**EPFL**
Mathematics section
**Prof Annalisa Buffa**

# Numerical Approximation of Partial Differential Equations

**MATH-451 EXAM** **23.06.2022** **9h15-12h15**

**Name:** ....................... **Forename:**....................... **Sciper:** ........................

**EXAM RULES:**

- **CAMIPRO card is mandatory and will be checked.**
- **The exam is recorded only after the student has signed.**
- **Do not detach any page. The colored sheets are draft papers and do not have to be handed in.**
- **Write with blue or black ink. No other colors are allowed.**
- **Mobile phones and other electronic devices must be turned off and in the bags.**
- **Please copy all MATLAB code into the exam. Results without code will not be graded.**
- **Please write one-sided.**
- **Justify all your answers. The clearness of the answers will be evaluated as well.**

☐ I read and understood the above rules. Signature : ....................................

| Exercises | Points | Grades |
|---|---|---|
| 1 | 8 | |
| 2 | 8 | |
| 3 | 10 | |
| 4 | 10 | |
| TOTAL | 36 | |

## Problem 1 (8 points)

Let $\mathcal{T}_h$ be a regular affine triangulation of a convex, polygonal domain $\Omega \subset \mathbb{R}^2$ and let $\hat{K} = \{(\hat{x}, \hat{y}),\ \hat{x} \geq 0, \hat{y} \geq 0, \hat{x} + \hat{y} \leq 1\}$ be the reference triangle.

(a) Define a set of degrees of freedom for $\mathbb{P}_2(\hat{K})$ and prove their unisolvence.

(b) Construct a corresponding Lagrangian basis.

(c) Construct a basis for

$$V_h = \left\{u_h \in C^0(\bar{\Omega}) : \ u_h|_K \in \mathbb{P}_2(K)\ \ \forall K \in \mathcal{T}_h\right\}. \tag{1}$$

(d) Let $I_h : C^0(\bar{\Omega}) \to V_h$ be the interpolation operator. What do you know about the error $\|u - I_h(u)\|_{L^2(\Omega)}$ for $u \in H^2(\Omega)$?

(e) Prove that $I_h\left(I_h(u)\right) = I_h(u)$, i.e., $I_h$ is a projector.

## Exercise 2 (8 points)

We are considering the general stationary advection-reaction-diffusion problem with homogeneous Dirichlet boundary conditions on an open polygonal domain $\Omega \subset \mathbb{R}^2$, divergence-free advection field $\mathbf{b} : \bar{\Omega} \to \mathbb{R}^2$ and reaction term $r \in \mathbb{R}^{\geq 0}$:

$$\begin{cases} -\Delta u + \mathbf{b}(\mathbf{x}) \cdot \nabla u + ru = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{2}$$

We are considering a regular affine triangulation $\mathcal{T}_h$ of $\Omega$ and the finite element space $X_h^1$ of continuous piecewise linear functions on $\mathcal{T}_h$ with canonical nodal basis $\{v_1, \dots, v_N\}$ satisfying $v_i(\mathbf{x}_j) = \delta_{ij}$, where $\mathbf{x}_j$ denotes the $j$-th vertex of $\mathcal{T}_h$.

Disregarding the Dirichlet boundary condition for now, the finite-element discretisation of the problem is associated with three matrices:

1. The mass matrix $M \in \mathbb{R}^{N \times N}$ with entries $M_{i,j} = \int_\Omega v_i v_j \mathrm{d}\Omega$;
2. The stiffness matrix $A \in \mathbb{R}^{N \times N}$ with entries $A_{i,j} = \int_\Omega \nabla v_i \cdot \nabla v_j \mathrm{d}\Omega$;
3. The advection matrix $B \in \mathbb{R}^{N \times N}$ with entries $B_{i,j} = \int_\Omega v_i \left(\mathbf{b}(\mathbf{x}) \cdot \nabla v_j\right) \mathrm{d}\Omega$.

