R. Hiptmair
L. Scarabosio
E. Fonn

Spring Term 2013

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 7

Introduction. The first two problems revisit Lagrangian finite element methods treated in [NPDE, Ch. 3]. Problems 7.3 and 7.4 examine the discretization of elliptic boundary value problems on tensor product meshes of tensor product domains, which is a common setting for finite difference methods addressed in [NPDE, Sect. 4.1].

Problem 7.1 Dirichlet Boundary Value Problem with Delta-Function Right-Hand Side (Core problem)

On of the messages of [NPDE, Sect. 2.3.2] was that not all linear variational problems on function spaces are well-posed, see [NPDE, Def. 2.3.10]. In this problem we will encounter a specimen of an innocent looking ill-posed linear variational problem that was discussed in detail in [NPDE, Ex. 2.3.14]. We are going to study how the ill-posed nature of the continuous variational problem will be reflected by the behavior of Galerkin solutions. This will be done empirically based on a LehrFEM MATLAB implementation.

In this problem we consider the linear variational problem

$$u \in H^1_0(\Omega): \quad \int_\Omega \operatorname{grad} u(\boldsymbol{x}) \cdot \operatorname{grad} v(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = v(0) \quad \forall v \in H^1_0(\Omega), \tag{7.1.1}$$

posed on a polygon Ω and attempt its numerical solution by means of a linear finite element Galerkin discretization.

(7.1a) Describe a physical system for which (7.1.1) provides a model of certain aspects. What is the meaning of u in this model.

HINT: Remember a particular model discussed in [NPDE, Ch. 2].

(7.1b) Which problem haunts the linear variational problem (7.1.1)? What can you say about the existence of a minimizer of the associated quadratic functional?

HINT: Refresh yourself on the contents of [NPDE, Sect. 2.1.3]. Again study the beginning on [NPDE, Sect. 2.2] and [NPDE, Rem. 2.3.14].

(7.1c) Let $\mathcal{M}$ be a triangular mesh of Ω. The Galerkin discretization of (7.1.1) based on $\mathcal{S}^0_{1,0}(\mathcal{M})$ leads to a discrete variational problem, *cf.* [NPDE, Sect. 3.1]. Explain why the associated discrete quadratic minimization problem (see [NPDE, Eq. (1.5.5)]) always has a unique solution.

(7.1d) Let $\mathcal{M}_k$ be a sequence of triangular meshes, where $\mathcal{M}_{k+1}$ is generated from $\mathcal{M}_k$ by regular refinement of all triangles. Guess how the minimal value of the quadratic functional

associated with the variational problem (7.1.1) will behave as $k \to \infty$.

(7.1e) The function

```
L = assemLoad_LFE(Mesh, QuadRule, FHandle, varargin)
```

in the directory `Lib/Assembly` assembles the right-hand side vector for finite element Galerkin discretization of the BVP $-\Delta u = f$, $u = 0$ on $\partial\Omega$, based on $\mathcal{S}^0_{1,0}(\mathcal{M})$, see [NPDE, Sect. 3.2.6]. The argument `Mesh` passes the LehrFEM mesh data structure for $\mathcal{M}$, `QuadRule` the quadrature rule, and `FHandle` is a function handle to the source function $f(\boldsymbol{x})$, see [LehrFEM, Sect. **??**].

Make a copy of this file called `assemLoad_Delta_LFE.m`. In this function, implement the specific right-hand side of (7.1.1). The function call should be

```
L = assemLoad_Delta_LFE(Mesh)
```

The function should abort with an error, in case the point 0 is not contained in any triangle.

HINT: For each triangle K with vertices $\boldsymbol{a}^1$, $\boldsymbol{a}^2$, $\boldsymbol{a}^3$, compute the barycentric coordinates $\lambda_i(\boldsymbol{x})$, $i = 1, 2, 3$, of $\boldsymbol{x} = 0$ by solving the linear system

$$\begin{pmatrix} a_1^1 & a_1^2 & a_1^3 \\ a_2^1 & a_2^2 & a_2^3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1(\boldsymbol{x}) \\ \lambda_2(\boldsymbol{x}) \\ \lambda_3(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} . \tag{7.1.2}$$

If one of the $\lambda_i(\boldsymbol{x})$ is negative, the point $\boldsymbol{x}$ lies outside the triangle. Otherwise the values $\lambda_i(\boldsymbol{x})$ already provide the values of the local shape functions at $\boldsymbol{x}$, see [NPDE, Ex. 3.3.13].

(7.1f) Now we consider (7.1.1) on the unit disk $\Omega := \{\boldsymbol{x} \in \mathbb{R}^2 : \|\boldsymbol{x}\| < 1\}$. An incomplete function `main_delta` is supplied, with header

```
[N, l2, h1] = main_delta(n)
```

It takes in a number $n \in \{1, 2, 3, 4\}$, which selects a mesh of the unit circle. The loading of the mesh and computation of the stiffness matrix is already implemented.

Extend this function so that it computes the load vector using the function you wrote in (7.1e), and returns the number of degrees of freedom in `N`, and the L^2-norm and H^1-seminorms of the solution in `l2` and `h1`, respectively.

