ETH Lecture 401-0663-00L Numerical Methods for PDEs

Final Exam

Spring Term 2018

16. August 2018, 14:00, HG G 1

Don't panic!

Family Name		**Grade**
First Name		
Department		
Legi Nr.		
Date	16. August 2018	

Points:

Task	1	2	3	4	Total
maximum pts.	30	25	30	50	135
achieved pts.					

(100% = 90Pt, 20% = 18Pt, pass = 30Pt)

Grade scale:

Points	Grade
0-2	1
3-5	1.25
6-8	1.5
9-11	1.75
12-13	2.0
14-15	2.25
16-18	2.5
19-20	2.75
21-22	3.0
23-24	3.25
25-27	3.5
28-29	3.75
30-37	4.0
38-44	4.25
45-52	4.5
53-60	4.75
61-68	5.0
69-76	5.25
77-82	5.5
83-89	5.75
$\geq$90	6.0

- **Duration: 180 minutes.**
- Keep calm!
- Cell phones and other communication devices are not allowed. Make sure that they are turned off and stowed away in your bag.
- Permitted materials (to be checked during the exam): summary of up to 10 A4 pages in the candidates own handwriting. No printouts and copies are allowed.
- You may cite theorems, lemmas, and equations from the lecture document by specifying their precise number in the supplied PDF.
- Begin each main problem on a new sheet of paper and write your name and the number of the exercise in the top right corner.
- *Write clearly with a non-erasable pen. Do not use red pen or green pen.* No more than one solution can be handed in per problem. Invalid attempts should be clearly crossed out.
- Include all considerations, auxiliary computations, etc. in your solution.
- Get a general idea of the problems. Pay attention to the number of points awarded for each subtask. It is roughly correlated with the amount of work the task will require.
- If you have failed to solve a sub-problem, do not give up on the entire problem, but try the next one. Dependency between sub-problems are stated when needed.
- At the beginning of the exam follow the steps listed below. This is not counted as exam time. **You are not allowed to start the exam until you have a clear indication from an assistant or the Professor.**
    1. Fill in the cover page lines and put your ETH ID card ("legi") on the table.
    2. You will get a separate form; fill it in and sign it. An assistant will collect it at the end of the exam.
    3. An assistant will give you blank exam sheets. Please put your name on each of them. Together with the problem sheet and the cover sheet they have to be handed in when you are finished with the exam.
    4. Please log into the exam computer with your first and last name plus NETHZ login.
    5. Go to the folder '`~/documentation`'.
    6. Open the main links `index-`....html in a web browser and verify that they work.
    7. Go to the folder '`~/questions`'. There you will find a folder for each problem involving implementation tasks.
    8. Open your preferred editor among the available ones. `Qt Creator` was tested, but other IDEs are also available. However, please notice that by using any of them you will take full responsibility of its setting and you should do so during your own exam time.
    9. Only when you are asked so, open the problem sheet. All the problems will be read aloud. You are only allowed to read through the problems with the Professor. Do not write or type until the actual start of the exam is announced.

Instructions concerning the C++ implementation

- Please **do not** modify the `CMake` files or any file in '`~/resources`'. The time required to restore them will be part of your own exam time.

- **Work and save all result files** (codes, plots, etc.) in the directory '∼**/questions**'**! Only the files in this folder will be corrected.**
- In order to compile your code, if you do not want to rely on an IDE, you can create a build folder in your `home` folder. E.g.,

```
mkdir build
cd build
cmake ../questions
```

- To compile one problem at a time, run

```
make problemX
```

  where `X` is the problem number.

At the end of the examination

- **Do not log out and do not turn off the computer!**
- Your written results will be collected together with the instruction and problem sheets.

Problems

- If you have problems with the computer, please raise your hand to get support.

Problem 1: First-order System Least-Squares Variational Formulation (30 pts)

This problem studies a non-standard variational formulation of a second-order elliptic boundary value problem. It is related to [Lecture → Chapter 2] and [Lecture → Chapter 5].

This problem does not involve coding.

Given a bounded computational domain $\Omega \subset \mathbb{R}^2$ we consider the quadratic minimization problem

$$(u, \mathbf{j}) = \operatorname*{argmin}_{(v,\mathbf{q})\in V} J\big((v,\mathbf{q})\big)\,, \tag{0.1.1}$$

$$J\big((v,\mathbf{q})\big) := \tfrac{1}{2}\left\| \sqrt{\alpha}\,\mathbf{grad}\, v - \frac{1}{\sqrt{\alpha}}\mathbf{q}\right\|^2_{L^2(\Omega)} + \tfrac{1}{2}\|\operatorname{div}\mathbf{q} - f\|^2_{L^2(\Omega)}\,, \tag{0.1.2}$$

(see [Lecture → Suppl. 2.5.11] for the definition of div) on the space

$$V := C^1_0(\overline{\Omega}) \times \left(C^1(\overline{\Omega})\right)^2. \tag{0.1.3}$$

Here, $\alpha \in C^0(\overline{\Omega})$ is a uniformly positive, real-valued function, and $f \in L^2(\Omega)$. Also recall the definition

$$C^1_0(\overline{\Omega}) := \{v : \overline{\Omega} \to \mathbb{R} \text{ continuously differentiable},\ v|_{\partial\Omega} = 0\}\,. \tag{0.1.4}$$

(1.a) ⚄ (5 pts) State a second-order scalar linear elliptic Dirichlet boundary value problem on Ω, whose solution will provide the u-component of a solution of (0.1.2).

HINT 1 for (1.a): Which equations will u and $\mathbf{j}$ satisfy, if $J\big((u,\mathbf{j})\big) = 0$? ⌟

SOLUTION of (1.a):

Since $J((v,\mathbf{q})) \geq 0\ \forall (v,\mathbf{q}) \in V$ we conclude that , u and $\mathbf{j}$ satisfying $J(v,\mathbf{q}) = 0$ are necessarily minimizers of J:

$$J(u,\mathbf{j}) = 0 \quad\Rightarrow\quad (u,\mathbf{j}) \in \operatorname{argmin}_{(v,\mathbf{q})\in V} J\big((v,\mathbf{q})\big)\,. \tag{0.1.5}$$

It is clear from the definition of J:

$$J(v,\mathbf{q}) = 0 \quad\Rightarrow\quad \begin{cases} \sqrt{\alpha}\,\mathbf{grad}\, v = \frac{1}{\sqrt{\alpha}}\mathbf{q} \\ \operatorname{div}\mathbf{q} = f \end{cases} \quad \text{in} \quad \Omega\,. \tag{0.1.6}$$

By applying the divergence operator to the first equation of (0.1.6), since $v|_{\partial\Omega} = 0$ we have the following second-order scalar linear elliptic Dirichlet boundary value problem on Ω for the u-component solution to (0.1.5):

$$\begin{cases} \operatorname{div}(\alpha\,\mathbf{grad}\, u) = f & \text{on } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{0.1.7}$$

The boundary condition for u is built into the space V, see (0.1.4).

The Dirichlet problem (0.1.7) always has a solution, from which we obtain $(u,\mathbf{j})$ with $J(u,\mathbf{q}) = 0$. This confirms that the minimal value 0 is attained by J.

▲

(1.b) ⚄ (5 pts) State a linear variational problem

(i) with a *symmetric* bilinear form $\mathsf{a}_F : V \times V \to \mathbb{R}$

(ii) for which every solution of the *quadratic minimization problem* (0.1.1) is also a solution.

SOLUTION of (1.b):

This problem can be tackled in two ways:

(I) We can rely on the calculus of variations approach:

To state the variational formulation of our problem, we write:

$$\lim_{t\to 0} \frac{J((v+tw,\mathbf{q}+t\mathbf{p})) - J((v,\mathbf{q}))}{t} = 0, \quad \forall (w,\mathbf{p}) \in V\,.$$

We obtain

$$\lim_{t\to 0} \frac{J((v+tw,\mathbf{q}+t\mathbf{p})) - J((v,\mathbf{q}))}{t} =$$
$$= \int_\Omega \alpha^{-1}\mathbf{q}\cdot\mathbf{p} + \alpha\,\mathbf{grad}\,v\cdot\mathbf{grad}\,w - \mathbf{grad}\,w\cdot\mathbf{q} - \mathbf{grad}\,v\cdot\mathbf{p} + \operatorname{div}\mathbf{q}\operatorname{div}\mathbf{p} - \operatorname{div}\mathbf{p}\, f\,\mathrm{d}\boldsymbol{x} = 0,$$

which leads to the following variational equation

$$\mathbf{a}_F((v,\mathbf{q}),(w,\mathbf{p})) - \ell((w,\mathbf{p})) = 0 \quad \forall (w,\mathbf{p}) \in V, \tag{0.1.8}$$

where

$$\ell((w,\mathbf{p})) = \int_\Omega \operatorname{div}\mathbf{p}\, f\,\mathrm{d}\boldsymbol{x}\,,$$

and $\mathbf{a}_F$ is a symmetric bilinear form defined by

$$\mathbf{a}_F((v,\mathbf{q}),(w,\mathbf{p})) = \int_\Omega \alpha^{-1}\mathbf{q}\cdot\mathbf{p} + \alpha\,\mathbf{grad}\,v\,\mathbf{grad}\,w - \mathbf{grad}\,w\cdot\mathbf{q} - \mathbf{grad}\,v\cdot\mathbf{p} + \operatorname{div}\mathbf{q}\operatorname{div}\mathbf{p}\,\mathrm{d}\boldsymbol{x}\,. \tag{0.1.9}$$

(II) We can start from the observation that (0.1.2) is a quadratic minimization problem:

$$J((v,\mathbf{q})) = \tfrac{1}{2}\mathbf{a}_F((v,\mathbf{q}),(v,\mathbf{q})) - \ell((v,\mathbf{q})) + \|f\|^2_{L^2(\Omega)}\,. \tag{0.1.10}$$

where $\mathbf{a}_F$ and ℓ are as in (0.1.9) and ((I)). The identity (0.1.10) emerges from straightforward elementary computations.

