R. Hiptmair
L. Scarabosio
E. Fonn

Spring Term 2013

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 6

Introduction. This assignment includes first a brief theoretical problem on mappings, and how they help us in the assembly of stiffness matrices. Here we assemble "by hand" the matrix of a small mesh composed of quadrilaterals.

Problem 6.2 provides a different use case for domain transformation, which allows us to solve an equation on a circle without having to introduce complicated boundary approximation.

In Problem 6.3 we take the idea of generic matrix assembly to its logical limit by solving our usual test problem with quadratic finite elements, of which the assembly routine is completely unaware.

Finally, in Problem 6.4 we offer an alternative method to treating Dirichlet boundary conditions by the use of so-called *penalized* formulations.

Problem 6.1 Mapped Quadrilaterals (Core problem)

[NPDE, Sect. 3.6.2] introduced "bilinear" finite elements on 2D meshes consisting of general quadrilaterals. The main idea was to obtain local shape functions through a "mapping technique" from the local shape functions on the unit square. To that end we relied on bilinear transformations [NPDE, Eq. (3.6.12)]. In this problem we practise the use of quadrilateral parametric bilinear finite elements.

Figure 6.1 depcits a mesh of quadrilaterals. The mesh approximates a circle of radius 1 centered at the origin, so vertices $2, 3, 7, 8$ have coordinates $x_1, x_2 = \pm 1/\sqrt{2}$. We use parametric bilinear finite elements on this mesh and this problem will demonstrate how we can use [NPDE, Eq. (3.6.18)] to compute element matrices. We will use the bilinear form (6.1.1) and the linear form (6.1.2).

$$\mathsf{a}(u,v) = \int_\Omega \mathbf{grad}\, u(\mathbf{x}) \cdot \mathbf{grad}\, v(\mathbf{x}) \mathrm{d}\mathbf{x}, \quad u, v \in H^1(\Omega), \tag{6.1.1}$$

$$\ell(v) = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) \mathrm{d}\mathbf{x}, \tag{6.1.2}$$

(6.1a) What is the dimension of the finite element space $\mathcal{S}_1^0(\mathcal{M})$?

In the rest of this problem, let K be the top-right quadrilateral (with vertices $1, 3, 6$ and 5).

(6.1b) Find a bilinear mapping $\Phi : \widehat{K} \to K$ that takes the unit square $\widehat{K}$ to K.

HINT: Map vertices $1, 5$ and 6 to themselves. Vertex $(1, 1)$ should be taken to vertex 3. See [NPDE, Eq. (3.6.12)].

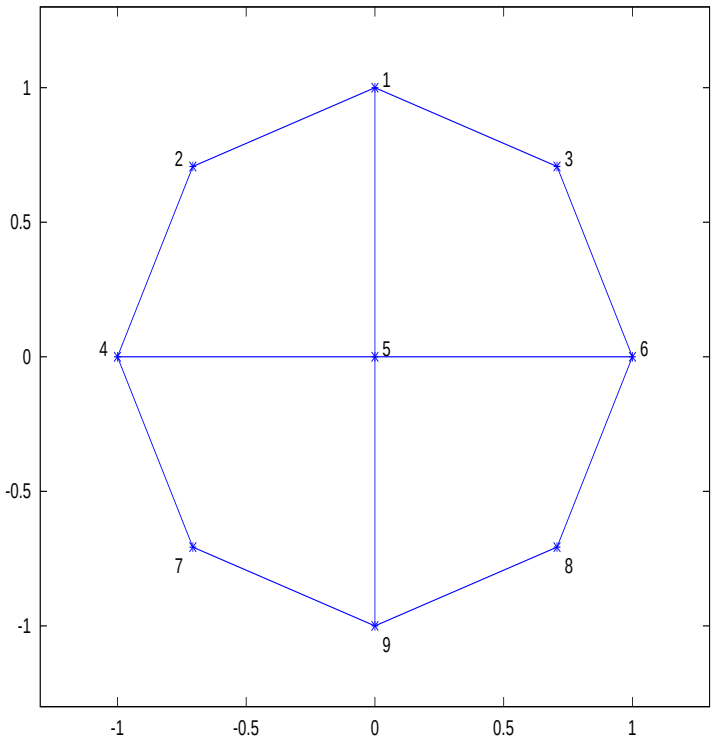

Figure 6.1: A mesh of quadrilaterals.

(6.1c) Compute the Jacobian $D\Phi(\mathbf{x})$ of Φ, its determinant, and its inverse transpose, as required by [NPDE, Eq. (3.6.18)].

(6.1d) Compute the gradients of the bilinear local shape functions $\widehat{b}^i$ on $\widehat{K}$.

HINT: The shape functions are given in [NPDE, Eq. (3.4.8)].

(6.1e) Compute the element Galerkin matrix for K using the formula [NPDE, Eq. (3.6.18)]. For integration, use the four-point quadrature rule (6.1.3) where $\mathbf{a}^i$ are the vertices of the unit square. Use a local numbering of the vertices where $(0,0)$ is vertex 1, $(1,0)$ is vertex 2, and so on.

$$\int_K f(\mathbf{x})\mathrm{d}\mathbf{x} \approx \frac{|K|}{4}\sum_{i=1}^{4} f(\mathbf{a}^i)\ . \tag{6.1.3}$$

(6.1f) Compute the element right-hand side vector for K. Use the quadrature formula (6.1.3) for approximating the integrals, and the same local numbering used in subproblem (6.1e).

