangular mesh ([NPDE, Sect. 3.2.1]) with a tent function basis ([NPDE, Sect. 3.2.3]). Compute the element (stiffness) matrix ([NPDE, Def. 3.5.13]) for a general triangle K analytically in terms of the angles, the area $|K|$, and edge lengths $|e_i|$.

HINT: The Galerkin matrix for the Laplacian part is given in [NPDE, Eq. (3.2.10)]. The Galerkin matrix for the reaction term is easily computed thanks to [NPDE, Lem. 3.5.29] and will depend only on the area of the triangle.

(5.2f) The function

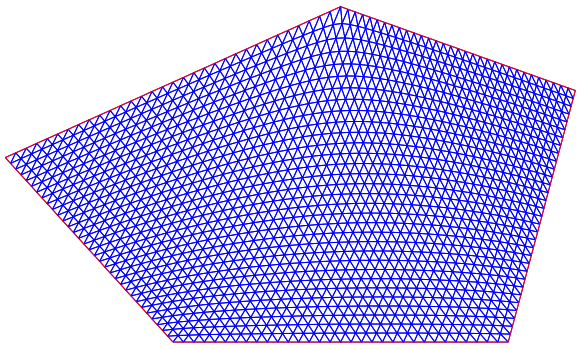


Figure 5.2: A triangulation of Ω.

```
Aloc = STIMA_Lapl_LFE(Vertices, varargin)
```

in the `Lib/Elements` directory computes the local Galerkin matrix for the "Laplacian" bilinear form

$$\mathsf{a}(u,v) = \int_K \operatorname{grad} u(\boldsymbol{x}) \cdot \operatorname{grad} v(\boldsymbol{x}) \mathrm{d}\boldsymbol{x},$$

and returns it in the 3×3 matrix `Aloc`. Here, `Vertices` is a 3×2 matrix, where each row denotes a vertex of the triangle, see [LehrFEM, Sect. 4.1]. Make a copy of this file called `STIMA_LaplMass_LFE.m` and extend the code in this file so that it computes the local Galerkin matrix for the bilinear form you found in subproblem (5.2c). The new function header should be

```
Aloc = STIMA_LaplMass_LFE(Vertices, flag, CHandle, varargin)
```

where `CHandle` is a function handle to $c(\boldsymbol{x})$ and `flag` is an element flag you can ignore for this problem. For integration, use midpoint quadrature,

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

where $\boldsymbol{m}_i$ are the midpoints of the edges of the triangle, and $|K|$ is the area.

HINT: You must add a contribution for the reaction term $\int_K c(\boldsymbol{x}) u(\boldsymbol{x}) v(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$ to the element matrix already computed in the code.

The values of two of the local shape functions at the midpoints are $\frac{1}{2}$, while the last is zero there. You can test your code by letting c be constant and comparing to the result from subproblem (5.2e)

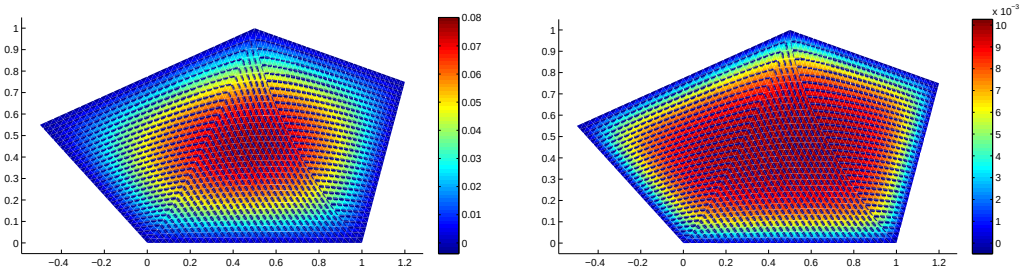
The area can be computed using Heron's formula,

$$|K| = \sqrt{s(s-a)(s-b)(s-c)},$$

where a, b, c are the side lengths of the triangle, and $s = (a+b+c)/2$.

