# Math 588 - Finite Element Method

# Final Project

Lazizbek Sadullaev

April 2025

## Contents

## Part A: Project Tasks

The final project for **Math 588 - Finite Element Method** consists of solving the following time-dependent boundary value problem (BVP):

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, \quad 0 < x < 1, \tag{1}$$

with the boundary conditions:

$$u(0,t) = u(1,t) = 0, \tag{2}$$

and initial condition:

$$u(x,0) = u_0(x), \tag{3}$$

where

$$u_0(x) = \begin{cases} 2x, & \text{for } x \in [0, 1/2], \\ 2 - 2x, & \text{for } x \in [1/2, 1]. \end{cases}$$

The project tasks are:

- Derive the semidiscrete variational formulation.
- Apply Forward Euler method for time discretization.
- Write the structure of the stiffness matrix **A**, mass matrix **B**, and load vector **f**.
- Solve the problem numerically by modifying the given Python code.

## 1 Methodology: Derivation of the Semidiscrete Formulation

We proceed step-by-step:

### 1.1 Function Spaces

Define the spatial domain and time interval as:

$$\Omega = (0, 1), \quad T = (0, T_{\text{final}}), \quad \text{for some } T_{\text{final}} > 0.$$

The admissible function space $V$ is defined as:

$$V := \{v \in C(\Omega) \mid v'(x) \text{ is piecewise continuous and bounded on } \Omega,\ v = 0 \text{ on } \partial\Omega\}.$$

### 1.2 Weak Formulation

Multiplying the PDE by a test function $v \in V$ and integrating over $\Omega$, we have:

$$\int_\Omega \frac{\partial u}{\partial t} v \, dx + \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \tag{4}$$

where $f(x) = 0$ in our problem.

The initial condition is incorporated as:

$$u(x, 0) = u_0(x), \quad x \in \Omega. \tag{5}$$

### 1.3 Finite Element Approximation

Let $V_h \subset V$ be a finite-dimensional subspace spanned by basis functions $\{\varphi_i\}_{i=1}^M$. We approximate:

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

where $u_i(t)$ are time-dependent coefficients.

Substituting into the weak form and choosing test functions $v = \varphi_j$ for each $j = 1, 2, \ldots, M$ gives:

$$\int_\Omega \left( \sum_{i=1}^M \frac{du_i(t)}{dt} \varphi_i(x) \right) \varphi_j(x) \, dx + \int_\Omega \left( \sum_{i=1}^M u_i(t) \nabla \varphi_i(x) \right) \nabla \varphi_j(x) \, dx = \int_\Omega f \varphi_j(x) \, dx.$$

Expanding and reordering sums:

$$\sum_{i=1}^M \left( \int_\Omega \varphi_i(x) \varphi_j(x) \, dx \right) \frac{du_i(t)}{dt} + \sum_{i=1}^M \left( \int_\Omega \nabla \varphi_i(x) \nabla \varphi_j(x) \, dx \right) u_i(t) = \int_\Omega f \varphi_j(x) \, dx.$$

## 1.4 Matrix Formulation

Define matrices:

$$\mathbf{B}_{ij} = \int_\Omega \varphi_i(x)\varphi_j(x)\,dx, \quad \mathbf{A}_{ij} = \int_\Omega \nabla\varphi_i(x)\nabla\varphi_j(x)\,dx,$$

and vectors:

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_M(t) \end{bmatrix}, \quad \mathbf{f}(t) = \begin{bmatrix} f_1(t) \\ f_2(t) \\ \vdots \\ f_M(t) \end{bmatrix}, \quad f_j(t) = \int_\Omega f(x)\varphi_j(x)\,dx.$$

The semidiscrete system is then:

$$\mathbf{B}\frac{d\mathbf{u}(t)}{dt} + \mathbf{A}\mathbf{u}(t) = \mathbf{f}(t), \quad t \in (0,T), \tag{6}$$

with initial condition:

$$\mathbf{B}\mathbf{u}(0) = \mathbf{u}_0, \quad \text{where} \quad (\mathbf{u}_0)_j = \int_\Omega u_0(x)\varphi_j(x)\,dx. \tag{7}$$

## 1.5 Time Discretization: Forward Euler Method

The time derivative is approximated by the Forward Euler scheme:

$$\frac{d\mathbf{u}(t)}{dt} \approx \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t},$$

where $\Delta t$ is the time step size.

Substituting into the semidiscrete system gives:

$$\mathbf{B}\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{A}\mathbf{u}^n = \mathbf{f}^n,$$

which can be rearranged to:

$$\mathbf{B}\mathbf{u}^{n+1} = (\mathbf{B} - \Delta t\mathbf{A})\,\mathbf{u}^n + \Delta t\mathbf{f}^n. \tag{8}$$

At each time step, we solve this linear system to update the solution.

