

Homework Problem Sheet 4

Introduction. The first three problems of this assignment concern elliptic PDEs.

The first two problems are fully theoretical and are aimed at making you more familiar with some formulas and properties that you have already used or seen at the lecture. The third problem tackles the discretization of the Poisson equation with a new kind of boundary conditions, the so-called Robin boundary conditions.

Finally, the fourth problem faces the discretization of a one-dimensional parabolic problem.

Problem 4.1 Green's Formula (Core problem)

To derive the variational formulation from a boundary value problem for a partial differential equation we needed multi-dimensional integration by parts as expressed through Green's first formula. In this problem we study the derivation of Green's formula, thus practicing elementary vector analysis and the application of Gauss' theorem.

(4.1a) Prove Green's formula for $\Omega \subset \mathbb{R}^2$, given by

$$-\int_{\Omega} \nabla \cdot \mathbf{j} v \, d\mathbf{x} = -\int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v \, dS + \int_{\Omega} \mathbf{j} \cdot \nabla v \, d\mathbf{x}, \quad (4.1.1)$$

where $\mathbf{j} \in C_{\text{pw}}^1(\Omega)^2$ and $v \in C_{\text{pw}}^1(\Omega)$.

HINT: Use Gauss' theorem

$$\int_{\Omega} \nabla \cdot \mathbf{F} \, d\mathbf{x} = \int_{\partial\Omega} \mathbf{F} \cdot \mathbf{n} \, dS,$$

where $\mathbf{F} \in C_{\text{pw}}^1(\Omega)^2$.

Solution: The vectorized product rule takes the form

$$\nabla \cdot (\mathbf{j}v) = \nabla \cdot \mathbf{j} v + \nabla v \cdot \mathbf{j},$$

which, when integrated, is

$$\int_{\Omega} \nabla \cdot (\mathbf{j}v) \, d\mathbf{x} = \int_{\Omega} \nabla \cdot \mathbf{j} v \, d\mathbf{x} + \int_{\Omega} \nabla v \cdot \mathbf{j} \, d\mathbf{x}.$$

The first integral fits Gauss' theorem, so we get

$$\int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v \, dS = \int_{\Omega} \nabla \cdot \mathbf{j} v \, d\mathbf{x} + \int_{\Omega} \nabla v \cdot \mathbf{j} \, d\mathbf{x}$$

which is what we wanted to show.

(4.1b) Use equation (4.1.1) to prove the formula

$$-\int_{\Omega} \Delta u(\mathbf{x})v(\mathbf{x}) \, d\mathbf{x} = \int_{\Omega} \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) \, d\mathbf{x} - \int_{\partial\Omega} \frac{\partial u}{\partial \mathbf{n}}(\mathbf{x})v(\mathbf{x}) \, dS(\mathbf{x})$$

for $\Omega \subset \mathbb{R}^2$ and $u, v \in H^1(\Omega)$.

Solution: Considering that $\Delta u = \nabla \cdot \nabla u$, this formula is an immediate consequence of (4.1.1) when taking $\mathbf{j} = \nabla u$.

Problem 4.2 Galerkin Orthogonality and energy norm (Core problem)

Let $\mathbf{a}(\cdot, \cdot)$ be a continuous, coercive and positive (i.e. with $\mathbf{a}(v, v) = |\mathbf{a}(v, v)| \geq 0$ for all $v \in V$) but *not* necessarily symmetric bilinear form on a function space V and let $\ell(\cdot)$ be a continuous linear functional on \mathcal{V} . Let u be the exact solution to the variational problem

$$u \in V, \quad \mathbf{a}(u, v) = \ell(v) \quad \forall v \in V, \quad (4.2.1)$$

and let u_N be the FE solution in a finite dimensional subspace $V_N \subset V$, given by

$$u_N \in V_N, \quad \mathbf{a}(u_N, v_N) = \ell(v_N) \quad \forall v_N \in V_N.$$

(4.2a) Show that the Galerkin orthogonality

$$\mathbf{a}(u - u_N, v_N) = 0 \quad (4.2.2)$$

holds.

Solution:

$$\mathbf{a}(u - u_N, v_N) = \mathbf{a}(u, v_N) - \mathbf{a}(u_N, v_N) = \ell(v_N) - \ell(v_N) = 0.$$

(4.2b) If $\mathbf{a}(\cdot, \cdot)$ is symmetric, then $\mathbf{a}(v_1, v_2)$ for all $v_1, v_2 \in V$ defines a scalar product on V . Show that this is a valid definition of scalar product

Solution: We have to verify that the properties in the definition of scalar product are fulfilled. The properties are bilinearity and symmetry and positive definiteness, and they follow directly from the assumptions on $\mathbf{a}(\cdot, \cdot)$.