(5.2g) Update `main_LFE_no_reac.m` to use `STIMA_LaplMass_LFE` instead of `STIMA_Lapl_LFE`.

HINT: After Line 6 of Listing 5.1, define a new function handle for $c(\boldsymbol{x})$. The call to `assemMat_LFE` in Line 36 should be altered into



(a) $c(\boldsymbol{x}) = 1$ (b) $c(\boldsymbol{x}) = 100$

Figure 5.3: Plots for subproblem (5.2h).

```
A = assemMat_LFE(Mesh, @STIMA_LaplMass_LFE, C_HANDLE);
```

This means that the handle to the function $c(x)$ is passed through the `varargin` facility of MATLAB. See [LehrFEM, Ch. 5] for details on the assembly routines, and [NPDE, Sect. 3.2.5] and [NPDE, Thm. 3.5.21] on the idea of assembly.

(5.2h) Solve and plot the solution of (5.2.1) for $c \equiv 1$ and $c \equiv 100$.

HINT: Use `plot_LFE` for plotting the solution, see [LehrFEM, Sect. 7.1] and Line 64 in Listing 7.3.

Listing 5.2: Testcalls for Problem 5.2

```
1 fprintf('\n##STIMA_LaplMass_LFE');
2 STIMA_LaplMass_LFE([0 0; 1 0; 0 1], 0, @(x)sum(x.^2,2))
```

Listing 5.3: Output for Testcalls for Problem 5.2

```
1 >> test_p1
2
3 ##STIMA_LaplMass_LFE
4 ans =
5     1.0208   -0.4896   -0.4896
6    -0.4896    0.5313    0.0208
7    -0.4896    0.0208    0.5313
```

Problem 5.3 Heat Conduction with Reaction Term and Convective Cooling [NPDE, Ex. 2.6.4] (Core problem)

This exercise involves the Galerkin discretization of a particular 2nd-order linear elliptic boundary value problems by means of linear Lagrangian finite elements and implementation in LehrFEM.

Let $\Omega \subset \mathbb{R}^2$ be a polygonal domain. In this problem we consider the boundary value problem

$$-\Delta u = f \quad \text{in } \Omega, \tag{5.3.1}$$

$$\operatorname{grad} u(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) + \gamma(\boldsymbol{x}) u(\boldsymbol{x}) = 0 \quad \text{on } \partial\Omega, \tag{5.3.2}$$

where, $f \in L^2(\Omega)$ and γ is a uniformly positive ($\rightarrow$)continuous function on the boundary $\partial\Omega$. Here we encounter so-called *Robin boundary conditions*, see [NPDE, Ex. 2.6.4] and [NPDE, Ex. 2.8.5].

We solve (5.3.1)–(5.3.2) approximately by means of Galerkin discretization based on piecewise linear Lagrangian finite elements on triangular meshes of Ω, see [NPDE, Sect. 3.2].

(5.3a) Derive the variational formulation of the boundary value problem (5.3.1)–(5.3.2). Do not forget to specify the trial and test spaces.

HINT: The boundary conditions are now contained in the bilinear form, not the trial and test spaces (that is, they are *natural*, see [NPDE, Ex. 2.8.5]).

(5.3b) Argue why the variational problem obtained in subproblem (5.3a) has a unique solution.

(5.3c) Assume that $\gamma \equiv \text{const}$. Compute the element (stiffness) matrix ($\rightarrow$ [NPDE, Def. 3.5.13]) for the bilinear form from subproblem (5.3a) and linear finite elements on a triangle K analytically in terms of the angles, the area $|K|$, and edge lengths $|e_i|$.

HINT: You may take knowledge of the edge lengths and angles of the triangle for granted and you may appeal to [NPDE, Eq. 3.2.10]. Special cases will occur if one or more edges are part of the boundary $\partial\Omega$! Then [NPDE, Lemma 3.5.29] for $d = 1$ may be useful.