▲

(1.c) ⚄ (5 pts) [depends on Sub-problem (1.b)]

Show that the bilinear form $\mathsf{a}_F : V \times V \to \mathbb{R}$ underlying the variational problem from Sub-problem (1.b) is continuous with respect to the energy norm

$$\|(v,\mathbf{q})\|^2_A := \|\mathbf{grad}\,v\|^2_{L^2(\Omega)} + \|\operatorname{div}\mathbf{q}\|^2_{L^2(\Omega)} + \|\mathbf{q}\|^2_{L^2(\Omega)}\,, \quad (v,\mathbf{q}) \in V\,. \tag{0.1.11}$$

HINT 1 for (1.c): Use the two versions of the Cauchy-Schwarz inequality

$$\int_\Omega v\,w\,\mathrm{d}\boldsymbol{x} \le \|v\|_{L^2(\Omega)}\,\|w\|_{L^2(\Omega)} \quad \forall v,w \in L^2(\Omega)\,, \tag{0.1.12}$$

$$\left|\sum_{i=1}^n \alpha_i\beta_i\right| \le \left(\sum_{i=1}^n \alpha_i^2\right)^{\frac{1}{2}} \cdot \left(\sum_{i=1}^n \beta_i^2\right)^{\frac{1}{2}} \quad \forall \alpha_i,\beta_i \in \mathbb{R}\,. \tag{0.1.13}$$

SOLUTION of (1.c):

We want to prove the continuity of $\mathbf{a}_F$ w.r.t. the energy norm $\|\cdot\|_A$, that is

$$|\mathbf{a}_F((v,\mathbf{q}),(w,\mathbf{p}))|^2 \le C\|(v,\mathbf{q})\|_A^2\|(w,\mathbf{p})\|_A^2 \quad \forall (v,\mathbf{q}),(w,\mathbf{p}) \in V\,. \tag{0.1.14}$$

By applying the triangle inequality and the Cauchy-Schwarz inequality to $\mathbf{a}_F$ in (0.1.9), we obtain

$$\begin{aligned}|\mathbf{a}_F((v,\mathbf{q}),(w,\mathbf{p}))| \le C_1\Big(&\|\mathbf{q}\|_{L^2(\Omega)}\|\mathbf{p}\|_{L^2(\Omega)} + \|\operatorname{\mathbf{grad}} v\|_{L^2(\Omega)}\|\operatorname{\mathbf{grad}} w\|_{L^2(\Omega)} + \\ &\|\mathbf{q}\|_{L^2(\Omega)}\|\operatorname{\mathbf{grad}} w\|_{L^2(\Omega)} + \|\operatorname{\mathbf{grad}} v\|_{L^2(\Omega)}\|\mathbf{p}\|_{L^2(\Omega)} + \|\operatorname{div}\mathbf{q}\|_{L^2(\Omega)}\|\operatorname{div}\mathbf{p}\|_{L^2(\Omega)}\Big)\end{aligned}$$

where $C_1 = \max\{1, \|\alpha^{1/2}\|_{L^\infty(\Omega)}, \|\alpha^{-1/2}\|_{L^\infty(\Omega)}\}$. Since, by (0.1.13), $(A+B+C+D)^2 \le 4(A^2+B^2+C^2$ for any real numbers A,B,C, and D, the following inequality holds:

$$\begin{aligned}|\mathbf{a}_F((v,\mathbf{q}),(w,\mathbf{p}))|^2 \le C\Big(&\|\mathbf{q}\|^2_{L^2(\Omega)}\|\mathbf{p}\|^2_{L^2(\Omega)} + \|\operatorname{\mathbf{grad}} v\|^2_{L^2(\Omega)}\|\operatorname{\mathbf{grad}} w\|^2_{L^2(\Omega)} + \\ &\|\mathbf{q}\|^2_{L^2(\Omega)}\|\operatorname{\mathbf{grad}} w\|^2_{L^2(\Omega)} + \|\operatorname{\mathbf{grad}} v\|^2_{L^2(\Omega)}\|\mathbf{p}\|^2_{L^2(\Omega)} + \|\operatorname{div}\mathbf{q}\|^2_{L^2(\Omega)}\|\operatorname{div}\mathbf{p}\|^2_{L^2(\Omega)}\Big)\end{aligned}$$

for $C = 4C_1^2$, which proves (0.1.14).

▲

Estimates employing the Helmholtz decomposition of vector fields give the following result for the bilinear form found in Sub-problem (1.b):

Theorem 0.1.15. Ellipticity of FOSLS bilinear form [CLM94]

There is a constant $\gamma > 0$ depending only on Ω such that

$$\mathrm{a}_F\big((v,\mathbf{q}),(v,\mathbf{q})\big) \ge \gamma\|(v,\mathbf{q})\|_A^2 \quad \forall (v,\mathbf{q}) \in V\,.$$

We restrict ourselves to polygonal domains Ω. Next we perform a Galerkin finite element discretization of the variational problem from Sub-problem (1.b). Based on a triangular mesh $\mathcal{M}$ of Ω we use the finite element space

$$V_{0,N} := \mathcal{S}^0_{2,0}(\mathcal{M}) \times \left(\mathcal{S}^0_2(\mathcal{M})\right)^2\,.$$

(1.d) ⚄ (5 pts) [depends on Sub-problem (1.b)]

Argue why $V_{0,N}$ can be used for the discretization of the variational problem from Sub-problem (1.b) or, equivalently, the minimization problem (0.1.2), though, owing to lack of smoothness, $V_{0,N} \not\subset V$ in general.

SOLUTION of (1.d):

We follow the considerations of [Lecture → Section 2.3.1]: We can use any finite element space for the Galerkin discretization of the minimization problem (0.1.2)/the FOSLS variational problem (0.1.8) for which the energy norm (0.1.11) is well defined, that is, finite.

In particular, by virtue of the trivial estimate $\|\operatorname{div}\mathbf{q}\|_{L^2(\Omega)} \leq \|\mathbf{q}\|_{H^1(\Omega)}$, this is the case for $(v,\mathbf{q}) \in H^1_0(\Omega) \times (H$ of which the proposed finite element space is a subspace.

Remark. It turns out that the continuity of v and the continuity of normal components of $\mathbf{q}$ are the key properties of valid arguments of a_F.

▲

(1.e) ⚂ (10 pts) For testing the Galerkin finite scheme introduced above we consider $\Omega =]0,1[^2$, $\alpha \equiv 1$, and choose f in (0.1.2) such that we obtain as exact solution

$$u(\boldsymbol{x}) = \sin(\pi x_1)\sin(\pi x_2) \quad , \quad \mathbf{j}(\boldsymbol{x}) = \pi \begin{bmatrix} \cos(\pi x_1)\sin(\pi x_2) \\ \sin(\pi x_2)\cos(\pi x_2) \end{bmatrix} .$$

Denote by $(u_N, \mathbf{j}_N) \in V_{0,N}$ the Galerkin finite element solution, given $N := \dim V_{0,N}$.

Describe qualitatively and quantitatively the asympotic convergence of

$$\|u - u_N\|_{H^1(\Omega)} \quad \text{and} \quad \|\operatorname{div}(\mathbf{j} - \mathbf{j}_N)\|_{L^2(\Omega)}$$

for $N \to \infty$, that one expects on a sequence of meshes obtained by regular uniform refinement of some coarse initial mesh.

HINT 1 for (1.e): Base your arguments on Thm. 0.1.15 and the assertion proved in Sub-problem (1.c).

SOLUTION of (1.e):

To begin with, we conclude from Thm. 0.1.15 that the bilinear form a_F is s.p.d. and, hence, induces a norm $\|\cdot\|_a$, which is *equivalent* to the energy norm $\|\cdot\|_A$.

Therefore we can apply Cea's lemma [Lecture → Thm. 5.1.15] and we obtain

$$\|(u - u_N, \mathbf{j} - \mathbf{j}_N)\|_a^2 = \inf_{(v_N,\mathbf{q}_N)\in V_{0,N}} \|(u - v_N, \mathbf{j} - \mathbf{q}_N)\|_a^2 .$$

We continue based on the result of Sub-problem (1.c):

$$\begin{aligned} \|(u - u_N, \mathbf{j} - \mathbf{j}_N)\|_A^2 &\leq C \inf_{(v_N,\mathbf{q}_N)\in V_{0,N}} \left(\|\operatorname{\mathbf{grad}}(u - v_N)\|_{L^2(\Omega)}^2 + \|\operatorname{div}(\mathbf{j} - \mathbf{q}_N)\|_{L^2(\Omega)}^2 + \|\mathbf{j} - \mathbf{q}_N\|_{L^2(\Omega)}^2 \right) \\ &\leq C \inf_{v_N\in\mathcal{S}^0_2(\mathcal{M})} \|\operatorname{\mathbf{grad}}(u - v_N)\|_{L^2(\Omega)}^2 + \inf_{\mathbf{q}_N\in(\mathcal{S}^0_2(\mathcal{M}))^2} \|\mathbf{j} - \mathbf{q}_N\|_{H^1(\Omega)}^2 . \end{aligned}$$

Then, to describe qualitatively and quantitatively the asympotic convergence of $\|u - u_N\|_{H^1(\Omega)}$ and $\|\operatorname{div}(\mathbf{j} - \mathbf{j}_N)\|_{L^2(\Omega)}$ for $N \to \infty$, we rely on [Lecture → Thm. 5.3.56].