—

# 2 Structure of the Stiffness Matrix, Mass Matrix, and Load Vector

## 2.1 Element Matrices for Linear Finite Elements

For each element $e$ between nodes $i$ and $i+1$ over an interval of length $h_e$, the local stiffness matrix and mass matrix are given by:

- **Local stiffness matrix**:
$$\mathbf{A}^{(e)} = \frac{1}{h_e}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$
- **Local mass matrix**:
$$\mathbf{B}^{(e)} = \frac{h_e}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Since the global domain $\Omega = (0,1)$ is divided uniformly into $n_{\text{elem}} = 6$ elements, each with equal size:
$$h = \frac{1-0}{6} = \frac{1}{6},$$
thus each element has the same size $h_e = h = \frac{1}{6}$.

## 2.2 Global Matrices Assembly

The global stiffness matrix $\mathbf{A}$ and mass matrix $\mathbf{B}$ are assembled by summing the contributions of the local matrices at shared nodes.

Since each element only connects two nodes, $\mathbf{A}$ and $\mathbf{B}$ are both $(n_{\text{node}} \times n_{\text{node}})$ matrices, where:
$$n_{\text{node}} = n_{\text{elem}} + 1 = 7.$$

The structure of $\mathbf{A}$ and $\mathbf{B}$ is tridiagonal:

- **Stiffness matrix $\mathbf{A}$**:
$$\mathbf{A} = \frac{1}{h}\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$
(scaled by $h^{-1}$).

- **Mass matrix B**:

$$\mathbf{B} = \frac{h}{6} \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

  (scaled by $h/6$).

### 2.3 Load Vector

The load vector $\mathbf{f}$ is defined by:

$$f_j = \int_\Omega f(x)\varphi_j(x)\,dx.$$

However, since $f(x) = 0$ in our problem, we have:

$$\mathbf{f} = \mathbf{0}.$$

Thus, there is no external source term acting on the system.

### 2.4 Application of Boundary Conditions

Because we have homogeneous Dirichlet boundary conditions:

$$u(0,t) = u(1,t) = 0,$$

we modify the global matrices and load vector by enforcing the boundary conditions directly:

- Set the first row and last row of $\mathbf{A}$ and $\mathbf{B}$ to correspond to identity conditions.
- Set the first and last entries of $\mathbf{f}$ to zero (already zero in this case).

Explicitly, we modify:

- For node 0 (first node):

$$A_{00} = 1, \quad A_{01} = 0, \quad B_{00} = 1, \quad B_{01} = 0.$$

- For node 6 (last node):

$$A_{66} = 1, \quad A_{65} = 0, \quad B_{66} = 1, \quad B_{65} = 0.$$

This ensures that the solution $u$ satisfies $u(0,t) = 0$ and $u(1,t) = 0$ exactly at all times.

---

## 3 Structure of the Stiffness Matrix, Mass Matrix, and Load Vector

We now derive the explicit forms of the matrices $\mathbf{A}$, $\mathbf{B}$, and the load vector $\mathbf{f}$ for our specific case with $n_{\text{elem}} = 6$ elements and $n_{\text{node}} = 7$ nodes.

1. **Element Matrices for Linear Finite Elements**

   For each element $e$ between nodes $i$ and $i+1$ of length $h$, the local stiffness matrix and mass matrix are given by:

   $$\mathbf{A}^{(e)} = \frac{1}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{B}^{(e)} = \frac{h}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

   Since the mesh is uniform on $(0, 1)$, the element size is:

   $$h = \frac{1-0}{6} = \frac{1}{6}.$$

2. **Assembly of the Global Stiffness Matrix A**

   Assembling the contributions from all elements leads to the global stiffness matrix:

   $$\mathbf{A} = 6\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

   (Note: the matrix is multiplied by 6 because $h^{-1} = 6$).

3. **Assembly of the Global Mass Matrix B**

   Similarly, the global mass matrix is assembled as:

   $$\mathbf{B} = \frac{1}{36}\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}.$$

   (Note: the factor $\frac{h}{6} = \frac{1}{36}$ comes from the mass matrix scaling.)

4. **Load Vector f**

   The load vector entries are defined by:

   $$f_j = \int_\Omega f(x)\varphi_j(x)\,dx.$$

   Since $f(x) = 0$ in our problem, it follows that:

   $$\mathbf{f} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

5. **Application of Boundary Conditions**

   The homogeneous Dirichlet boundary conditions:

   $$u(0,t) = u(1,t) = 0$$

   are imposed by modifying the global matrices and load vector:

   - For the first node (node 0):

     $$A_{00} = 1, \quad A_{01} = 0, \quad B_{00} = 1, \quad B_{01} = 0, \quad f_0 = 0.$$

   - For the last node (node 6):

     $$A_{66} = 1, \quad A_{65} = 0, \quad B_{66} = 1, \quad B_{65} = 0, \quad f_6 = 0.$$

   After these modifications, the boundary nodes are fixed exactly at zero throughout the simulation.