(7.1g) Solve (7.1.1) and plot the solution on the meshes `Coord_Circ_N.dat` and `Elem_Circ_N.dat` for `N=1,...,4`. Plot the L^2-norm and H^1-norm of the solutions vs. the number of degrees of freedom.

(7.1h) Consider an abstract linear variational problem

$$u \in V_0 : \quad \mathsf{a}(u, v) = \ell(v) \quad \forall v \in V_0, \tag{7.1.3}$$

with s.p.d. ($\to$ Def. [NPDE, Def. 2.1.25]) bilinear form a and a linear form ℓ that is continuous in the energy norm in the sense of [NPDE, Eq. (2.2.1)].

Elaborate the relationship between the energy norm of the solution of (7.1.3) and the minimal value of the associated quadratic functional according to [NPDE, Rem. 1.4.5].

(7.1i) Explain your observations on the behavior of the H^1-norm in subproblem (7.1g) in light of the theoretical conclusions obtained in subproblems (7.1d) and (7.1h).

Listing 7.1: Testcalls for Problem 7.1

```
fprintf('\n##assemLoad_LFE');
Mesh.Coordinates = [-1 0; 1 -2; 1 3];
Mesh.Elements = [1 2 3];
assemLoad_Delta_LFE(Mesh)
```

Listing 7.2: Output for Testcalls for Problem 7.1

```
>> test_p1

##assemLoad_LFE
ans =

    0.5000
    0.3000
    0.2000
```

Problem 7.2 Basis Transformation

So far the presentation in class has probably created the impression that there is a "canonical" basis for every Lagrangian finite element space ($\rightarrow$ [NPDE, DEf. 3.4.1]), characterized by the duality condition [NPDE, Eq. (3.4.3)] with respect to a set of interpolation nodes. However, for higher degree Lagrangian finite element spaces $\mathcal{S}_p^0(\mathcal{M})$, $p \geq 1$, there are many ways to choose basis functions without sacrificing the crucial *local support property* from [NPDE, Sect. 3.3.3]. In this problem we will see how this works for $p = 2$ and how coefficient representations with respect to different bases can be converted into each other.

Let a polygon $\Omega \subset \mathbb{R}^2$ be equipped with a triangular mesh $\mathcal{M}$, whose vertices (set $\mathcal{V}(\mathcal{M})$) and edges (set $\mathcal{E}(\mathcal{M})$) are numbered from 1 to $\sharp\mathcal{V}(\mathcal{M})$ and $\sharp\mathcal{E}(\mathcal{M})$, respectively.

We consider the space $V_N = \mathcal{S}_2^0(\mathcal{M})$ of quadratic Lagrangian finite element functions on $\mathcal{M}$. A basis of this space is given by the standard ordered nodal basis from [NPDE, Ex. 3.4.2]

$$\mathcal{B}_q = \{\widetilde{b}_N^j\}_{j=1}^N, \qquad N = \sharp\mathcal{V}(\mathcal{M}) + \sharp\mathcal{E}(\mathcal{M}),$$

where we number the $\sharp\mathcal{V}(\mathcal{M})$ vertex associated basis functions before the $\sharp\mathcal{E}(\mathcal{M})$ edge associated basis functions.

Another basis for V_N is the hierarchical basis given by

$$\mathcal{B}_h = \{\widehat{b}_N^j\}_{j=1}^N, \qquad \widehat{b}_N^j = \begin{cases} b_N^j, & j = 1, \ldots, \sharp\mathcal{V}(\mathcal{M}), \\ \widetilde{b}_N^j, & j = \sharp\mathcal{V}(\mathcal{M}), \ldots, \sharp\mathcal{V}(\mathcal{M}) + \sharp\mathcal{E}(\mathcal{M}), \end{cases}$$

where $\{b_N^j\}_{j=1}^{\sharp v}$ is the nodal basis for the space $\mathcal{S}_1^0(\mathcal{M})$ of linear Lagrangian finite element functions, see [NPDE, Sect. 3.2.3], and $\widetilde{b}_N^j$ are the edge associated nodal basis functions from $\mathcal{B}_q$.

(7.2a) Let $\vec{\mu}_q$ be the coefficient vector of a function $u_N \in V_N$ w.r.t. the nodal basis $\mathcal{B}_q$. Further let $\vec{\mu}_h$ be the coefficient vector of the *same* function u_N w.r.t. the hierarchical basis $\mathcal{B}_h$.

For the mesh $\mathcal{M}$ depicted in Figure 7.1 find the matrix $\boldsymbol{S} \in \mathbb{R}^{N\times N}$, $N = 4+5$ such that $\boldsymbol{\mu}_h = \boldsymbol{S}\boldsymbol{\mu}_q$. Use the numbering of the basis functions as indicated by edge and vertex numbers in Figure 7.1.