Since the exact solutions u and $\mathbf{j}$ are smooth for this problem, by [Lecture → Thm. 5.3.56] we can rely on the following best approximation estimates for quadratic Lagrangian finite elements ($p = 2$):

$$\inf_{v_N\in\mathcal{S}^0_2(\mathcal{M})} \|u - v_N\|_{H^1(\Omega)} \leq C\frac{h_{\mathcal{M}}^2}{2}\|u\|_{H^3(\Omega)} ,$$

$$\inf_{\mathbf{q}_N \in \mathcal{S}_2^0(\mathcal{M})} \|\operatorname{div}(\mathbf{j}-\mathbf{q}_N)\|_{L^2(\Omega)} \leq \inf_{\mathbf{q}_N \in \mathcal{S}_2^0(\mathcal{M})} \|\mathbf{j}-\mathbf{q}_N\|_{H^1(\Omega)} \leq C\frac{h_{\mathcal{M}}^2}{2}\|\mathbf{j}\|_{H^3(\Omega)} ,$$

with constants depending only on the shape-regularity measure of the meshes.

As $N \approx h_{\mathcal{M}}^{-2}$, this means that we have the following asymptotic convergence for $N \to \infty$:

$$\|u-u_N\|_{H^1(\Omega)} = O(N^{-1}) ,$$

$$\|\operatorname{div}(\mathbf{j}-\mathbf{j}_N)\|_{L^2(\Omega)} = O(N^{-1}) .$$

In each case we expect algebraic convergence with rate $(-)1$.

▲

End Problem 1

Problem 2: Blended parametric representation of curvi-linear triangles (25 pts)

This problem explores a new way for the construction of parametric finite elements. It relates to [Lecture → Section 3.8]

This problem relies on EIGEN for an implementation in C++.

The figure beside displays a so-called curvi-linear triangle that is a bounded domain $K \subset \mathbb{R}^2$ with three corners $\boldsymbol{a}^0$, $\boldsymbol{a}^1$, and $\boldsymbol{a}^2$, joined by non-intersecting curves γ_{01}, γ_{12}, and γ_{20}.

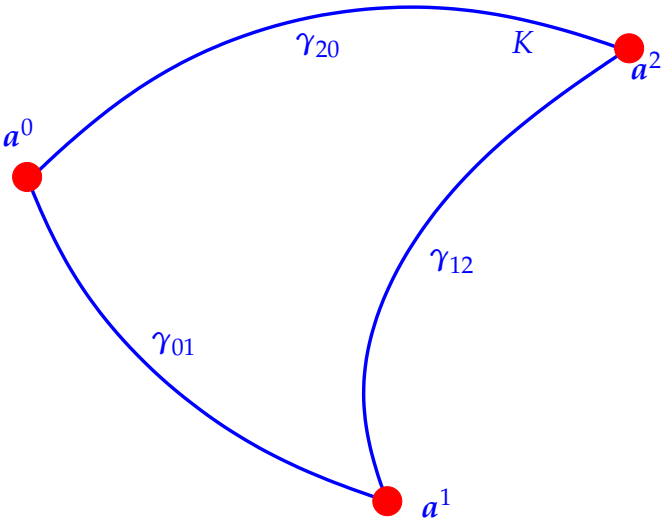

Fig. 1

These three curves are given by smooth parameterizations:

$$\gamma_{01}, \gamma_{12}, \gamma_{20} : [0,1] \to \mathbb{R}^2$$

such that

$$\begin{aligned} \gamma_{01}(0) &= \boldsymbol{a}^0 , & \gamma_{01}(1) &= \boldsymbol{a}^1 , \\ \gamma_{12}(0) &= \boldsymbol{a}^1 , & \gamma_{12}(1) &= \boldsymbol{a}^2 , \\ \gamma_{20}(0) &= \boldsymbol{a}^2 , & \gamma_{20}(1) &= \boldsymbol{a}^0 . \end{aligned}$$

Based on this description of the curvi-linear triangle, we introduce the mapping $\boldsymbol{\Phi} : [0,1]^2 \to \mathbb{R}^2$,

$$\begin{aligned} \boldsymbol{\Phi}(\widehat{\boldsymbol{x}}) := \gamma_{01}(\widehat{x}_1) + \gamma_{20}(1-\widehat{x}_2) - \gamma_{01}(0) + \widehat{x}_1\gamma_{12}(\widehat{x}_2) + \widehat{x}_2\gamma_{12}(1-\widehat{x}_1) - \\ \widehat{x}_1(\gamma_{01}(1-\widehat{x}_2) + \gamma_{20}(1-\widehat{x}_2) - \gamma_{01}(0)) - \widehat{x}_2(\gamma_{01}(\widehat{x}_1) + \gamma_{20}(\widehat{x}_1) - \gamma_{01}(0)) , \end{aligned} \tag{0.2.1}$$

for $\widehat{\boldsymbol{x}} = [\widehat{x}_1, \widehat{x}_2]^\top \in [0,1]^2$.

(2.a) ⚂ (5 pts) Show that $\boldsymbol{\Phi}$ from (0.2.1) satisfies

$$\boldsymbol{\Phi}\left(\begin{bmatrix}\xi\\0\end{bmatrix}\right) = \gamma_{01}(\xi)\,, \quad 0 \le \xi \le 1\,. \tag{0.2.2}$$

SOLUTION of (2.a):

By miraculous cancellation we have

$$\begin{aligned}
\boldsymbol{\Phi}\left(\begin{bmatrix}\xi\\0\end{bmatrix}\right) &= \gamma_{01}(\xi) + \gamma_{02}(1) - \gamma_{01}(0) + \xi\gamma_{12}(0) - \xi\gamma_{01}(1) - \xi\gamma_{20}(1) + \xi\gamma_{01}(0)\\
&= \gamma_{01}(\xi)\,,\\
\boldsymbol{\Phi}\left(\begin{bmatrix}0\\\eta\end{bmatrix}\right) &= \gamma_{01}(0) + \gamma_{20}(1-\eta) - \gamma_{01}(0) + \eta\gamma_{12}(1) - \eta\gamma_{01}(0) - \eta\gamma_{20}(0) + \eta\gamma_{01}(0)\\
&= \gamma_{20}(1-\eta)\,,\\
\boldsymbol{\Phi}\left(\begin{bmatrix}1-\tau\\\tau\end{bmatrix}\right) &= \gamma_{01}(1-\tau) + \gamma_{20}(1-\tau) - \gamma_{01}(0) + (1-\tau)\gamma_{12}(\tau) + \tau\gamma_{12}(\tau) -\\
&\quad (1-\tau)\gamma_{01}(1-\tau) - (1-\tau)\gamma_{20}(1-\tau) +\\
&\quad (1-\tau)\gamma_{01}(0) - \tau\gamma_{01}(1-\tau) - \tau\gamma_{20}(1-\tau) + \tau\gamma_{01}(0)\\
&= \gamma_{12}(\tau)\,.
\end{aligned}$$

for $0 \le \xi\,\eta, \tau \le 1$. The first identity is (0.2.2) and is all that is required to solve this sub-problem.

▲

(2.b) ⚂ (3 pts) Give a geometric interpretation of (0.2.2).

SOLUTION of (2.b):

The mapping $\zeta \mapsto \begin{bmatrix}\zeta\\0\end{bmatrix}$, $0 \le \zeta \le 1$, parameterizes the line segment $\left[\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right]$. Hence, the mapping $\boldsymbol{\Phi}$ maps this line segment to the curve given by the parameterization γ_{01}.

▲

(2.c) ⚂ (3 pts) We write $\widehat{K}$ for the (reference) triangle with vertices $\begin{bmatrix}0\\0\end{bmatrix}$, $\begin{bmatrix}1\\0\end{bmatrix}$, and $\begin{bmatrix}0\\1\end{bmatrix}$.

State a formula analogous to (0.2.2) that expresses the fact that $\boldsymbol{\Phi}$ from (0.2.1) maps the edge of $\widehat{K}$ connecting $\begin{bmatrix}1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$ to the edge of K connecting $\boldsymbol{a}^1$ and $\boldsymbol{a}^2$.

SOLUTION of (2.c):

This formula has already been given in the solution of Sub-problem (2.a):

$$\boldsymbol{\Phi}\left(\begin{bmatrix}1-\tau\\\tau\end{bmatrix}\right) = \gamma_{12}(\tau)\,, \quad 0 \le \tau \le 1\,.$$

▲

(2.d) ⚄ (3 pts) We write $\dot{\gamma}_*$ for the derivative of γ_* with respect to its argument, given $* \in \{01, 12, 20\}$.

Using this notation give a formula for the Jacobian $D\boldsymbol{\Phi} : [0,1]^2 \to \mathbb{R}^{2,2}$ of $\boldsymbol{\Phi}$.