(6.1g) Assemble the full 9×9 Galerkin matrix from the local contributions found in subproblem (6.1e).

HINT: The element Galerkin matrices are the same for all elements.

(6.1h) Compute the full right-hand side vector using the local contributions found in subproblem (6.1f).

Problem 6.2 Poisson Problem in Polar Coordinates

In the problem we will come across an important case of *transformation* of the domain of a boundary value problem prior to its discretization. The formulas derived in [NPDE, Sect. 3.6.3], in particular [NPDE, Lemma 3.6.16] and [NPDE, Eq. (3.6.18)], can be used to obtain the variational formulation of the new boundary value problem after the transformation from the original variational formulation.

Please be aware, that a transformation of the domain can also be viewed as a *change of coordinates*, and this will be the perspective adopted in this problem, because we study the concrete case of using *polar coordinates*, see also [NPDE, Eq. (2.3.16)] and [NPDE, Eq. (2.3.18)], to switch from the unit disc domain to a simple square domain. The rationale for doing this is obvious, because triangulating a square is straightforward, whereas the unit disc will entail relying on some kind of boundary approximation as discussed in [NPDE, Sect. 3.6.4].

Invariably, the change of coordinates will lead to variational problem with non-constant coefficients of a particular form posed on spaces with non-standard boundary condition. In this problem we will also practice the implementation of local computations and global assembly connected with a Lagrangian finite element discretization of the transformed problem.

Consider the boundary value problem

$$\begin{aligned} \Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{6.2.1}$$

on the domain $\Omega = \{x \in \mathbb{R}^2 : |x| < 1\}$. The transformation from polar coordinates (r, ϕ), $r \geq 0$, $0 \leq \phi < 2\pi$, to Cartesian coordinates $(x_1, x_2) \in \mathbb{R}^2$ is given by, *cf.* [NPDE, Eq. (2.3.16)],

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \mathbf{\Phi}(r, \phi) := r \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix}, \tag{6.2.2}$$

and we have $\Omega = \mathbf{\Phi}(\Omega_p)$, $\Omega_p := [0, 1] \times [0, 2\pi]$.

From [NPDE, Eq. (2.3.17)] and [NPDE, Eq. (2.3.18)] we derive the variational formulation for (6.2.1) in polar coordinates with $u = u(r, \phi)$: find $u \in V$

$$\underbrace{\int_0^1 \int_0^{2\pi} \frac{\partial u}{\partial r} \frac{\partial v}{\partial r} r \, \mathrm{d}r\mathrm{d}\phi}_{=:\mathsf{a}(u,v)} + \underbrace{\int_0^1 \int_0^{2\pi} \frac{1}{r} \frac{\partial u}{\partial \phi} \frac{\partial v}{\partial \phi} \, \mathrm{d}r\mathrm{d}\phi}_{=:\mathsf{b}(u,v)} = \underbrace{\int_0^1 \int_0^{2\pi} f(r, \phi)\, v r \, \mathrm{d}r\mathrm{d}\phi}_{=:\ell(v)} \quad \forall v \in V, \tag{6.2.3}$$

where V is a suitable Sobolev space of functions $\Omega_p \mapsto \mathbb{R}$.

We equip Ω_p with a tensor product mesh $\mathcal{M}$ as depicted in Figure 6.2. For the discretization of (6.2.3) we use $V_N \subset V$, the space of the bilinear Lagrangian finite element functions satisfying additional constraints (a)–(c)

$$\begin{aligned} V_N := \{ v_N \in \mathcal{S}_1^0(\mathcal{M}) : \text{(a)} \quad & v_N\big|_{r=1} = 0, \\ \text{(b)} \quad & v_N\big|_{r=0} = \text{const}, \\ \text{(c)} \quad & v_N\big|_{\phi=0} = v_N\big|_{\phi=2\pi} \}. \end{aligned}$$

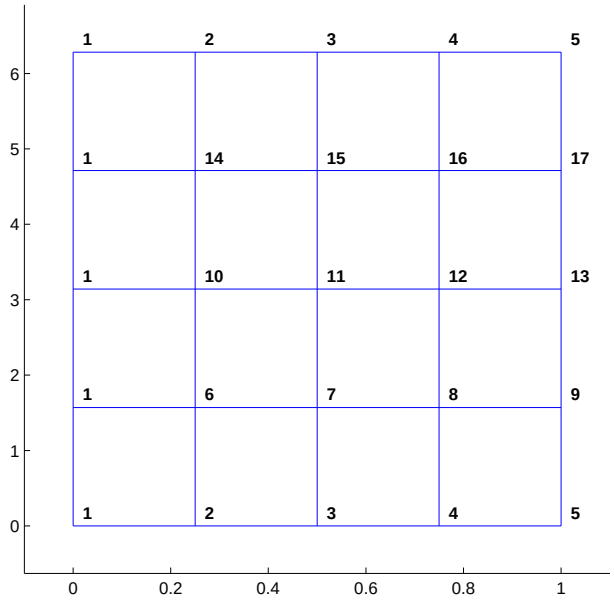

Figure 6.2: Mesh for Problem 6.2 with $n = 4$. Indicated is the numbering that should be used for the global shape functions in subproblem (6.2l).

