Numerical Methods for Differential Equations

Exercise on the Finite Element Method

Numerical solution of the heat equation

Contents

1 Problem statement

Let us consider a domain $\Omega \subset \mathbb{R}^2$, with $\Gamma = \partial\Omega$ its boundary. This boundary is partitioned into two non-overlapping regions $\Gamma_D = \Gamma_{D,1} \cup \Gamma_{D,2}$ and Γ_N, such that $\Gamma_D \cap \Gamma_N = \varnothing$ and $\Gamma_D \cup \Gamma_N = \Gamma$, where Dirichlet and Neumann boundary conditions, respectively, are applied. Meaning with $\mathbf{x}$ the position vector, t the time variable, $\mathbf{n}$ the outer normal unit vector and $\mathbb{T} = (0, t_{\max}]$ the time domain of interest, the strong form of the initial boundary value problem can be stated as:

$$\begin{gathered}\text{Given } f(\mathbf{x},t) : \Omega \times \mathbb{T} \to \mathbb{R},\ d_1(\mathbf{x},t) : \Gamma_{D,1} \times \mathbb{T} \to \mathbb{R},\ d_2(\mathbf{x},t) : \Gamma_{D,2} \times \mathbb{T} \to \mathbb{R},\\ q(\mathbf{x},t) : \Gamma_N \times \mathbb{T} \to \mathbb{R} \text{ and } u_0(\mathbf{x}) : \overline{\Omega} \to \mathbb{R},\ \text{find } u(\mathbf{x},t) \in \overline{\Omega} \times \mathbb{T} \text{ such that:}\\ \begin{cases} \dfrac{\partial u(\mathbf{x},t)}{\partial t} - \dfrac{\partial^2 u(\mathbf{x},t)}{\partial x^2} - \dfrac{\partial^2 u(\mathbf{x},t)}{\partial y^2} = f(\mathbf{x},t) & \text{in } \Omega \times \mathbb{T}\\ u(\mathbf{x},t) = d_1(\mathbf{x},t) & \text{on } \Gamma_{D,1} \times \mathbb{T}\\ u(\mathbf{x},t) = d_2(\mathbf{x},t) & \text{on } \Gamma_{D,2} \times \mathbb{T}\\ \dfrac{\partial u(\mathbf{x},t)}{\partial \mathbf{n}} = q(\mathbf{x},t) & \text{on } \Gamma_N \times \mathbb{T}\\ u(\mathbf{x},0) = u_0(\mathbf{x}) & \text{in } \overline{\Omega} \end{cases}\end{gathered} \tag{1}$$

The domain Ω has a horseshoe shape as reported in Fig. 1. In the problem (1), the source term is such that the problem is homogeneous, i.e., $f(\mathbf{x},t) = 0$. The Dirichlet functions are $d_1(\mathbf{x},t) = 0$ and $d_2(\mathbf{x},t)$ linear up to 1, reached at $t = t_{\max}/2$, then equal to 1, as reported in Fig. 1. The final time is $t_{\max} = 10$. The Neumann function and the initial solution are $q(\mathbf{x},t) = 0$ and $u_0(\mathbf{x}) = 0$, respectively.

Problem (1) governs, for instance, the evolution of the temperature $u(\mathbf{x},t)$ at point $\mathbf{x}$ and time t of a metal body, where $d_1(t)$ and $d_2(t)$ are known time functions of the temperature at the Dirichlet extremes $\Gamma_{D,1}$ and $\Gamma_{D,2}$. The initial temperature is $u_0(\mathbf{x})$. For this reason, the governing equation is called *heat equation*.

The purpose of this exercise is to solve the initial boundary value problem (1) with the Finite Element Method (FEM) using triangular elements and linear basis functions. We are interested in both the transient solution in the entire time domain and the steady-state solution, i.e., the solution for $t \to \infty$, when no time derivatives are present anymore and the problem changes its nature, from a **parabolic** second order PDE to an **elliptic** one.

2 FEM solution

The initial boundary value problem (1) can be solved numerically using the FEM. The variational formulation leads to the Galerkin method.