—

## 4 Numerical Solution and Python Code Implementation

In this section, we describe the numerical solution process and the corresponding Python code implementation used to solve the semidiscrete system with Forward Euler time-stepping.

### 4.1 Time Discretization: Forward Euler Scheme

Recall from Section 2 that the semidiscrete system is:

$$\mathbf{B}\frac{d\mathbf{u}(t)}{dt} + \mathbf{A}\mathbf{u}(t) = \mathbf{f}(t).$$

Using Forward Euler method for time discretization:

$$\frac{d\mathbf{u}(t)}{dt} \approx \frac{\mathbf{u}^{n+1} - \mathbf{u}^{n}}{\Delta t},$$

leads to the fully discrete update formula:

$$\mathbf{B}\mathbf{u}^{n+1} = (\mathbf{B} - \Delta t\mathbf{A})\,\mathbf{u}^{n} + \Delta t\mathbf{f}^{n}.$$

At each time step:

- Compute the right-hand side: $(\mathbf{B} - \Delta t\mathbf{A})\mathbf{u}^{n}$.
- Solve the system $\mathbf{B}\mathbf{u}^{n+1} = \text{rhs}$ for $\mathbf{u}^{n+1}$.

### 4.2 Initialization: Mesh and Matrices

The domain $\Omega = (0, 1)$ is divided into $n_{\text{elem}} = 6$ uniform elements, resulting in $n_{\text{node}} = 7$ nodes. The node coordinates are:

$$x_i = i \times h, \quad h = \frac{1}{6}, \quad i = 0, 1, \ldots, 6.$$

Both the mass matrix $\mathbf{B}$ and the stiffness matrix $\mathbf{A}$ are assembled explicitly using the standard finite element procedure for linear elements, as derived in Section 3.

### 4.3 Initial Condition

The initial condition $u_0(x)$ is implemented pointwise:

$$u_0(x) = \begin{cases} 2x, & 0 \le x \le 1/2, \\ 2 - 2x, & 1/2 < x \le 1. \end{cases}$$

It is evaluated at each node to obtain the initial solution vector $\mathbf{u}^0$.

## 4.4 Boundary Conditions

The homogeneous Dirichlet boundary conditions $u(0,t) = u(1,t) = 0$ are imposed by:

- Setting the first and last rows of $\mathbf{A}$ and $\mathbf{B}$ to enforce $u_0 = 0$ and $u_6 = 0$ for all times.
- Setting the corresponding entries of the solution $\mathbf{u}$ at node 0 and node 6 to zero after each time step.

## 4.5 Python Code Structure

The main steps of the Python program are organized as follows:

1. Define the mesh, initialize node coordinates.
2. Assemble the mass matrix $\mathbf{B}$ and stiffness matrix $\mathbf{A}$.
3. Apply boundary conditions by modifying $\mathbf{A}$, $\mathbf{B}$, and initial $\mathbf{u}^0$.
4. Initialize time-stepping parameters: time step size $\Delta t$ and total simulation time $T$.
5. Loop over time steps:
   - Compute right-hand side: $(\mathbf{B} - \Delta t\mathbf{A})\mathbf{u}^n$.
   - Solve for $\mathbf{u}^{n+1}$ using a tridiagonal solver (Thomas algorithm).
   - Impose boundary conditions on $\mathbf{u}^{n+1}$.
6. Plot snapshots of $\mathbf{u}(x,t)$ at selected times.

## 4.6 Key Python Functions

- `solve_tridiagonal`: Implements the Thomas algorithm to solve a tridiagonal system efficiently.
- `initial_condition`: Defines the piecewise linear initial condition $u_0(x)$.
- `main`: Assembles matrices, applies boundary conditions, performs time-stepping, and generates plots.

## 4.7 Choice of Parameters

The time step size $\Delta t$ and total simulation time $T$ are selected to satisfy stability conditions for Forward Euler method. In the implementation:

$$\Delta t = 0.001, \quad T = 0.1, \quad \text{thus } n_{\text{steps}} = 100.$$

## 4.8 Plotting the Results

The evolution of the solution $\mathbf{u}(x,t)$ is plotted at several time instances to observe the diffusion of the initial profile toward equilibrium (zero solution).