(6.2a) This sub-problem takes a closer look at the specific formulas for polar coordinates. All these should have been treated in an introductory calculus course, so that this is a mere repetition.

- Compute the Jacobian $D\mathbf{\Phi}(r, \phi)$ of the transformation (6.2.2) from polar to Cartesian coordinates.
- Derive [NPDE, Eq. (2.3.18)] from the general transformation formula for multi-dimensional integrals [NPDE, Eq. (3.6.15)]
- Use Lemma [NPDE, Lemma 3.6.16] to confirm [NPDE, Eq. (2.3.17)].

(6.2b) Explain why constraint (a) has to be enforced on the trial and test functions for the boundary value problem (6.2.1)

(6.2c) Explain why condition (b) is essential.

(6.2d) Explain why constraint (c) has to be imposed.

(6.2e) What is the dimension N of V_N when using a tensor product mesh as in Figure 6.2 with n mesh cells both in radial and angular direction (mesh points $0 = r_0 < r_1 < \cdots < r_n = 1$ and $0 = \phi_0 < \phi_1 < \cdots < \phi_n = 2\pi$, see Figure 6.2).

(6.2f) Consider the tensor product mesh shown in Figure 6.2 and use the numbering of global shape functions indicated there. If several vertices carry the same number, the associated shape function is obtained as a suitable linear combination of standard nodal shape functions associated with all those vertices.

Copy the mesh from Figure 6.2 on paper and mark the supports of the global shape functions b_N^1, b_N^3, and b_N^7.

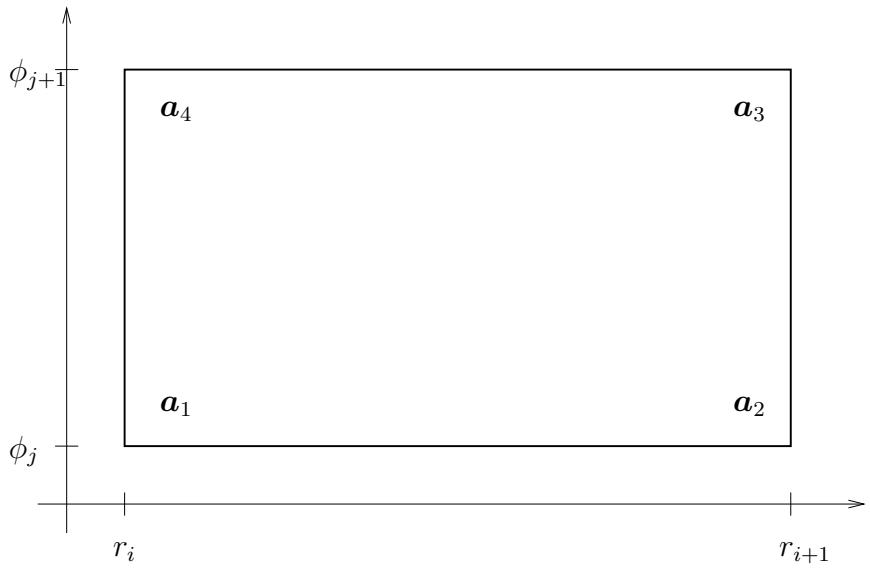

Figure 6.3: Local numbering of vertices of rectangle $K_{i,j}$

(6.2g) Write a MATLAB function

```
function Aloc = STIMA_LaplPolar_a(i,j,n)
```

that computes the 4×4 element matrix corresponding to the bilinear form a from (6.2.3) for the rectangle $K_{i,j}$, $0 \leq i, j < n$, of an $n \times n$ equidistant tensor product mesh, see Figure 6.2 for the numbering scheme for $n = 4$.

The vertices of $K_{i,j}$ are numbered as shown in Figure 6.3 and the associated bilinear local shape functions are

$$\begin{aligned} b_K^1(r,\phi) &:= \frac{r - r_{i+1}}{r_i - r_{i+1}} \frac{\phi - \phi_{j+1}}{\phi_j - \phi_{j+1}}, \\ b_K^2(r,\phi) &:= \frac{r - r_i}{r_{i+1} - r_i} \frac{\phi - \phi_{j+1}}{\phi_j - \phi_{j+1}}, \\ b_K^3(r,\phi) &:= \frac{r - r_i}{r_{i+1} - r_i} \frac{\phi - \phi_j}{\phi_{j+1} - \phi_j}, \\ b_K^4(r,\phi) &:= \frac{r - r_{i+1}}{r_i - r_{i+1}} \frac{\phi - \phi_j}{\phi_{j+1} - \phi_j}. \end{aligned} \tag{6.2.4}$$

(6.2h) Write a MATLAB function

```
function Bloc = STIMA_LaplPolar_b(i,j,n)
```

that computes the element matrix corresponding to the bilinear form b from (6.2.3) for the rectangle $K_{i,j}$, $1 \leq i < n$, $0 \leq j < n$, of an $n \times n$ equidistant tensor product mesh.