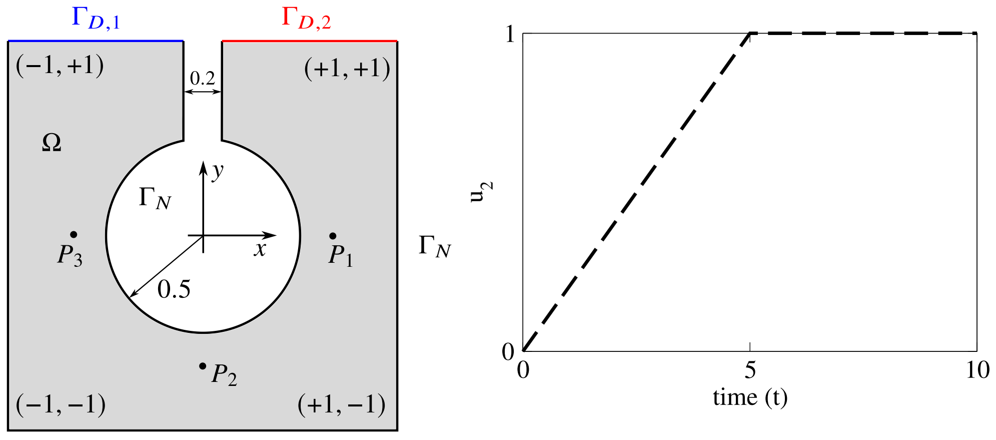

Figure 1: Domain and $u_2(t)$ time behavior.

The approximate solution $u_h(\mathbf{x}, t)$ can be written as:

$$u_h(\mathbf{x}, t) = \sum_{i=1}^{n} u_i(t)\varphi_i(\mathbf{x}) \tag{2}$$

where $\varphi_i(\mathbf{x})$, $i = 1, \ldots, n$ are suitable basis functions associated to the chosen discretization of the spatial domain and $u_i(t)$ are the time-dependent nodal values on n points in the domain Ω. These are the unknowns of our problem.

2.1 Spatial solution

Substituting u_h in (1), we can write the residual

$$L(u_h) = \frac{\partial u_h}{\partial t} - \frac{\partial^2 u_h}{\partial x^2} - \frac{\partial^2 u_h}{\partial y^2} - f \tag{3}$$

and then impose the orthogonality to the n basis functions $\varphi_i(x, y)$, obtaining:

$$\int_\Omega L(u_h)\,\varphi_i(\mathbf{x})d\Omega = 0 \quad i = 1, \ldots, n \tag{4}$$

Using the divergence (Gauss) theorem, Eq. (4) becomes:

$$\int_\Omega \left(\frac{\partial u_h}{\partial t}\varphi_i\right) d\Omega + \int_\Omega \left(\frac{\partial u_h}{\partial x}\frac{\partial \varphi_i}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial \varphi_i}{\partial y}\right) d\Omega - \int_\Gamma \left(\frac{\partial u_h}{\partial x}n_x + \frac{\partial u_h}{\partial y}n_y\right)\varphi_i d\gamma - \int_\Omega f\varphi_i d\Omega = 0 \tag{5}$$

Substituting the definition (2) into (5), we have:

$$\int_\Omega \left[\sum_{j=1}^{n} \left(\frac{\partial u_j}{\partial t} \varphi_j \varphi_i \right) \right] d\Omega + \int_\Omega \left[\sum_{j=1}^{n} \left(\frac{\partial \varphi_j}{\partial x} \frac{\partial \varphi_i}{\partial x} + \frac{\partial \varphi_j}{\partial y} \frac{\partial \varphi_i}{\partial y} \right) u_j \right] d\Omega - \int_\Omega f \varphi_i d\Omega - \int_{\Gamma_N} q \varphi_i d\gamma = 0 \tag{6}$$

where q represent the known normal derivative $\partial u/\partial \mathbf{n}$, i.e., the flux of u, on the Neumann boundary. In our problem, given that $q(\mathbf{x}, t) = 0$, this contribution vanishes. Also the term related to the source function f vanished, being $f(\mathbf{x}, t) = 0$.