The scalar product $\mathbf{a}(v_1, v_2)$ induces a norm on V defined by

$$\|v\|_a = \sqrt{\mathbf{a}(v, v)} \quad \forall v \in V,$$

called the *energy norm*.

For the rest of this problem we assume the bilinear form to be symmetric, so that the energy norm is well defined.

(4.2c) Show that the norms $\|\cdot\|_a$ and $\|\cdot\|_V$ are equivalent.

HINT: Two norms $\|\cdot\|_1$ and $\|\cdot\|_2$ on a space V are equivalent if there exist two constants $C_1, C_2 > 0$ such that

$$C_1\|v\|_2 \leq \|v\|_1 \leq C_2\|v\|_2 \quad \forall v \in V.$$

HINT: Use the properties of the bilinear form.

Solution: From the coercivity and the continuity of $a(\cdot, \cdot)$ we get:

$$\alpha\|v\|_V^2 \leq \|v\|_a^2 \leq \nu\|v\|_V^2 \quad \forall v \in V,$$

where α is the coercivity constant and ν the continuity constant. Taking the square root we get the result with $C_1 = \sqrt{\alpha}$ and $C_2 = \sqrt{\nu}$.

The Galerkin orthogonality property (4.2.2) together to the property $V_N \subset V$ (characteristic of Galerkin discretization schemes) show that the Finite Element solution u_N is the orthogonal projection of u on V_N when using $a(\cdot, \cdot)$ as scalar product. This justifies the following statement.

(4.2d) Show that u_N is the best approximation in V_N of u with respect to the energy norm $\|\cdot\|_a$ induced by $a(\cdot, \cdot)$, i.e. $\inf_{v_N \in V_N} \|u - v_N\|_a = \|u - u_N\|_a$.

Solution:

$$\begin{aligned} a(u - u_N, u - u_N) &= a(u - v_N, u - u_N) + \underbrace{a(v_N - u_N, u - u_N)}_{=0 \text{ for Galerkin orthog + symmetry}} \\ &= a(u - v_N, u - v_N) + a(u - v_N, v_N - u_N) = \\ &= a(u - v_N, u - v_N) - a(v_N - u_N, v_N - u_N) + \underbrace{a(u - u_N, v_N - u_N)}_{=0 \text{ for Galerkin orthog}}. \end{aligned}$$

Thus

$$\|u - v_N\|_a^2 = \|u - u_N\|_a^2 + \|v_N - u_N\|_a^2;$$

since the two addends on the right-handside are both positive, the infimum is achieved when $\|v_N - u_N\|_a = 0$ and thus $v_N = u_N$.

We now want to compare Finite Element solutions of the same problem when using different and nested Finite Element spaces.

(4.2e) Show that $a(u - u_N, u - u_N) = a(u, u) - a(u_N, u_N)$.

Solution:

$$\begin{aligned} a(u - u_N, u - u_N) &= a(u, u - u_N) - \underbrace{a(u_N, u - u_N)}_{=0 \text{ (using symmetry!)}} \\ &= a(u, u) - a(u, u_N) \\ &= a(u, u) - \underbrace{a(u - u_N, u_N)}_{=0} - a(u_N, u_N). \end{aligned}$$

(4.2f) Let $V_N \subset V_{N'}$. Show that $a(u - u_N, u - u_N) \geq a(u - u_{N'}, u - u_{N'})$, where $u_{N'}$ is the Finite Element solution given by

$$u_{N'} \in V_{N'}, \quad a(u_{N'}, v_{N'}) = f(v_{N'}) \quad \forall v_{N'} \in V_{N'}.$$

Solution: Since $V_N \subset V_{N'}$, we have, using (4.2e),

$$0 \leq a(u_{N'} - u_N, u_{N'} - u_N) = a(u_{N'}, u_{N'}) - a(u_N, u_N),$$

and hence $a(u_N, u_N) \leq a(u_{N'}, u_{N'})$.

Using (4.2e) two times we obtain

$$\begin{aligned} a(u - u_N, u - u_N) &= a(u, u) - a(u_N, u_N) \\ &\geq a(u, u) - a(u_{N'}, u_{N'}) \geq a(u - u_{N'}, u - u_{N'}). \end{aligned}$$

Alternatively, the claim follows just considering what proved in task (4.2d): since $V_N \subset V_{N'}$, the infimum over V_N can only be greater or equal than the infimum over $V_{N'}$.