The vertices of $K_{i,j}$ are numbered as shown in Figure 6.3 and the associated bilinear local shape functions are given in (6.2.4).

(6.2i) What is the problem when trying to call `STIMA_LaplPolar_b` with $i = 0$? Which entries of the element matrix will turn out to be ill defined?

HINT: A scrambled reference implementation of `STIMA_LaplPolar_b` is provided in the file `STIMA_LaplPolar_b_ref.p`.

(6.2j) On $K_{0,j}$, $0 \leq j < n$, the problem addressed in the previous sub-problem can be overcome by using only three local shape functions. Characterize them as linear combinations of $b_K^1, \ldots, b_K^4$, see (6.2.4). Give a formula for the new "non-standard" local shape functions.

HINT: Remember constraint (b) in the definition of V_N.

(6.2k) Extend your implementation of `STIMA_LaplPolar_b` from subproblem (6.2h) so that for $i = 0$ it uses the three local shape functions found in the previous subproblem and returns the corresponding 3×3 element matrix belonging to the bilinear form `b`.

HINT: The constraint (b) means that there is only one basis function b_N^N of V_N, $N := \dim V_N$, associated with all vertices at $r = r_0 = 0$. Its restriction to each element abutting $r = 0$ is the sum of the local shape functions associated with vertices located on the $r = 0$ side of Ω_p.

(6.2l) A key device for cell oriented assembly of finite element Galerkin matrices is the local→global index mapping `locaglobmap` discussed in [NPDE, Sect. 3.5.3.1], see [NPDE, Eq. (3.5.14)]. Write a MATLAB function

```
function [i1, i2, i3, i4] = locglobmap(i, j, n)
```

that, for the rectangular cell $K_{i,j} = [r_i, r_{i+1}] \times [\phi_j, \phi_{j+1}]$, computes the global indices $i_1, \ldots, i_4$ corresponding to the vertices $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_4$ of K, where the local numbering of the vertices is given in Figure 6.3. The global numbering of vertices and their associated basis functions should follow the *lexikographic numbering* scheme indicated in Figure 6.2.

(6.2m) The scrambled MATLAB file `STIMA_Lapl_Polar.p` supplies the function

```
Mloc = STIMA_Lapl_Polar(i,j,n)
```

that returns the element matrix for the bilinear form `a` + `b` from (6.2.3). It makes use of the local shape functions from (6.2.4) for $i > 0$, and of the special local shape functions found in subproblem (6.2j) for $i = 0$. In former case it returns a 4×4-matrix, in the latter a 3×3 matrix.

Use this function to implement the assembly of the full Galerkin matrix for the variational problem (6.2.3) and trial and test space V_N in the MATLAB function

```
function A = assemMatPolar_BFE(n)
```

where an equidistant tensor product mesh with n cells in r- and ϕ-direction is used (see Figure 6.2).

HINT: Do not forget to take into account the constraints (b)–(c) imposed on the finite element space. You may ignore constraint (a) (so for example, with $n = 4$, the returned matrix should be 17×17.

(6.2n) Solve the BVP (6.2.1) in polar coordinates for $f \equiv 1$ for $n = 3$ and output the solution vector. Write a script `solve.m` for this.

HINT: You may use `L=assemLoadPolar_BFE(n)` which assembles the load vector for $f \equiv 1$.

Listing 6.1: Testcalls for Problem 6.2

```
fprintf('## assemLoadPolar_BFE\n');
assemLoadPolar_BFE(3)'

fprintf('## assemMatPolar_BFE\n');
full(assemMatPolar_BFE(2))

fprintf('## locglobmap\n');
[a,b,c,d] = locglobmap(0, 0, 3);
[e,f,g,h] = locglobmap(2, 2, 3);
[a,b,c,d,e,f,g,h]

fprintf('## STIMA_LaplPolar_a\n');
STIMA_LaplPolar_a(0, 0, 3)

fprintf('## STIMA_LaplPolar_b\n');
STIMA_LaplPolar_b(0, 0, 3)
```

Listing 6.2: Output for Testcalls for Problem 6.2

```
## assemLoadPolar_BFE

ans =

   -0.1164   -0.3879   -1.3187   -0.6206   -0.3879   -1.3187
      -0.6206   -0.3879   -1.3187   -0.6206

## assemMatPolar_BFE

ans =

    3.1416   -1.5708         0   -1.5708         0
   -1.5708    4.6806   -3.0692    1.6025   -1.6432
         0   -3.0692    3.2646   -1.6432    1.4478
   -1.5708    1.6025   -1.6432    4.6806   -3.0692
         0   -1.6432    1.4478   -3.0692    3.2646

## locglobmap

ans =

     1     2     5     1     9    10     4     3

## STIMA_LaplPolar_a

```

```
ans =

    0.3491   -0.3491   -0.1745    0.1745
   -0.3491    0.3491    0.1745   -0.1745
   -0.1745    0.1745    0.3491   -0.3491
    0.1745   -0.1745   -0.3491    0.3491

## STIMA_LaplPolar_b

ans =

         0         0         0
         0    0.2387   -0.2387
         0   -0.2387    0.2387
```

Problem 6.3 Generic Assembly in LehrFEM (Core problem)

This problem examines algorithmic aspects of cell-oriented the assembly of finite element Galerkin matrices in LehrFEM, see [NPDE, Sect. 3.5.3.1]and [LehrFEM, Sect. 5.1]. It is meant to demonstrate the fundamental algorithmic paradigms and foster awareness of the abstract principles of assembly: local computations and local→global relationships.