(5.3d) Now solve subproblem (5.3c) for general continuous $\gamma \in \mathcal{C}^0(\partial\Omega)$, but using the one-dimensional trapezoidal rule [NPDE, Eq. (1.5.59)], see also [NPDE, Fig. 82], for the approximate evaluation of integrals along edges of K that are contained in $\partial\Omega$.

HINT: Contributions from the boundary terms will only enter the diagonal of the element matrices.

(5.3e) Implement a MATLAB function

```
Aloc = STIMA_LaplRobin_LFE(Vertices,flag,BdEdges,EHandle)
```

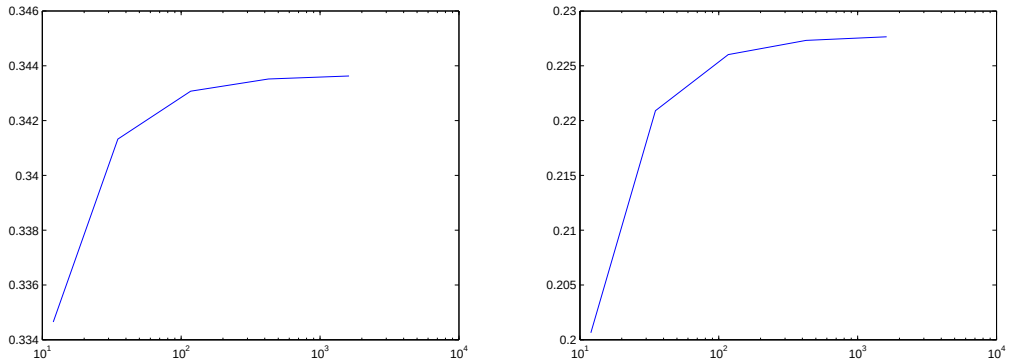
that returns the element matrix for the bilinear form from subproblem (5.3a) and linear Lagrangian finite elements. The 1D trapezoidal rule is to be used for the evaluation of integrals along edges on $\partial\Omega$,see subproblem (5.3d).

Here `Vertices` is a 3×2-matrix passing the coordinates of the triangle's vertices in its rows, `flag` can be ignored, and `BdEdges` is a possibly empty column vector of integer indices telling which edges of the current triangle are located on the boundary. The convention is that edge no. i is opposite to vertex i. Finally, `Ehandle` passes a handle of type `@(x)` to the function γ.

HINT: Of course, you may use the `STIMA_Lapl_LFE` function that comes with the LehrFEM library.

Remember that in MATLAB an empty vector can be detected by a call to `isempty()`. You can base this test also on the `length()`-function.

(5.3f) Create a copy of `assemMat_LFE.m` in the `Lib` directory of the LehrFEM distribution and rename it to `assemMatLapRobin_LFE.m`. Modify this function so that it can provide a global assembly routine that can accommodate the function `STIMA_LaplRobin_LFE` implemented in subproblem (5.3e). The syntax of your new function should be



(a) L^2-norm versus N (b) H^1-seminorm versus N

Figure 5.4: Plots for subproblem (5.3h).

```
varargout = assemMatRobin_LFE(Mesh,gammaHandle),
```

where `gammaHandle` is a function implementing $\gamma(\boldsymbol{x})$. Your implementation can take for granted that complete edge information is available for `Mesh` and that its `BdFlags` field is initialized as done in Listing 5.1.

HINT: By convention, in LehrFEM, edges on the boundary have a negative boundary flag, and interior edges have zero or positive values.

(5.3g) Make a copy of `main_LFE_no_reac.m` called `main_LFE_robin.m`. Modify this script so that it computes a linear finite element solution of (5.3.1)–(5.3.2) making use of the new function `assemMatRobin_LFE` from subproblem (5.3f).

