F.E.M

Valentina Zonta

September 11, 2024

1 Introduction

The heat equation is a fundamental partial differential equation (PDE) governing the distribution of temperature in a given domain over time. This equation arises in various physical phenomena involving heat transfer, such as conduction in solid materials. In this exercise, we aim to solve the initial boundary value problem for the heat equation in a two-dimensional horseshoe-shaped domain using the *Finite Element Method (FEM)*.

The domain boundary is divided into two regions, Γ_D and Γ_N, where Dirichlet and Neumann boundary conditions are applied, respectively. The Dirichlet boundary is further split into two parts, $\Gamma_{D,1}$ and $\Gamma_{D,2}$, with different time-dependent boundary conditions. The governing PDE is time-dependent (parabolic) and becomes elliptic in the steady-state, where the solution no longer changes over time.

The main objective is to compute the transient and steady-state solutions using triangular elements and linear basis functions, which will allow us to observe the evolution of the temperature field over time and its behavior as it approaches equilibrium. Specifically, we will employ *FEM* to discretize the spatial domain and apply time integration to capture the transient behavior. The problem will be solved for both the transient case (for a time interval $T = (0, t_{\max}]$) and the steady-state case, where the temperature no longer varies with time.

By solving this problem, we will gain insight into the numerical solution of parabolic and elliptic PDEs using FEM and observe how boundary conditions influence the temperature distribution in time.

2 Problem Statement

The problem involves solving the heat equation over a two-dimensional domain Ω for a finite time interval $T = [0, t_{\max}]$, where $t_{\max} = 10$. The equation has a unit diffusion constant and no drift term. Mixed boundary conditions are applied to the domain, with Dirichlet conditions on the upper boundary and Neumann conditions on the inner and outer sides. The initial configuration of the temperature distribution is also provided at $t = 0$.

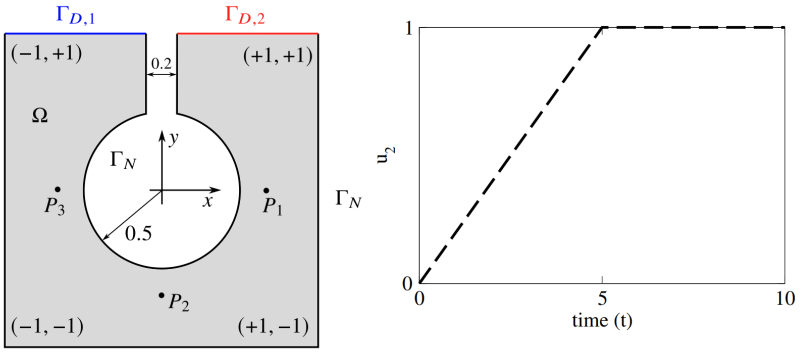


Figure 1: Domain and $u_2(t)$ time behavior.

The problem is formulated as follows:

$$
\begin{aligned}
&\frac{\partial u(x,t)}{\partial t} - \nabla^2 u(x,t) = f(x,t) && \text{in } \Omega \times T,\\
&u(x,t) = d_1(x,t) && \text{on } \Gamma_{D,1} \times T,\\
&u(x,t) = d_2(x,t) && \text{on } \Gamma_{D,2} \times T,\\
&\frac{\partial u(x,t)}{\partial n} = q(x,t) && \text{on } \Gamma_N \times T,\\
&u(x,0) = u_0(x) && \text{in } \overline{\Omega}.
\end{aligned}
$$

Here, $u(x,t)$ represents the temperature distribution over time and space. The initial condition $u_0(x) \equiv 0$ states that the entire domain is initially at zero temperature, representing thermal equilibrium at $t = 0$. There is no internal heat source, meaning $f(x,t) \equiv 0$. Dirichlet boundary conditions are applied: on the upper boundary, $d_1(x,t) \equiv 0$, keeping this side at a constant temperature of zero throughout the time interval. On the progressively heated side, $d_2(x,t)$ increases linearly from zero at $t = 0$ to a value of one at $t = \frac{t_{\max}}{2}$, after which it remains constant at one for $t > \frac{t_{\max}}{2}$. The Neumann condition $q(x,t) \equiv 0$ ensures that no heat flux occurs along the inner and outer sides of the boundary.

At $t = 0$, the temperature across the domain satisfies the Laplace equation, meaning $u(x,0) = 0$. As time progresses, the temperature distribution evolves, and the transient solution will be computed using the Finite Element Method (FEM). Finally, the steady-state solution, $\lim_{t\to\infty} u(x,t)$, will be determined by setting the time derivative to zero, reducing the problem to an elliptic equation (the Laplace equation).

3 FEM Solution

The initial boundary value problem (1) can be solved numerically using the Finite Element Method (FEM). By applying the FEM, the problem is reformulated into a variational form, leading to the Galerkin method.

The approximate solution $u_h(x,t)$ can be expressed as a linear combination of basis functions:

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

where $\varphi_i(x)$, for $i = 1, \ldots, n$, are the basis functions corresponding to the chosen discretization of the spatial domain, and $u_i(t)$ are the time-dependent nodal values at the n discretization points within the domain Ω. These nodal values $u_i(t)$ represent the unknowns of the problem that must be solved at each time step.

In this formulation, the spatial domain is discretized into a finite number of elements, and the solution is approximated over these elements using the basis functions. By substituting this expression into the variational form of the original problem, we obtain a system of equations for the unknowns $u_i(t)$, which can be solved iteratively over time.