Starting point is the MATLAB code given in Listing 6.3, which is a "generic" assembly function largely adhering to LehrFEM conventions, see [LehrFEM, Sect. 5.1]. This function `assemMat_generic` is available for download.

Listing 6.3: Generic LehrFEM assembly routine

```
function A =
   assemMat_generic(Mesh,EHandle,locglobmap,varargin)
% Generic LehrFEM assembly function for a finite element Galerkin
% matrix, see also [NPDE, Sect. 3.5.3.1] and, in particular
% [NPDE, Code 3.5.16] and [NPDE, Code 3.5.19].

% Mesh is supposed to contain a LehrFEM mesh data structure, see
% [NPDE, Sect. 3.5.2], maybe of basic type only, cf.
% [NPDE, Example 3.5.8]

% EHandle is a handle to a function performing the computation of
% the element (stiffness) matrices, called getElementMatrix in
% [NPDE, Code 3.5.16].  This function expects a 3×2-matrix
% argument, whose columns contain the coordinates of the vertices
% of the triangle whoses element matrix is requested.

% locaglobmap is a handle to a MATLAB function locglobmap that
% realizes the local→global index mapping from [NPDE, Eq. 3.5.14].

A = sparse(1, 1);

% number of triangles in the mesh
```

```
nElements = size(Mesh.Elements, 1);

% loop over all cells:  cell-oriented assembly
for i = 1:nElements
    % obtain vertex coordinates
    Vertices = Mesh.Coordinates(Mesh.Elements(i,:),:);

    % compute element matrix (→ [NPDE, Def. 3.5.13]) for current
    % triangle
    Aloc = EHandle(Vertices, varargin{:});

    % number of local shape functions
    Qk = size(Aloc, 1);

    % obtain a row vector of global indices associated with the
    % local shape functions, see the discussion of the local→global
    % index map in [NPDE, Sect. 3.5.3.1] and [NPDE, Eq. 3.5.14]
    idx = locglobmap(i, 1:Qk);

    % increase the size of A if we must
    if max(idx) > size(A,1)
        A(max(idx),max(idx)) = 0;
    end

    % Add element matrix to a suitable sub-matrix of the Galerkin
    % matrix in the spirit of [NPDE, Eq. 3.5.23]
    A(idx,idx) = A(idx,idx) + Aloc;
end
```

HINT: Note that the function `assemMat_generic` takes a function handle argument **locglobmap**. This function will have to access mesh information. In order to pass it, you can use *partial application*, a feature that permits you to define handles to functions with some of the arguments already specified, see Listing 6.4.

Listing 6.4: Partial application in MATLAB

```
function out = myfun(a, b, c, d, e)
    % Function definition here.
end

b_val = % Something...
d_val = % Something...
e_val = % Something...
partial_myfun = @(a, c) myfun(a, b_val, c, d_val, e_val)
% partial_myfun now takes two variables.
```

We focus on the generation of the finite element Galerkin matrix for the bilinear form

$$\mathsf{a}(u,v) = \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x}\ , \quad u,v \in H^1(\Omega)\ , \tag{6.3.1}$$

where the Lagrangian finite elements space $\mathcal{S}_2^0(\mathcal{M})$ on a triangular mesh of a polygonal domain $\Omega \subset \mathbb{R}^2$ is used as trial and test space. The use of the nodal basis as defined [NPDE, Ex. 3.4.2] is assumed throughout.

(6.3a) Give formulas for the entries of a generic element (stiffness) matrix for the bilinear form a from (6.3.1) and $\mathcal{S}_2^0(\mathcal{M})$ for a generic triangle K in terms of the angles ω_1, ω_2, ω_3 and the area $|K|$ of that triangle.

HINT: Use the barycentric coordinate representation of local shape functions from [NPDE, Eq. 3.4.4], [NPDE, Eq. 3.2.10], the formula [NPDE, Eq. (3.5.27)], the fact that gradients of barycentric coordinate functions are constant, and [NPDE, Lemma 3.5.30] for $d = 2$.

(6.3b) Write a MATLAB function

```
function Aloc = STIMA_Lapl_QFE_nt(Vertices,varargin)
```

that computes the $\mathcal{S}_2^0(\mathcal{M})$-element matrix for the bilinear form a from (6.3.1) on a triangle K, based on the coordinates of the vertices of the triangle, which are passed in the 3×2-matrix `Vertices`, see [LehrFEM, Sect. 4.1]. The `varargin` argument can be ignored, since it was introduced only for LehrFEM compatibility. Avoid using any trigonometric functions or their inverses!

HINT: A scrambled reference implementation of this function is available for download. The `varargin` argument has been included for compatibility with LehrFEM conventions, but it can be ignored in the implementation. Note that LehrFEM by convention places edges 4, 5 and 6 opposite of vertices 3, 1 and 2 respectively!