HINT: This time you can solve the linear system directly, i.e. `U = A\L`, because without Dirichlet boundary conditions, *all* nodes of the mesh carry active basis functions.

(5.3h) Solve and plot the linear finite element solution of (5.2.1) with boundary conditions (5.3.2) based on the meshes found in the files `Coord_Polygon_N.dat` and `Elem_Polygon_N.dat` with `N = 2,...,6`. Use $\gamma \equiv 1$.

Compute the L^2- and H^1-seminorm of each solution, and plot these quantities vs. the dimension N of the finite element spaces.

HINT: You can compute the norms of the solutions by calling `L2Err_LFE` and `H1SErr_LFE` with `FHANDLE` being the zero-function, see [LehrFEM, Sect. 8], in particular [LehrFEM, Sect. 8.4] and [LehrFEM, Sect. 8.2].

Listing 5.4: Testcalls for Problem 5.3

```
fprintf('\n##STIMA_LaplRobin_LFE');
STIMA_LaplRobin_LFE([0 0; 1 0; 0 1], 0, [1; 3], @(x)sum(x.^2,2))

clear Mesh
Mesh.Coordinates = 2*[0 0; 1 0; 1 1; 0 1];
Mesh.Elements = [1 2 3; 1 3 4];
Mesh.ElemFlag = ones(size(Mesh.Elements,1),1);
```

```
Mesh = add_Edges(Mesh);
Loc = get_BdEdges(Mesh);
Mesh.BdFlags = zeros(size(Mesh.Edges,1),1);
Mesh.BdFlags(Loc) = -1;
Mesh = add_Edge2Elem(Mesh);

fprintf('\n##assemMatRobin_LFE');
assemMatRobin_LFE(Mesh, @(x)sum(x.^2,2))
```

Listing 5.5: Output for Testcalls for Problem 5.3

```
>> test_p2

##STIMA_LaplRobin_LFE
ans =

    1.0000   -0.5000   -0.5000
   -0.5000    1.7071    0.0000
   -0.5000    0.0000    1.2071

##assemMatRobin_LFE
ans =

   (1,1)       1.0000
   (2,1)      -0.5000
   (3,1)       0.0000
   (4,1)      -0.5000
   (1,2)      -0.5000
   (2,2)       9.0000
   (3,2)      -0.5000
   (1,3)       0.0000
   (2,3)      -0.5000
   (3,3)      17.0000
   (4,3)      -0.5000
   (1,4)      -0.5000
   (3,4)      -0.5000
   (4,4)       9.0000
```

Problem 5.4 Vector-Valued Linear Finite Elements

This problem is dedicated to the Galerkin finite element discretization of a rather non-standard 2D variational problem and implementation in MATLAB based on the LehrFEM library introduced in [NPDE, Sect. 3.5]. This exercise cannot be solved by' numbly following the standard steps outlined in [NPDE, Sect. 3.2], but it takes grasp of the abstract principles of Galerkin discretization as discussed in [NPDE, Sect. 3.1]. Also the implementation will require insight into the principles of finite element assembly, see [NPDE, Sect. 3.5.3] and [NPDE, Code 3.5.18].

In this problem we consider the bilinear form

$$\mathsf{a}(\mathbf{u},\mathbf{v}) = \int_\Omega \operatorname{div}\mathbf{u}(\boldsymbol{x})\operatorname{div}\mathbf{v}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \tag{5.4.1}$$

for *vector-valued* functions $\mathbf{u}, \mathbf{v} \in (H_0^1(\Omega))^2$, that is, $\mathbf{u}, \mathbf{v} : \Omega \mapsto \mathbb{R}^2$ have both components in $H_0^1(\Omega)$. Below we will study its Galerkin finite element discretization. Here $\Omega \subset \mathbb{R}^2$ is a bounded polygonal domain. The divergence operator div is introduced in [NPDE, Lemma 2.4.3].