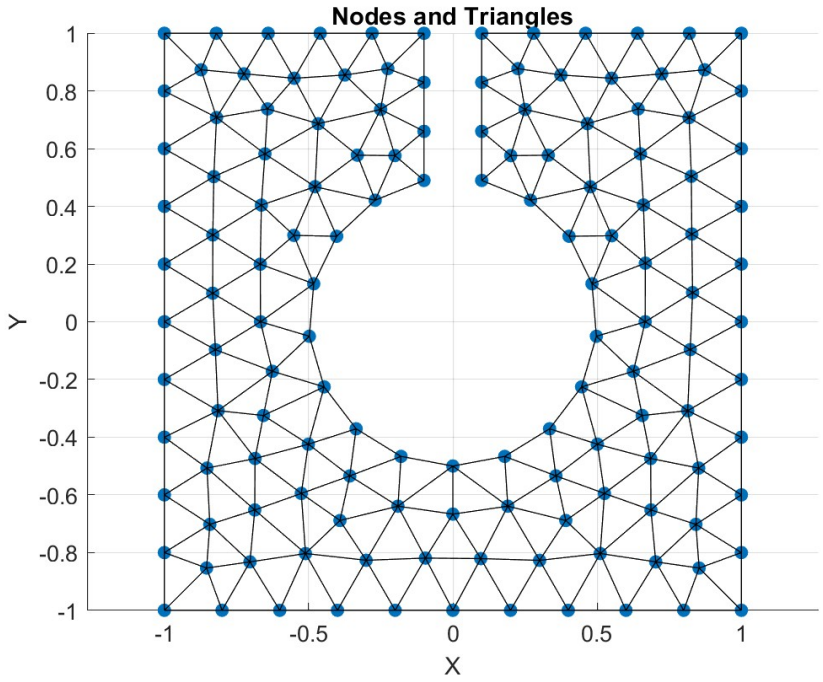


Figure 1: domain and fem discretisation

The solution evolves over time as shown in Figure 1, where the domain and time-dependent behavior of $u_2(t)$ is depicted.

3.1 Basis Functions and Approximation

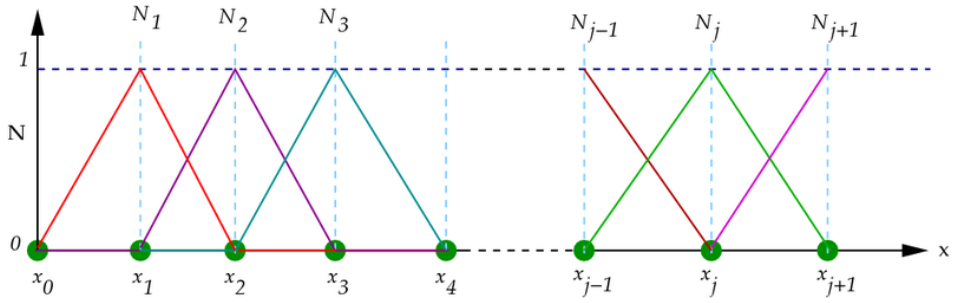
Let us explain how the basis function we use work. The simplest basis functions used in finite element methods are piecewise linear functions. Corresponding elements in different dimensions are as follows: line segments in 1D, triangles and quadrilaterals in 2D, and tetrahedra, prisms, and hexahedra in 3D.

These basis functions must be selected from the function space $\mathcal{H}_0^n$. Let Ω_n denote the space of piecewise linear functions. It can be demonstrated that $\Omega_n \subset \mathcal{H}_0^n$.

Consider a partition of the interval $[0, 1]$ into subintervals $I_j = [x_j, x_{j+1}]$, where

$$0 = x_0 < x_1 < x_2 < \cdots < x_n < x_{N+1} = 1$$

is the partition of $[0, 1]$ into N subintervals. Let $\Delta x_j = x_{j+1} - x_j$ for $j = 1, 2, \ldots, N$, and let $h = \max(\Delta x_j)$ be a measure of the overall grid fineness. Basis functions are chosen as triangular functions, as illustrated in Figure 2.



In algebraic form, the basis functions $N_j(x)$ are defined as:

$$N_j(x) = \begin{cases} \frac{x - x_{j-1}}{x_j - x_{j-1}}, & x_{j-1} \le x \le x_j, \\ \frac{x_{j+1} - x}{x_{j+1} - x_j}, & x_j \le x \le x_{j+1}, \\ 0, & \text{otherwise.} \end{cases}$$

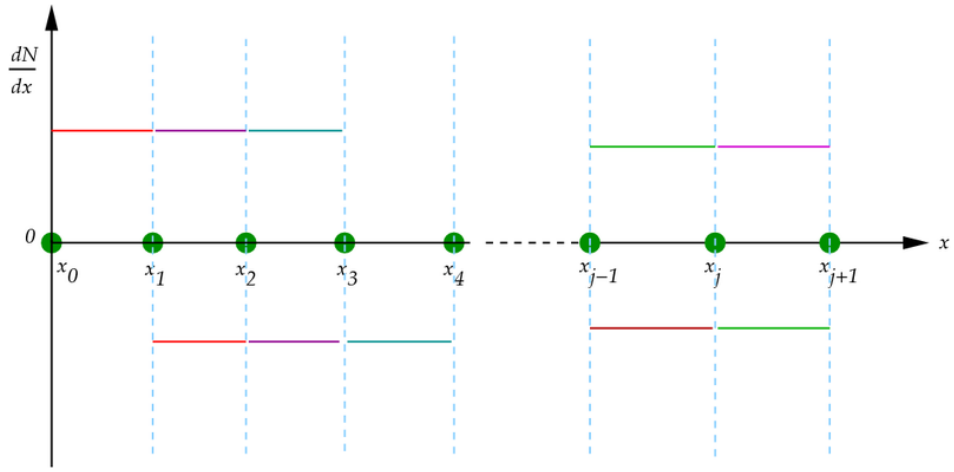
These shape functions satisfy the Kronecker delta property:

$$N_i(x_j) = \delta_{ij} = \begin{cases} 1, & \text{at } x = x_j, \\ 0, & \text{at } x \neq x_j. \end{cases}$$