The point of this exercise is designing Matlab functions for the *local* contributions to the mass, stiffness and advection matrices. The functions are of the form:

```
Mloc = LocalMass(BK, bk, xhat, w, shapeF, gradshapeF, bhandle);
Aloc = LocalStiff(BK, bk, xhat, w, shapeF, gradshapeF, bhandle);
Bloc = LocalAdv(BK, bk, xhat, w, shapeF, gradshapeF, bhandle);
```

and take as input

- `BK` of shape `[2 2]` and `bk` of shape `[2 1]` that map the reference element $\hat{K} = \{\hat{x} \geq 0, \hat{y} \geq 0, \hat{x} + \hat{y} \leq 1\}$ onto the current element $K \in \mathcal{T}_h$ via

$$F_K(\hat{\mathbf{x}}) = B_K \hat{\mathbf{x}} + b_k, \quad \text{for } \hat{\mathbf{x}} \in \hat{K}$$

- `xhat`: a $7 \times 2$ array of quadrature points in $\hat{K}$ corresponding to a 7-point Gauss quadrature of order 6 on the reference element;
- `w`: a $7 \times 1$ array of quadrature weights corresponding to the same 7-point Gauss quadrature of order 6 on the reference element;
- `shapeF`: a $7 \times 3$ array containing the evaluations of the set of the locally defined basis functions $\{\hat{\phi}_1, \hat{\phi}_2, \hat{\phi}_3\}$ in the quadrature points `xhat`.
- `gradshapeF`: a $7 \times 6$ array containing the evaluations of $\hat{\nabla}\hat{\phi}_i$ in `xhat` in columns of two.
- `bhandle`: a function representing $\mathbf{b}(\mathbf{x})$ by mapping any $N \times 2$ array of values $\mathbf{x} \in \Omega$ onto an $N \times 2$ array of function evaluations of $\mathbf{b}$ in those values.

Answer the following points, **keeping in mind that some functions may not use all of their inputs and that your implementation need not be efficient**:

(a) Complete the template for the local contribution to the mass matrix by filling the blanks of Listing 1.

Listing 1: Template for the implementation of the local mass matrix

```
function Mloc = LocalMass(BK, bk, xhat, w, shapeF, gradshapeF, bhandle)
% Local mass matrix for reaction in 2D using
% linear Lagrangian finite elements (hat functions).
% The integrals are computed using a 7-point Gauss quadrature rule.

% compute BK^-1 and det(BK). These values may or may not be needed
invBK = inv(BK);
detBK = det(BK);

Mloc = zeros(3,3);
```

```
% Compute element mass matrix using two for loops
for i = 1:3
    for j = i:3

        % FILL IN THE BLANK LINE(S)
        Mloc(i,j) =


    end
end

Mloc(2, 1) = Mloc(1, 2);
Mloc(3, 1) = Mloc(1, 3);
Mloc(3, 2) = Mloc(2, 3);
end
```

(b) Complete the template for the local contribution to the stiffness matrix by filling the blanks of Listing 2

Listing 2: Template for the implementation of the local stiffness matrix

```
function Aloc = LocalStiff(BK, bk, xhat, w, shapeF, gradshapeF, bhandle)
% Local stiffness matrix 2D using
% linear Lagrangian finite elements (hat functions).
% The integrals are computed using a 7-point Gauss quadrature rule.

% compute BK^-1 and det(BK). These values may or may not be needed
invBK = inv(BK);
detBK = det(BK);

% Create an empty array of size(gradshapeF). This array represents the
% push-forward of gradshapeF onto the element.
gradshapeF_global = zeros(size(gradshapeF));

% Fill the array with the correct values
for j = 1:3

    % FILL IN THE BLANK LINE(S)
    gradshapeF_global(:, 2*j - 1: 2*j) =


end

Aloc = zeros(3,3);


for i = 1:3
    for j = i:3

        % FILL IN THE BLANK LINE(S)
        Aloc(i,j) =


    end
end

Aloc(2,1) = Aloc(1,2);
Aloc(3,1) = Aloc(1,3);
Aloc(3,2) = Aloc(2,3);
end
```