(4.2g) Let u_L and u_Q be the Finite Element solutions to (4.2.1) when using respectively linear and quadratic Finite Elements on the same mesh.

Show that

$$\|u - u_Q\|_a \leq \|u - u_L\|_a.$$

Solution: This is an immediate consequence of (4.2f) since, when using the same mesh, the space of piecewise linear polynomials is a subspace of piecewise quadratic polynomials.

Problem 4.3 Heat Conduction with Reaction Term and Convective Cooling

This exercise involves the Galerkin discretization of a particular 2nd-order linear elliptic boundary value problems by means of linear Lagrangian finite elements. In this problem you will need most of the routines that you should have implemented for the previous assignment.

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

$$-\Delta u = f \quad \text{in } \Omega, \tag{4.3.1}$$

$$\nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) + \gamma(\mathbf{x})u(\mathbf{x}) = 0 \quad \text{on } \partial\Omega, \tag{4.3.2}$$

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*.

We solve (4.3.1)–(4.3.2) approximately by means of Galerkin discretization based on piecewise linear Lagrangian finite elements on triangular meshes of Ω .

(4.3a) Derive the variational formulation of the boundary value problem (4.3.1)–(4.3.2). 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*).

HINT: If $u, v \in H^1(\Omega)$, then $\int_{\partial\Omega} uv \, dS$ is well defined.

Solution: Like before, we multiply with a test function v (that is *not* generally zero on the boundary), and integrate:

$$\begin{aligned} - \int_{\Omega} v(\mathbf{x}) \Delta u(\mathbf{x}) \, d\mathbf{x} &= - \int_{\partial\Omega} v(\mathbf{x}) \nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) \, d\mathbf{x} + \int_{\Omega} \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) \, d\mathbf{x} \\ &= \int_{\partial\Omega} \gamma(\mathbf{x}) u(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x} + \int_{\Omega} \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) \, d\mathbf{x}. \end{aligned}$$

So the variational formulation is to find $u \in H^1(\Omega)$ so that $a(u, v) = \ell(v)$ for all $v \in H^1(\Omega)$, where

$$a(u, v) = \int_{\partial\Omega} \gamma(\mathbf{x})u(\mathbf{x})v(\mathbf{x}) \, d\mathbf{x} + \int_{\Omega} \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) \, d\mathbf{x},$$

and

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

(4.3b) Argue why the variational problem obtained in [subproblem \(4.3a\)](#) has a unique solution (proof of existence is not required).

Solution: The bilinear form is coercive. Indeed, assume $a(u, u) = 0$ for some u , so that

$$\int_{\partial\Omega} \gamma(\mathbf{x})|u(\mathbf{x})|^2 \, d\mathbf{x} + \int_{\Omega} \|\nabla u(\mathbf{x})\|^2 \, d\mathbf{x} = 0.$$

Since γ is uniformly positive, this means that both integrals must be zero. First, $\|\nabla u(\mathbf{x})\| = 0$ everywhere, so u is constant. But we also know that $u = 0$ on the boundary (from the first integral), so u must be zero everywhere.

(4.3c) Assume that $\gamma \equiv \text{const}$. Compute the element (stiffness) matrix for the bilinear form from [subproblem \(4.3a\)](#) and linear finite elements on a triangle K analytically in terms of the area $|K|$ and edge lengths $|e_i|$. You can denote by \mathbf{A}_K^Δ the part of the matrix associated to the gradients, without computing it explicitly.

HINT: You may take knowledge of the edge lengths and angles of the triangle for granted. Special cases will occur if one or more edges are part of the boundary $\partial\Omega$!

Solution: The Laplacian part is already given. We concern ourselves here with the boundary term.

We compute the contributions of the boundary integral to the local Galerkin matrix. Suppose the edge between vertices i and j is part of $\partial\Omega$. Then this edge will give a contribution to $\mathbf{A}_{i,i}$, $\mathbf{A}_{j,j}$, $\mathbf{A}_{i,j}$ and $\mathbf{A}_{j,i}$, which we can compute by transformation to $[0, 1]$. On this unit interval, the basis functions are of course x and $1 - x$:

$$h \int_0^1 \gamma x^2 \, dx = \frac{h\gamma}{3}, \quad h \int_0^1 \gamma x(1 - x) \, dx = \frac{h\gamma}{6},$$

where h is the length of the edge in question.