HINT 1 for (2.d): The computations boil down to applying the product rule. Recall that the columns of the Jacobian are partial derivatives. ⌟

SOLUTION of (2.d):

The Jacobian is defined as

$$D\boldsymbol{\Phi}(\widehat{\boldsymbol{x}}) = \left[\frac{\partial \boldsymbol{\Phi}}{\partial \widehat{x}_1}(\widehat{\boldsymbol{x}}), \frac{\partial \boldsymbol{\Phi}}{\partial \widehat{x}_2}(\widehat{\boldsymbol{x}})\right] \in \mathbb{R}^{2,2} .$$

Thus the colunns of the Jacobian are

$$\begin{aligned}
\left[D\boldsymbol{\Phi}(\widehat{\boldsymbol{x}})\right]_{:,1} &= \dot{\gamma}_{01}\widehat{x}_1 + \gamma_{12}(\widehat{x}_2) - \widehat{x}_2\dot{\gamma}_{12}(1-\widehat{x}_1) - \gamma_{01}(1-\widehat{x}_2) - \gamma_{20}(1-\widehat{x}_2) \\
&\quad + \gamma_{01}(0) - \widehat{x}_2\dot{\gamma}_{01}(\widehat{x}_1) - \widehat{x}_2\dot{\gamma}_{20}(\widehat{x}_1) , \\
\left[D\boldsymbol{\Phi}(\widehat{\boldsymbol{x}})\right]_{:,2} &= -\dot{\gamma}_{20}(1-\widehat{x}_2) + \widehat{x}_1\dot{\gamma}_{12}(\widehat{x}_2) + \gamma_{12}(1-\widehat{x}_1) + \widehat{x}_1\dot{\gamma}_{01}(1-\widehat{x}_2) + \widehat{x}_1\dot{\gamma}_{20}(1-\widehat{x}_2) \\
&\quad - \gamma_{01}(\widehat{x}_1) - \gamma_{20}(\widehat{x}_1) + \gamma_{01}(0) .
\end{aligned}$$

▲

In some EIGEN-based C++ code curves $[0,1] \mapsto \mathbb{R}^2$ are represented by child classes of the following virtual base class:

```
class Curve {
  public:
  virtual Eigen::Vector2d operator()(double parameter) const = 0;
  virtual Eigen::Vector2d derivative(double parameter) const = 0;
};
```

The `operator()` returns a point on the curve when given a parameter $\in [0,1]$, whereas the method `derivative()` returns the derivative vector.

(2.e) ⚅ (3 pts) [depends on Sub-problem (2.d)]

Implement a C++ function

```
Eigen::MatrixXd JacobianPhi(const Eigen::Vector2d& point,
const Curve& gamma01, const Curve& gamma12, const Curve& gamma20)
```

that provides the Jacobian of $\boldsymbol{\Phi}$ from (0.2.1) when one supplies the coordinates of a point $\in [0,1]^2$, given γ_{01}, γ_{12}, and γ_{20}.

Template. A template for the function **JacobianPhi** is given in the file `Problem2/BlendedParameterizat`

SOLUTION of (2.e):

C++ code 0.2.3: Sub-problem (2.e): Implementation of `JacobianPhi`

```
2 matrix_t JacobianPhi(const coord_t& point,
3                      const Curve& gamma01, const Curve& gamma12,
```

```
                                const Curve& gamma20)
{
    matrix_t J(2,2); // Variable for returning Jacobian
    // The formulas for the columns of the jacobian have been derived
       in Sub-problem (2.d)
    J.col(0) = gamma01.derivative(point[0]) + gamma12(point[1]) -
               point[1]*gamma12.derivative(1.-point[0]) -
               (gamma01(1.-point[1]) + gamma20(1.-point[1]) -
                  gamma01(0.)) -
               point[1]*(gamma01.derivative(point[0]) +
                  gamma20.derivative(point[0]));
    J.col(1) = -gamma20.derivative(1.-point[1]) +
               point[0]*gamma12.derivative(point[1]) +
                  gamma12(1.-point[0]) +
               point[0]*(gamma01.derivative(1.-point[1]) +
                  gamma20.derivative(1.-point[1])) -
               (gamma01(point[0]) + gamma20(point[0]) - gamma01(0.));
    return J;
}
```

▲

(2.f) ⚄ (8 pts) The observations made in Sub-problem (2.b) and Sub-problem (2.c) hint that $\boldsymbol{\Phi}(\widehat{K}) = K$, which we are going to assume.

Implement a C++ function

```
Eigen::MatrixXd evalBlendLocMat(const Curve& gamma01,
                                const Curve& gamma12,
                                const Curve& gamma20);
```

that *approximately* computes the local element matrix for the bilinear form

$$(u,v) \mapsto \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x}$$

for the Laplacian $-\Delta$ on K using parametric linear Lagrangian finite elements based on the parametrization of K through $\boldsymbol{\Phi}$ from (0.2.1). Function arguments `gamma01`, `gamma12`, and `gamma20` of type `const Curve&` specify the curves γ_{01}, γ_{12}, and γ_{20}, respectively.

Consider the following quadrature rule to approximately evaluate any integrals over triangles: for a generic triangle T with vertices $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z} \in \mathbb{R}^2$, use

$$\int_T f \,\mathrm{d}\boldsymbol{x} \approx \frac{|T|}{3}\left(f\left(\tfrac{1}{2}(\boldsymbol{x}+\boldsymbol{y})\right) + f\left(\tfrac{1}{2}(\boldsymbol{y}+\boldsymbol{z})\right) + f\left(\tfrac{1}{2}(\boldsymbol{z}+\boldsymbol{x})\right)\right).$$

Template. A template for the function `evalBlendLocMat` is given in the file `Problem2/BlendedParamet`

HINT 1 for (2.f): Matrices of EIGEN are supplied with methods `determinant()`, `inverse()`, and `transpose()`. ⌟

SOLUTION of (2.f):

C++ code 0.2.4: Sub-problem (2.f): Implementation of `evalBlendLocMat`

```
matrix_t evalBlendLocMat(const Curve& gamma01,
                         const Curve& gamma12,
                         const Curve& gamma20)
{
    // Variable for returning 3×3 element matrix
    matrix_t lclMat(3,3); lclMat.setZero();
    matrix_t xi(2,3); // coordinates of midpoints of curves
    xi.col(0) << 0.5, 0.; xi.col(1) = 0.5, 0.5; xi.col(2) = 0., 0.5;
    // (constant) gradients of barycentric coordinate function on the
       reference element,
    // which are the preimages of the local shape functions under
       pullback
    matrix_t gradEval(3,2);
    gradEval << -1., -1., // grad λ̂_1
                 1.,  0., // grad λ̂_2
                 0.,  1.; // grad λ̂_3
    // Generate element matrix by adding up rank-1 matrices formed
       from gradients
    // of local shape functions at midpoints of edges.
    for(int l=0; l<xi.cols(); ++l) {
        coord_t  xi_l = xi.col(l);
        // Call auxiliary function implemented in Sub-problem (2.e)
        matrix_t Ji_l = JacobianPhi(xi_l, gamma01, gamma12, gamma20);
        numeric_t detJi_l = std::abs(Ji_l.determinant());
        // Transformation matrix for gradients, see [Lecture →
           Lemma 3.8.26]
        matrix_t  invJT_l = Ji_l.inverse().transpose();
        // Transformed gradient
        matrix_t grad_b_l = gradEval * invJT_l;
        // Rank-1 update of element matrix
        lclMat += detJi_l * grad_b_l * grad_b_l.transpose();
    }
    // Don't forget the quadrature weight 1/6:  area
    // of the reference triangle = 1/2!
    return lclMat / 6.;
}
```

▲

End Problem 2

Problem 3: A Special Neumann problem (30 pts)

This problem deals with a scalar second-order elliptic boundary with non-standard boundary conditions. It draws on techniques from [Lecture → Chapter 2], [Lecture → Chapter 3], and [Lecture → Chapter 5].

Coding for this problem relies on EIGEN and BETL.

For a connected bounded Lipschitz polygon $\Omega \subset \mathbb{R}^2$ define

$$H^1_\partial(\Omega) := \{v \in H^1(\Omega) : \int_{\partial\Omega} v \,\mathrm{d}S = 0\} \,. \tag{0.3.1}$$

On this Sobolev space we consider the variational problem

$$u \in H^1_\partial(\Omega): \quad \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} = \int_\Omega f v \,\mathrm{d}\boldsymbol{x} \quad \forall v \in H^1_\partial(\Omega) \,, \tag{0.3.2}$$

with source function $f \in L^2(\Omega)$.

(3.a) ⚀ (3 pts) Discuss the uniqueness of solutions of (0.3.2).

SOLUTION of (3.a):

We follow the considerations from [Lecture → § 2.2.54].

For uniqueness, we have to prove that the bilinear form a stemming from the variational formulation of the problem is **positive definite**, that is

$$\mathsf{a}(v,v) > 0 \quad \forall v \in H^1_\partial(\Omega) \setminus \{v = 0 \text{ in } \Omega\}. \tag{0.3.3}$$

Using the variational formulation, we write

$$\mathsf{a}(u,v) = \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} \,,$$

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

where a is a bilinear form and ℓ is linear. To prove (0.3.3), we write

$$\mathbf{a}(v,v) = \int_\Omega \|\mathbf{grad}\, v\|^2 \,\mathrm{d}\boldsymbol{x} = 0 \quad \text{if and only if} \quad \mathbf{grad}\, v = 0 \,.$$

Then, $v = C$ on Ω, where C is a constant. Since $v \in H^1_\partial(\Omega)$, then $\int_{\partial\Omega} C \,\mathrm{d}S = 0$, that is $C = 0$, and a must be positive definite.