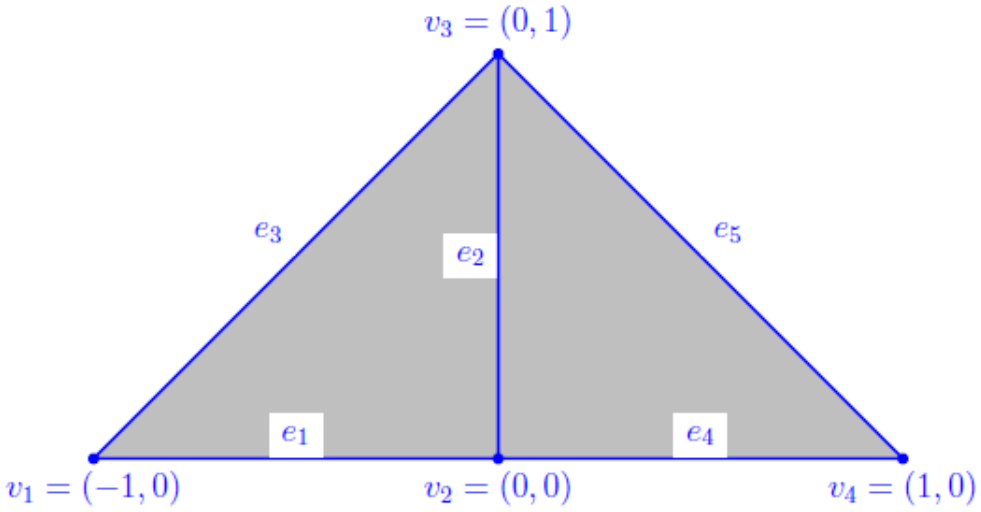

Figure 7.1: Mesh for subproblem (7.2a)

(7.2b) Now we consider a general polygon Ω equipped with a triangular mesh $\mathcal{M}$. Retain the notation $\boldsymbol{S}$ from subproblem (7.2a) for the $N \times N$ matrix that transforms a coefficient vector w.r.t. to $\mathcal{B}_q$ into a coefficient vector w.r.t. $\mathcal{B}_h$.

Let $\mathsf{a}(\cdot,\cdot)$ be a continuous bilinear form on $H^1(\Omega)$, $\boldsymbol{A}_q$ and $\boldsymbol{A}_h$ be the Galerkin matrices of a w.r.t. to $\mathcal{B}_q$ and $\mathcal{B}_h$, respectively. How can $\boldsymbol{A}_h$ be obtained from $\boldsymbol{A}_q$ using the matrix $\boldsymbol{S}$?

HINT: Refresh yourself on [NPDE, Lemma 3.1.9] and its proof.

(7.2c) Given the mesh $\mathcal{M}$ in the usual LehrFEM data structure `Mesh` complete with edge information, see [NPDE, Sect. 3.5.2], write an *efficient* MATLAB function

```
function mu_hier = basistrf(Mesh, mu_nod),
```

where `mu_hier` and `mu_nod` correspond to the coefficient vectors $\vec{\mu}_h$ and $\vec{\mu}_q$, of a $u_N \in V_N$ with respect to $\mathcal{B}_h$ and $\mathcal{B}_q$, respectively. The argument `Mesh` passes a LehrFEM mesh data structure complete with edge information.

HINT: The key to efficiency is to avoid creating the matrix $\boldsymbol{S}$!

Listing 7.3: Testcalls for Problem 7.2

```
fprintf('\n##basistrf');
clear Mesh
Mesh.Coordinates = [0 0; 1 0; 0 1; -1 0; 0 -1];
Mesh.Elements = [1 2 3; 1 3 4; 1 4 5; 1 5 2];
```

```
Mesh = add_Edges(Mesh);
nCoord = length(Mesh.Coordinates);
nEdge = length(Mesh.Edges);
muTilde = 1:(nCoord+nEdge);
muHat = basistrf(Mesh, muTilde);
muHat
```

Listing 7.4: Output for Testcalls for Problem 7.2

```
>> test

##basistrf
muHat =

  Columns 1 through 6

    1.0000    2.0000    3.0000    4.0000    5.0000    4.5000

  Columns 7 through 12

    5.0000    5.5000    6.0000    7.5000    7.5000    8.5000

  Column 13

    8.5000
```

Problem 7.3 Assembly in Tensor Product Setting (Core problem)

On tensor product meshes, as they underly finite difference methods (→ [NPDE, Sect. 4.1]), the generation of the global system matrix can often be accomplished by forming *Kronecker products* of matrices, see the documentation of MATLAB's `kron` for further information. In this problem we study this technique in a particular setting.

We consider the 2nd-order linear scalar elliptic boundary value problem

$$-\operatorname{div}(\alpha(\boldsymbol{x})\,\mathbf{grad}\,u) = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega, \tag{7.3.1}$$

on the unit square $\Omega := (0,1)^2$. Here $\alpha \in C^0(\overline{\Omega})$ is a uniformly positive function on Ω, which satisfies:

Assumption. The coefficient function α is given in *separable* form $\alpha(\boldsymbol{x}) = \alpha_1(x_1)\alpha_2(x_2)$, $\boldsymbol{x} = \binom{x_1}{x_2} \in \Omega$, where both α_1 and α_2 are positive continuous functions on $[0,1]$.