Eq. (6) are to be intended for any i in $1, \dots, n$, thus they constitute a system of ordinary differential equations (ODEs). With the n unknown functions in time $u_1(t), u_2(t), \dots, u_n(t)$ collected in vector $\mathbf{u}(t)$, the problem can be written as:

$$M \frac{d\mathbf{u}(t)}{dt} + H\mathbf{u}(t) = \mathbf{0}. \tag{7}$$

The chosen basis functions are piecewise linear polynomials with local support. Thus, the generic coefficient h_{ij} of the stiffness matrix, referring to the i-th equation, is the result of the assembly of local contributions $h_{ij}^{(e)}$ from all the finite elements sharing node i

$$h_{ij} = \sum_e h_{ij}^{(e)} = \sum_e \int_{\Omega^{(e)}} \left(\frac{\partial \varphi_j^{(e)}}{\partial x} \frac{\partial \varphi_i^{(e)}}{\partial x} + \frac{\partial \varphi_j^{(e)}}{\partial y} \frac{\partial \varphi_i^{(e)}}{\partial y} \right) d\Omega, \tag{8}$$

while the mass matrix entry m_{ij} reads:

$$m_{ij} = \sum_e m_{ij}^{(e)} = \sum_e \int_{\Omega^{(e)}} \varphi_j^{(e)} \varphi_i^{(e)} d\Omega. \tag{9}$$

From the definition of the single entries $h_{i,j}$ and $m_{i,j}$ in Eqs. (8) and (9), we can observe that both H and M are sparse symmetric positive definite matrices.

2.2 Time integral

The system of ODEs is then solved using a θ-method, that discretizes the temporal derivative by a simple difference quotient and replaces the other terms with a linear combination of the value at time t_k and at time t_{k+1}, depending on the real parameter θ $(0 \leq \theta \leq 1)$. The method applied to Eq. (7) reads:

$$M \frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} + H \left[\theta \mathbf{u}^{k+1} + (1 - \theta) \mathbf{u}^k \right] = 0, \tag{10}$$

where the real positive parameter $\Delta t = t_{k+1} - t_k$, $k = 0, 1, \dots$, denotes the discretization step and the superscript k indicates that the quantity under consideration refers to the time t_k. To advance in time, one has to solve the linear system:

$$\left(\frac{M}{\Delta t} + \theta H\right) \mathbf{u}^{k+1} = \left(\frac{M}{\Delta t} - (1-\theta)H\right) \mathbf{u}^{k}. \tag{11}$$

To obtain the unknown vector at any time step, the linear system in Eq. (11) must be solved. In a more compact, we have:

$$K_1 \mathbf{u}^{k+1} = K_2 \mathbf{u}^{k} \tag{12}$$

where:

$$\begin{aligned} K_1 &= \frac{M}{\Delta t} + \theta H \\ K_2 &= \frac{M}{\Delta t} - (1-\theta)H \end{aligned}$$

Being M and H symmetric and positive definite matrices, $\theta > 0$ and $\Delta t > 0$, K_1 is a symmetric and positive definite matrix too. In this application, we use only $\theta = 0.5$, i.e., Crank-Nicolson time marching scheme, with constant $\Delta t = 0.02$. A detailed study on time convergence is beyond the scope of this homework.

3 The FEM code

The initial boundary value problem (1) can be solved thanks to a code. Given the high level of complexity, this code has to exploit a well-defined structure, with different functions for different tasks, e.g., see the flowchart in Fig. 2.

3.1 Input data

Let us consider a non overlapping triangulation of Ω with triangular finite elements. This triangulation is defined with a table providing the topology for each triangle, i.e., the list of nodes in anticlockwise order. For instance:

```
 66  103  104
107   67  108
 72   86  129
  ⋮    ⋮    ⋮
```

means that the first element is identified by the vertices $(66, 103, 104)$, the second by $(107, 67, 108)$, the third by $(72, 86, 129)$ and so on. Finally, the position of each node is determined by two coordi-