(6.3c) Write a MATLAB function

```
function A = assemMat_Lapl_QFE(Mesh)
```

that takes a LehrFEM mesh data structure complete with edge information as argument `Mesh` and returns the Galerkin finite element matrix for $\mathcal{S}_2^0(\mathcal{M})$ and the bilinear form (6.3.1), that is, this function serves the same purpose as [NPDE, Code 3.5.19] when give a suitable `EHandle` function. Your implementation is supposed to rely on the functions `STIMA_Lapl_QFE` from sub-problem (6.3b) and `assemMat_generic`.

HINT: Here you will have to rely on partial application, see Listing 6.4.

(6.3d) Write a MATLAB function

```
function A = computeMat_Lapl_QFE
```

that reads in a mesh from the files `Coord.dat` and `Elem.dat`, see [NPDE, Sect. 3.5.1], and computes the $\mathcal{S}_2^0(\mathcal{M})$-finite element Galerkin matrix for the bilinear form a from (6.3.1) on it. It should also output the number of non-zero matrix entries and the "`spy`-pattern" of the Galerkin matrix. Example files `Coord.dat` and `Elem.dat` are available for download.

(6.3e) Explain why the function `assemMat_generic` will usually perform very poorly in terms of computing time, when passed a mesh with many cells.

(6.3f) Copy the file `assemMat_generic.m` to `assemMat_generic_fast.m`. In this new file improve the implementation of `assemMat_generic` so that it runs much more efficiently. Explain, why your redesigned functions can be expected to perform better.

HINT: Study the implementation of [NPDE, Code 3.5.19].

(6.3g) Compare the runtimes of `assemMat_Lapl_QFE` for the mesh supplied in `Coord.dat` and `Elem.dat`

1. when using the original implementation of `assemMat_generic` given in Listing 6.3,
2. when using `assemMat_generic_fast` instead.

HINT: Use MATLAB's `tic/toc` timing facility. Take the average runtime over several runs. You can make the mesh bigger by calling `refine_REG`.

Listing 6.5: Testcalls for Problem 6.3

```
Mesh.Coordinates = [0 0; 1 0; 1 1; 0 1];
Mesh.Elements = [1 2 4; 2 3 4];
Mesh = add_Edges(Mesh);

fprintf('## assemMat_Lapl_QFE\n');
round(6*full(assemMat_Lapl_QFE(Mesh))

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

Listing 6.6: Output for Testcalls for Problem 6.3

```
## assemMat_Lapl_QFE

ans =

     6     1     0     1    -4    -4     0     0     0
     1     6     1     0    -4     0     0    -4     0
     0     1     6     1     0     0     0    -4    -4
     1     0     1     6     0    -4     0     0    -4
    -4    -4     0     0    16     0    -8     0     0
    -4     0     0    -4     0    16    -8     0     0
     0     0     0     0    -8    -8    32    -8    -8
     0    -4    -4     0     0     0    -8    16     0
     0     0    -4    -4     0     0    -8     0    16

## STIMA_Lapl_QFE_nt

ans =

    1.0000    0.1667    0.1667   -0.6667         0   -0.6667
```

```
20     0.1667    0.5000   -0.0000   -0.6667    0.0000         0
21     0.1667   -0.0000    0.5000         0    0.0000   -0.6667
22    -0.6667   -0.6667         0    2.6667   -1.3333    0.0000
23          0    0.0000    0.0000   -1.3333    2.6667   -1.3333
24    -0.6667         0   -0.6667    0.0000   -1.3333    2.6667
```

Problem 6.4 Treatment of Dirichlet boundary conditions by penalty approach

In [NPDE, Sect. 2.9] we learned that for second-order elliptic boundary value problems (BVPs) Dirichlet boundary conditions are *essential boundary conditions* in the sense that they have to be incorporated into trial and test spaces in the variational formulation, for instance, by relying on the Sobolev spaces $H^1_0(\Omega)$.

However, sometimes, it may be desirable to dispense with boundary conditions in trial and test spaces, though one wants to tackle a Dirichlet BVP. This problem discusses a way how to take into account Dirichlet boundary conditions in the weak formulation only by means of a technique called *penalization*.

Throughout, we consider the Poisson problem, see [NPDE, Remark 2.4.12],

$$-\Delta u = f \quad \text{in } \Omega \quad , \quad u = g \quad \text{on } \partial\Omega \, , \tag{6.4.1}$$

for $f \in L^2(\Omega)$ and polygon $\Omega \subset \mathbb{R}^2$.

Temporarily, we restrict ourselves to $g \equiv 0$. As has been explained in [NPDE, Sect. 2.4] and [NPDE, Sect. 2.1.3], the BVP (6.4.1) arises from a quadratic minimization problem posed on $H^1_0(\Omega)$. However, now we consider the *penalized* quadratic minimization problem

$$u_* = \operatorname*{argmin}_{v \in H^1(\Omega)} J_\alpha(v) \, , \quad J_\alpha(v) := \tfrac{1}{2} \int_\Omega \|\mathbf{grad}\, v\|^2 \, \mathrm{d}\boldsymbol{x} + \tfrac{1}{2}\alpha \int_{\partial\Omega} |v|^2 \, \mathrm{d}S - \int_\Omega f v \, \mathrm{d}\boldsymbol{x} \, , \tag{6.4.2}$$

with a *penalty parameter* $\alpha > 0$.

(6.4a) Write down the linear variational problem equivalent to (6.4.2).

(6.4b) Explain, why (6.4.2) has a unique solution.

HINT: You may re-examine [NPDE, Rem. 2.8.13].

(6.4c) Show that $\|u_*\|_{L^2(\partial\Omega)} \to 0$ as $\alpha \to \infty$

HINT: Use that the solution of (6.4.2) on $H^1_0(\Omega)$ does not depend on α and that the minimum of J over the larger space $H^1(\Omega)$ will invariably less or equal its minimal value on $H^1_0(\Omega)$.

(6.4d) Write down the second-order elliptic boundary value problem solved by u_* from (6.4.2).

HINT: The solution can be found in [NPDE, Sect. 2.8].

(6.4e) Now consider

- the unit disk domain $\Omega = \{\boldsymbol{x} \in \mathbb{R}^2 : \|\boldsymbol{x}\| < 1\}$,

- $g \equiv 0$ and $f(\boldsymbol{x}) = \frac{\pi^2}{4}\left(\cos\left(\frac{\pi}{2}\left\|\boldsymbol{x}\right\|\right) + \operatorname{sinc}\left(\frac{\pi}{2}\left\|\boldsymbol{x}\right\|\right)\right)$,

which leads to the rotationally symmetric smooth solution $u(\boldsymbol{x}) = \cos(\frac{\pi}{2}\left\|\boldsymbol{x}\right\|)$ of the BVP (6.4.1). Recall that $\operatorname{sinc}(x) = \frac{1}{x}\sin(x)$.

The LehrFEM function