The first derivatives of the basis functions are:

$$\frac{dN_j(x)}{dx} = \begin{cases} \frac{1}{x_j - x_{j-1}}, & x_{j-1} \le x \le x_j, \\ -\frac{1}{x_{j+1} - x_j}, & x_j \le x \le x_{j+1}, \\ 0, & \text{otherwise.} \end{cases}$$

As shown in Figure 2, there are discontinuities in the first derivatives at $x = x_j$.



The Galerkin trial solution is expressed as:

$$u_h = N_0 + \sum_{i=1}^{n} a_i N_i$$

where n is the number of nodes in an element. For two nodes per element, as depicted in Figure 2, the solution within each element is:

$$u_h^e = a_1 N_1^e + a_2 N_2^e$$

Here, N_i^e are the local (element-specific) basis functions, distinct from the global basis functions centered at nodes.

In an element connecting nodes x_j and x_{j+1}, the local basis functions are:

$$N_1^e = \frac{x_{j+1} - x}{x_{j+1} - x_j}$$

$$N_2^e = \frac{x - x_j}{x_{j+1} - x_j}$$

Thus, the element-wise solution is:

$$u_h^e = a_1 \left(\frac{x_{j+1} - x}{x_{j+1} - x_j} \right) + a_2 \left(\frac{x - x_j}{x_{j+1} - x_j} \right)$$

To satisfy boundary conditions, let u_h^e be equal to u_j at $x = x_j$ and u_{j+1} at $x = x_{j+1}$. This yields:

$$a_1 = u_j \quad \text{and} \quad a_2 = u_{j+1}$$

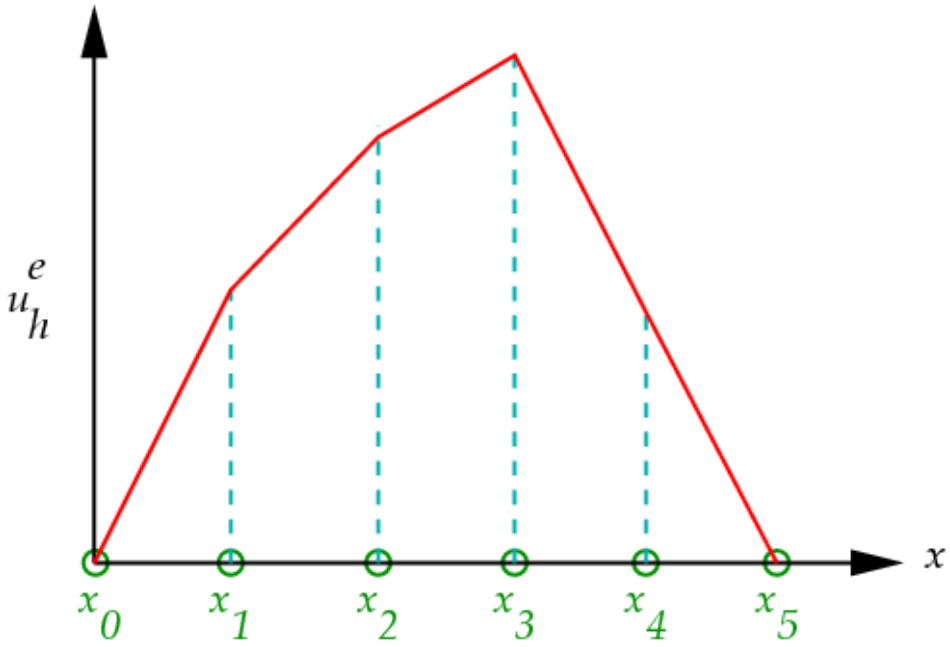
Therefore, the approximate solution is:

$$u_h^e = u_j \left(\frac{x_{j+1} - x}{x_{j+1} - x_j} \right) + u_{j+1} \left(\frac{x - x_j}{x_{j+1} - x_j} \right)$$

Alternatively, this can be written in parametric form as:

$$u_h^e = (1 - t)u_j + tu_{j+1}$$

where $t = \frac{x - x_j}{x_{j+1} - x_j}$.



This parametric representation illustrates that u_h^e is a linear approximation. Figure 3 presents a schematic of the finite element solution using linear shape functions.