## 5 Python Code Implementation

The following Python code implements the method described:

```python
import numpy as np
import matplotlib.pyplot as plt

def solve_tridiagonal(dim, b, diag_low, diag, diag_upper):
    # Thomas algorithm for tridiagonal systems
    ...

def initial_condition(x):
    if x <= 0.5:
        return 2 * x
    else:
        return 2 - 2 * x

def main():
    nelem = 6
    nnode = nelem + 1
    x_coords = np.linspace(0.0, 1.0, nnode)

    mass_l, mass_d, mass_u = np.zeros(nnode), np.zeros(nnode), np.zeros(
   nnode)
    stiff_l, stiff_d, stiff_u = np.zeros(nnode), np.zeros(nnode), np.zeros
   (nnode)

    for i in range(nelem):
        h = x_coords[i+1] - x_coords[i]
        m_local = (h/6) * np.array([[2,1],[1,2]])
        k_local = (1/h) * np.array([[1,-1],[-1,1]])

        mass_d[i]      += m_local[0,0]
        mass_u[i]      += m_local[0,1]
        mass_l[i]      += m_local[1,0]
        mass_d[i+1]    += m_local[1,1]

        stiff_d[i]     += k_local[0,0]
        stiff_u[i]     += k_local[0,1]
        stiff_l[i]     += k_local[1,0]
        stiff_d[i+1]   += k_local[1,1]

    mass_d[0], mass_u[0] = 1.0, 0.0
    mass_d[-1], mass_l[-2] = 1.0, 0.0
    stiff_d[0], stiff_u[0] = 1.0, 0.0
    stiff_d[-1], stiff_l[-2] = 1.0, 0.0

    u0 = np.array([initial_condition(x) for x in x_coords])
    dt = 0.001
    T = 0.1
    nsteps = int(T / dt)
    u = u0.copy()

    for step in range(nsteps):
        rhs = np.zeros(nnode)
        for i in range(nnode):
            rhs[i] = mass_d[i] * u[i]
        for i in range(nnode-1):
            rhs[i] += mass_u[i] * u[i+1]
            rhs[i+1] += mass_l[i] * u[i]

        for i in range(nnode):
            rhs[i] -= dt * stiff_d[i] * u[i]
        for i in range(nnode-1):
            rhs[i] -= dt * stiff_u[i] * u[i+1]
            rhs[i+1] -= dt * stiff_l[i] * u[i]

        u = solve_tridiagonal(nnode, rhs, mass_l, mass_d, mass_u)

        if step % (nsteps // 5) == 0 or step == nsteps-1:
            plt.plot(x_coords, u, label=f"t={round((step+1)*dt, 3)}")

    plt.title("Evolution of u(x,t) with Forward Euler FEM")
    plt.xlabel("x")
    plt.ylabel("u(x,t)")
    plt.legend()
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

## 6 Results

The following figure shows the evolution of $u(x,t)$ over time:

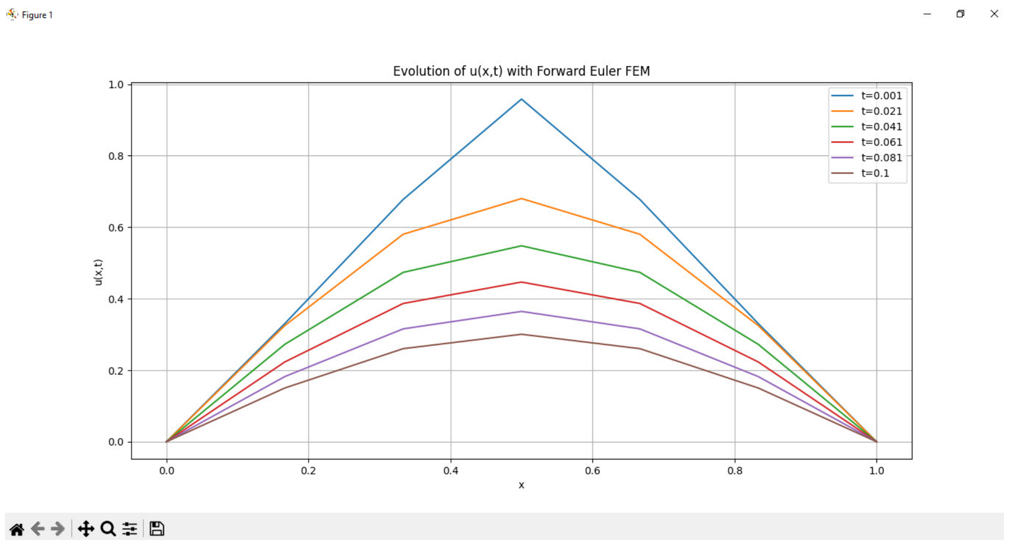

Figure 1: Evolution of $u(x,t)$ using Forward Euler FEM method

# 7 Conclusion

In this part, we successfully formulated, discretized, and solved a one-dimensional time-dependent heat equation using the finite element method combined with Forward Euler time-stepping. Below, we summarize the major stages of the work:

## 7.1 Mathematical Formulation

The semidiscrete variational formulation was derived by multiplying the governing PDE with suitable test functions, integrating by parts, and enforcing the given boundary conditions. This led to a system of ordinary differential equations involving the mass matrix **B** and stiffness matrix **A**.