▲

(3.b) ⚅ (5 pts) State the second-order elliptic boundary value problem for which (0.3.2) is the weak formulation.

SOLUTION of (3.b):

The approach is similar to that elaborated in [Lecture → Ex. 2.5.23].

We first test with functions $v \in H^1_0(\Omega)$ and tease out the *PDE* by integration by parts: $-\Delta u = f$.

Then test the PDE with $v \in H^1_\partial(\Omega)$ and find after integration by parts that

$$\int_{\partial\Omega} \mathbf{grad}\, u \cdot \boldsymbol{n}\, v \,\mathrm{d}S = 0 \quad \forall v \in H^1_\partial(\Omega) \,. \tag{0.3.4}$$

Since the average of v over $\partial\Omega$ vanishes, a constant value for $\mathbf{grad}\, u \cdot \boldsymbol{n}$ satisfies this condition. If $\mathbf{grad}\, u \cdot \boldsymbol{n}$ is not constant, then (0.3.4) cannot hold. Thus we have found the *boundary condition*

$$\mathbf{grad}\, u \cdot \boldsymbol{n} \equiv \text{const} \quad \text{on} \quad \partial\Omega\,.$$

▲

Since there is no simple finite element subspace of $H^1_\partial(\Omega)$, the following **augmented variational formulation** is often used:

$$\begin{array}{c} u \in H^1(\Omega) \\ \eta \in \mathbb{R} \end{array} : \quad \begin{array}{lcll} \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} & + \int\limits_{\partial\Omega} \eta\, v \,\mathrm{d}\boldsymbol{x} & = \int\limits_\Omega f v \,\mathrm{d}\boldsymbol{x} & \forall v \in H^1(\Omega)\,, \\ \int\limits_{\partial\Omega} \mu\, u \,\mathrm{d}\boldsymbol{x} & & = 0 & \forall \mu \in \mathbb{R}\,. \end{array} \tag{0.3.5}$$

(3.c) ⚀ (3 pts) Show that the solution u of the variational problem (0.3.5) solves the variational problem (0.3.2).

SOLUTION of (3.c):

Since $\int_{\partial\Omega} \mu u \,\mathrm{d}S = 0 \quad \forall \mu \in \mathbb{R}$, the second equation implies $u \in H^1_\partial(\Omega)$. For $v \in H^1_\partial(\Omega) \subset H^1(\Omega)$, $\int\limits_{\partial\Omega} \eta\, v \,\mathrm{d}\boldsymbol{x} = 0$ and therefore the solution u of the variational problem (0.3.5) solves the variational problem (0.3.2).

▲

(3.d) ⚄ (5 pts) [depends on Sub-problem (3.b)]

How is the value for the scalar solution component $\eta \in \mathbb{R}$ in (0.3.5) related to the u-component of the solution of (0.3.5)?

SOLUTION of (3.d):

We follow the same procedure of Sub-problem (3.b) for (0.3.5). Using integration by parts, the first equation of (0.3.2) becomes

$$\eta \int_{\partial\Omega} v \,\mathrm{d}S = -\int_{\partial\Omega} \frac{\partial u}{\partial \boldsymbol{n}} v \,\mathrm{d}S\,.$$

Since $\frac{\partial u}{\partial \boldsymbol{n}} = \text{const}$ on Ω, as we have seen in Sub-problem (3.b). we obtain $\eta = -\text{const}$.

▲

We equip Ω with a triangular mesh and consider the finite element Galerkin discretization of (0.3.5) based on the piecewise linear Lagrangian finite element space $\mathcal{S}^0_1(\mathcal{M})$ and its nodal ordered basis $\{b^\ell_N\}_{\ell=1}^N$, $N := \dim \mathcal{S}^0_1(\mathcal{M})$, of "tent functions". This converts (0.3.5) into a linear system of equations of the following block form

$$\begin{bmatrix} \mathbf{A} & \mathbf{c} \\ \mathbf{c}^\top & 0 \end{bmatrix} \begin{bmatrix} \vec{\boldsymbol{\mu}} \\ \eta \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{\varphi}} \\ 0 \end{bmatrix}\,, \tag{0.3.6}$$

where $\vec{\boldsymbol{\mu}}$ is the vector of expansion coefficients of the Galerkin solution $u_N \in H^1(\Omega)$ w.r.t. the nodal basis.

(3.e) ⚀ (3 pts) Give formulas for the entries of the matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ and of the column vectors $\mathbf{c} \in \mathbb{R}^N$ and $\vec{\boldsymbol{\varphi}} \in \mathbb{R}^N$.

SOLUTION of (3.e):

Entries of matrices and vectors arise from plugging basis functions into the (bi-)linear forms of (0.3.5):

$$\mathbf{A}_{i,j} = \int_\Omega \mathbf{grad}\, b_i^N \,\mathbf{grad}\, b_j^N \,\mathrm{d}\boldsymbol{x}\,,$$
$$\mathbf{c}_i = \int_{\partial\Omega} b_i^N \,\mathrm{d}S\,,$$
$$\vec{\boldsymbol{\varphi}}_i = \int_\Omega f\, b_i^N \,\mathrm{d}\boldsymbol{x}\,,$$

$i, j = 1, \ldots, N$.

▲

We have at our disposal a BETL-based C++ function

```
template <typename FESPACE, typename FUNC_ALPHA, typename FUNC_GAMMA,
   typename FUNC_BETA>
std::vector <Eigen::Triplet<double> >
compGalerkinMatrix(const FESPACE& fe_spc,
                   FUNC_ALPHA&& alpha, FUNC_GAMMA&& gamma,
                      FUNC_BETA&& beta);
```

that, given a mesh $\mathcal{M}$, computes the $\mathcal{S}_1^0(\mathcal{M})$-based finite element Galerkin matrix *in triplet format* for the variational boundary value problem

$$u \in H^1(\Omega)\colon \int_\Omega \alpha(\boldsymbol{x})\,\mathbf{grad}\, u(\boldsymbol{x}) \cdot \mathbf{grad}\, v(\boldsymbol{x}) + \gamma(\boldsymbol{x})\, u(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \beta(\boldsymbol{x})\, u(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) = \int_\Omega f(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \quad \forall v \in H^1(\Omega)\,, \tag{0.3.7}$$

where $\alpha, \gamma : \Omega \to \mathbb{R}$, $\beta : \partial\Omega \to \mathbb{R}$ are bounded, uniformly positive coefficient functions, and $f \in L^2(\Omega)$. The argument `fe_spc` passes the FE space defined on $\mathcal{M}$, while `alpha`, `gamma`, and `beta` are functors for the scalar coefficients α, γ, and β.

Code. The function `compGalerkinMatrix` is implemented in the file

`Problem3/SolAvgBoundary.cpp`,

while the auxiliary classes **AssemblerLocalMat** and **AssemblerLocalVec** can be found in the file `utils/AssemblerLocal.hpp`.

(3.f) ⚁ (6 pts) [depends on Sub-problem (3.f)]

Based on the function `compGalerkinMatrix`, write a BETL-based C++ function ,

```
template <typename FESPACE>
Eigen::SparseMatrix<double> augmentMatrix(const FESPACE& fe_spc,
   const Eigen::VectorXd& c)
```

that augments the appropriate sparse coefficient matrix assembled by `compGalerkinMatrix` in triplet format to obtain the full Galerkin matrix for the linear system of equations (0.3.6). Argument

`c` passes vector $\mathbf{c}$ from (0.3.6), while argument `fe_test` passes an object describing the piecewise linear Lagrangian finite element space.

Template. A template for the function `augmentMatrix` is given in the file `Problem3/SolAvgBoundary.c`

HINT 1 for (3.f): Carry out matrix manipulations in triplet format and build the sparse matrix a final step. ⌟

SOLUTION of (3.f):

C++ code 0.3.8: Sub-problem (3.f): Implementation of `augmentMatrix`

```
2  template<typename FESPACE_TEST_T>
3  sparseMatrix_t augmentMatrix(const FESPACE_TEST_T& fe_test, const
       vector_t& c)
4  {
5      const auto alpha = [](const coord_t& x) { return 1.; };
6      const auto gamma = [](const coord_t& x) { return 0.; };
7      const auto beta  = [](const coord_t& x) { return 0.; };
8
9      tripletMatrix_t A = compGalerkinMatrix(fe_test, alpha, gamma,
           beta);
10
11     size_t numDofs = c.size();
12     for(int i=0; i<numDofs; ++i) {
13
14         A.push_back( triplet_t( i, numDofs, c(i) ) );
15         A.push_back( triplet_t( numDofs, i, c(i) ) );
16     }
17
18     sparseMatrix_t Ac(numDofs+1,numDofs+1);
19     Ac.setFromTriplets(A.begin(), A.end());
20     return Ac;
21 }
```

▲

(3.g) ⚄ (5 pts) [depends on Sub-problem (3.f)]

Based on the function `compGalerkinMatrix`, write another C++ function

```
template<typename FESPACE>
Eigen::VectorXd computeCVector(const FESPACE& fe_test)
```

that returns the column vector $\mathbf{c} \in \mathbb{R}^N$ as defined in (0.3.6).

For your implementation you should use *only* only the function `compGalerkinMatrix` and tools offered by EIGEN.