4 Finite Element Method Solution

4.1 Spatial Solution

To solve the initial boundary value problem, we substitute u_h into the governing equation:

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

The residual $L(u_h)$ is then orthogonal to the basis functions $\phi_i(x, y)$:

$$\int_\Omega L(u_h)\phi_i(x, y)\, d\Omega = 0 \quad \text{for } i = 1, \ldots, n$$

Applying the divergence (Gauss) theorem to this equation results in:

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

Substituting the approximation u_h from Eq. (2) into this equation yields:

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

In our case, with $q(x,t) = 0$ and $f(x,t) = 0$, this simplifies to:

$$\int_\Omega \left[\sum_{j=1}^{n}\left(\frac{\partial \phi_j}{\partial x}\frac{\partial \phi_i}{\partial x} + \frac{\partial \phi_j}{\partial y}\frac{\partial \phi_i}{\partial y}\right)u_j\right] d\Omega = 0$$

This leads to a system of ordinary differential equations (ODEs):

$$M\frac{du(t)}{dt} + Hu(t) = 0$$

where M is the mass matrix and H is the stiffness matrix. The stiffness matrix entries are:

$$h_{ij} = \sum_e \int_{\Omega(e)} \left(\frac{\partial \phi_j^{(e)}}{\partial x}\frac{\partial \phi_i^{(e)}}{\partial x} + \frac{\partial \phi_j^{(e)}}{\partial y}\frac{\partial \phi_i^{(e)}}{\partial y}\right) d\Omega$$

and the mass matrix entries are:

$$m_{ij} = \sum_e \int_{\Omega(e)} \phi_j^{(e)}\phi_i^{(e)}\, d\Omega$$

Both H and M are sparse, symmetric, and positive definite matrices.

4.2 Time Integration

The system of ODEs is solved using the θ-method, which approximates the time derivative with a simple difference quotient and incorporates both current and previous time steps with a parameter θ (where $0 \le \theta \le 1$):

$$M\frac{u^{k+1} - u^k}{\Delta t} + H\left(\theta u^{k+1} + (1-\theta)u^k\right) = 0$$

where $\Delta t = t_{k+1} - t_k$ is the time step, and k denotes the time index. Rearranging, we solve the linear system:

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

In compact form:

$$K_1 u^{k+1} = K_2 u^k$$

where:

$$K_1 = \frac{M}{\Delta t} + \theta H$$

$$K_2 = \frac{M}{\Delta t} - (1-\theta)H$$

For this application, we use $\theta = 0.5$ (Crank-Nicolson scheme) with a constant time step $\Delta t = 0.02$. The matrices K_1 and K_2 are symmetric and positive definite, ensuring stable and accurate time integration.

4.3 Computation of Local Stiffness and Mass Matrices and Their Assembly

The domain Ω is subdivided into triangular elements e. For each element, the solution $u_h^{(e)}$ is approximated as:

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

where $\phi_k^{(e)}(x,y)$ are the basis functions for the k-th node of the element. In a generic triangular element, the solution is linear with respect to the nodal unknowns u_i, u_j, and u_m. To satisfy the boundary conditions at the nodes, the basis functions are computed as:

$$\phi_i^{(e)}(x,y) = \frac{a_i + b_i x + c_i y}{2\Delta}$$

$$\phi_j^{(e)}(x,y) = \frac{a_j + b_j x + c_j y}{2\Delta}$$

$$\phi_m^{(e)}(x,y) = \frac{a_m + b_m x + c_m y}{2\Delta}$$

where Δ is the area of the element:

$$\Delta = \frac{1}{2}\left|x_i y_j + x_j y_m + x_m y_i - x_i y_m - x_j y_i - x_m y_j\right|$$

The coefficients a_i, b_i, and c_i are determined as follows:

$$a_i = \frac{x_j y_m - x_m y_j}{2\Delta}, \quad b_i = \frac{y_j - y_m}{2\Delta}, \quad c_i = \frac{x_m - x_j}{2\Delta}$$

Similarly:

$$a_j = \frac{x_m y_i - x_i y_m}{2\Delta}, \quad b_j = \frac{y_m - y_i}{2\Delta}, \quad c_j = \frac{x_i - x_m}{2\Delta}$$

$$a_m = \frac{x_i y_j - x_j y_i}{2\Delta}, \quad b_m = \frac{y_i - y_j}{2\Delta}, \quad c_m = \frac{x_j - x_i}{2\Delta}$$

The local stiffness matrix $H^{(e)}$ for a triangular element is computed from:

$$h_{ij}^{(e)} = \int_{\Omega^{(e)}} \left(\frac{\partial \phi_j^{(e)}}{\partial x}\frac{\partial \phi_i^{(e)}}{\partial x} + \frac{\partial \phi_j^{(e)}}{\partial y}\frac{\partial \phi_i^{(e)}}{\partial y} \right) d\Omega$$

$$h_{ij}^{(e)} = \frac{1}{4\Delta}\left(b_i b_j + c_i c_j\right)$$

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

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

The mass matrix $M^{(e)}$ for a triangular element is computed from:

$$m_{ij}^{(e)} = \int_{\Omega^{(e)}} \phi_j^{(e)} \phi_i^{(e)} d\Omega$$

$$m_{ij}^{(e)} = \begin{cases} \frac{\Delta}{6} & \text{if } i = j \\ \frac{\Delta}{12} & \text{if } i \neq j \end{cases}$$

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

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

Once the local stiffness matrices $H^{(e)}$ and mass matrices $M^{(e)}$ are computed, they are assembled into the global stiffness matrix H and mass matrix M by summing the contributions at the appropriate positions according to the global nodal numbering.

4.4 Boundary Conditions Enforcement

In numerical methods, enforcing a Dirichlet boundary condition at a node involves specifying the value of the function at that particular location. To achieve this, the corresponding equation for that node must be modified to include the known value. For a node i, the modified equation is:

$$u_i = u_{D,i}$$

where $u_{D,i}$ is the prescribed Dirichlet value for node i. Consequently, the i-th row of the system matrix is altered to contain only a single non-zero entry: a diagonal element set to 1. The corresponding entry on the right-hand side of the equation is updated to $u_{D,i}$.

This modification disrupts the symmetry of the matrix. To restore symmetry, the entire i-th column of the matrix must also be set to zero, and the right-hand side of the equation adjusted accordingly.