Thus, the local Galerkin matrix will look like this:

$$\mathbf{A} = \mathbf{A}_K^\Delta + \delta_{(1,2)} \frac{|e_{(1,2)}|\gamma}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} + \delta_{(2,3)} \frac{|e_{(2,3)}|\gamma}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} + \delta_{(1,3)} \frac{|e_{(1,3)}|\gamma}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix},$$

where \mathbf{A}_K^Δ is the Laplacian local Galerkin matrix, $|e_{(i,j)}|$ is the length of edge (i, j) , and $\delta_{(i,j)} = 1$ if the edge (i, j) is part of $\partial\Omega$, and $\delta_{(i,j)} = 0$ when not.

(4.3d) Now solve [subproblem \(4.3c\)](#) for general continuous $\gamma \in \mathcal{C}^0(\partial\Omega)$, but using the one-dimensional trapezoidal rule 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.

Solution: According to the trapezoidal rule, the integrals from (4.3c) read

$$h \int_0^1 \gamma(x) x^2 dx \approx \frac{h\gamma(1)}{2}, \quad h \int_0^1 \gamma(x)(1-x)^2 dx \approx \frac{h\gamma(0)}{2}, \quad h \int_0^1 \gamma x(1-x) dx \approx 0.$$

So we get

$$\mathbf{A} = \mathbf{A}_L + \delta_{(1,2)} \frac{|e_{(1,2)}|}{2} \begin{pmatrix} \gamma_1 & \\ & \gamma_2 \end{pmatrix} + \delta_{(2,3)} \frac{|e_{(2,3)}|}{2} \begin{pmatrix} & \gamma_2 \\ & \gamma_3 \end{pmatrix} + \delta_{(1,3)} \frac{|e_{(1,3)}|}{2} \begin{pmatrix} \gamma_1 & \\ & \gamma_3 \end{pmatrix},$$

where we have used the notation $\gamma_i = \gamma(\mathbf{a}^i)$.

(4.3e) Implement a function

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

that returns the element matrix for the bilinear form from subproblem (4.3a) 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 (4.3d).

Here `Vertices` is a 3×2 -matrix passing the coordinates of the triangle's vertices in its rows 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 to the function γ .

HINT: You may use the `STIMA_Heat_LFE` function that you already implemented.

Solution: See Listing 4.1.

Listing 4.1: Implementation for `STIMA_LaplRobin_LFE`

```
1 function Aloc = STIMA_LaplRobin_LFE(Vertices, BdEdges,
2   EHandle)
3
4   % Preallocate memory
5   Aloc = STIMA_Heat_LFE(Vertices, P706(), @(x) 1);
6
7   % Boundary
8   for i = BdEdges'
9       verts = setdiff(1:3, i);
10      h = norm(Vertices(verts(1), :) - Vertices(verts(2), :));
11      gamma1 = EHandle(Vertices(verts(1), :));
12      gamma2 = EHandle(Vertices(verts(2), :));
13      Aloc(verts(1), verts(1)) = ...
14          Aloc(verts(1), verts(1)) + h/2 * gamma1;
15      Aloc(verts(2), verts(2)) = ...
16          Aloc(verts(2), verts(2)) + h/2 * gamma2;
17   end
```

17

18

end

(4.3f) Create a copy of `assemMat_LFE.m` 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_LapRobin_LFE` implemented in [subproblem \(4.3e\)](#). The syntax of your new function should be

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

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

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

The field `Mesh.Vert2Edge` may come in hand.

Solution: See [Listing 4.2](#).