The finite element Galerkin discretization of (7.3.1) is carried out in the following way:

- Given points

$$\begin{aligned} &0 = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = 1, \\ &0 = y_0 < y_1 < y_2 < \cdots < y_{n-1} < y_m = 1, \end{aligned} \quad n, m \in \mathbb{N}, \tag{7.3.2}$$

  we rely on a tensor product mesh, see [NPDE, Eq. (3.3.2)],

$$\mathcal{M} := \{(x_{i-1}, x_i) \times (y_{i-1}, y_i) : \ 1 \le i \le n, \ 1 \le j \le m\}. \tag{7.3.3}$$

- As finite element space we use *bilinear* Lagrangian finite elements on $\mathcal{M}$, the space $\mathcal{S}^0_{1,0}(\mathcal{M})$, see [NPDE, Ex. 3.4.6]
- Assembly of the Galerkin matrix is based on the canonical global shape functions of $\mathcal{S}^0_{1,0}(\mathcal{M})$, see [NPDE, Eq. (3.4.7)], and the midpoint quadrature formula

$$\int_K g(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx |K| g(\boldsymbol{c})\ , \tag{7.3.4}$$

  where $\boldsymbol{c}$ is the barycenter of the square K.
- The approximate evaluation of the right-hand side linear form ($\rightarrow$ [NPDE, Eq. (2.8.4)]) associated with (7.3.1) is done by means of local vertex based quadrature,

$$\int_K f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \tfrac{1}{4}|K| \sum_{i=1}^{4} f(\boldsymbol{a}^i), \tag{7.3.5}$$

  where $\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3, \boldsymbol{a}^4$ are the vertices of the mesh rectangular mesh cell K.

(7.3a) What is the dimension N of $\mathcal{S}^0_{1,0}(\mathcal{M})$?

(7.3b) Write a MATLAB function

```
T = galmat1D(x,gamma)
```

that expects a sorted vector of grid points $0 = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = 1$, a function handle `gamma` to a continuous function $\gamma : [0,1] \mapsto \mathbb{R}$, and returns the Galerkin matrix arising from the discretization of the bilinear form

$$\mathsf{a}(u,v) = \int_0^1 \gamma(x)\,\frac{\mathrm{d}u}{\mathrm{d}x}(x)\,\frac{\mathrm{d}v}{\mathrm{d}x}(x)\,\mathrm{d}x, \quad u,\, v \in H^1_0((0,1)). \tag{7.3.6}$$

by means of piecewise linear continuous finite elements on the mesh $\mathcal{M} := \{(x_{i-1}, x_i) : \ i = 1,\ldots,n\}$], see [NPDE, Sect. 1.5.1.2]. The usual tent functions [NPDE, Eq. (1.5.52)] are to serve as global shape functions. They are ordered "from left to right". Use the composite midpoint rule [NPDE, Eq. 1.5.61] for the approximate evaluation of the integrals. Take care that sparse matrices are really stored in sparse format.

(7.3c) Write a MATLAB function

```
M = massmat1D(x,gamma)
```

that solves the task of subproblem (7.3b), but this time for the bilinear form

$$\mathsf{m}(u,v) = \int_0^1 \gamma(x)\,u(x)v(x)\,\mathrm{d}x, \quad u,\, v \in H^1_0((0,1)). \tag{7.3.7}$$

using the composite midpoint rule [NPDE, Eq. 1.5.61] for the approximate evaluation of the integral.

(7.3d) Write b_x^j for the 1D tent function associated with vertex x_j, $j = 1, \ldots, n-1$ of the mesh $\mathcal{M}_x := \{(x_{i-1}, x_i) : \ i = 1, \ldots, n\}]$, and b_y^j for the 1D tent function associated with vertex y_j, $j = 1, \ldots, m-1$ of the mesh $\mathcal{M}_y := \{(y_{i-1}, y_i) : \ i = 1, \ldots, m\}]$. Then the global shape function $b_N^{\boldsymbol{p}}$ of $\mathcal{S}_{1,0}^0(\mathcal{M})$ belonging to vertex $\boldsymbol{p} = \binom{x_i}{y_j}$, $1 \le i < n$, $1 \le j < m$, can be written as

$$b_N^{\boldsymbol{p}}(\boldsymbol{x}) = b_x^i(x_1) b_y^j(x_2), \quad \boldsymbol{x} \in \Omega. \tag{7.3.8}$$

Show that, *without numerical quadrature*, the entries of the Galerkin matrix $\mathbf{A} \in \mathbb{R}^N$ for the finite element discretization of (7.3.1) described above can be computed according to

$$\begin{aligned}(\mathbf{A})_{\boldsymbol{p},\boldsymbol{q}} = & \int_0^1 \alpha_1(x) \frac{\mathrm{d}b_x^i}{\mathrm{d}x}(x) \frac{\mathrm{d}b_x^l}{\mathrm{d}x}(x)\,\mathrm{d}x \cdot \int_0^1 \alpha_2(y)\, b_y^j(y) b_y^k(y)\,\mathrm{d}y \\ & + \int_0^1 \alpha_2(y) \frac{\mathrm{d}b_y^j}{\mathrm{d}y}(y) \frac{\mathrm{d}b_y^k}{\mathrm{d}y}(y)\,\mathrm{d}y \cdot \int_0^1 \alpha_1(x)\, b_x^i(x) b_x^l(x)\,\mathrm{d}x,\end{aligned} \tag{7.3.9}$$

where $\boldsymbol{q} = \binom{x_l}{y_k}$, $1 \le l < n$, $1 \le k < m$. Here, $(\mathbf{A})_{\boldsymbol{p},\boldsymbol{q}}$ refers to the matrix entry corresponding to the interior mesh vertices $\boldsymbol{p}$ and $\boldsymbol{q}$.