(c) Complete the template for the local contribution to the advection matrix by filling the blanks of Listing 3

Listing 3: Template for the implementation of the local advection matrix

```
function Bloc = LocalAdv(BK, bk, xhat, w, shapeF, gradshapeF, bhandle)
% Local advection matrix in 2D using
% linear Lagrangian finite elements (hat functions).
% The integrals are computed using a 7-point Gauss quadrature rule.

% compute BK^-1 and det(BK). These values may or may not be needed
invBK = inv(BK);
detBK = det(BK);

% Create an empty array of size(gradshapeF). This array represents the
% push-forward of gradshapeF onto the element.
gradshapeF_global = zeros(size(gradshapeF));
```

```
% Fill the array with the correct values
for j = 1:3

    % FILL IN THE BLANK LINE(S)
    gradshapeF_global(:, 2*j - 1: 2*j) =


end

% create an array of global values x by mapping xhat from the reference
% element onto the current element K
x =

% compute the values of b(x) from x computed above
b =

Bloc = zeros(3,3);

% Compute element advection matrix using two for loops
for i = 1:3
    for j = 1:3

        % FILL IN THE BLANK LINE(S)
        Bloc(i, j) =


    end
end

end
```

Suppose our code is capable of assembling $M, A$ and $B$ using above routines as well as the right-hand side vector $\mathbf{f} \in \mathbb{R}^N$ (again, disregarding the boundary conditions). We define the matrix

$$S = A + B + rM \tag{3}$$

and the index-set $\mathcal{I}_{\text{inner}}$ of trace-free basis functions in $\Omega$, i.e.,

$$\mathcal{I}_{\text{inner}} = \left\{ i \in \{1, \ldots, N\} \mid v_i \in X_h^1 \cap H_0^1(\Omega) \right\}. \tag{4}$$

Consider the function

```
uinner = SolveWithHomogeneousDirichlet(S, f, Iinner)
```

taking as input

- `S`: the **full** matrix $S \in \mathbb{R}^{N \times N}$ disregarding any boundary conditions, as defined in (3).
- `f`: the full right-hand side vector $\mathbf{f} \in \mathbb{R}^n$, again disregarding the BC.
- `Iinner` a $N_0 \times 1$ vector containing the $i \in \mathcal{I}_{\text{inner}}$ in ascending order. Here $N_0$ denotes the cardinality of $\mathcal{I}_{\text{inner}}$.

(d) Complete the template of the routine that solves for a $N_0 \times 1$ vector `uinner` containing the approximate solution's weights corresponding to the $v_i$, $i \in \mathcal{I}_{\text{inner}}$ in ascending order. For this, fill in the blanks of Listing 4

Listing 4: Template for solving for the vector of inner degrees of freedom

```
function uinner = SolveWithHomogeneousDirichlet(S, f, Iinner)
% Solve for and return the solution's weights corresponding to the inner
% degrees of freedom.

% You may use the following lines to define auxiliary quantities for
% their use later on.






% FILL IN THE BLANK LINE
uinner =



end
```

## Exercise 3 (10 points)

We are considering the heat equation with time-independent source term $f(x) \in C^2([0,1])$:

$$\begin{cases} u_t = Au_{xx} + f(x) & \text{for} \quad x \in (0,1), \ \ t \in (0,T] \\ u(0,t) = u(1,t) = 0 & \text{for} \quad t \in (0,T] \\ u(x,0) = u_0(x) & \text{for} \quad x \in [0,1] \end{cases} \tag{5}$$

and constant diffusivity $A > 0$.
We introduce a uniform grid with spacing $h = 1/N$, $x_j = jh, j = 0 \dots N$ and $\Delta t = T/M$, where $M \in \mathbb{Z}$ is the total number of time-steps.
We discretise this equation in the usual way, using a forward Euler scheme for the time derivative and a central scheme for the Laplacian. This leads to the discrete scheme