Listing 4.2: Implementation for `assemMatRobin_LFE`.

```
1 function A = assemMatLapRobin_LFE(Mesh, gammaHandle)
2
3     % Initialize constants
4     nElements = size(Mesh.Elements,1);
5
6     % Preallocate memory
7     I = zeros(9*nElements,1);
8     J = zeros(9*nElements,1);
9     A = zeros(9*nElements,1);
10
11    % Assemble element contributions
12    loc = 1:9;
13    for i = 1:nElements
14
15        % Extract vertices of current element
16        idx = Mesh.Elements(i,:);
17        Vertices = Mesh.Coordinates(idx,:);
18
19        % Extract boundary data
20        % BdEdges should contain 1 if the edge opposite of
21        % vertex 1 is a boundary edge. If BdEdges is empty
22        % then the element is in the interior.
23        BdEdges = [];
24        if Mesh.BdFlags(Mesh.Vert2Edge(idx(2),idx(3))) < 0
25            BdEdges = [BdEdges; 1];
26        end
27        if Mesh.BdFlags(Mesh.Vert2Edge(idx(1),idx(3))) < 0
```

```

28         BdEdges = [BdEdges; 2];
29     end
30     if Mesh.BdFlags(Mesh.Vert2Edge(idx(1),idx(2))) < 0
31         BdEdges = [BdEdges; 3];
32     end
33
34     % Compute element contributions
35     Aloc = STIMA_LaplRobin_LFE(Vertices,BdEdges,
36         gammaHandle);
37
38     % Add contributions to stiffness matrix
39     Iloc = idx(ones(3,1),:)' ;
40     I(loc) = Iloc(:);
41
42     Jloc = idx(ones(1,3),:);
43     J(loc) = Jloc(:);
44
45     A(loc) = Aloc(:);
46     loc = loc+9;
47 end
48
49 A = sparse(I,J,A);
50
51 return

```

(4.3g) Make a copy of `main_LFE.m` called `main_LFE_robin.m`. Modify this script so that it computes a linear finite element solution of (4.3.1)–(4.3.2) making use of the new function `assemMatRobin_LFE` from [subproblem \(4.3f\)](#).

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

Solution: See [Listing 4.3](#).

Listing 4.3: Implementation for `main_LFE_robin`

```

1 function [N, l2, h1] = main_LFE_robin(Mesh)
2
3     % Initialize constants
4     F_HANDLE = @(x,varargin) 1;
5     E_HANDLE = @(x,varargin) 1;
6
7     % Assemble stiffness matrix and load vector
8     A = assemMatRobin_LFE(Mesh, E_HANDLE);
9     L = assemLoad_LFE(Mesh,P706(),F_HANDLE);
10
11     % Solve the linear system
12     U = A \ L;
13
14     % Plot solution

```



```

15     plot_LFE(U,Mesh); colorbar;
16
17     % Output
18     N = size(Mesh.Coordinates,1);
19     l2 = L2Err_LFE(Mesh, U, P7O6(),...
20                   @(x,varargin) zeros(size(x,1),1));
21     h1 = H1SErr_LFE(Mesh, U, P7O6(),...
22                    @(x,varargin) zeros(size(x,1),2));
23
24 end

```

(4.3h) Solve and plot the linear finite element solution of (5.2.1) with boundary conditions (4.3.2) based on the meshes found in the zip-file Polygon.zip. 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.

Solution: See Listing 4.3 and Listing 4.4 for the code. The plots can be found in Figure 4.1.

Listing 4.4: Implementation for driver_robin

```

1  N = zeros(5,1);
2  L2 = zeros(5,1);
3  H1 = zeros(5,1);
4
5  for nv = 2:6,
6      Mesh = load(['Polygon_' num2str(nv) '.mat']);
7      [n,l2,h1] = main_LFE_robin(Mesh);
8      N(nv-1) = n;
9      L2(nv-1) = l2;
10     H1(nv-1) = h1;
11 end
12
13 figure(1); semilogx(N, L2);
14 figure(2); semilogx(N, H1);

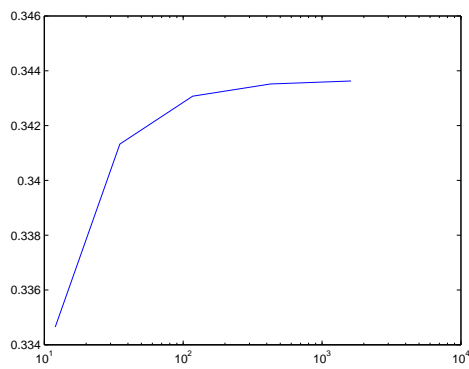
```

Listing 4.5: Testcalls for Problem 4.3

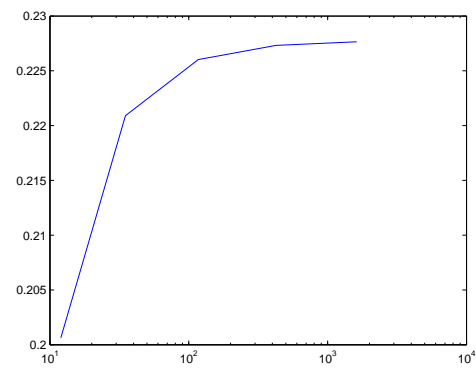
```

1  fprintf(' \n##STIMA_LaplRobin_LFE');
2  STIMA_LaplRobin_LFE([0 0; 1 0; 0 1], [1; 3], @(x) sum(x.^2,2))
3
4  clear Mesh
5  Mesh = load(['Polygon_' num2str(2) '.mat']);
6
7  fprintf(' \n##assemMatRobin_LFE');
8  assemMatRobin_LFE(Mesh, @(x) sum(x.^2,2))

```



(a) L^2 -norm versus N



(b) H^1 -seminorm versus N

Figure 4.1: Plots for [subproblem \(4.3h\)](#).