Template. A template for the function `computeCVector` is given in the file `Problem3/SolAvgBoundary.`

HINT 1 for (3.g): You may use the method `setFromTriplets` of **Eigen::SparseMatrix<double>** to create a matrix from the data returned by `compGalerkinMatrix`. ⌟

SOLUTION of (3.g):

Let $\mathbf{B}$ be the matrix returned by `compGalerkinMatrix` for $\alpha \equiv 0$, $\gamma \equiv 0$, and $\beta \equiv 1$. Then we have

$$\mathbf{c} = \mathbf{B}\mathbf{1}\,, \quad \mathbf{1} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^{\top} \in \mathbb{R}^N\,.$$

C++ code 0.3.9: Sub-problem (3.f): Implementation of `computeCVector`

```
template<typename FESPACE_TEST_T>
vector_t computeCVector(const FESPACE_TEST_T& fe_test)
{
    const auto alpha = [](const coord_t& x) { return 0.; };
    const auto gamma = [](const coord_t& x) { return 0.; };
    const auto beta  = [](const coord_t& x) { return 1.; };

    tripletMatrix_t B = compGalerkinMatrix(fe_test, alpha, gamma,
       beta);
     sparseMatrix_t B_(fe_test.numDofs(), fe_test.numDofs());
    B_.setFromTriplets(B.begin(), B.end());

    return B_ * vector_t::Ones(fe_test.numDofs());
}
```

▲

End Problem 3

Problem 4: Implicit-Explicit Runge-Kutta Single-Step Methods (50 pts)

This problem concerns a special class of timestepping schemes for parabolic evolution problems.

This problem relies on EIGEN and BETL.

The article [**ARS97**] introduces s-stage ($s \in \mathbb{N}$) so-called **implicit-explicit (IMEX)** partitioned Runge-Kutta single-step methods (RK-SSMs). Their definition relies on two elementary RK-SSMs given through their respective Butcher schemes, see [Lecture → § 6.1.79]:

$$\text{(A)} \quad \begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} \;\widehat{=}\; \begin{array}{c|ccccc} c_1 & a_{11} & 0 & \dots & \dots & 0 \\ c_2 & a_{21} & a_{22} & & & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ c_s & a_{s1} & \dots & & & a_{ss} \\ \hline & b_1 & b_2 & \dots & \dots & b_s \end{array}\,, \quad \begin{array}{l} \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \\ \mathfrak{A} \in \mathbb{R}^{s,s}\,, \end{array} \tag{0.4.1}$$

$$\text{(B)}\quad \begin{array}{c|c} \widehat{\mathbf{c}} & \widehat{\mathfrak{A}} \\ \hline & \widehat{\mathbf{b}}^T \end{array} \;\widehat{=}\; \begin{array}{c|cccccc} 0 & 0 & & & & & 0 \\ \widehat{c}_2 & \widehat{a}_{21} & 0 & \dots & \dots & & 0 \\ \widehat{c}_3 & \widehat{a}_{31} & \widehat{a}_{32} & 0 & & & 0 \\ \vdots & \vdots & & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & & \ddots & \ddots & \vdots \\ \widehat{c}_{s+1} & \widehat{a}_{s+1,1} & \dots & & \dots & \widehat{a}_{s+1,s} & 0 \\ \hline & \widehat{b}_1 & \widehat{b}_2 & \dots & & \dots & \widehat{b}_{s+1} \end{array} \quad , \quad \begin{array}{l} \widehat{\mathbf{c}}, \widehat{\mathbf{b}} \in \mathbb{R}^{s+1}, \\ \widehat{\mathfrak{A}} \in \mathbb{R}^{s+1,s+1} . \end{array} \tag{0.4.2}$$

The IMEX RK-SSM is applied for the numerical integration of an autonomous ordinary differential equation of the form

$$\dot{\mathbf{u}} = \mathbf{f}(\mathbf{u}) + \mathbf{g}(\mathbf{u}) , \quad \mathbf{f}, \mathbf{g} : \mathbb{R}^N \to \mathbb{R}^N , \quad N \in \mathbb{N} . \tag{0.4.3}$$

This results in a discrete evolution given by

$$\mathbf{u}_1 := \Psi^{t,t+\tau}\mathbf{u}_0 : \left\{ \begin{array}{l} \widehat{\mathbf{k}}_1 := \mathbf{f}(\mathbf{u}_0) \\ \textbf{for} \quad i = 1, \dots, s \quad \textbf{do} \\ \left[\begin{array}{l} \mathbf{u}_i := \mathbf{u}_0 + \tau \sum_{j=1}^{i} a_{i,j}\mathbf{k}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\mathbf{k}}_j , \\ \mathbf{k}_i := \mathbf{g}(\mathbf{u}_i) , \\ \widehat{\mathbf{k}}_{i+1} := \mathbf{f}(\mathbf{u}_i) , \end{array} \right. \\ \mathbf{u}_1 := \mathbf{u}_0 + \tau \sum_{j=1}^{s} b_j\mathbf{k}_j + \tau \sum_{j=1}^{s+1} \widehat{b}_j\widehat{\mathbf{k}}_j . \end{array} \right. \tag{0.4.4}$$

(4.a) ⚂ (5 pts) Algorithm (0.4.4) entails solving potentially non-linear equations to determine the increments $\mathbf{k}_i$, $i = 1, \dots, s$, which depend upon $\mathbf{k}_i = \mathbf{g}(\mathbf{u}_i)$. For the scalar case $N = 1$ formulate the Newton iteration that can be employed to approximately find $\mathbf{u}_i$, $i = 1, \dots, s$. The function $\mathbf{g}$ can be assumed to be continuously differentiable.

SOLUTION of (4.a):

The scalar Newton iteration for solving $\tilde{f}(u) = 0$ for a function $\tilde{f}(u) \in C^1(\mathbb{R})$ is

$$u^{(n+1)} = u^{(n)} - \frac{\tilde{f}(u^{(n)})}{\tilde{f}'(u^{(n)})} \quad \text{provided that the derivative } \tilde{f}'(u^{(n)}) \neq 0 .$$

In Eq. (0.4.4), we can define $\tilde{f}(u_i)$ as

$$\tilde{f}(u_i) := \underbrace{u_0 + \tau \sum_{j=1}^{i-1} a_{i,j}k_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{k}_j}_{\text{independent of } u_i} + \tau a_{i,i} g(u_i) - u_i .$$

Hence, the Newton iteration is

$$u_i^{(n+1)} = u_i^{(n)} - \frac{1}{\tau a_{i,i} g'(u_i^{(n)}) - 1} \left(u_0 + \tau \sum_{j=1}^{i-1} a_{i,j}k_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{k}_j + \tau a_{i,i} g(u_i^{(n)}) - u_i^{(n)} \right) .$$

▲

(4.b) (5 pts) State the definition of the IMEX RK-SSM from (0.4.4) in the "explicit form"

$$\mathbf{u}_1 := \mathbf{\Psi}^{t,t+\tau}\mathbf{u}_0 : \quad \begin{cases} \widehat{\mathbf{k}}_1 := \mathbf{f}(\mathbf{u}_0) \\ \textbf{for} \quad i = 1, \dots, s \quad \textbf{do} \\ \quad \left[\begin{array}{l} \mathbf{u}_i := \text{[redacted]} \, , \\ \mathbf{k}_i := \mathbf{g}(\mathbf{u}_i) \, , \\ \widehat{\mathbf{k}}_{i+1} := \mathbf{f}(\mathbf{u}_i) \, , \end{array} \right. \\ \mathbf{u}_1 := \mathbf{u}_0 + \tau \sum_{j=1}^{s} b_j \mathbf{k}_j + \tau \sum_{j=1}^{s+1} \widehat{b}_j \widehat{\mathbf{k}}_j \, , \end{cases} \tag{0.4.5}$$

when it is applied to the ODE

$$\dot{\mathbf{u}} = \mathbf{f}(\mathbf{u}) + \mathbf{g}(\mathbf{u}) \quad \text{with} \quad \mathbf{g}(\mathbf{u}) := -\mathbf{A}\mathbf{u} \, , \quad \mathbf{A} \in \mathbb{R}^{N,N} \text{ s.p.d.} \, , \tag{0.4.6}$$

and a generic continuous function $\mathbf{f} : \mathbb{R}^N \to \mathbb{R}^N$.

You need only write down the contents of the black box, but neither $\mathbf{k}_i$ nor $\mathbf{u}_i$ must occur in your expression.

SOLUTION of (4.b):

Rewrite $\mathbf{u}_i$ in (0.4.4) as

$$\mathbf{u}_i := \mathbf{u}_0 + \tau \sum_{j=1}^{i-1} a_{i,j}\mathbf{k}_j + \tau a_{i,i}\mathbf{g}(\mathbf{u}_i) + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\mathbf{k}}_j \, .$$

Then, by replacing $\mathbf{g}(\mathbf{u}) := -\mathbf{A}\mathbf{u}$ with $\mathbf{A}$ given and s.p.d., you get

$$\mathbf{u}_i = (\mathbf{I} + \tau a_{i,i}\mathbf{A})^{-1}\left(\mathbf{u}_0 + \tau \sum_{j=1}^{i-1} a_{i,j}\mathbf{k}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\mathbf{k}}_j\right) \, .$$

Note that for s.p.d. $\mathbf{A}$ also the matrix $\mathbf{I} + \tau a_{i,i}\mathbf{A}$ is s.p.d. and, hence, invertible.