(7.3e) How do you have to rewrite (7.3.9), if the one-point local quadrature formula ((7.3.4)) is used for assembly of the Galerkin matrix.

(7.3f) Familiarize yourself with the *Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ of two matrices $\mathbf{A}$, $\mathbf{B}$, available in MATLAB through the `kron` built-in function. Sketch the structure of the Kronecker product of two large tridiagonal squares matrices. What is the maximal number of non-zero entries of their Kronecker product.

(7.3g) Assume a lexikographic (row-wise) ordering of the vertices (x_1-direction first) of the mesh as introduced in [NPDE, Sect. 4.1], see [NPDE, Fig. 130]. Show that

$$\mathbf{A} = \mathbf{T}_y \otimes \mathbf{M}_x + \mathbf{M}_y \otimes \mathbf{T}_x. \tag{7.3.10}$$

where $\mathbf{T}_x$, $\mathbf{T}_y$ are the Galerkin matrices computed in subproblem (7.3b) for the meshes with gridpoints x_j and y_j, and the coefficient functions $\gamma := \alpha_1$ and $\gamma := \alpha_2$, respectively, see (7.3.2). Further, $\mathbf{M}_x$, $\mathbf{M}_y$ stand for the corresponding mass matrices computed in subproblem (7.3c), again for suitable coefficient functions $\gamma := \alpha_1$ or $\gamma := \alpha_2$.

(7.3h) Write a MATLAB function

```
[A,phi] = tpass(f,alpha1,alpha2,x,y)
```

that computes the Galerkin matrix $\mathbf{A}$ and right-hand side vector $\vec{\varphi}$ for the finite element discretization of (7.3.1) introduced above. The functions handles `alpha1` and `alpha2` of type `@(x)` provide the coefficient functions α_1 and α_2 defined on $[0, 1]$. Besides, the function handle `f` of type `f = @(x1,x2)` passes the source function f.

HINT: Use the MATLAB functions implemented in sub-problems (7.3b) and (7.3c). Verify that, for equidistant meshes in x_1 and x_2-direction with meshwidth h, you obtain the Poisson matrix [NPDE, Eq. (4.1.3)].

Listing 7.5: Testcalls for Problem 7.3

```
fprintf('\n##galmat1D');
x=[0, 0.3, 0.4, 0.7, 0.95, 1]';
y=[0, 0.2, 0.4, 0.6, 0.9, 1]';
alpha1 = @(x) x;
alpha2 = @(x) x.^2;
A = galmat1D(x,alpha1)
fprintf('\n##massmat1D');
M = massmat1D(x,alpha1)
fprintf('\n##tpass');
f = @(x,y) sin(pi*x).*sin(pi*y);
[A,phi]=tpass(f,alpha1,alpha2,x,y)
```