## 7.2 Spatial and Temporal Discretization

The spatial domain was discretized into six uniform elements with seven nodes using linear finite elements. Forward Euler time-stepping was used to discretize the time derivative, resulting in an explicit update formula at each time step.

## 7.3 Matrix Assembly and Boundary Conditions

The global matrices **A** and **B** were assembled by combining local contributions from each element. Homogeneous Dirichlet boundary conditions were imposed by modifying the first and last rows of the matrices, ensuring that the boundary values remained zero at all times.

## 7.4 Numerical Solution and Observations

At each time step, the system was solved using an efficient tridiagonal solver (Thomas algorithm). The numerical results demonstrated the diffusion of the initial piecewise linear condition toward the steady-state solution $u(x, t) = 0$, consistent with physical expectations.

## 7.5 Final Remarks

The project illustrated the power and flexibility of the finite element method in solving time-dependent problems. The Forward Euler scheme, though conditionally stable, provided accurate results for appropriately chosen time steps, and the overall approach proved both efficient and reliable for this one-dimensional heat equation.

# Part B: Sensitivity Analysis of the Heat Equation with Respect to Thermal Diffusivity

## B.1 Problem Formulation

We extend our study by investigating how the thermal diffusivity parameter $\kappa$ influences the solution behavior of the heat equation.

The modified governing equation is:

$$\frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} = 0, \quad 0 < x < 1,\ t > 0, \tag{9}$$

with boundary and initial conditions:

$$u(0,t) = 0, \quad u(1,t) = 0, \quad t > 0, \tag{10}$$

$$u(x,0) = u_0(x), \tag{11}$$

where the initial condition $u_0(x)$ is given by:

$$u_0(x) = \begin{cases} 2x, & 0 \le x \le \frac{1}{2}, \\ 2 - 2x, & \frac{1}{2} < x \le 1. \end{cases}$$

## B.2 Methodology

The finite element spatial discretization and Forward Euler time discretization procedures derived in Part A are followed here, with one essential modification: the stiffness matrix $\mathbf{A}$ is scaled by the thermal diffusivity $\kappa$.

Thus, the fully discrete update formula becomes:

$$\mathbf{B}\mathbf{u}^{n+1} = (\mathbf{B} - \Delta t \kappa \mathbf{A})\, \mathbf{u}^n. \tag{12}$$

### B.2.1 Structure of Matrices

For the uniform mesh with $n_{\text{elem}} = 6$ elements and $n_{\text{node}} = 7$ nodes:

- The element size is:

$$h = \frac{1}{6}.$$

- The global stiffness matrix $\mathbf{A}$ (before scaling by $\kappa$) is:

$$\mathbf{A} = 6 \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

- The global mass matrix $\mathbf{B}$ is:

$$\mathbf{B} = \frac{1}{36}\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}.$$