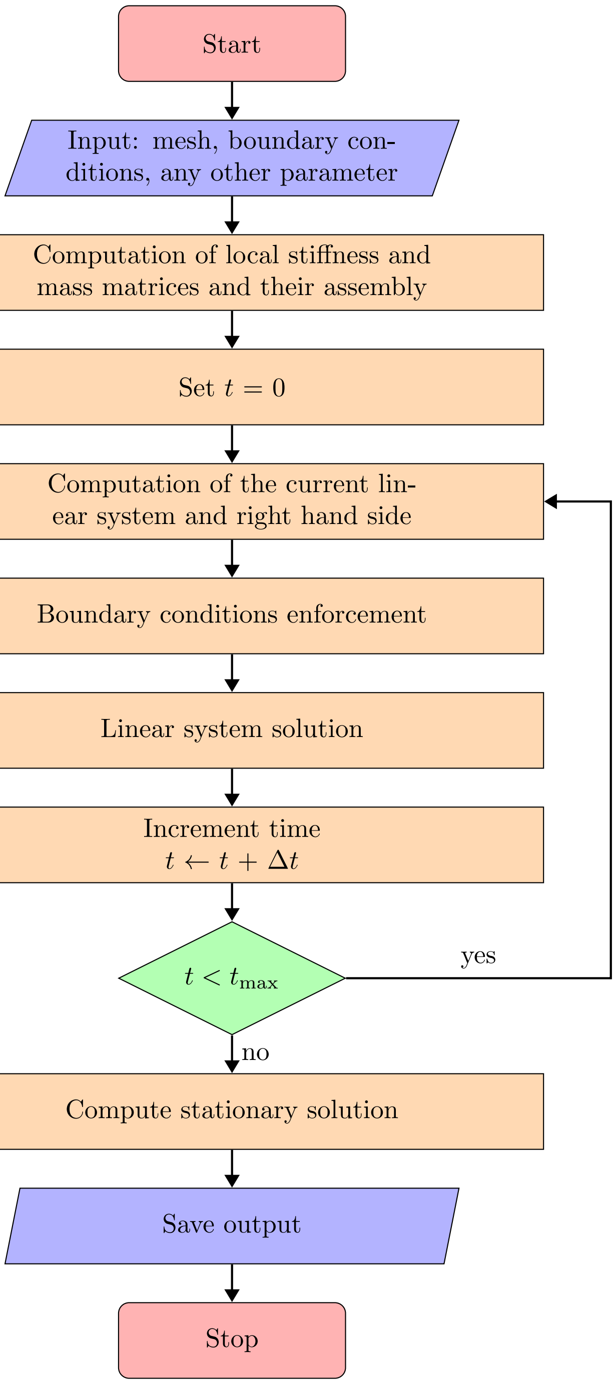
Start
Input: mesh, boundary conditions, any other parameter
Computation of local stiffness and mass matrices and their assembly
Set $t = 0$
Computation of the current linear system and right hand side
Boundary conditions enforcement
Linear system solution
Increment time $t \leftarrow t + \Delta t$
$t < t_{\max}$
yes
no
Compute stationary solution
Save output
Stop

Figure 2: Flowchart of the code with the main steps.

nates, e.g.:

```
-1.0  -1.0
-1.0   1.0
 1.0   1.0
   ⋮     ⋮
```

means that the first node has coordinates $(-1,-1)$, the second $(-1,1)$, the third $(1,1)$, and so on.

Another required piece of information are the boundary conditions. Generally speaking, both the set of nodes located on the Dirichlet and Neumann boundaries have to be provided. In the specific application, we already know that $\Gamma_N = \varnothing$, thus only the list of Dirichlet nodes is given. For instance

```
 2  0.0
 5  0.0
29  0.0
 ⋮    ⋮
 3  1.0
 6  1.0
 ⋮    ⋮
```

means that 2, 5, 29, 3, 6, etc., are the nodes where the value of the solution, i.e., the Dirichlet boundary condition, have to be enforced. In the second column the value at time $t_{\max}$ of the boundary condition is reported, i.e., $d_1(t_{\max})$ and $d_2(t_{\max})$. Since d_1 is constant, there is no need to update its value, while the current value of $d_2(t)$ has to be analytically computed according to the behavior reported in Fig. 1, i.e.,

$$d_2(t) = \begin{cases} \dfrac{2t}{t_{\max}}\, d_2(t_{\max}) & \text{if } t \leq \dfrac{t_{\max}}{2} \\ d_2(t_{\max}) & \text{elsewhere} \end{cases} \tag{13}$$

Three ASCII files are used to represent a computational grid, characterized by different extensions:

1. `topol`: the file containing the triangle topology;
2. `coord`: the file containing the node coordinates;
3. `bound`: the file containing the set of Dirichlet nodes.

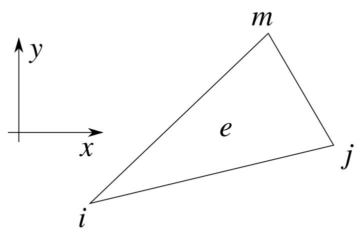

Figure 3: Triangular element with nodes i, j, m.