Listing 7.6: Output for Testcalls for Problem 7.3

```
>> test_call

##galmat1D
A =

   (1,1)        4.0000
   (2,1)       -3.5000
   (1,2)       -3.5000
   (2,2)        5.3333
   (3,2)       -1.8333
   (2,3)       -1.8333
   (3,3)        5.1333
   (4,3)       -3.3000
   (3,4)       -3.3000
   (4,4)       22.8000

##massmat1D
M =

   (1,1)        0.0200
   (2,1)        0.0088
   (1,2)        0.0088
   (2,2)        0.0500
   (3,2)        0.0412
   (2,3)        0.0412
   (3,3)        0.0928
   (4,3)        0.0516
   (3,4)        0.0516
   (4,4)        0.0638

##tpass
A =

   (1,1)        0.0300
   (2,1)       -0.0131
```

```
   (5,1)       0.0090
   (6,1)      -0.0197
   (1,2)      -0.0131
   (2,2)       0.0517
   (3,2)       0.0115
   (5,2)      -0.0197
   (6,2)       0.0015
   (7,2)      -0.0268
   (2,3)       0.0115
   (3,3)       0.0721
   (4,3)       0.0093
   (6,3)      -0.0268
   (7,3)      -0.0187
   (8,3)      -0.0381
   (3,4)       0.0093
   (4,4)       0.1459
   (7,4)      -0.0381
   (8,4)       0.0739
   (1,5)       0.0090
   (2,5)      -0.0197
   (5,5)       0.1020
   (6,5)      -0.0446
   (9,5)       0.0250
  (10,5)      -0.0547
   (1,6)      -0.0197
   (2,6)       0.0015
   (3,6)      -0.0268
   (5,6)      -0.0446
   (6,6)       0.1757
   (7,6)       0.0390
   (9,6)      -0.0547
  (10,6)       0.0042
  (11,6)      -0.0745
   (2,7)      -0.0268
   (3,7)      -0.0187
   (4,7)      -0.0381
   (6,7)       0.0390
   (7,7)       0.2450
   (8,7)       0.0316
  (10,7)      -0.0745
  (11,7)      -0.0518
  (12,7)      -0.1057
   (3,8)      -0.0381
   (4,8)       0.0739
   (7,8)       0.0316
   (8,8)       0.4960
  (11,8)      -0.1057
  (12,8)       0.2053
   (5,9)       0.0250
   (6,9)      -0.0547
```

```
    (9,9)        0.2812
   (10,9)       -0.1641
   (13,9)        0.1312
   (14,9)       -0.1641
    (5,10)      -0.0547
    (6,10)       0.0042
    (7,10)      -0.0745
    (9,10)      -0.1641
   (10,10)       0.4479
   (11,10)       0.0286
   (13,10)      -0.1641
   (14,10)       0.1312
   (15,10)      -0.1547
    (6,11)      -0.0745
    (7,11)      -0.0518
    (8,11)      -0.1057
   (10,11)       0.0286
   (11,11)       0.5708
   (12,11)      -0.0193
   (14,11)      -0.1547
   (15,11)       0.0425
   (16,11)      -0.2359
    (7,12)      -0.1057
    (8,12)       0.2053
   (11,12)      -0.0193
   (12,12)       1.4461
   (15,12)      -0.2359
   (16,12)       0.8423
    (9,13)       0.1312
   (10,13)      -0.1641
   (13,13)       0.4770
   (14,13)      -0.1312
    (9,14)      -0.1641
   (10,14)       0.1312
   (11,14)      -0.1547
   (13,14)      -0.1312
   (14,14)       0.8903
   (15,14)       0.3309
   (10,15)      -0.1547
   (11,15)       0.0425
   (12,15)      -0.2359
   (14,15)       0.3309
   (15,15)       1.3440
   (16,15)       0.3484
   (11,16)      -0.2359
   (12,16)       0.8423
   (15,16)       0.3484
   (16,16)       2.1712

phi =
```

```
136
137     0.0190
138     0.0224
139     0.0262
140     0.0028
141     0.0308
142     0.0362
143     0.0423
144     0.0045
145     0.0385
146     0.0452
147     0.0529
148     0.0056
149     0.0100
150     0.0118
151     0.0138
152     0.0015
```

Problem 7.4 Mehrstellenverfahren for Poisson Equation

This problem deals with a special finite difference method (→ [NPDE, Sect. 4.1]) for the Poisson equation (→ [NPDE, Eq. (2.4.13)]). In line with the course's philosophy, we examine it from the perspective of a finite element method and derive the linear system of equation by means of local assembly, see [NPDE, Sect. 3.2.5] and [NPDE, Sect. 3.5.3.1].

We consider the scalar second-order boundary value problem

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega := (0,1)^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{7.4.1}$$

for a bounded and continuous function $f \in \mathcal{C}^0(\overline{\Omega})$.

The so-called *Collatz Mehrstellenverfahren* can be viewed as a modified finite element Galerkin discretization using bilinear Lagrangian finite elements (→ [NPDE, Ex. 3.4.6]) on a uniform quadrilateral tensor product mesh of Ω with meshwidth $h := (M+1)^{-1}$, $M \in \mathbb{N}$, see Figure 7.3. We take for granted that standard nodal basis functions are used.

Then the Mehrstellenverfahren is obtained when using the following 4×4 element matrix on each cell K of the mesh

$$\mathbf{A}_K := \frac{1}{6} \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 5 & -2 & -1 \\ -1 & -2 & 5 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix}, \tag{7.4.2}$$

and the following element vector

$$\vec{\varphi}_K = \frac{h^2}{12} \begin{pmatrix} 2f_1 + f_2 + f_4 \\ 2f_2 + f_3 + f_1 \\ 2f_3 + f_4 + f_2 \\ 2f_4 + f_1 + f_3 \end{pmatrix}, \quad f_i := f(\boldsymbol{a}_i), \quad i = 1,2,3,4. \tag{7.4.3}$$