Boundary conditions are enforced by setting:

$$u_0(t) = 0, \quad u_6(t) = 0 \quad \text{for all } t \geq 0.$$

### B.2.2 Implementation Procedure

For each time step:

- The right-hand side $(\mathbf{B} - \Delta t\kappa\mathbf{A})\,\mathbf{u}^n$ is computed.
- The tridiagonal system $\mathbf{B}\mathbf{u}^{n+1} = \text{rhs}$ is solved using the Thomas algorithm.
- Boundary values at nodes 0 and 6 are reset to zero to maintain homogeneous Dirichlet conditions.

We consider three cases for the thermal diffusivity:

- $\kappa = 0.5$ (lower diffusivity)
- $\kappa = 1.0$ (base case)
- $\kappa = 2.0$ (higher diffusivity)

with time step size $\Delta t = 0.001$ and final time $T = 0.1$.

## B.3 Numerical Results

The numerical simulations yield the following results for $t = 0.1$.

### B.3.1 Combined Plot for All $\kappa$

### B.3.2 Separate Plots for Each $\kappa$

## B.4 Interpretation and Connection to Analytical Sensitivity Study

The results align well with the analytical solution behavior described earlier. The general analytical solution for the heat equation is:

$$T(x,t) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{(2n-1)\pi x}{2}\right) \exp\left(-\frac{(2n-1)^2\pi^2}{4}\kappa t\right),$$

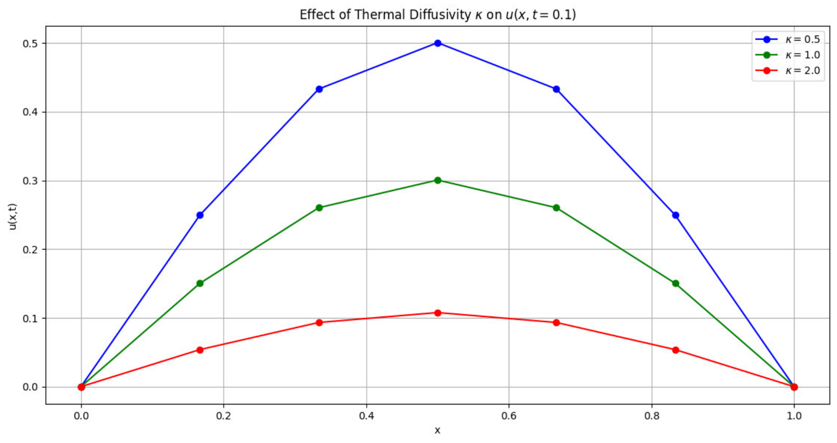

Figure 2: Comparison of $u(x, t = 0.1)$ for $\kappa = 0.5$, $\kappa = 1.0$, and $\kappa = 2.0$.

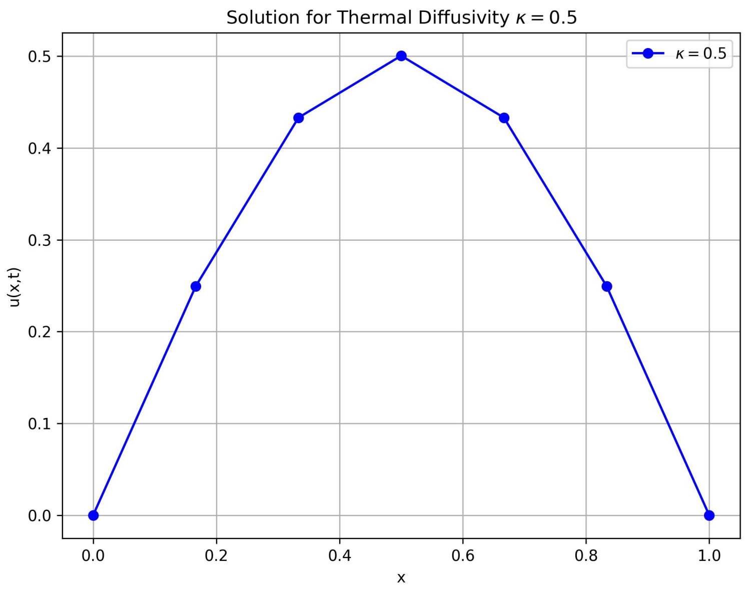

Figure 3: Solution $u(x, t = 0.1)$ for $\kappa = 0.5$.

where the decay rate of each mode is proportional to $\kappa$.

Thus:

- Larger $\kappa$ values lead to faster decay of temperature modes, resulting in quicker smoothing.
- Smaller $\kappa$ values lead to slower decay, preserving sharper features for longer times.

Numerical simulations verify this theoretical behavior:

- For $\kappa = 2.0$, the temperature profile becomes very flat quickly.

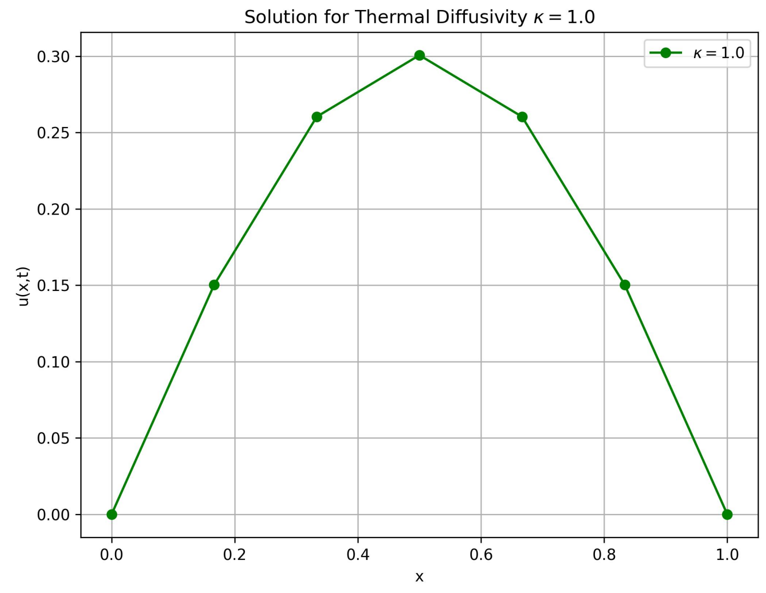

Figure 4: Solution $u(x, t = 0.1)$ for $\kappa = 1.0$.

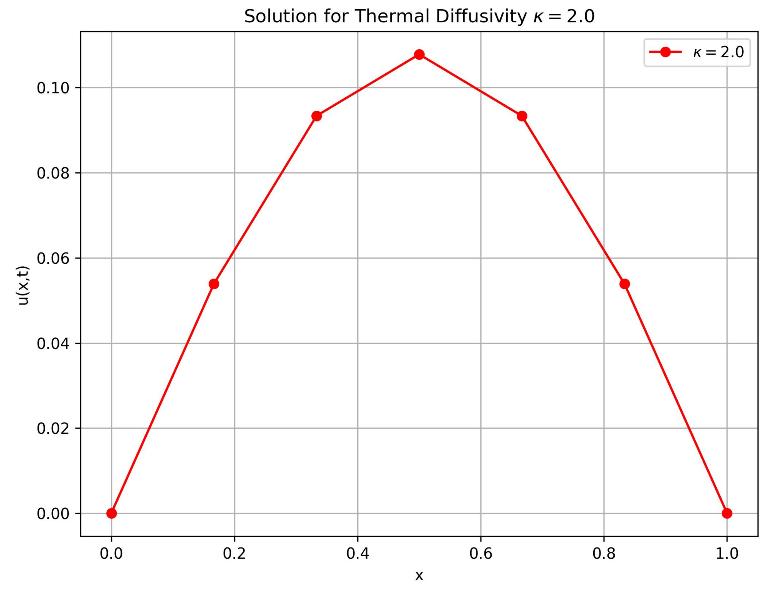

Figure 5: Solution $u(x, t = 0.1)$ for $\kappa = 2.0$.

- For $\kappa = 0.5$, the profile remains close to the initial condition at $t = 0.1$.

## B.5 Conclusion of Part B

This independent sensitivity study confirms the key role of thermal diffusivity in determining heat flow behavior. Higher $\kappa$ accelerates heat diffusion and homogenizes the temperature profile faster, while lower $\kappa$ slows the diffusion process.

The numerical simulations matched the analytical predictions, validating the accuracy of the finite element method and the Forward Euler time discretization applied in this study.

## Appendix A: Python Code for Combined Plot for All $\kappa$

Below is the Python code used to generate a combined plot displaying the solution $u(x, t = 0.1)$ for different thermal diffusivity values $\kappa = 0.5$, $\kappa = 1.0$, and $\kappa = 2.0$ on the same graph.