For instance, consider imposing the condition $u_2 = u_{D,2}$ in a 4×4 linear system. To maintain symmetry, the second column of the matrix must be removed, and the right-hand side updated with the values from the removed column.

5 2.6 Computation of Integral and Error

To verify the accuracy of the computed solution, we compare it against an exact solution provided a priori, which is stored in the file 'solRef.dat'. This file includes three columns:

- The first column lists the x-coordinates.
- The second column lists the y-coordinates.
- The third column contains the exact stationary solution values corresponding to the given coordinates.

The exact solution is read from the file and interpolated to create a fine-grained reference solution for comparison with the computed solution.

The overall error ϵ is computed using the following formula:

$$\epsilon = \sqrt{\int_\Omega (u_h - u_{\text{exact}})^2 \, d\Omega}$$

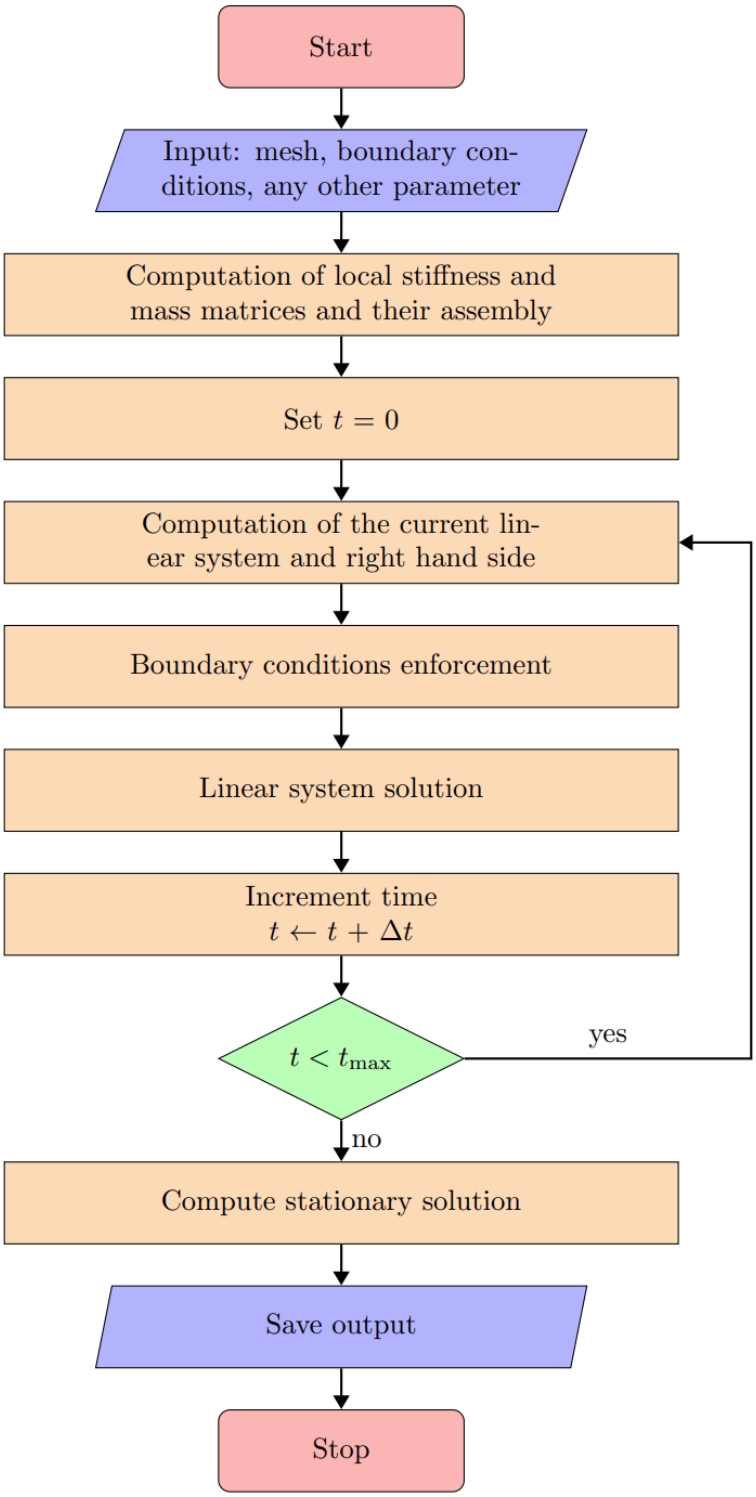
Evaluating the integral and using midpoint approximation the error formula becomes:

$$\epsilon \approx \sqrt{\sum_{i=1}^{n} (u_{h,i} - u_{\text{exact},i})^2 \sum_{e:i\in e} \frac{\Delta^{(e)}}{3}}$$

This approach provides an estimate of the error by accounting for the contributions from each element connected to the node, allowing for an assessment of the solution's accuracy.

6 Code explanation

For our code we followed this procedure:



First we uploaded the mesh and the data as follow:

Since five distinct meshes were provided for solving the PDE equation, a general class was implemented to handle each grid. For every mesh, five different files were supplied:

- `meshX.coord`: This file contains the comprehensive list of all nodes arranged in order, with two columns detailing the x and y coordinates of each node, respectively.
- `meshX.topol`: This file includes the ordered list of triangular elements, where each row specifies three integers representing the node numbers of the triangle's vertices.
- `meshX.trace`: This file lists all nodes belonging to Γ_N and provides the arc length of the curve up to each node, which is necessary for plotting the solution profile along the boundary at four different time instances.
- `meshX.track`: This file contains three node numbers whose values over time will be monitored and plotted. These nodes correspond to the points marked as P1, P2, and P3 in .
- `meshX.bound`: This file lists the boundary points where Dirichlet conditions need to be applied, along with the corresponding solution values at $t_{\max}$.