Listing 4.6: Output for Testcalls for [Problem 4.3](#)

```

1 >> test_call
2
3 ##STIMA_LaplRobin_LFE
4 ans =
5
6     1.0000    -0.5000    -0.5000
7     -0.5000     1.7071         0
8     -0.5000         0     1.2071
9
10 ##assemMatRobin_LFE
11 ans =
12
13     (1,1)     1.3436
14     (6,1)    -0.5580
15     (7,1)    -0.5200
16     (8,1)    -0.2656
17     (2,2)     1.5040
18     (8,2)    -0.3096
19     (9,2)    -0.2684
20    (10,2)    -0.4820
21     (3,3)     1.8477
22    (10,3)    -0.5217
23    (11,3)    -0.5652
24     (4,4)     1.9935
25     (6,4)    -0.2294
26     (9,4)    -0.2572
27    (11,4)    -0.6167
28    (12,4)    -0.3152
29     (5,5)     1.0606
30     (7,5)    -0.6129
31    (12,5)    -0.1935
32     (1,6)    -0.5580

```

33	(4, 6)	-0.2294
34	(6, 6)	3.6617
35	(7, 6)	-0.4417
36	(8, 6)	-0.5524
37	(9, 6)	-0.8640
38	(12, 6)	-1.0161
39	(1, 7)	-0.5200
40	(5, 7)	-0.6129
41	(6, 7)	-0.4417
42	(7, 7)	1.9506
43	(12, 7)	-0.3246
44	(1, 8)	-0.2656
45	(2, 8)	-0.3096
46	(6, 8)	-0.5524
47	(8, 8)	1.8610
48	(9, 8)	-0.6085
49	(2, 9)	-0.2684
50	(4, 9)	-0.2572
51	(6, 9)	-0.8640
52	(8, 9)	-0.6085
53	(9, 9)	3.6930
54	(10, 9)	-0.9173
55	(11, 9)	-0.7776
56	(2, 10)	-0.4820
57	(3, 10)	-0.5217
58	(9, 10)	-0.9173
59	(10, 10)	2.4882
60	(11, 10)	-0.0431
61	(3, 11)	-0.5652
62	(4, 11)	-0.6167
63	(9, 11)	-0.7776
64	(10, 11)	-0.0431
65	(11, 11)	2.5556
66	(4, 12)	-0.3152
67	(5, 12)	-0.1935
68	(6, 12)	-1.0161
69	(7, 12)	-0.3246
70	(12, 12)	2.1788

Problem 4.4 Finite differences and implicit Euler for the heat equation

Let $\Omega := (0, 1)$ and consider a one-dimensional *heat equation* (an example of a parabolic PDE) with homogeneous Dirichlet boundary conditions:

$$\frac{\partial u}{\partial t} - \Delta u = f(x) \quad \text{in } (0, T] \times \Omega, \quad (4.4.1)$$

$$u = 0 \quad \text{on } (0, T] \times \partial\Omega, \quad (4.4.2)$$

$$u = u_0 \quad \text{on } \{0\} \times \Omega. \quad (4.4.3)$$

The initial data is given by $u_0(x) = \sin(\pi x)$, and we will mainly consider the homogeneous case, i.e. $f(x) \equiv 0$. The aim of this exercise is to discretize the above heat equation using central finite

differences coupled with the implicit Euler time-stepping scheme.

To discretize the spatial domain $\Omega = (0, 1)$, we subdivide the interval $[0, 1]$ in $N + 1$ subintervals using equispaced grid points $\{x_0 = 0, x_1, \dots, x_N, x_{N+1} = 1\}$.