$$\begin{cases} \frac{U_j^{m+1} - U_j^m}{\Delta t} = \frac{A}{h^2}\left(U_{j-1}^m - 2U_j^m + U_{j+1}^m\right) + F_j, \quad j = 1, \dots, N-1 \\ U_0^m = U_N^m = 0 \quad \forall m \end{cases} \tag{6}$$

where the first iterate satisfies $U_j^0 = u_0(x_j)$ and $F_j = f(x_j)$. In what follows, we define $\mathbb{U} = \left(\mathbf{U}^0, \mathbf{U}^1, \dots, \mathbf{U}^M\right)$ as the column matrix of discrete time iterates $\mathbf{U}^m = (U_0^m, \dots, U_N^m)^T$, $\forall m = 0, \dots, M$ and $\kappa = \frac{A\Delta t}{h^2}$. We write the system compactly as $\mathcal{L}\mathbb{U} = \mathcal{F}$.

Answer the following questions:

(a) Give the linear operator $\mathcal{L}$ and right-hand side $\mathcal{F}$ corresponding to (6), where $M \in \mathbb{Z}$ denotes the total number of discrete time steps we perform. Show that $\mathcal{L}$ is inverse monotone for $\kappa \le \frac{1}{2}$ and derive a bound on $\max_{j,m} |\mathbb{U}_{jm}^M|$ in terms of $\|f\|_{C([0,1])}$ using a suitable comparison function. You may assume that $u_0 = 0$.

(b) The recurrence from (6) can be written in the matrix-form $\tilde{\mathbf{U}}^{m+1} = S\tilde{\mathbf{U}}^m + \Delta t\tilde{\mathbf{F}}$, where $\tilde{\mathbf{U}}^m$ is the vector of **inner** values, i.e.,

$$\mathbf{U}^m = \begin{pmatrix} U_0^m \\ \tilde{\mathbf{U}}^m \\ U_N^m \end{pmatrix}, \quad \text{while} \quad \tilde{\mathbf{F}} = \begin{pmatrix} F_1 \\ \vdots \\ F_{N-1} \end{pmatrix}.$$

Given the matrix

$$K = \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & & \\ 0 & & & \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N-1)}$$

provide the matrix $S \in \mathbb{R}^{(N-1)\times(N-1)}$ in closed form. Moreover, knowing that the eigenvalues of $K$ are given by

$$\lambda_j(K) = 2\left(1 - \cos\left(\frac{j\pi}{N}\right)\right), \quad j = 1, \dots, N-1,$$

provide a formula for the eigenvalues $\lambda_j(S)$, $j = 1, \dots, N-1$ of $S$.

(c) Give an expression for $\tilde{\mathbf{U}}^M$ in terms of $\tilde{\mathbf{U}}^0$, $\tilde{\mathbf{F}}$ and S

(d) Derive a condition on $\kappa$ such that the spectral radius $\rho(S) < 1$ for all $N$. Is this result to be expected ? Starting from your result in (c), explain what happens if the condition is violated.

(e) Consider problem (5) again. Propose a discretisation via piecewise polynomials of degree 1 and a forward-Euler discretisation in time.

- Comment on differences and similarities with (6);
- Discuss the stability of the FEM scheme.

## Exercise 4 (10 points)

We are interested in approximating the solution of the following nonlinear PDE:

$$\begin{cases} \Delta u + R(u) = 0 & \text{in } \Omega = (0,1)^2 \\ u|_{\partial\Omega} = 0 & \end{cases}, \tag{7}$$

where $R(u) = ru(1-u)$, with $r \in \mathbb{R}^{>0}$ is a nonlinear reaction term. In what follows, we disregard the trivial solution $u = 0$ and look for solutions $u \neq 0$.
Rather than seeking the solution directly, we look for the steady-state solution of the nonlinear reaction-diffusion problem

$$\begin{cases} u_t = \Delta u + R(u) & \text{in } \Omega = (0,1)^2, t > 0 \\ u(x,y,0) = u_0(x,y) & \\ u|_{\partial\Omega} = 0 & \forall t \geq 0 \end{cases}, \tag{8}$$