3.2 Computation of local stiffness and mass matrices and their assembly

The domain Ω is subdivided into triangular elements e. Let us name $u_h^{(e)}$ the solution restricted to the element e:

$$u_h^{(e)}(x, y) = \sum_{k=1}^{3} u_k(t)\varphi_k^{(e)}(x, y) \tag{14}$$

Referring to the generic finite element of Fig. 3, we have that the solution is linear with respect to the nodal unknowns u_i, u_j, and u_m. Those values are the solution at the corresponding nodes, since $\varphi_k(x_k, y_k) = 1$. To respect this constraint and be null on the other nodes, the basis functions are computed as:

$$\begin{aligned} \varphi_i^{(e)}(x, y) &= \frac{a_i + b_i x + c_i y}{2\Delta} \\ \varphi_j^{(e)}(x, y) &= \frac{a_j + b_j x + c_j y}{2\Delta} \\ \varphi_m^{(e)}(x, y) &= \frac{a_m + b_m x + c_m y}{2\Delta} \end{aligned} \tag{15}$$

where Δ is the element surface measure:

$$\Delta = \frac{1}{2} \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{vmatrix},$$

and the coefficients a_i, b_i e c_i are given by:

$$\begin{aligned} a_i &= x_j y_m - x_m y_j \\ b_i &= y_j - y_m \\ c_i &= x_m - x_j \end{aligned}$$

The others are obtained thanks to an anticlockwise indices permutation:

$$a_j = x_m y_i - x_i y_m \qquad b_j = y_m - y_i \qquad c_j = x_i - x_m$$

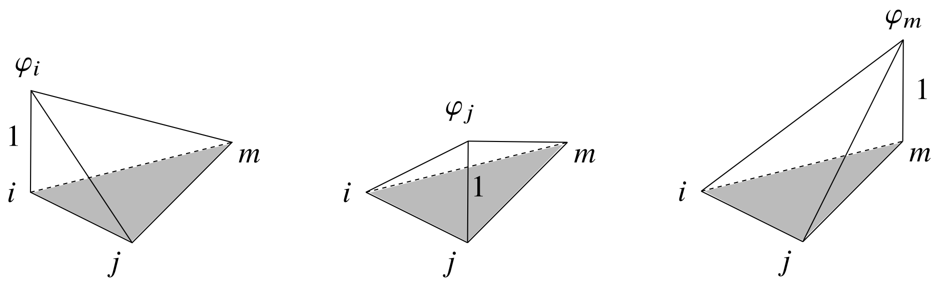

Figure 4: Linear basis functions for a generic triangular element.

$$a_m = x_i y_j - x_j y_i \qquad b_m = y_i - y_j \qquad c_m = x_j - x_i$$

The behavior of the basis functions $\varphi_k^{(e)}$ is shown in Fig. 4.

From Eq. (8), we have the expression of any entry of the stiffness matrix H:

$$h_{ij}^{(e)} = \int_{\Omega^{(e)}} \left[\frac{\partial \varphi_j^{(e)}}{\partial x} \frac{\partial \varphi_i^{(e)}}{\partial x} + \frac{\partial \varphi_j^{(e)}}{\partial y} \frac{\partial \varphi_i^{(e)}}{\partial y} \right] d\Omega = \frac{1}{4\Delta} \left(b_i b_j + c_i c_j \right) \tag{16}$$

Thus, the local stiffness matrix $H^{(e)}$ for a triangular element is:

$$H^{(e)} = \frac{1}{4\Delta} \left\{ \begin{bmatrix} b_i b_i & b_i b_j & b_i b_m \\ b_j b_i & b_j b_j & b_j b_m \\ b_m b_i & b_m b_j & b_m b_m \end{bmatrix} + \begin{bmatrix} c_i c_i & c_i c_j & c_i c_m \\ c_j c_i & c_j c_j & c_j c_m \\ c_m c_i & c_m c_j & c_m c_m \end{bmatrix} \right\} \tag{17}$$

Similarly, from Eq. (9), we have the expression of any entry of the mass matrix M:

$$m_{ij}^{(e)} = \int_{\Omega^{(e)}} \varphi_j^{(e)} \varphi_i^{(e)} d\Omega = \begin{cases} \dfrac{\Delta}{6} & \text{if } i = j \\ \dfrac{\Delta}{12} & \text{if } i =\neq j \end{cases} \tag{18}$$

Thus, the local mass matrix $M^{(e)}$ for a triangular element is:

$$M^{(e)} = \frac{\Delta}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{19}$$

Once the local contributions $H^{(e)}$ and $M^{(e)}$ are available, they have to be assembled in the global stiffness and mass matrices H and M, i.e., they have to summed to the existing values in the *right* position. The right position is given by the global numbering of the nodal indices i, j and m.