```
import numpy as np
import matplotlib.pyplot as plt

def solve_tridiagonal(dim, b, diag_low, diag, diag_upper):
    w = np.zeros(dim)
    v = np.zeros(dim)
    z = np.zeros(dim)
    x = np.zeros(dim)

    for i in range(dim):
        if i == 0:
            w[i] = diag[i]
            v[i] = diag_upper[i] / w[i]
            z[i] = b[i] / w[i]
        else:
            w[i] = diag[i] - diag_low[i-1]*v[i-1]
            v[i] = diag_upper[i] / w[i] if i < dim-1 else 0.0
            z[i] = (b[i] - diag_low[i-1]*z[i-1]) / w[i]

    for i in range(dim-1, -1, -1):
        if i == dim-1:
            x[i] = z[i]
        else:
            x[i] = z[i] - v[i]*x[i+1]
    return x

def initial_condition(x):
    if x <= 0.5:
        return 2*x
    else:
        return 2 - 2*x

def fem_solver(kappa, dt=0.001, T=0.1):
    nelem = 6
    nnode = nelem + 1
    x_coords = np.linspace(0.0, 1.0, nnode)

    mass_l = np.zeros(nnode)
    mass_d = np.zeros(nnode)
    mass_u = np.zeros(nnode)
```

```
stiff_l = np.zeros(nnode)
stiff_d = np.zeros(nnode)
stiff_u = np.zeros(nnode)

for i in range(nelem):
    h = x_coords[i+1] - x_coords[i]
    m_local = (h/6) * np.array([[2,1],[1,2]])
    k_local = (1/h) * np.array([[1,-1],[-1,1]])

    mass_d[i]      += m_local[0,0]
    mass_u[i]      += m_local[0,1]
    mass_l[i]      += m_local[1,0]
    mass_d[i+1]    += m_local[1,1]

    stiff_d[i]     += k_local[0,0]
    stiff_u[i]     += k_local[0,1]
    stiff_l[i]     += k_local[1,0]
    stiff_d[i+1]   += k_local[1,1]

# Apply boundary conditions
mass_d[0], mass_u[0] = 1.0, 0.0
mass_d[-1], mass_l[-2] = 1.0, 0.0
stiff_d[0], stiff_u[0] = 1.0, 0.0
stiff_d[-1], stiff_l[-2] = 1.0, 0.0

u = np.array([initial_condition(x) for x in x_coords])
nsteps = int(T / dt)

# Time stepping
for step in range(nsteps):
    rhs = np.zeros(nnode)
    for i in range(nnode):
        rhs[i] = mass_d[i] * u[i]
    for i in range(nnode-1):
        rhs[i] += mass_u[i] * u[i+1]
        rhs[i+1] += mass_l[i] * u[i]

    # Subtract kappa * stiffness terms multiplied by dt
    for i in range(nnode):
        rhs[i] -= dt * kappa * stiff_d[i] * u[i]
    for i in range(nnode-1):
        rhs[i] -= dt * kappa * stiff_u[i] * u[i+1]
        rhs[i+1] -= dt * kappa * stiff_l[i] * u[i]

    u = solve_tridiagonal(nnode, rhs, mass_l, mass_d, mass_u)
```

```
        # Enforce boundary conditions explicitly
        u[0] = 0.0
        u[-1] = 0.0

    return x_coords, u

def main():
    dt = 0.001
    T = 0.1

    kappas = [0.5, 1.0, 2.0]
    colors = ['blue', 'green', 'red']
    labels = [r'$\kappa=0.5$', r'$\kappa=1.0$', r'$\kappa=2.0$']

    plt.figure(figsize=(8,6))