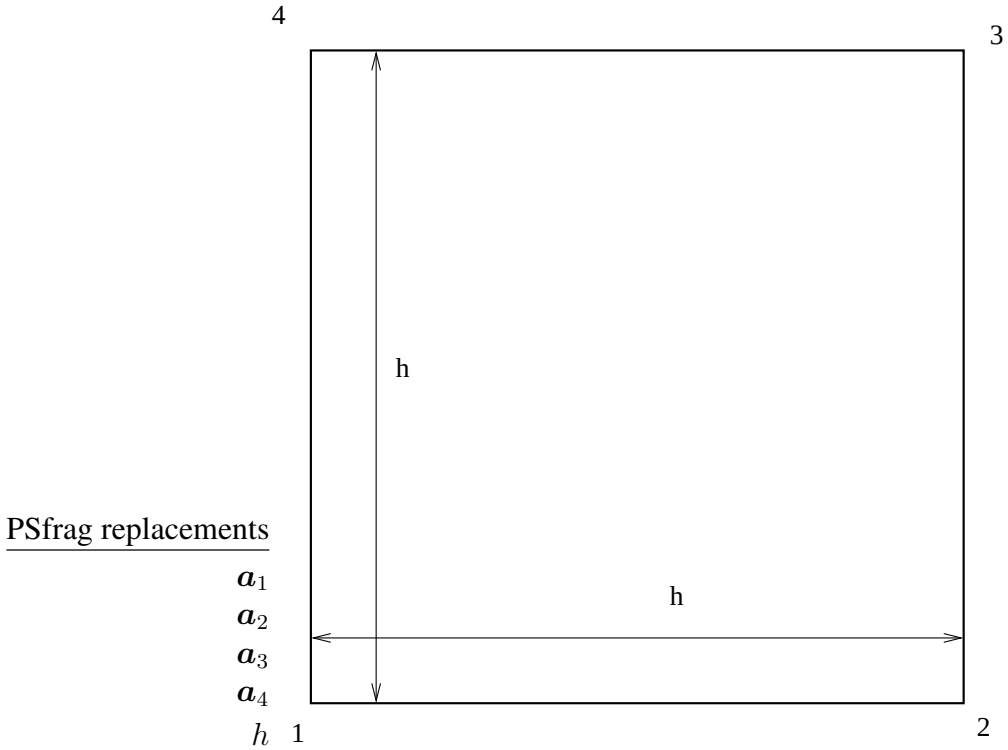

Figure 7.2: Local numbering of vertices and associated local shape functions for a square cell K.

Here, we have denoted the vertices of the square cell K by $\boldsymbol{a}_i$, $i = 1, 2, 3, 4$, and have used the counter-clockwise local numbering indicated in Figure 7.2.

Throughout this problem we assume a lexikographic numbering of the interior vertices of the mesh and of the associated basis functions, see Figure 7.3.

Eventually, we arrive at a linear system of equations $\mathbf{A}\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}$ encoded by (7.4.2) and (7.4.3).

(7.4a) What is the size of the matrix $\mathbf{A}$ and of the right hand side vector $\vec{\boldsymbol{\varphi}}$?

(7.4b) What is the meaning of the entries of the solution vector $\vec{\boldsymbol{\mu}}$?

(7.4c) Write a MATLAB function

```
function A = compMehrstellenA(M)
```

that assembles the *sparse* Galerkin matrix of the Mehrstellenverfahren.

HINT: The matrix $\mathbf{A}$ is a block tridiagonal matrix with tridiagonal blocks.

HINT: In MATLAB `A = gallery('tridiag',n,c,d,e)`, where `c`, `d`, and `e` are all scalars, yields the Toeplitz tridiagonal matrix of order `n` with subdiagonal elements `c`, diagonal elements `d`, and superdiagonal elements `e`.

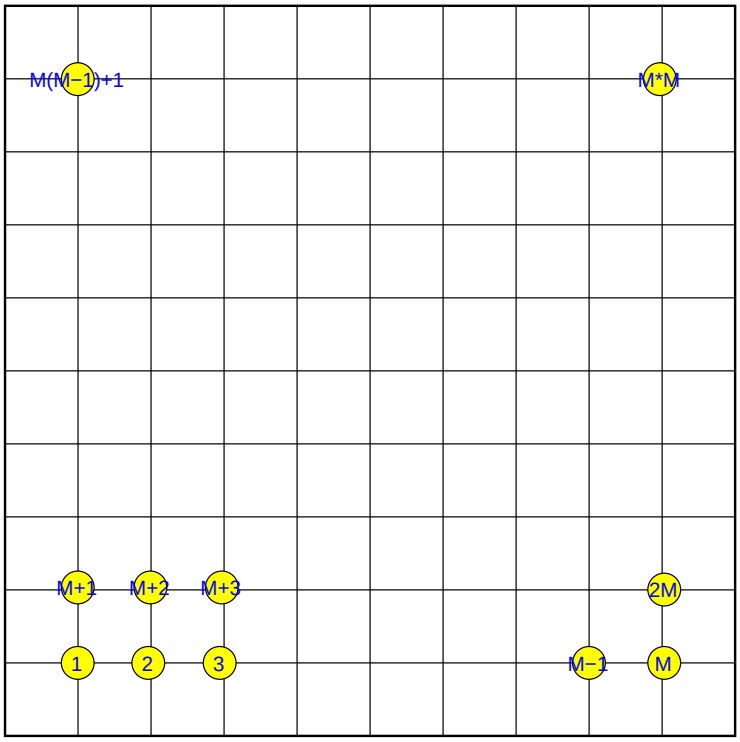

Figure 7.3: Lexikographic numbering of vertices of the equidistant tensor product mesh

HINT: Here, the assembly of the system matrix follows the policy discussed in Problem 7.3. The MATLAB command `kron` that computes the Kronecker product of two matrices

$$\mathbf{R} \otimes \mathbf{Q} = \begin{pmatrix} r_{1,1}\mathbf{Q} & r_{1,2}\mathbf{Q} & \cdots & r_{1,M}\mathbf{Q} \\ r_{2,1}\mathbf{Q} & r_{2,2}\mathbf{Q} & \cdots & r_{2,M}\mathbf{Q} \\ \vdots & & \ddots & \vdots \\ r_{M,1}\mathbf{Q} & \cdots & \cdots & r_{M,M}\mathbf{Q} \end{pmatrix} \in \mathbb{R}^{MN \times MN}, \quad \mathbf{R} \in \mathbb{R}^{M \times M}, \ \mathbf{Q} \in \mathbb{R}^{N \times N},$$

comes handy.