As second step we use a function that create H and M matrices following the instruction in 4.3 section, and assembling the local contribution of both to create the sparse matrices as follow:

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
```

Then we find the solution of the heat

we compute the solution to a heat equation using a time-stepping method with Crank-Nicolson scheme.

where:

- `M` and `H` are the mass and stiffness matrices, respectively.
- `h` is the time step size.
- `tmax` is the maximum time.
- `numnod` is the number of nodes.
- `bound` is an array specifying the Dirichlet boundary conditions.

The function performs the following steps:

1. Compute matrices K_1 and K_2:
$$K_1 = \frac{M}{h} + \frac{H}{2}$$
$$K_2 = \frac{M}{h} - \frac{H}{2}$$
2. Modify matrix A to enforce Dirichlet boundary conditions:

```
A(:, bound(i,1)) = 0;
A(bound(i,1), :) = 0;
A(bound(i,1), bound(i,1)) = 1;
```

3. Compute the incomplete Cholesky factorization L of A:

```
L = ichol(A, struct('type','ict','droptol',1e-3));
```

4. Initialize solution matrix:

$$\texttt{solution} = \texttt{zeros(num\_nod, (tmax/h) + 1);}$$

5. Time-stepping loop:

   - For $i > 1$, compute the right-hand side vector b:

$$b = K_2 \cdot \texttt{solution}(:, i-1)$$

   - For $i < (tmax/(2 \cdot h)) + 2$, update boundary condition values:

$$\mathrm{d}_2(t) = \begin{cases} \frac{2t}{t_{\max}} \cdot d_2(t_{\max}) & \text{if } t \leq \frac{t_{\max}}{2} \\ d_2(t_{\max}) & \text{if } t > \frac{t_{\max}}{2} \end{cases}$$

6. Adjust b to account for Dirichlet boundary conditions:

```
b = b - (K1(:,bound(k,1)) .* rbound(k));
b(bound(k,1)) = rbound(k);
```

7. Solve the linear system using the preconditioned conjugate gradient method:

```
[solution(:,i),~] = pcg(A, b, tol, maxit, L, L');
```

Now we are interested in 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. We have then $\dot{u} = 0$ (or equivalently M = 0) obtaining the discretization of the Laplace equation. We impose the boundary conditions , this time H should be modified. The computation are the same as the previous section and using pcg with e incomplete Cholesky precondition we find the solution.

The next step is to check it with refined solution, provided for a very refined case (Ref). 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.

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)
```

To ensure the accuracy of the computed solution, a comparison will be made with a pre-provided exact solution, stored in the file `solRef.dat`. This file contains data related to the stationary solution, with the first column representing the x-coordinate, the second column representing the y-coordinate, and the third column representing the stationary solution value at the corresponding coordinates. The error norm reads as show before:

$$\epsilon \approx \sqrt{\sum_{i=1}^{n} (u_i - u(x_i, y_i))^2 \sum_{e:i \in e} \Delta e/3}$$

Finally, the code calculates the absolute error between the computed value at time 2.5, 5, 7,5, 10 s of the three tracking point and the one given in the file exercise.

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

7 Results

The trace of the solution at times 2.5 s, 5 s, 7.5 s, 10 s along the external boundary for each mesh:

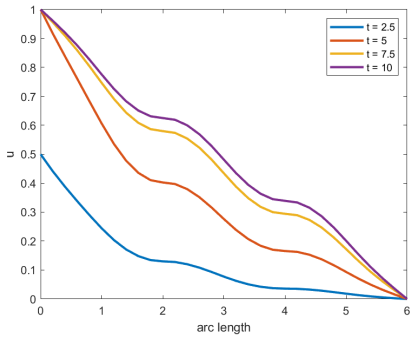


Figure 2: mesh0

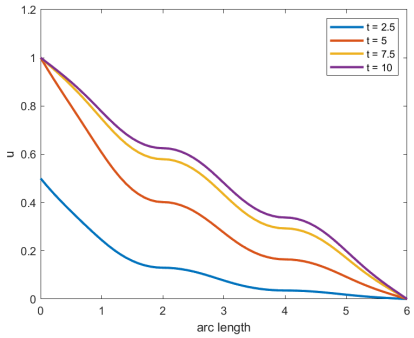


Figure 3: mesh1

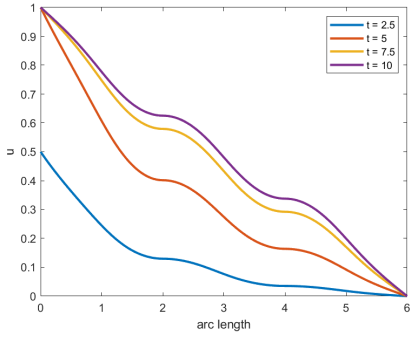


Figure 4: mesh2

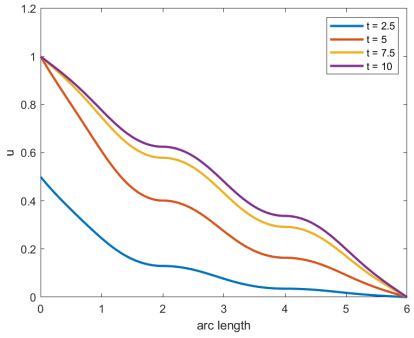


Figure 5: mesh3

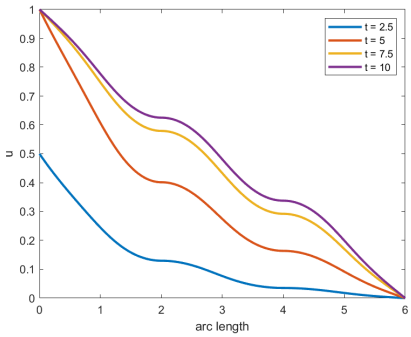


Figure 6: mesh4

The time evolution of the solution for the three points P1, P2 and P3 are reported for each mesh:

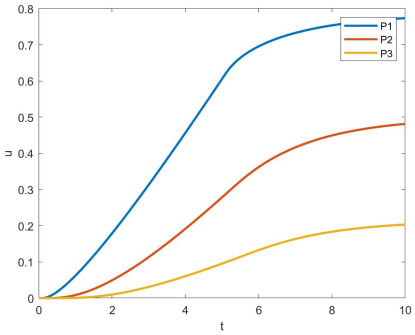


Figure 7: mesh0

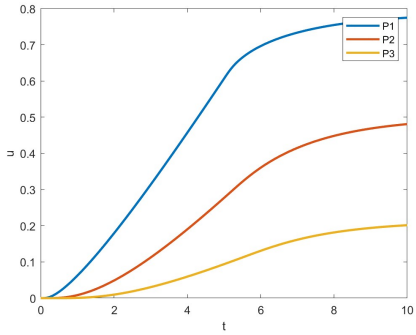


Figure 8: mesh1

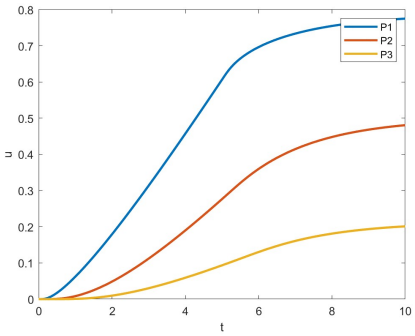


Figure 9: mesh2

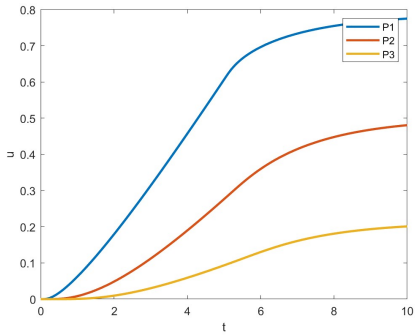


Figure 10: mesh3

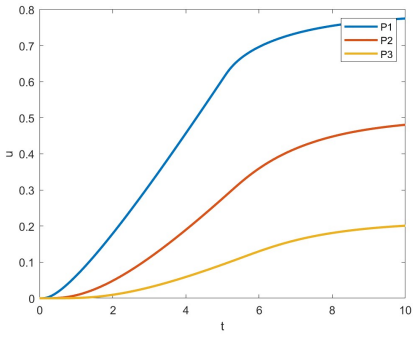


Figure 11: mesh4

Stationary solution plots for each mesh

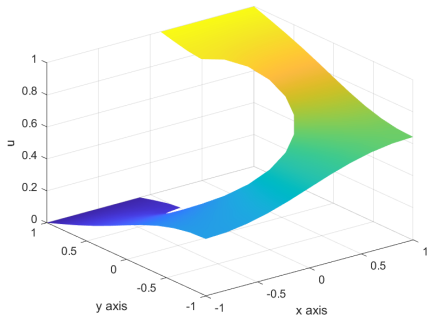


Figure 12: mesh0

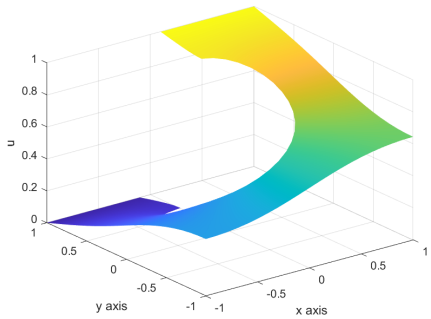


Figure 13: mesh1

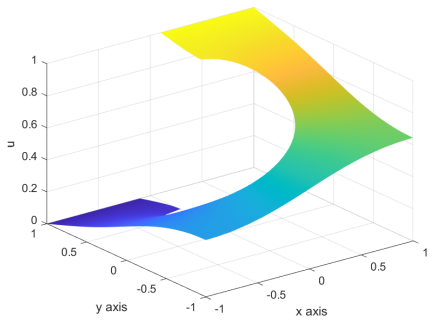


Figure 14: mesh2

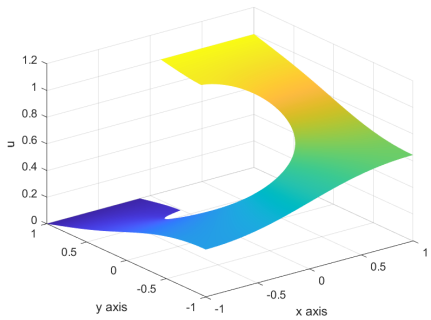


Figure 15: mesh3

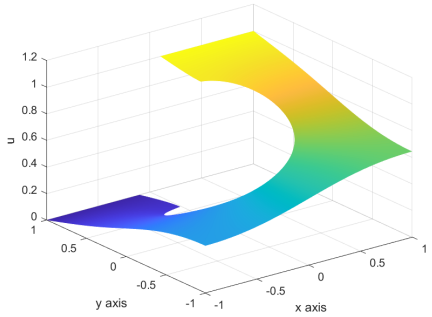


Figure 16: mesh4

The semilogarithmic convergence profiles for the stationary solution computation for incomplete Cholesky precondition