```
u = solveRobinBVP(Mesh,CoeffHandle,FHandle)
```

solves the following second-order elliptic BVP

$$-\Delta u = f \quad \text{in } \Omega \quad , \quad \mathbf{grad}\, u \cdot \boldsymbol{n} + \eta(\boldsymbol{x})u = 0 \quad \text{on } \partial\Omega \, , \tag{6.4.3}$$

where `CoeffHandle` supplies the (positive) function $\eta \in C^0(\partial\Omega)$ and `FHandle` the function $f \in C^0(\Omega)$. The return value provides the (nodal) basis expansion coefficient of the finite element solution in $\mathcal{S}_1^0(\mathcal{M})$.

Use the function `solveRobinBVP` with the mesh read in from the files `Coords.dat` and `Elem.dat` to compute the $\mathcal{S}_1^0(\mathcal{M})$ finite element solutions $u_N(\alpha) \in \mathcal{S}_1^0(\mathcal{M})$ for $\alpha = 2^j$, $j = 0, \ldots, 16$. Then compute the error norms $\|u - u_N(\alpha)\|_{L^2(\Omega)}$ approximately using the LehrFEM function `L2Err_LFE`, see [LehrFEM, Sect. 8.4] and the seven-point quadrature rule of order six available in `P706.m`, see [LehrFEM, Chapter 3]. Plot these error norms versus the values of the parameter α. Do all this in a MATLAB script `BdPenalTest.m`.

(6.4f) We first introduce an important concept related to variational formulations:

Definition. A variational formulation for a (linear) second-order boundary value problem is called *consistent* with the boundary value problem, if its solution u, which is assumed to be sufficiently smooth, also provides a solution of the boundary value problem.

Show that the variational problem from sub-problem (6.4a) is *not* consistent with the boundary value problem (6.4.1).

(6.4g) Write a_α, $\alpha > 0$, for the bilinear form occurring in the linear variational problem found in sub-problem (6.4a). Show that the modified variational problem: seek $u \in C^1_{\mathrm{pw}}(\overline{\Omega}) \subset H^1(\Omega)$ such that

$$\widehat{\mathsf{a}}_\alpha(u,v) := \mathsf{a}_\alpha(u,v) - \int_{\partial\Omega} v\ \mathbf{grad}\, u \cdot \boldsymbol{n}\, \mathrm{d}S = \int_\Omega fv\, \mathrm{d}\boldsymbol{x} \quad \forall v \in H^1(\Omega) \, , \tag{6.4.4}$$

is *consistent* with the boundary value problem (6.4.1) in the case $g \equiv 0$.

HINT: Start from (6.4.1), multiply with a test function v and use Green's formula from [NPDE, Thm. 2.4.7]. Essentially follow the steps laid out in the beginning of [NPDE, Sect. 2.8], but take into account that v does not vanish on $\partial\Omega$.

(6.4h) A shortcoming of (6.4.4) is that the underlying bilinear form is *not symmetric*. This can be remedied by augmenting it by an additional term; we consider the augmented variational problem: seek $u \in C^1_{\mathrm{pw}}(\overline{\Omega})$ such that

$$\widetilde{\mathsf{a}}_\alpha(u,v) := \widehat{\mathsf{a}}_\alpha(u,v) - \int_{\partial\Omega} u\ \mathbf{grad}\, v \cdot \boldsymbol{n}\, \mathrm{d}S = \int_\Omega fv\, \mathrm{d}\boldsymbol{x} \quad \forall v \in C^1_{\mathrm{pw}}(\overline{\Omega}) \, . \tag{6.4.5}$$

Confirm that (6.4.5) is still consistent with (6.4.1) in the case $g \equiv 0$. The variational formulation (6.4.5) is called the *Nitsche boundary panelization* for the Dirichlet problem for the Laplacian.

HINT: Notice that the solution u satisfies a boundary condition.

(6.4i) Why do you think the variational problem (6.4.5) was stated on the space $C^1_{\mathrm{pw}}(\overline{\Omega})$ instead of $H^1(\Omega)$?

(6.4j) So far we focused on homogeneous Dirichlet boundary conditions $g \equiv 0$. Augment the right hand side functional in (6.4.5) by suitable terms in order to achieve a variational formulation that is consistent with the BVP (6.4.1) for general $g \in C^0(\partial\Omega)$.

(6.4k) Write a MATLAB function