▲

Now we focus on the concrete 2-stage IMEX RK-SSM defined by

$$\text{(A)} \quad \begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} \quad \widehat{=} \quad \begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-\gamma & 1-2\gamma & \gamma \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \, , \quad \gamma = \frac{3+\sqrt{3}}{6} \, , \tag{0.4.7}$$

$$\text{(B)} \quad \begin{array}{c|c} \widehat{\mathbf{c}} & \widehat{\mathfrak{A}} \\ \hline & \widehat{\mathbf{b}}^T \end{array} \quad \widehat{=} \quad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & \gamma & 0 & 0 \\ 1-\gamma & \gamma-1 & 2(1-\gamma) & 0 \\ \hline & 0 & \frac{1}{2} & \frac{1}{2} \end{array} \, . \tag{0.4.8}$$

(4.c) (10 pts) [depends on Sub-problem (4.b)]

Perform an empirical study of the order of the IMEX RK-SSM given by (0.4.7) and (0.4.8) by solving the scalar initial value problem (IVP)

$$\dot{y} = f(y) + g(y) \, , \quad f(y) := -y^2 \, , \quad g(y) = y \, , \quad y(0) = \tfrac{1}{4} \, , \tag{0.4.9}$$

with exact solution

$$y(t) = \frac{\exp(t)}{3+\exp(t)}\,,$$

on a sequence of equidistant meshes of $[0,1]$ with meshwidths $\tau := 2^{-3}, 2^{-4}, \ldots, 2^{-10}$ and monitoring the maximal error in the meshpoints:

$$\texttt{err}(\tau) := \max_{j=1,\ldots,M} |y(j\tau) - y^{(j)}|\,, \quad \tau := M^{-1}\,.$$

To that end write a C++ function

```
double IMEXError(int M)
```

that computes $\texttt{err}(M)$ for the IVP given in (0.4.9). Then tabulate the error values for $M = 2^3, 2^4, \ldots, 2^{10}$ and describe the convergence qualitatively and quantitatively.

Template. A template for the function `IMEXError` is given in the file `Problem4/IMEX.hpp`.

SOLUTION of (4.c):

C++ code 0.4.10: Sub-problem (4.c): Implementation of `IMEXError`

```
2  double IMEXError(int M)
3  {
4      double tau = 1./M;
5      double y   = 0.25;
6      double t   = 0.;
7      double err = 0.;
8
9      double gamma = 0.5 + std::sqrt(3.)/6.;
10
11     for(int i=0; i<M; ++i) {
12
13         double kh_1 = -y*y;
14         double y_1  = (y + tau*gamma*kh_1) / (1. - tau*gamma);
15         double k_1  =  y_1;
16         double kh_2 = -y_1*y_1;
17         double y_2  = (y + tau*(1-2.*gamma)*k_1 +
               tau*(gamma-1.)*kh_1 + tau*2.*(1.-gamma)*kh_2) / (1. -
               tau*gamma);
18         double k_2  =  y_2;
19         double kh_3 = -y_2*y_2;
20         y += tau*0.5*k_1 + tau*0.5*k_2 + tau*0.5*kh_2 + tau*0.5*kh_3;
21
22         t += tau;
23         double y_ex  = std::exp(t)/(3.+std::exp(t));
24         double err_t = std::abs(y - y_ex);
25         if(err_t > err) {
26             err = err_t;
27         }
28     }
```

```
29
30      return err;
31  }
```

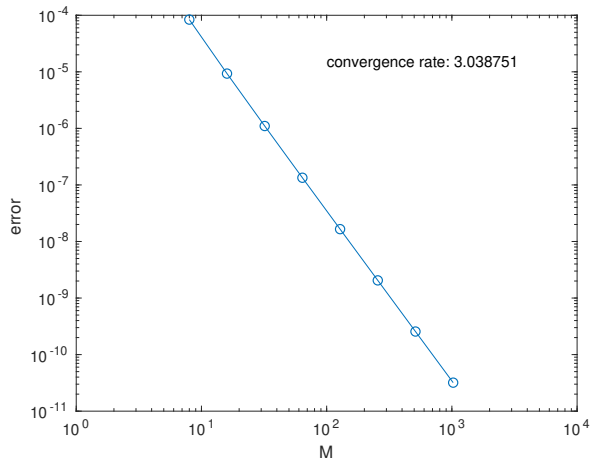

Fig. 2

Table of error:

M	error
2^3	8.38411e-05
2^4	9.30965e-06
2^5	1.10098e-06
2^6	1.33980e-07
2^7	1.65278e-08
2^8	2.05249e-09
2^9	2.55726e-10
2^{10}	3.19141e-11

◁ Log-log plot of `err` vs. M. The error curve is almost a line and from its slope we can read off a convergence rate of ~ 3.

▲

For a bounded connected polygonal $\Omega \subset \mathbb{R}^2$ we consider the following *non-linear* 2nd-order parabolic evolution problem in weak form: seek $u : [0,T] \to H^1(\Omega)$ such that

$$\begin{aligned} \int_\Omega \frac{\partial u}{\partial t} v + \mathbf{grad}\, u \cdot \mathbf{grad}\, v + \sinh(u)\, v \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} u\, v \,\mathrm{d}S = \int_{\partial\Omega} v \,\mathrm{d}S \quad \forall v \in H^1(\Omega)\,, \\ u(0) = u_0 \in H^1(\Omega)\,, \end{aligned} \tag{0.4.11}$$

where u_0 is given.

(4.d) ⚄ (5 pts) State the initial-boundary value problem in strong (PDE) form for which (0.4.11) is the associated spatial variational formulation.

SOLUTION of (4.d):

It boils down to "undoing" integration by parts using Green's formula. The linear terms are the standard terms occurring in the variational formulation of a scalar linear parabolic boundary value problem with spatial radiation/impedance boundary conditions, see [Lecture → Eq. (2.9.8)]. The non-linear term is of order zero and is not affected by integration by parts.

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u + \sinh(u) = 0 \quad \text{in} \quad \Omega\,, \\ \mathbf{grad}\, u \cdot \boldsymbol{n} + u = 1 \quad \text{on} \quad \partial\Omega\,, \\ u(0) = u_0\,. \end{aligned}$$

▲

In the spirit of the method of lines (MOL) we perform the spatial semidiscretization of (0.4.11) by means of a Galerkin finite element discretization based on the Lagrange finite element space $\mathcal{S}_1^0(\mathcal{M})$, $\mathcal{M}$ a

triangular mesh of Ω, and end up with an initial value problem for an ordinary differential equation of the form

$$\begin{aligned} \mathbf{M}\dot{\vec{\mu}} + \mathbf{A}\vec{\mu} + \mathbf{r}(\vec{\mu}) &= \vec{\varphi}\,, \\ \vec{\mu}(0) &= \vec{\mu}_0\,, \end{aligned} \tag{0.4.12}$$

with suitable matrices $\mathbf{M}, \mathbf{A} \in \mathbb{R}^{N,N}$, a non-linear function $\mathbf{r} : \mathbb{R}^N \to \mathbb{R}^N$, and $\vec{\varphi} \in \mathbb{R}^N$, $N := \dim \mathcal{S}_1^0(\mathcal{M})$. Here $t \to \vec{\mu}(t)$ is the time-dependent vector of basis expansion coefficients of the semidiscrete solution $t \to u_N(t) \in \mathcal{S}_1^0(\mathcal{M})$. The use of standard nodal basis functions $\{b_N^\ell\}_{\ell=1}^N$ is assumed.

(4.e) ⚄ (3 pts) Give formulas for the entries of the matrices $\mathbf{M}$ and $\mathbf{A}$ and of the column vector $\vec{\varphi} \in \mathbb{R}^N$ in terms of suitable integrals of expressions involving the basis functions b_N^ℓ.

SOLUTION of (4.e):

From [Lecture → Section 6.1.4] it is immediate:

$$\begin{aligned} (\mathbf{M})_{ij} &= \int_\Omega b_N^i\, b_N^j \,\mathrm{d}\boldsymbol{x}\,, \\ (\mathbf{A})_{ij} &= \int_\Omega \mathbf{grad}\, b_N^i \cdot \mathbf{grad}\, b_N^j \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} b_N^i\, b_N^j \,\mathrm{d}S\,, \\ (\vec{\varphi})_{ij} &= \int_{\partial\Omega} b_N^i \,\mathrm{d}S\,, \end{aligned}$$

with $i, j = 1, \ldots, N$.

▲

We have at our disposal a BETL-based C++ function

```
template <typename FESPACE_TEST_T, typename FUNC_ALPHA, typename
   FUNC_GAMMA, typename FUNC_BETA>
Eigen::SparseMatrix<double>
compGalerkinMatrix(const FESPACE_TEST_T& fe_test,
FUNC_ALPHA&& alpha, FUNC_GAMMA&& gamma, FUNC_BETA&& beta)
```

that, given a mesh $\mathcal{M}$, computes the $\mathcal{S}_1^0(\mathcal{M})$-based finite element Galerkin matrix for the variational boundary value problem

$$u \in H^1(\Omega)\colon \int_\Omega \alpha(\boldsymbol{x})\, \mathbf{grad}\, u(\boldsymbol{x}) \cdot \mathbf{grad}\, v(\boldsymbol{x}) + \gamma(\boldsymbol{x})\, u(\boldsymbol{x})\, v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \beta(\boldsymbol{x})\, u(\boldsymbol{x})\, v(\boldsymbol{x}) \,\mathrm{d}S(\boldsymbol{x}) = \int_\Omega f(\boldsymbol{x})\, v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} \quad \forall v \in H^1(\Omega)\,, \tag{0.4.13}$$

where $\alpha, \gamma : \Omega \to \mathbb{R}$, $\beta : \partial\Omega \to \mathbb{R}$ are bounded, uniformly positive coefficient functions, and $f \in L^2(\Omega)$. The argument `fe_test` passes the FE space defined on $\mathcal{M}$, while `alpha`, `gamma`, and `beta` are functors for coefficient functions α, γ, and β.