The discretized (in space domain Ω) problem can be written as the following linear system,

$$\frac{\partial}{\partial t} \mathbf{u} + \mathbf{A} \mathbf{u} = \mathbf{L}, \quad (4.4.4)$$

where \mathbf{A} is a $N \times N$ matrix, \mathbf{L} a $N \times 1$ vector and \mathbf{u} the $N \times 1$ vector containing the unknowns $u(x_j)$, $j = 1, \dots, N$, i.e. denoting the values of the function u at the grid points.

Let us denote by $h = |x_1 - x_0|$ the mesh size.

(4.4a) Write the matrix \mathbf{A} and the right-handside \mathbf{L} in (4.4.4), which will be analogous as in the case of Laplace equation you had in Problem 1 of the Exercise sheet 1.

HINT: Careful with the factor $\frac{1}{h^2}$.

Solution: We have:

$$\mathbf{A} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & -1 & 2 & -1 \\ 0 & \dots & \dots & -1 & 2 \end{pmatrix} \quad \mathbf{L} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix} \equiv 0.$$

(4.4b) Implement a function

```
function A = PoissonMatrix(N)
```

which computes the matrix \mathbf{A} for (4.4.4). Here the input parameter N denotes the number of *internal* grid points.

Solution: See listing 4.7 for the code.

Listing 4.7: Implementation for PoissonMatrix

```
1 function A = PoissonMatrix(N)
2
3 e = ones(N,1);
4 A = spdiags((N+1)^2 * [-e 2*e -e], -1:1, N, N);
```

(4.4c) Implement a function

```
function L = RHS(FHandle,N)
```

to compute the right-handside \mathbf{L} (also for $f \neq 0$) for (4.4.4). The input parameter `FHandle` is the function handle for the right-handside $f(x)$ and N is again the number of interior grid points.

Solution: See listing 4.8 for the code.

Listing 4.8: Implementation for RHS

```

1 function L = RHS(FHandle,N)
2
3 h = 1/(N+1);
4 L = FHandle( (h:h:1-h)' );

```

(4.4d) Next, we discretize the time domain $[0, T]$ by considering $M \in \mathbb{N}$ equispaced time points

$$t_0 = 0, \quad t_1, \quad \dots, \quad t_M = T,$$

and denote the time step size by $\Delta t := |t_{n+1} - t_n| = T/M$. Then, at a time point $t = t_n$ with $n = 0, \dots, M-1$, the solution vector with values $u(x_0, t^n), \dots, u(x_{N+1}, t^n)$ is denoted by \mathbf{u}^n . Show that the implicit Euler scheme applied to (4.4.4) at a time point $t = t_n$ (for $n = 0, \dots, M-1$), gives the following linear system for \mathbf{u}^{n+1} :

$$(\mathbf{1} + \Delta t \mathbf{A}) \mathbf{u}^{n+1} = \mathbf{u}^n, \quad (4.4.5)$$

where $\mathbf{1}$ denotes the $N \times N$ identity matrix.

HINT: For an ODE $\frac{\partial}{\partial t} \mathbf{u} = g(t, \mathbf{u})$, the implicit Euler scheme with timestep Δt is

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t g(t_{n+1}, \mathbf{u}^{n+1}).$$

Solution: The implicit Euler scheme is given as follows:

$$\begin{aligned}
 \frac{\partial}{\partial t} \mathbf{u}(t) &= -\mathbf{A} \mathbf{u}(t) \\
 \Rightarrow \mathbf{u}^{n+1} &= \mathbf{u}^n - \Delta t \mathbf{A} \mathbf{u}^{n+1} \\
 \Rightarrow (\mathbf{1} + \Delta t \mathbf{A}) \mathbf{u}^{n+1} &= \mathbf{u}^n.
 \end{aligned}$$

(4.4e) Implement the function

```
function un = implicitEuler(u0,A,T,M)
```

which performs implicit Euler time stepping by recursive solves of the linear system (4.4.5).

The input parameter `u0` is the $N \times 1$ vector containing the values $\{u_0(x_j)\}_{j=1}^N$ of the initial data u_0 at the interior grid points, `A` is as in subproblem (4.4b), `M` is the number of time steps, and `T` is the final time, i.e. $t^M = T$. The output `un` is the array $\{u_h(x_j, t^n)\}_{j=1}^N$ containing the approximate value of the solution u at the interior grid points $\{x_j\}_{j=1}^N$ and at the time point $t = t^M$.

Solution: See listing 4.9 for the code.