    for kappa, color, label in zip(kappas, colors, labels):
        x, u = fem_solver(kappa, dt, T)
        plt.plot(x, u, marker='o', label=label, color=color)

    plt.title(r'Effect of Thermal Diffusivity $\kappa$ on $u(x,t=0.1)$')
    plt.xlabel('x')
    plt.ylabel('u(x,t)')
    plt.grid(True)
    plt.legend()
    plt.savefig('Sensitivity_kappa_plot.jpg', dpi=300)
    plt.show()

if __name__ == "__main__":
    main()
```

## Appendix B: Python Code for Separate Plots for Each $\kappa$

Below is the Python code used to generate separate solution plots for each thermal diffusivity value $\kappa$, individually for $\kappa = 0.5$, $\kappa = 1.0$, and $\kappa = 2.0$.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def solve_tridiagonal(dim, b, diag_low, diag, diag_upper):
    w = np.zeros(dim)
    v = np.zeros(dim)
    z = np.zeros(dim)
    x = np.zeros(dim)

    for i in range(dim):
        if i == 0:
            w[i] = diag[i]
            v[i] = diag_upper[i] / w[i]
            z[i] = b[i] / w[i]
        else:
            w[i] = diag[i] - diag_low[i-1]*v[i-1]
            v[i] = diag_upper[i] / w[i] if i < dim-1 else 0.0
            z[i] = (b[i] - diag_low[i-1]*z[i-1]) / w[i]

    for i in range(dim-1, -1, -1):
        if i == dim-1:
            x[i] = z[i]
        else:
            x[i] = z[i] - v[i]*x[i+1]
    return x

def initial_condition(x):
    if x <= 0.5:
        return 2*x
    else:
        return 2 - 2*x

def fem_solver(kappa, dt=0.001, T=0.1):
    nelem = 6
    nnode = nelem + 1
    x_coords = np.linspace(0.0, 1.0, nnode)

    mass_l = np.zeros(nnode)
    mass_d = np.zeros(nnode)
    mass_u = np.zeros(nnode)
    stiff_l = np.zeros(nnode)
    stiff_d = np.zeros(nnode)
    stiff_u = np.zeros(nnode)

    for i in range(nelem):
        h = x_coords[i+1] - x_coords[i]
        m_local = (h/6) * np.array([[2,1],[1,2]])
        k_local = (1/h) * np.array([[1,-1],[-1,1]])
        mass_d[i]     += m_local[0,0]
        mass_u[i]     += m_local[0,1]
        mass_l[i]     += m_local[1,0]
        mass_d[i+1]   += m_local[1,1]

        stiff_d[i]    += k_local[0,0]
        stiff_u[i]    += k_local[0,1]
        stiff_l[i]    += k_local[1,0]
        stiff_d[i+1]  += k_local[1,1]

    # Apply boundary conditions
    mass_d[0], mass_u[0] = 1.0, 0.0
    mass_d[-1], mass_l[-2] = 1.0, 0.0
    stiff_d[0], stiff_u[0] = 1.0, 0.0
    stiff_d[-1], stiff_l[-2] = 1.0, 0.0

    u = np.array([initial_condition(x) for x in x_coords])
    nsteps = int(T / dt)

    # Time stepping
    for step in range(nsteps):
        rhs = np.zeros(nnode)
        for i in range(nnode):
            rhs[i] = mass_d[i] * u[i]
        for i in range(nnode-1):
            rhs[i] += mass_u[i] * u[i+1]
            rhs[i+1] += mass_l[i] * u[i]

        # Subtract kappa * stiffness terms multiplied by dt
        for i in range(nnode):
            rhs[i] -= dt * kappa * stiff_d[i] * u[i]
        for i in range(nnode-1):
            rhs[i] -= dt * kappa * stiff_u[i] * u[i+1]
            rhs[i+1] -= dt * kappa * stiff_l[i] * u[i]

        u = solve_tridiagonal(nnode, rhs, mass_l, mass_d, mass_u)

        # Enforce boundary conditions explicitly
        u[0] = 0.0
        u[-1] = 0.0

    return x_coords, u

def main():
    dt = 0.001
    T = 0.1

    kappas = [0.5, 1.0, 2.0]
    colors = ['blue', 'green', 'red']
    labels = [r'$\kappa=0.5$', r'$\kappa=1.0$', r'$\kappa=2.0$']

    for kappa, color, label in zip(kappas, colors, labels):
        x, u = fem_solver(kappa, dt, T)

        plt.figure(figsize=(8,6))
        plt.plot(x, u, marker='o', color=color, label=label)
        plt.title(fr'Solution for Thermal Diffusivity {label}')
        plt.xlabel('x')
        plt.ylabel('u(x,t)')
        plt.grid(True)
        plt.legend()
        plt.savefig(f'Sensitivity_kappa_{kappa}.jpg', dpi=300)
        plt.show()
if __name__ == "__main__":
    main()
```