(5.4a) Which properties do all Galerkin matrices arising from the discretization of linear variational problems with bilinear form $\mathsf{a}(\cdot,\cdot)$ using the *same* basis for the test and trial spaces share?

To discretize a linear variational problem based on $\mathsf{a}(\cdot,\cdot)$, we rely on a triangular mesh $\mathcal{M}$, and use as trial and test space $V_{0,N} = \left(\mathcal{S}^0_{1,0}(\mathcal{M})\right)^2$. This means that each component of both $\mathbf{u}$ and $\mathbf{v}$ is approximated by means of piecewise linear Lagrangian finite element functions.

We use the standard "tent function" basis of $\mathcal{S}^0_{1,0}(\mathcal{M})$, see [NPDE, Sect. 3.2.3], with the ordering of the basis functions induced by the numbering of the vertices of the mesh. For the space of vector valued finite element functions we use the *ordered* basis

$$\mathcal{B}_N = \left\{ \begin{pmatrix} \varphi_N^1 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} \varphi_N^L \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \varphi_N^1 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ \varphi_N^L \end{pmatrix} \right\} \tag{5.4.2}$$

where L is the dimension of $\mathcal{S}^0_{1,0}(\mathcal{M})$, and φ_N^j are the tent functions associated with the vertices $\boldsymbol{x}_j$ of $\mathcal{M}$.

(5.4b) Again, study [NPDE, Ex. 1.5.66] as an example of a discretization of a vector valued function, the curve in this case, by means of piecewise linear finite elements.

(5.4c) Given an arbitrary triangular element K with vertices $\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3$, there are six local shape functions associted to K, namely $(\lambda_i, 0)$ and $(0, \lambda_i)$, for $i = 1, 2, 3$, where λ_i are the barycentric coordinate functions, see the discussion in [NPDE, Sect. 3.2.5].

Compute analytic expressions for the divergences of these local shape functions.

HINT: Use the gradients given in [NPDE, Sect. 3.2.5].

(5.4d) Write a MATLAB function

```
Aloc = STIMA_Div_LFE(Vertices, varargin)
```

that computes the 6×6 local Galerkin matrix for a triangle whose vertex coordinates are given in the 3×2 matrix `Vertices`.

HINT: For a triangle with vertices $\boldsymbol{a}^1$, $\boldsymbol{a}^2$, and $\boldsymbol{a}^3$ the area can be computed by the formula

$$|K| = \tfrac{1}{2} \left| \det \begin{pmatrix} \boldsymbol{a}^2 - \boldsymbol{a}^1 & \boldsymbol{a}^3 - \boldsymbol{a}^1 \end{pmatrix} \right| .$$

Keep in mind the ordering of the basis functions!

(5.4e) Make a copy of the LehrFEM file `assemMat_LFE.m` (residing in the `Assembly` director), rename it to `assemMat_2DLFE.m`, and in it, write a function

```
A = assemMat_2DLFE(Mesh, EHandle, varargin)
```

for assembling the full Galerkin matrix for two-dimensional vector-valued linear finite elements.

HINT: The local matrix will now be of size 6×6, which means each local contribution has 36 elements instead of just 9. Your function will also have to keep track of the numbering of the basis functions as given in (5.4.2). Before you tackle this task carefully examine the code of [NPDE, Ex. 3.5.18].

(5.4f) Write a function

```
d = main_2D(Mesh)
```

that takes in a LehrFEM mesh complete with vertex, edge and element data, computes the global Galerkin matrix for the bilinear form `a` on the mesh, and then implements boundary conditions (i.e., discarding the superfluous degrees of freedom), and returns in `d` the dimension of the nullspace of $\mathbf{A}$ (i.e. if $\mathbf{A}$ is $N \times N$, then $d = N - \operatorname{rank} \mathbf{A}$).