Listing 4.9: Implementation for implicitEuler

```

1 function un = implicitEuler(u0,A,T,M)
2
3 % compute time step size
4 dt = T / M;
5

```

```

6  % initialize time t = 0
7  un = u0;
8
9  % compute eye(N) + dt * A for efficiency
10 eye_dtA = eye(size(A)) + dt * A;
11
12 % run the for-loop for each time step
13 for n = 1:M
14     un = eye_dtA \ un;
15 end

```

(4.4f) Implement the function

```
function uhn = HeatSolve(N,M)
```

to solve the homogeneous heat equation (4.4.1) up to time $T = 0.1$.

The input parameters are as in subproblems (4.4b) and (4.4e). The output uhn is the array $\{u_h(x_j, t^M)\}_{j=0}^{N+1}$ containing the approximate value of the solution u at the grid points $\{x_j\}_{j=0}^{N+1}$ and at the final time point $t = T = t^M$.

HINT: Use the routines from subproblems (4.4b) and (4.4e).

Solution: See listing 4.10 for the code.

Listing 4.10: Implementation for HeatSolve

```

1  function uhn = HeatSolve(N,M)
2
3  % time horizon
4  T = 0.1;
5
6  % spatial grid
7  xr = linspace(0,1,N+2)';
8
9  % initial data
10 u0 = sin(pi*xr);
11
12 % solver
13 A = PoissonMatrix(N);
14 uhn = zeros(N+2,1);
15 uhn(2:end-1) = implicitEuler(u0(2:end-1), A, T, M);

```

(4.4g) Run the routine HeatSolve for $N = 50$ and $M = 100$ and plot the solution at initial time $t = 0$ and final time $t = T$. Due to (zero) Dirichlet boundary conditions, you will observe that the initial temperature distribution u_0 is gradually diffusing towards zero.

Solution: See listing 4.11 for the code and Fig. for the plot.

Listing 4.11: Implementation for plotSolution

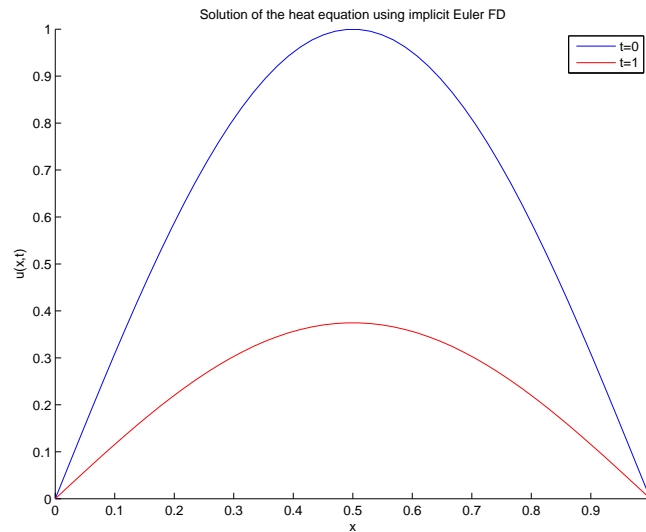


Figure 4.2: Plot for subproblem (4.4g)

```

1 function plotSolution ()
2
3 N = 50;
4 M = 100;
5 xr = linspace(0,1,N+2)';
6
7 uh0 = HeatSolve(N,0);
8 uhn = HeatSolve(N,M);
9
10 hold on;
11 plot(xr, uh0, 'b');
12 plot(xr, uhn, 'r');
13 legend('t=0', 't=1');
14 xlabel('x');
15 ylabel('u(x,t)');
16 title('Solution of the heat equation using implicit Euler FD')
17 hold off;
18
19 print -depsc '../fig/sol.eps'

```

(4.4h) If we kept the parameters the same, i.e. $N = 50$ and $M = 100$, and replaced the *implicit* Euler time stepping by *explicit* Euler time stepping scheme, would our numerical method still be appropriate for approximation of the solution u ? If not, what could N or M be modified to achieve this?

HINT: Consider stability and the CFL condition.

HINT: Alternatively, you could to implement the *explicit* Euler time stepping scheme and see what happens. Do not waist too much time in debugging – it is not possible to obtain a stable

approximation without modifying N or M .

Solution: The CFL condition

$$\frac{\Delta t}{h^2} \leq \frac{1}{2} \quad (4.4.6)$$

is violated for $N = 50$ and $M = 100$. Hence, the *explicit* Euler scheme will be unstable and will not approximate solution u . If we increase M to $M = 1000$, the CFL condition is satisfied and the scheme becomes stable.

Published on April 14.

To be submitted on April 30.

Last modified on May 2, 2014