# Homework 4

*Due Date: November 21, 2025*

## 2D Heat Equation with Finite Element Method

Consider the time-dependent heat equation as follows:

$$\frac{\partial u}{\partial t} - \nu \Delta u = f(x, y, t) \quad \text{in} \quad \Omega = (-2, 2) \times (-2, 2), \quad t \in (0, 1]$$

with diffusion coefficient $\nu = 0.05$ and corresponding homogeneous Dirichlet boundary conditions:

$$u(x, y, \cdot) = 0 \quad \text{for} \quad (x, y) \in \partial\Omega$$

We assume the exact solution is given by:

$$u_{exact}(x, y, t) = e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

We will be using rectangular elements. You will use the bilinear $Q_1$ element as your basis function to solve the heat equation. The corresponding four shape functions defined in the reference element $(\xi, \eta) \in (-1, 1) \times (-1, 1)$ are:

$$\Phi_1(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 - \eta), \quad \Phi_2(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 - \eta),$$

$$\Phi_3(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 + \eta), \quad \Phi_4(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 + \eta)$$

# Preliminary Setup

This is a general outline of the finite element method for solving the 2D heat equation. The specifics for our problem will be addressed in the subsequent questions.

## Weak Formulation

Let $v$ be a test function belonging to the function space:

$$V = \{v \in H_0^1(\Omega) \mid v, v' \in L^2(\Omega)\}$$

Note that $v = 0$ on $\partial\Omega$. Multiplying the PDE by $v$ and integrating over the domain $\Omega$, we have:

$$\int_\Omega u_t v - \nu v \Delta u \; dx = \int_\Omega f v \; dx$$

Integrating by parts on the second term of the left-hand side:

$$\int_\Omega u_t v \; dx + \nu[\int_\Omega \nabla u \cdot \nabla v \; dx - \int_{\partial\Omega} v(\nabla u \cdot n) \; ds] = \int_\Omega f v \; dx$$

Since $v = 0$ on $\partial\Omega$, the boundary integral vanishes. Therefore, the weak formulation is given by:

$$\int_\Omega u_t v \; dx + \nu \int_\Omega \nabla u \cdot \nabla v \; dx = \int_\Omega f v \; dx$$

Where $u, v \in V$. Denote the left hand side as $a(u, v)$ and the right hand side as $L(v)$.

## Discretization and Global System

We discretize the spatial domain $\Omega$ into rectangular elements. Let $V_h \subset V$ be the finite-dimensional subspace spanned by the basis functions $\{\phi_i\}_{i=1}^N$, where $N$ is the total number of nodes in the mesh (will be reduced later on based on BC). Each bilinear $\phi_n$ corresponds to some node $(x_i, y_j)$ satisfying $\phi_{i,j} = \delta_{i,j}$. We assume $u_h \in V_h$ satisfies the weak formulation $a(u_h, v_h) = L(v_h)$ for all $v_h \in V_h$.

Approximate the solution of $u$ as:

$$u_h = \sum_{j=1}^N U_j(t)\phi_j(x, y)$$

where $U_j$ are time-dependent coefficients to be determined. The test function $v$ is also chosen from the same space and discretized similarly:

$$v_h = \sum_{i=1}^N V_i(t)\phi_i(x, y)$$

Substituting these approximations into the weak formulation, we obtain the system:

$$\sum_{i,j=1}^N V_j \left(\int_\Omega \phi_i \phi_j \; dx\right) \frac{dU_i}{dt} + \nu \sum_{i,j=1}^N V_j \left(\int_\Omega \nabla\phi_i \cdot \nabla\phi_j \; dx\right) U_i = \sum_{j=1}^N V_j \int_\Omega f \phi_j \; dx$$

We may factor out the $V_j$ to give us:

$$\sum_{i,j=1}^N \left(\int_\Omega \phi_i \phi_j \; dx\right) \frac{dU_i}{dt} + \nu \sum_{i,j=1}^N \left(\int_\Omega \nabla\phi_i \cdot \nabla\phi_j \; dx\right) U_i = \sum_{j=1}^N \int_\Omega f \phi_j \; dx$$