HINT: Base your code on one of the functions written previously, which should already contain most of what is needed. Use the MATLAB `rank` function that tells you the rank of a full matrix. Use `full` to convert to this format.

(5.4g) The supplied function `Mesh = makeMesh(n)` returns a "criss-cross" mesh with $(n+1)^2 + n^2$ vertices and $4n^2$ elements, complete with vertex, edge and element data. Let your code compute the dimension of the kernel for the Galerkin matrix for these meshes from $n = 1$ to $n = 10$. What do you observe?

Listing 5.6: Testcalls for Problem 5.4

```
fprintf('\n##STIMA_Div_LFE');
STIMA_Div_LFE([0 0; 1 0; 0 1])

clear Mesh
Mesh.Coordinates = 2*[0 0; 1 0; 1 1; 0 1];
Mesh.Elements = [1 2 3; 1 3 4];

fprintf('\n##assemMat_2DLFE (depends on STIMA_Div_LFE)');
assemMat_2DLFE(Mesh, @STIMA_Div_LFE)

fprintf('\n##main_p3 (depends on assemMat_2DLFE and STIMA_Div_LFE)');
main_p3(makeMesh(5))
```

Listing 5.7: Output for Testcalls for Problem 5.4

```
>> test_p3

##STIMA_Div_LFE
ans =

    0.5000   -0.5000         0    0.5000         0   -0.5000
   -0.5000    0.5000         0   -0.5000         0    0.5000
    0         0         0         0         0         0
    0.5000   -0.5000         0    0.5000         0   -0.5000
    0         0         0         0         0         0
   -0.5000    0.5000         0   -0.5000         0    0.5000
```

```
##assemMat_2DLFE (depends on STIMA_Div_LFE)
ans =

   (1,1)       0.5000
   (2,1)      -0.5000
   (6,1)       0.5000
   (7,1)      -0.5000
   (1,2)      -0.5000
   (2,2)       0.5000
   (6,2)      -0.5000
   (7,2)       0.5000
   (3,3)       0.5000
   (4,3)      -0.5000
   (5,3)      -0.5000
   (8,3)       0.5000
   (3,4)      -0.5000
   (4,4)       0.5000
   (5,4)       0.5000
   (8,4)      -0.5000
   (3,5)      -0.5000
   (4,5)       0.5000
   (5,5)       0.5000
   (8,5)      -0.5000
   (1,6)       0.5000
   (2,6)      -0.5000
   (6,6)       0.5000
   (7,6)      -0.5000
   (1,7)      -0.5000
   (2,7)       0.5000
   (6,7)      -0.5000
   (7,7)       0.5000
   (3,8)       0.5000
   (4,8)      -0.5000
   (5,8)      -0.5000
   (8,8)       0.5000

##main_p3 (depends on assemMat_2DLFE and STIMA_Div_LFE)
ans =

     9
```

Problem 5.5 Heat Distribution in a Submerged Wire

This problem is a continuation of Problem 3.5 and supplies a complete LehrFEM implementation for the Galerkin finite element discretization based on piecewise linear finite element functions. The implementation is challenging, but mastery of this problem will really hone your LehrFEM skills.

In this problem we solve (3.5.1) on a domain which is the annulus between two non-concentric

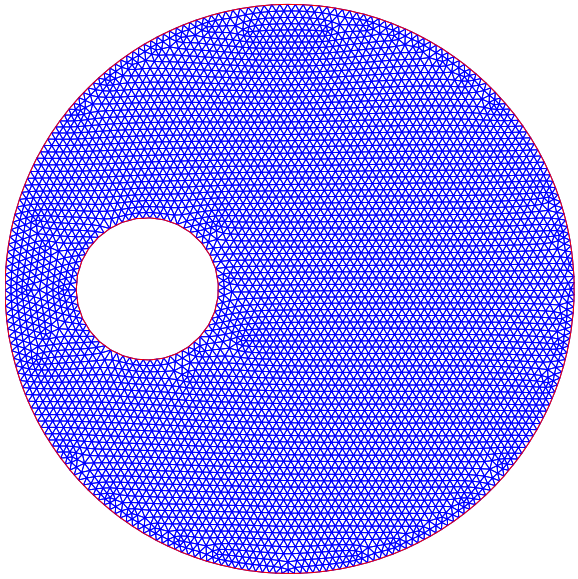


Figure 5.5: The mesh used in Problem 5.5

circles, just as explained in Problem 3.5.