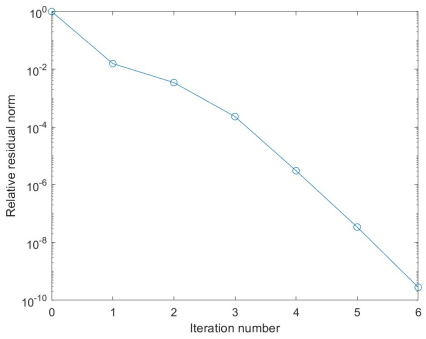


Figure 17: mesh0

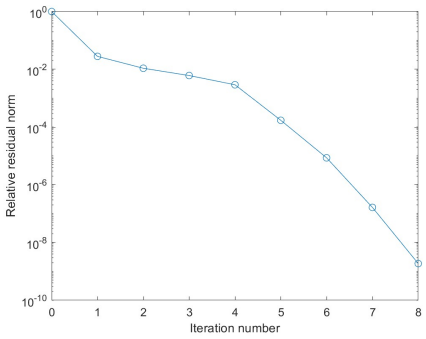


Figure 18: mesh1

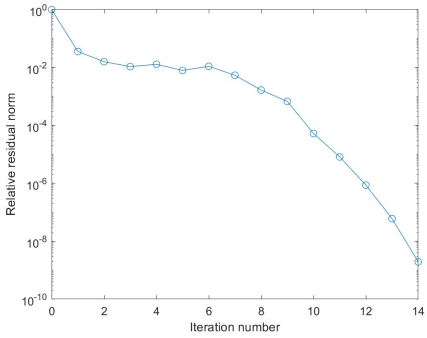


Figure 19: mesh2

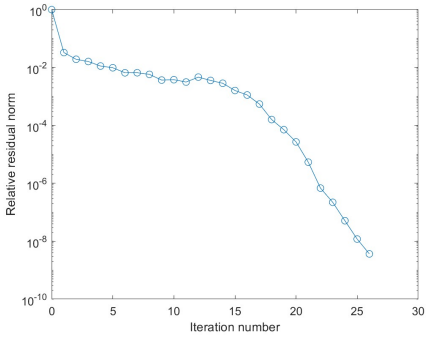


Figure 20: mesh3

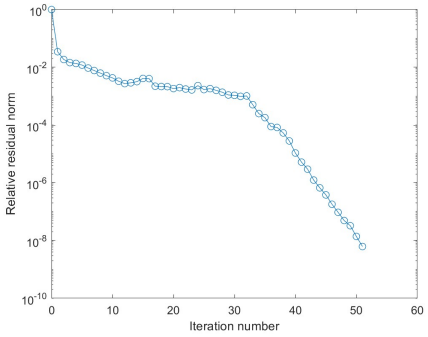


Figure 21: mesh4

In the following tables we show the three points values at fixed times together with the error computed as the difference between the solution and the reference values and finally the summary values requested:

	P1	P2	P3
mesh0	0.244	0.0789	0.0192
	0.605	0.278	0.0953
	0.745	0.435	0.175
	0.774	0.481	0.203
mesh1	0.245	0.0781	0.0187
	0.606	0.277	0.0937
	0.746	0.434	0.173
	0.775	0.481	0.201
mesh2	0.245	0.0779	0.0186
	0.606	0.276	0.0933
	0.746	0.433	0.172
	0.775	0.481	0.201
mesh3	0.245	0.0779	0.0185
	0.606	0.276	0.0932
	0.746	0.433	0.172
	0.775	0.481	0.201
mesh4	0.245	0.0779	0.0185
	0.606	0.276	0.0932
	0.746	0.433	0.172
	0.775	0.481	0.201

Table 1: Rounded data values for P1, P2, and P3

Errors	P1	P2	P3
mesh0	0.0008839	0.0016548	0.0008600
	0.0006358	0.0031186	0.0024793
	0.0008491	0.0021410	0.0029301
	0.0014095	0.0009465	0.0024145
mesh1	0.0012971	0.0009107	0.0003717
	0.0014253	0.0015085	0.0009015
	6.23E-05	0.0008389	0.0008857
	0.0002187	0.0003249	0.0006253
mesh2	0.0013405	0.0007062	0.0002495
	0.0014887	0.0010448	0.0005046
	0.0001591	0.0004488	0.0003860
	2.44E-05	0.0001407	0.0002161
mesh3	0.0013500	0.0006562	0.0002170
	0.0015153	0.0009371	0.0003969
	0.0002078	0.0003671	0.0002465
	4.87E-05	0.0001125	9.71E-05
mesh4	0.0013455	0.0006413	0.0002083
	0.0015083	0.0009047	0.0003692
	0.0002072	0.0003408	0.0002116
	5.53E-05	0.0001002	6.80E-05

Table 2: Error values for P1, P2, and P3

	ϵ	h	rk
mesh 0	0.0022872	0.2163297	
mesh 1	0.0006056	0.1243409	1.0985
mesh 2	0.0001464	0.0645342	1.2190
mesh 3	3.68e-5	0.0310225	0.7254
mesh 4	8.78e-6	0.0161198	0.5646

Table 3: Values for ϵ, h, and rk for different meshes