```
function Aloc = ...
      STIMA_Lapl_Nitsche(Vertices,bdedges,alpha,varargin)
```

that computes the element (stiffness) matrices for the bilinear form $\widetilde{\mathsf{a}}_\alpha$ from (6.4.5) and the Lagrangian finite element space $\mathcal{S}^0_1(\mathcal{M})$ on a triangular mesh, of course equipped with the "tent function" nodal basis.

The argument `Vertices` passes a 3×2 matrix, whose rows contain the coordinates of the vertices of the triangle. The integer vector `bdedges` gives local numbers of those edges of the triangle located on $\partial\Omega$. Here an edge carries local number i, if it is opposite to local vertex i, $i = 1, 2, 3$. If the triangle is not adjacent to $\partial\Omega$, then `bdedges` will be empty, which can be checked by calling the MATLAB function `isempty()`. The parameter `alpha` contains α. The `varargin` argument can be ignored and is included only for the sake of consistency with LehrFEM conventions, see [LehrFEM, Sect. 4.1].

HINT: You may use [NPDE, Eq. 3.2.10], but please avoid evaluating any direct or inverse trigonometric functions! A scrambled reference implementation of `STIMA_Lapl_Nitsche` can be downloaded.

(6.4l) Write a MATLAB assembly routine

```
function A = assemMat_Nitsche(Mesh,alpha)
```

that efficiently computes the finite element Galerkin matrix for the bilinear form $\widetilde{\mathsf{a}}_\alpha$ from (6.4.5) and the finite element space $\mathcal{S}^0_1(\mathcal{M})$ on a triangular mesh $\mathcal{M}$. Here `Mesh` passes a LehrFEM mesh data structure ($\rightarrow$ [NPDE, Sect. 3.5.2], [LehrFEM, Sect. 1.1]) with complete edge information, see [NPDE, Ex. 3.5.8], whereas `alpha` gives the value of α.

HINT: Start from `assemMat_LFE` and extend it. Of course, the function `STIMA_Lapl_Nitsche` from sub-problem (6.4k) should be used. Again, a reference implementation of `assemMat` is provided.

(6.4m) Implement a MATLAB function

```
function u = solveNitsche(Mesh,FHandle,alpha)
```

that computes the finite element solution (in terms of its basis expansion coefficients) in $\mathcal{S}_1^0(\mathcal{M})$ of (6.4.5). Here, `Mesh` passes a simple LehrFEM mesh data structure that may not contain edge information, see [NPDE, Sect. 3.5.2]). `FHandle` provides a function handle of type `@(x)` for the source function f, where `x` is a 2-vector of point coordinates. Finally, `alpha` contains the value of the penalty parameter α.

HINT: A scrambled reference implementation of `solveNitsche` is supplied.

(6.4n) Repeat the evaluations of sub-problem (6.4e) with `solveNitsche` replacing `solveRobinBVP`.

Listing 6.7: Testcalls for Problem 6.4

```
Mesh.Coordinates = [0 0; 1 0; 1 1; 0 1];
Mesh.Elements = [1 2 4; 2 3 4];
Mesh = add_Edges(Mesh);
loc = get_BdEdges(Mesh);
Mesh.BdEdges = zeros(size(Mesh.Edges,1),1);
Mesh.BdEdges(loc) = -1;

fprintf('## assemMat\n');
full(assemMat(Mesh, 1))

fprintf('## solveNitsche\n');
solveNitsche(Mesh, @(x,varargin) sum(x.^2,2), 1)'

fprintf('## STIMA_Lapl_Nitsche\n');
STIMA_Lapl_Nitsche([0 0; 1 0; 0 0.5], [1 2 3], 1)
```

Listing 6.8: Output for Testcalls for Problem 6.4

```
## assemMat

ans =

   -0.3333   -0.3333         0   -0.3333
   -0.3333    1.6667   -0.3333    2.0000
         0   -0.3333   -0.3333   -0.3333
   -0.3333    2.0000   -0.3333    1.6667

## solveNitsche

ans =

   -0.0933   -0.0033   -0.5933   -0.0033

## STIMA_Lapl_Nitsche

ans =

   -0.7500    0.4167    1.0833
    0.4167    0.4560    0.1863
    1.0833    0.1863   -0.4607
```

Published on 04.04.2013.

To be submitted on 11.04.2013.

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 # 54024.

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

[LehrFEM] LehrFEM manual.

Last modified on April 11, 2013