Most of the implementation will be done in a script called `wire.m` which we will be develop throughout this problem. Thus, when no filename is specified, the code should be incorporated into this file.

(5.5a) Load the mesh in `Coord_Annulus.dat` and `Elem_Annulus.dat` into the variable `Mesh`. Add element flags, edges and edge to element data to the mesh (as is done in `main_LFE_p2.m`). Plot the mesh.

HINT: See [LehrFEM, Sect. 1.1] for a description of the mesh data structure and the functions you are supposed to call. Use `plot_Mesh` ([LehrFEM, Sect. 1.3.2]) to plot the mesh.

(5.5b) We now need to add boundary flags to the mesh. This is a little challenging, because there are two distinct parts of the boundary, which should be treated separately. Create a function

```
Mesh = wire_add_BdFlags(Mesh)
```

In this function, call `loc = get_BdEdges(Mesh)` ([LehrFEM, Sect. 1.1]) to get a vector with the indices of the boundary edges. Then, allocate a vector `Mesh.BdFlags` of zeros, with one index per edge. An element of this vector should be zero if it corresponds to an internal edge, -1 if it corresponds to an edge on Γ_0 (the outer boundary), and -2 if it corresponds to an edge on Γ_1 (the inner boundary).

Loop through each edge index in `loc` and get the endpoints from the `Mesh.Coordinates` matrix. Γ_1 is a circle centered at the origin with radius 1. Therefore, a boundary edge will be part of Γ_1 if both the endpoints are, say, within distance 1.1 from the origin.

Finally, run this function on your mesh.

(5.5c) Assemble the full Galerkin matrix `A` and right-hand side load vector `L`. Also, assemble the initial solution vector `U` using `assemDir_LFE`. We will use a constant heat conductivity of

$\kappa \equiv 1$.

HINT: Use `assemMat_LFE` to assemble the Galerkin matrix, see [LehrFEM, Sect. 5.1]. `EHandle` should point to `STIMA_Heat_LFE` (see [LehrFEM, Sect. 4.1]).

To assemble the right-hand side vector, you must loop over all edges on the inner boundary (those with boundary flag -2). Use the trapezoidal rule.

Have a look at [LehrFEM, Sect. 6.1] to see how the Dirichlet boundary data assembly works, and how the linear system should be modified.

(5.5d) Now, we will deal with the boundary condition on Γ_1, namely that the solution and test functions should be constant there. This means that all the nodal basis functions $\{b_N^{\mathbf{x}} : \mathbf{x} \in \mathcal{V}(\mathcal{M}) \cap \Gamma_1\}$ corresponding to vertices on Γ_1 should be replaced with a single *non-local* basis function

$$\tilde{b}_N = \sum_{\mathbf{x} \in \mathcal{V}(\mathcal{M}) \cap \Gamma_1} b_N^{\mathbf{x}}.$$

To accomplish this, we must sum the rows and columns in $\mathbf{A}$ corresponding to these degrees of freedom into one row and column. First, use the `find` function to locate the edges corresponding to Γ_1, i.e.

```
>> g_edges = find(Mesh.BdFlags == -2);
```

Then, the degrees of freedom corresponding to Γ_1 (`g`), and those who do not (`r`):

```
>> g = Mesh.Edges(g_edges,:); g = g(:); g = unique(g);
>> r = setdiff(1:size(Mesh.Coordinates,1), g);
```

Then, replace `A` with a block matrix

```
>> A = [A(r,r) B; C D];
```

where `B,C,D` are `A(r,g), A(g,r), A(g,g)` summed in the horizontal, the vertical and both directions respectively.