To simplify the computation, it is worth to create a dedicated function for the computation of the global matrices. Given the topology and the coordinates, the function will loop over all elements, compute the local contributions and assemble them in the corresponding position. The output will be the assembled stiffness and mass matrices H and M. Once `Hloc` is formed for the element k, the pseudo-code of Alg. 3.1 assembles it. The extension to handle the mass matrix is straightforward.

Algorithm 3.1 Assembly pseudo-code

```
1: H = sparse(n,n)                                  ▷ Initialize $H$ as a sparse matrix
2: for i = 1, 3 do
3:     row = elem(k,i)                              ▷ Retrieve global row index
4:     for j = 1, 3 do
5:         col = elem(k,j)                          ▷ Retrieve global column index
6:         H(row,col) = H(row,col) + Hloc(i,j)      ▷ Assembly local term in global position
7:     end for
8: end for
```

3.3 Computation of nodal areas

Even if in this application there is no source term, we investigate how to compute it because we need a portion of the algorithm involved in the right-hand side computation to evaluate the global error.

The i-th component of the right-hand side f is:

$$f_i = \int_\Omega f(\mathbf{x})\varphi_i d\Omega. \tag{20}$$

Thanks to the local nature of φ_i, the integral in (20) can be restricted from Ω to the set of triangles sharing node i. Since i is a sort of centroid of this element patch, we can assume the function f to be constant over this subdomain and equal to the value in the node i. Thus, $f(x_i, y_i)$ can be extracted from the integral (20), reading:

$$f_i \simeq f\left(\mathbf{x}_i\right) \int_\Omega \varphi_i d\Omega = f\left(\mathbf{x}_i\right) \frac{\sum_e \Delta_e}{3} \tag{21}$$

where the summation is extended to all the triangles sharing node i. The term $\sum_e \Delta_e/3$ is the surface measure relative to this node, i.e., one third of the area of the triangle patch surrounding node i. To compute this contribution, an array storing the surface measure for each element can be saved during the assembly loop, where Δ has to be determined, then, thanks to the connection list of the topology, these areas can be redistributed across all the nodes.

3.4 Boundary conditions enforcement

From a numerical viewpoint, to impose a Dirichlet boundary condition on a node means we have to set the function value on that location. To do so, we need to modify the equation relative to this node and to impose the known value. The new equation reads:

$$u_i = u_{D,i} \tag{22}$$

i.e., the i-th row of the system matrix is composed by one value only, a diagonal 1, and the right-hand side is the fixed value $u_{D,i}$ for that node. This operation clearly destroys the symmetry of the matrix. To restore this beneficial property, we need to modify also the i-th column, setting it to zero, and update the right-hand side accordingly.

Let us assume to impose the condition $u_2 = u_{D,2}$ on a 4×4 linear system, as in Eq. (23). To preserve the symmetry the right-hand side has to be updated with the removed column, in this case the second one. This procedure is sketched in Alg. 3.2.

$$\begin{gathered}
\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \\
\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & 1 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ u_{D,2} \\ f_3 \\ f_4 \end{bmatrix} \\
\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} \\ 0 & 1 & 0 & 0 \\ a_{31} & 0 & a_{33} & a_{34} \\ a_{41} & 0 & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 - a_{12}u_{D,2} \\ u_{D,2} \\ f_3 - a_{32}u_{D,2} \\ f_4 - a_{42}u_{D,2} \end{bmatrix}
\end{gathered} \tag{23}$$

Algorithm 3.2 Enforcement of Dirichlet boundary condition pseudo-code

```
1: H(bound(:,1),:)  = 0                          ▷ Set all the Dirichlet rows to zero
2: rhs = rhs - H(:,bound(:,1)) * bound(:,2)      ▷ Update rhs with to-be-removed columns
3: rhs(bound(:,1)) = bound(:,2)                  ▷ Set the rhs values on the Dirichlet rows
4: H(:,bound(:,1)) = 0                           ▷ Actually remove the Dirichlet columns
5: for i = 1, nBound do
6:     H(bound(i,1),bound(i,1)) = 1              ▷ Finally set diagonal term to 1
7: end for
```

If there were non-homogeneous Neumann boundary conditions, they would have contributed to the right-hand side.