Code. The function `compGalerkinMatrix` is implemented in the file

`Problem4/IMEXRKSSM.cpp`,

while the auxiliary classes **AssemblerLocalMat** and **AssemblerLocalVec** can be found in the file `utils/AssemblerLocal.hpp`.

(4.f) ⚄ (8 pts) Write a BETL-based C++ function

```
template <typename FESPACE_TEST_T>
Eigen::VectorXd compNonlinearTerm(const FESPACE_TEST_T& fe_test,
   const Eigen::VectorXd& u)
```

that globally assembles $\mathbf{r}(\vec{\mu})$, where r is the non-linear mapping (0.4.12). The vector of coefficients $\vec{\mu}$ is passed as argument `u`.

To that end you have to complete the `eval` method of the C++ `struct` **AssemblerLocalFunc**,

```
template <typename BUILDER_DATA, typename ELEMENT>
inline static result_t eval( const BUILDER_DATA& data, const
   ELEMENT& el );
```

which is called by `compNonlinearTerm` and locally assembles $\mathbf{r}(\vec{\mu})$. For the evaluation of the integral involved in the definition of $\mathbf{r}$, rely on the composite trapezoidal rule over the mesh $\mathcal{M}$, see [Lecture → Eq. (3.4.51)]. The member vector `u_loc` of the `data` argument stores the coefficients of the local FE basis functions.

Templates. Templates for the function `compNonlinearTerm` and the auxiliary class **AssemblerLocalFunc** are given in the file `Problem4/AssemblerFunc.hpp`.

SOLUTION of (4.f):

C++ code 0.4.14: Sub-problem (4.f): Implementation of `AssemblerLocalFunc`

```
2       template<typename BUILDER_DATA, typename ELEMENT>
3       inline static result_t eval( const BUILDER_DATA& data, const
           ELEMENT& el )
4       {
5           ETH_ASSERT_MSG( el.refElType() ==
               eth::base::RefElType::TRIA,
6                           "This assembler only works with 2D
                               triangles." );
7
8           result_t result; result.setZero();
9
10          const auto& geom = el.geometry();
11          double elem_area = geom.volume();
12
13          for(int l=0; l<3; ++l) {
14              result(l) = elem_area/3. * std::sinh(data.u_loc(l));
15          }
16
17          return std::move(result);
18      }
```

C++ code 0.4.15: Sub-problem (4.f): Implementation of `compNonlinearTerm`

```
2       template<typename FESPACE_TEST_T>
3       Eigen::VectorXd compNonlinearTerm( const FESPACE_TEST_T& fe_test,
```

```
                                        const Eigen::VectorXd& u )
{
    struct data_t {
        Eigen::VectorXd u;
        Eigen::Vector3d u_loc;
        data_t(const Eigen::VectorXd& u):
            u(u) {}
    } data(u);

    Eigen::VectorXd vector(fe_test.numDofs()); vector.setZero();
    AssemblerLocalFunc::initialize();
    for(const auto& el : fe_test) {

        const auto id_test = fe_test.indices(el);
        for(const auto test_idx : id_test) {
            data.u_loc(test_idx.local()) =
               data.u(test_idx.global());
        }

        Eigen::VectorXd lclVec = AssemblerLocalFunc::eval(data,
           el);

        const auto id = fe_test.indices(el);
        for(const auto test_idx : id) {

            unsigned row_loc = test_idx.local();
            unsigned row_glo = test_idx.global();

            vector(row_glo) += lclVec(row_loc);
        }
    }

    return std::move(vector);
}
```

▲

(4.g) ⚄ (14 pts) [depends on Sub-problem (4.f)]

The method-of-lines ODE (0.4.12) can be rewritten in the form

$$\dot{\vec{\mu}} = \mathbf{f}(\vec{\mu}) + \mathbf{g}(\vec{\mu}) \,,$$

where $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^N$ is affine linear and collects all linear terms in (0.4.12), whereas $\mathbf{f} : \mathbb{R}^N \to \mathbb{R}^N$ represents the non-linear terms. This splitting forms the foundation for applying an IMEX RK-SSM.

Based on the functions `compGalerkinMatrix` and `compNonlinearTerm` from Sub-problem (4.f), complete the implementation of a C++ class

```
class IMEXTimestep {
public:
```

```
  template<typename FESPACE_TEST_T>
  IMEXTimestep(const FESPACE_TEST_T& fe_test);
  template<typename FESPACE_TEST_T>
  void compTimestep(const FESPACE_TEST_T& fe_test,
                    double tau, Eigen::VectorXd& y) const;
private:
  Eigen::SparseMatrix<double> M_;
  Eigen::SparseMatrix<double> A_;
  Eigen::VectorXd phi_;
  // Feel free to add more data members
};
```

The data `M_`, `A_`, and `phi_` store the time-independent Galerkin matrices and vectors denoted by $\mathbf{M}$, $\mathbf{A}$, and $\vec{\varphi}$ in (0.4.12). They should be initialized by the constructor `IMEXTimestep`. Argument `fe_test` passes an object describing the piecewise linear Lagrangian finite element space.

The method `compTimestep` realizes a single step of the IMEX RK-SSM (0.4.7)+(0.4.8) for (0.4.12). Argument `y` passes the vector of coefficients $\vec{\mu}_0$ of the previous timestep and the method should update `y` with $\vec{\mu}_1$ after one step of the IMEX RK-SSM.

Template. A template for the class **IMEXTimestep** is given in the file `Problem4/IMEX.hpp`.

HINT 1 for (4.g): You can also use a `compGalerkinMatrix` to initialize the right-hand-side vector `phi_`. ⌟

SOLUTION of (4.g):

C++ code 0.4.16: Sub-problem (4.g): Implementation of `IMEXTimestep`

```
2  class IMEXTimestep {
3
4  public:
5
6      template<typename FESPACE_TEST_T>
7      IMEXTimestep(const FESPACE_TEST_T& fe_test)
8      {
9          {
10             const auto alpha = [](const Eigen::Vector2d& x) { return
                   0.; };
11             const auto gamma = [](const Eigen::Vector2d& x) { return
                   1.; };
12             const auto beta  = [](const Eigen::Vector2d& x) { return
                   0.; };
13             M_ = compGalerkinMatrix(fe_test, alpha, gamma, beta);
14         }
15
16         {
17             const auto alpha = [](const Eigen::Vector2d& x) { return
                   1.; };
18             const auto gamma = [](const Eigen::Vector2d& x) { return
                   0.; };
19             const auto beta  = [](const Eigen::Vector2d& x) { return
```

```
                1.; };
            A_ = compGalerkinMatrix(fe_test, alpha, gamma, beta);
        }

        using localVecAssembler_t = betl2::NPDE::AssemblerLocalVec;
        using linearForm_t =
            betl2::NPDE::LoadVectorAssembler<localVecAssembler_t>;

        const auto f = [](const Eigen::Vector2d& x) { return 1.; };
        linearForm_t rhs;
        phi_ = rhs.assembleRhs(fe_test, f);
    }

    template<typename FESPACE_TEST_T>
    void compTimestep(const FESPACE_TEST_T& fe_test,
                      double tau, Eigen::VectorXd& y) const
    {
        Eigen::SimplicialLDLT<Eigen::SparseMatrix<double> >
            solver_M; solver_M.compute(M_);
        Eigen::SparseMatrix<double> MA = solver_M.solve(A_);
        Eigen::VectorXd Mphi = solver_M.solve(phi_);

        double gamma = 0.5 + std::sqrt(3.)/6.;

        Eigen::VectorXd kh_1
            =-solver_M.solve(betl2::NPDE::compNonlinearTerm(fe_test,
            y));
        Eigen::SparseMatrix<double> IMA = tau*gamma*MA;
            IMA.diagonal().array() += 1.;
        Eigen::SimplicialLDLT<Eigen::SparseMatrix<double> >
            solver_IMA; solver_IMA.compute(IMA);
        Eigen::VectorXd y_1  = solver_IMA.solve(y + tau*gamma*Mphi +
            tau*gamma*kh_1);
        Eigen::VectorXd k_1  = Mphi - MA*y_1;
        Eigen::VectorXd kh_2
            =-solver_M.solve(betl2::NPDE::compNonlinearTerm(fe_test,
            y_1));
        Eigen::VectorXd y_2  = solver_IMA.solve(y +
            tau*(1-2.*gamma)*k_1 + tau*gamma*Mphi +
            tau*(gamma-1.)*kh_1 + tau*2.*(1.-gamma)*kh_2);
        Eigen::VectorXd k_2  = Mphi - MA*y_2;
        Eigen::VectorXd kh_3
            =-solver_M.solve(betl2::NPDE::compNonlinearTerm(fe_test,
            y_2));
        y += tau*0.5*k_1 + tau*0.5*k_2 + tau*0.5*kh_2 + tau*0.5*kh_3;
    }

private:
    Eigen::SparseMatrix<double> M_;
    Eigen::SparseMatrix<double> A_;
```

```
56         Eigen::VectorXd phi_;
57 };
```

▲

End Problem 4