(7.4d) Devise a MATLAB function

```
function f = compMehrstellenf(f,M)
```

that returns the right hand side vector $\vec{\varphi}$ for the Mehrstellenverfahren. The argument `f` passes a handle of type `@(x)` (with a 2-vector `x`) to the function f, while $M+1$ gives the number of grid cells in one direction.

(7.4e) Write a MATLAB function

```
function u = solveMehrstellen(f,M)
```

that computes the coefficient vector $\vec{\mu}$ for the Mehrstellen discretization of (7.4.1), when supplied with a handle `f` to the source function f and the discretization parameter `M` (see subproblem (7.4d)).

(7.4f) We consider $f(x,y) = \sin(\pi x)\sin(\pi y)$, which yields the exact solution $u(x,y) = (2\pi^2)^{-1} f(x,y)$, $(x,y) \in \Omega$.

Write a MATLAB function

```
function err = compgriderr(M)
```

that computes the following "discrete norm" of the discretization error

$$\|u - u_M\|_{\mathcal{M},\infty} := \max\{|u(\boldsymbol{x}) - u_M(\boldsymbol{x})| : \boldsymbol{x} = (ih, jh),\ 1 \le i,j \le M,\ h := (M+1)^{-1}\}. \tag{7.4.4}$$

where u_M is the "Mehrstellen solution" for discretization parameter `M` defined at the vertices of the mesh.

HINT: A reference implementation of `solveMehrstellen.m` is supplied in the file `solveMehrstellen_ref.p`.

(7.4g) Write a MATLAB script `convergence.m` that estimates the (algebraic) order of convergence of the Mehrstellenverfahren with respect to the error norm (7.4.4) and for the specific source function $f(x,y) = \sin(\pi x)\sin(\pi y)$. To achieve this, evaluate the error for $M = 5, 10, 20, 40, 80, 160$ using the function `compgriderr` from subproblem (7.4f).

HINT: A reference implementation of `compgriderr.m` can be accessed through the file `compgriderr_ref.p`.

Listing 7.7: Testcalls for Problem 7.4

```
M=3;
fprintf('\n#compMehrstellenA');
A=compMehrstellenA(M)
fprintf('\n#compMehrstellenf');
func = @(x) sin(pi*x(1))*sin(pi*x(2));
f=compMehrstellenf(func,M)
fprintf('\n#solveMehrstellen');
u=solveMehrstellen(func,M)
fprintf('\n#compgriderr');
err=compgriderr(M)
```

Listing 7.8: Output for Testcalls for Problem 7.4

```
>> test_call

#compMehrstellenA
A =

   (1,1)        3.3333
   (2,1)       -0.6667
   (4,1)       -0.6667
   (5,1)       -0.1667
   (1,2)       -0.6667
   (2,2)        3.3333
```

```
    (3,2)       -0.6667
    (4,2)       -0.1667
    (5,2)       -0.6667
    (6,2)       -0.1667
    (2,3)       -0.6667
    (3,3)        3.3333
    (5,3)       -0.1667
    (6,3)       -0.6667
    (1,4)       -0.6667
    (2,4)       -0.1667
    (4,4)        3.3333
    (5,4)       -0.6667
    (7,4)       -0.6667
    (8,4)       -0.1667
    (1,5)       -0.1667
    (2,5)       -0.6667
    (3,5)       -0.1667
    (4,5)       -0.6667
    (5,5)        3.3333
    (6,5)       -0.6667
    (7,5)       -0.1667
    (8,5)       -0.6667
    (9,5)       -0.1667
    (2,6)       -0.1667
    (3,6)       -0.6667
    (5,6)       -0.6667
    (6,6)        3.3333
    (8,6)       -0.1667
    (9,6)       -0.6667
    (4,7)       -0.6667
    (5,7)       -0.1667
    (7,7)        3.3333
    (8,7)       -0.6667
    (4,8)       -0.1667
    (5,8)       -0.6667
    (6,8)       -0.1667
    (7,8)       -0.6667
    (8,8)        3.3333
    (9,8)       -0.6667
    (5,9)       -0.1667
    (6,9)       -0.6667
    (8,9)       -0.6667
    (9,9)        3.3333

#compMehrstellenf
f =

    0.0282
    0.0399
    0.0282
```

```
    0.0399
    0.0564
    0.0399
    0.0282
    0.0399
    0.0282

#solveMehrstellen
u =

    0.0253
    0.0358
    0.0253
    0.0358
    0.0506
    0.0358
    0.0253
    0.0358
    0.0253

#compgriderr
err =

   5.1319e-05
```

Published on April 15.
To be submitted on April 22.

References

[NPDE] Lecture Slides for the course “Numerical Methods for Partial Differential Equations”, SVN revision # 54041.

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

[LehrFEM] LehrFEM manual.

Last modified on April 16, 2013