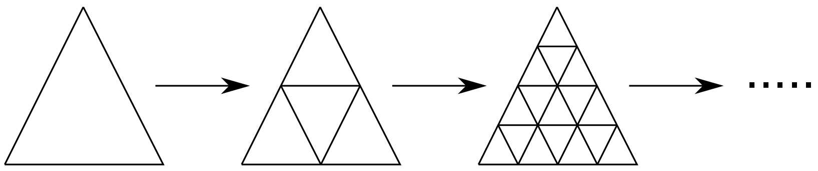
Figure 5: Refinement scheme for the computational grid.

3.5 Linear system solution

Since the system matrix $K_1 = M/\Delta t + \theta H$ is large, sparse and symmetric positive definite (SPD), a Krylov subspace iterative method can be efficiently used to solve the associated linear system. In particular, the Preconditioned Conjugate Method (PCG) is the reference method to solve the linear system (11). Among the simplest preconditioners, we remind Jacobi, i.e., the inverse of the diagonal elements, and the incomplete Cholesky factorization. The exit criterion is met when the residual norm is less then 10^{-8} times the right-hand side norm. The null vector is taken as initial solution.

3.6 Error computation

Since for the problem at hand the analytical solution is missing, one idea is to adopt as reference the numerical solution on a very refined grid. Thanks to this reference solution, the FEM convergence can be studied. We restrict ourselves to the convergence study at the stationary solution, i.e., the solution for $t \to \infty$. The refined solution is provided for a very refined case (`Ref`). In particular, a file with `xRef`, `yRef` and `uRef` is provided. Note that this solution has to be interpolated on the current grid. This interpolation can be done thanks to the functions provided in Alg. 3.3.

Refining the grid, as shown in Fig. 5, it is expected an error reduction. For this kind of refinement, where on average a node is added for each edge midpoint and triangle centroid, the total number of triangles increases of a factor 4 and the characteristic length decreases by a factor 2 at each level. The Euclidean norm of the error decreases as ℓ^2, where ℓ represents the average mesh size, i.e., the characteristic length of the edges. The error norm reads:

$$\varepsilon = \sqrt{\int_\Omega (u_h - u)^2 \, d\Omega} \tag{24}$$

where u_h is the numeric solution and u is the reference solution on a very fine grid.

The integral (24) can be computed thanks to the midpoint rule, i.e.:

$$\varepsilon \simeq \left\{ \sum_{i=1}^{n} \left[(u_i - u(x_i, y_i))^2 \frac{\sum_e \Delta_e}{3} \right] \right\}^{1/2} \tag{25}$$

where the summation is meant over all the elements e sharing node i.

Algorithm 3.3 Interpolation routines (both can be used with vectors of x_i and y_i)

```
For MATLAB users
% function handle of two variables linearly interpolating uRef
interp = scatteredInterpolant(xRef, yRef, uRef);
soli = interp(xi, yi);                    ▷ interpolated solution at node (x_i, y_i)

For python users
from scipy.interpolate import LinearNDInterpolator
# piecewise linear interpolant of uRef on xRef, yRef
interp = LinearNDInterpolator((xRef, yRef), uRef)
soli = interp(xi, yi)                     ▷ interpolated solution at node (x_i, y_i)
```

Table 1: Solution at different times for points P_1, P_2 and P_3.

t	$u(P_1)$	$u(P_2)$	$u(P_3)$
2.5	0.2434390	0.0772287	0.0183037
5.0	0.6046775	0.2751718	0.0928241
7.5	0.7454968	0.4328630	0.1716526
10.0	0.7751273	0.4805514	0.2008722

4 Results

To assess the solution, qualitatively reported in Fig. 6, some quantities can be monitored. They are:

- the trace of the solution at different times ($t = t_{\max}/4$, $t = t_{\max}/2$ and $t = 3/4\ t_{\max}$ and $t = t_{\max}$) on the external boundary Γ_N, from $(+1, +1)$ to $(-1, +1)$ passing through $(+1, -1)$ and $(-1, -1)$ (see Fig. 1). The nodes along this path are provided for each mesh in the right order in the `trace` file;
- the solution at all the times on the three tracking points P_1, P_2 and P_3 shown in Fig. 1. For each mesh, the indices of these points are provided in the `track` file;
- the values of the solution on the three tracking points P_1, P_2 and P_3 at the end of the simulation time, i.e., $t = t_{\max}$.