For this, we introduce a computational grid

$$\Omega_h = \{(ih, jh), i, j = 0, \dots, N\}, \;\; h = \frac{1}{N}$$

with boundary

$$\partial\Omega_h = \{(ih, jh), i \in \{0, N\} \text{ or } j \in \{0, N\}\}$$

and corresponding index-sets

$$\mathcal{I}_{\text{inner}} = \{(i,j) \mid i, j = 1, \dots, N-1\} \quad \text{and} \quad \mathcal{I}_{\text{boundary}} = \{(i,j) \mid i \in \{0, N\} \text{ or } j \in \{0, N\}\}$$

of inner and boundary vertices, resepectively.

In this problem, we seek to approximate the solution of (8) by using a mixed implicit-explicit quadrature in time that treats the diffusion implicitly, while the reaction is treated explicity, i.e,

$$\frac{u^{m+1} - u^m}{\Delta t} \approx \Delta u^{m+1} + R(u^m), \tag{9}$$

where $u^m = u(t = m\Delta t)$, for some time-step $\Delta t > 0$.

We discretise in space using the usual second-order accurate central finite-difference scheme. For this we introduce $U_{i,j}^m$ as the approximate solution at time-instance $t = m\Delta t$ and vertex $(ih, jh)$, taking as an initialisation $U_{i,j}^0 = u_0(ih, jh)$.

(a) Write down the recursion associated with the numerical scheme as described above. Here, make a distinction between the indices $(i,j) \in \mathcal{I}_{\text{inner}}$ and $(i,j) \in \mathcal{I}_{\text{boundary}}$ while including the initialisation and boundary conditions.

We introduce the vector $\mathbf{U}^m$ containing the $U_{i,j}^m$ corresponding to the **inner** indices $(i,j) \in \mathcal{I}_{\text{inner}}$ in the usual lexicographic ordering.

For the vector of inner degrees of freedom, the scheme can be written in matrix form

$$\left(I - \frac{\Delta t}{h^2} A\right) \mathbf{U}^{m+1} = \mathbf{U}^m + \Delta t r \mathbf{U}^m * (\mathbf{1} - \mathbf{U}^m),$$

where the operator $*$ denotes entry-wise multiplication and $\mathbf{1}$ is a vector of ones of appropriate size.

(b) Explain how you would implement the matrix $A$ in Matlab using matrix tensor products.
**HINT:** Thanks to the elimination of the boundary vertices, $A$ can be constructed from univariate matrices of size $(N-1) \times (N-1)$.

(c) What happens if we take $u_0(x,y) = 0$ ?

(d) Implement the scheme for $r = 100$, $N = 50$ and $dt = h^2$. Use the function $u_0(x,y) = x(1-x)y(1-y)$ to initialise the scheme. Use sparse matrices and Matlab's backslash command to invert them.
Terminate the scheme once $\|\frac{1}{h^2} A\mathbf{U}^m + r\mathbf{U}^m * (\mathbf{1} - \mathbf{U}^m)\|_\infty < 10^{-6}$ and sketch the plot of the solution.

**COPY ALL YOUR MATLAB CODE INTO THE EXAM !!**

**HINT 1:** if you could not answer question 1, you may use

```
E = ones((N-1)^2, 1);
Em1 = repmat([ones(N - 2, 1); 0], N-1, 1);
E1 = repmat([0; ones(N - 2, 1)], N-1, 1);
A = spdiags([E Em1 -4*E E1 E], [-(N-1) -1 0 1 (N-1)], (N-1)^2, (N-1)^2);
```

**HINT 2:** $\mathbf{U}^0$ can be constructed using

```
x = linspace(0, 1, N+1);
xinner = x(2:end - 1);
u0 = xinner.*(1 - xinner);
U0 = kron(u0, u0)';
```

Adhering to the lexicographic ordering, you can plot using

```
[X, Y] = meshgrid(xinner, xinner);
surf(X, Y, reshape(U, [N-1 N-1]))
```

You need not plot the points located on the boundary.