Note that the procedure above renumbers all degrees of freedom!

HINT: Use the MATLAB `sum` command.

(5.5e) Apply the same summing procedure to the load vector `L` and the initial solution vector `U`. That is, replace `L` with `[L(r); sum(L(g))]`, and the same with `U`.

(5.5f) Now we need to update the `FreeDofs` vector returned to us from `assemDir_LFE` according to the new degrees of freedom.

Write a function

```
fd = wire_update_FreeDofs(FreeDofs, r)
```

which takes as input the `FreeDofs` vector of free nodes *before* the renumbering, and the vector `r` from subproblem (5.5d), and returns the updated numbers of the free nodes. Afterwards, use this function to update your `FreeDofs` vector.

HINT: If `FreeDofs(i) == r(k)` then the updated number for that node is `k`. Remember that the elements of `FreeDofs` and `r` are unique.

Some elements in `FreeDofs` will not correspond to any element in `r`. These are nodes on Γ_1. The best way to deal with these is to simply ignore them while looping, and then add the number of the "big" degree of freedom at Γ_1 to `fd` in the end. The number of this is `length(r)+1`.

(5.5g) The time has now come to solve the linear system by calling

```
>> L = L - A*U;
>> U(FreeDofs) = A(FreeDofs,FreeDofs)\L(FreeDofs);
```

as described in [LehrFEM, Sect. 6.1]. Then we need to restore the original node numbering. Create a new column vector `V` with all zeros. The length of this vector should be equal to the size of the original system (that is `length(r) + length(g)`). Then, set all the elements of `V` corresponding to indices in `r` equal to the appropriate elements in `U`. Finally, set the elements of `V` corresponding to indices in `g` equal to `U(end)`, the element corresponding to the "big" degree of freedom at Γ_1.

(5.5h) Plot the solution!

HINT: Use `plot_LFE` [LehrFEM, Sect. 7.1].

Listing 5.8: Testcalls for Problem 5.5

```
Mesh.Coordinates = [1, 0; 0, 1; -1, 0; 0, -1;

                    2, 0; 1, 1; 0, 2; -1, 1;
                    -2, 0; -1, -1; 0, -2; 1, -1];
Mesh.Elements = [1, 5, 6; 1, 6, 2; 2, 6, 7; 2, 7, 8;
                 2, 8, 3; 3, 8, 9; 3, 9, 10; 3, 10, 4;
                 4, 10, 11; 4, 11, 12; 4, 12, 1; 1, 12, 5];
Mesh.ElemFlag = ones(size(Mesh.Elements,1),1);
Mesh = add_Edges(Mesh);
Mesh = add_Edge2Elem(Mesh);
Mesh = wire_add_BdFlags(Mesh);

disp('##wire_add_BdFlags');
find(Mesh.BdFlags)'

fd = [1,2,5,7,9,10,12,15,20,25,31];
r = [2,7,5,12,15,9];
disp('##wire_update_FreeDofs');
wire_update_FreeDofs(fd, r)
```

Listing 5.9: Output for Testcalls for Problem 5.5

```
>> test_call
##wire_add_BdFlags

ans =

     3     4     9    13    17    18    19    20    21    22    23
           24

##wire_update_FreeDofs
```

```
ans =

     1     3     2     6     4     5     7
```

Published on March 25.

To be submitted on April 1.

MATLAB: Submit all file in the online system. Include the files that generate the plots. Label all your plots. Include commands to run your functions. Comment on your results.

References

[NPDE] Lecture Slides for the course “Numerical Methods for Partial Differential Equations”, SVN revision # 53425.

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

[LehrFEM] LehrFEM manual.

Last modified on March 28, 2013