For the finest grid, these behaviors are reported in Fig. 7 and Tab. 1.

5 Requirements

Four subsequent refinement levels of the original computational grid are provided, for a total of five meshes. The candidate has to describe their development in a report, with this minimum set of requirements:

- plots of the solution, as described in the Result section, for each mesh;

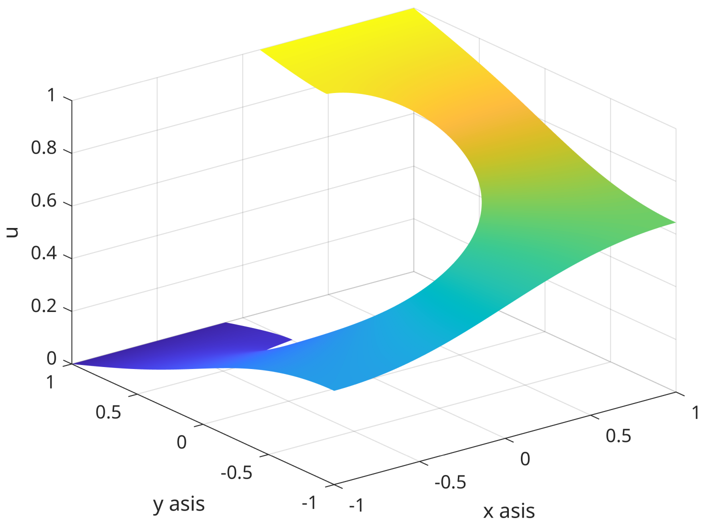

Figure 6: Stationary solution on the finest grid.

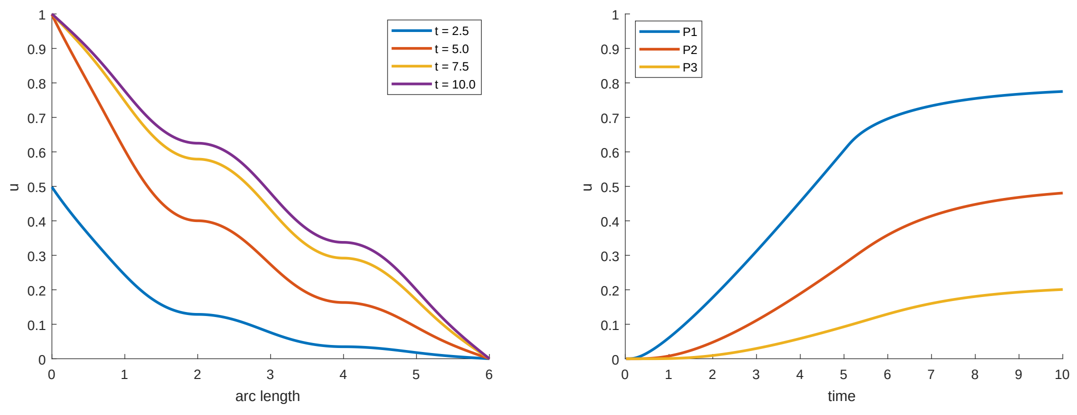

Figure 7: Solution along the boundary for given times and on the three tracking points for all the times for the finest grid.

- table of solution values, as described in the Result section, and the associated error, measured as difference with respect to values in Tab. 1, for each mesh;
- semi-logarithmic convergence plots for the relative residual norm during the PCG iterations for the linear system arising from the stationary solution. Two plots with the convergence histories for all the refinement levels are required: one using Jacobi as preconditioner, the other with the incomplete Cholesky factorization;
- summary table for the FEM convergence, with the error value ε_k and the ratio r_k, function of the error and the mesh size at the current and previous refinement levels, for each level k;

$$r_k = \frac{\varepsilon_{k-1}}{\varepsilon_k} \left(\frac{h_k}{h_{k-1}} \right)^2 \tag{26}$$

- the source code, with appropriate comments for each function.