More compactly, let:

$$U = [U_1, U_2, \dots, U_N]^T, \quad M_{ij} = \int_\Omega \phi_i \phi_j \, dx, \quad K_{ij} = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, dx, \quad F_j = \int_\Omega f \phi_j \, dx$$

such that $M = (M_{ij})$, $K = (K_{ij})$, and $F = [F_1, F_2, \dots, F_N]^T$. We can rewrite the system as:

$$M \frac{dU}{dt} + \nu K U = F$$

where $M$ is the mass matrix, $K$ is the stiffness matrix, and $F$ is the force vector.

## Elemental-Level Systems and Assembly

We suppose element-wise, each $u^{(e)}$ satisfies the weak formulation over its own domain $\Omega^{(e)}$:

$$\int_{\Omega^{(e)}} u_t^{(e)} v^{(e)} \, dx + \nu \int_{\Omega^{(e)}} \nabla u^{(e)} \cdot \nabla v^{(e)} \, dx = \int_{\Omega^{(e)}} f v^{(e)} \, dx$$

We suppose the local approximations are given by:

$$u_h^{(e)} = \sum_{j=1}^{4} U_j^{(e)} \phi_j^{(e)}(x, y), \quad v_h^{(e)} = \sum_{i=1}^{4} V_i^{(e)} \phi_i^{(e)}(x, y)$$

since we have rectangular elements with four nodes each. Using the same process as before, we can derive the elemental system:

$$M^{(e)} \frac{dU^{(e)}}{dt} + \nu K^{(e)} U^{(e)} = F^{(e)}$$

We can express the global matrices and vector as sums over all elements:

$$M = \sum_{e=1}^{E} M^{(e)}, \quad K = \sum_{e=1}^{E} K^{(e)}, \quad F = \sum_{e=1}^{E} F^{(e)}$$

where $E$ is the total number of elements, and the elemental matrices and vector are defined as:

$$M_{ij}^{(e)} = \int_{\Omega^{(e)}} \phi_i^{(e)} \phi_j^{(e)} \, dx, \quad K_{ij}^{(e)} = \int_{\Omega^{(e)}} \nabla \phi_i^{(e)} \cdot \nabla \phi_j^{(e)} \, dx, \quad F_j^{(e)} = \int_{\Omega^{(e)}} f \phi_j^{(e)} \, dx$$

Here, $\Omega^{(e)}$ is the domain of element $e$, and $\phi_i^{(e)}$ are the local shape functions associated with element $e$.

We can use quadrature to numerically compute the integrals for $M^{(e)}$, $K^{(e)}$, and $F^{(e)}$ on each element, then assemble them into the global system.

# Question 1

By substituting $u_{exact}$ into the PDE, determine the forcing term $f(x, y, t)$ such that:

$$\frac{\partial u_{exact}}{\partial t} - \nu \Delta u_{exact} = f(x, y, t)$$

*Solution.* For the purposes of this question, denote $u = u_{exact}$. Where $u$ is the exact solution given by:

$$u(x, y, t) = e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

Let us first compute the time derivative:

$$\frac{\partial u}{\partial t} = -8\pi^2 \nu e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

Next, we want to find the Laplacian. Computing the first and second derivative with respect to $x$:

$$\frac{\partial u}{\partial x} = 2\pi e^{-8\pi^2 \nu t} \cos(2\pi x) \sin(2\pi y)$$

$$\frac{\partial^2 u}{\partial x^2} = -4\pi^2 e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

The 2nd derivative with respect to $y$ is the same:

$$\frac{\partial^2 u}{\partial y^2} = -4\pi^2 e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

Therefore, the Laplacian is:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -8\pi^2 e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)$$

Substituting these results into the heat equation, we have:

$$\frac{\partial u}{\partial t} - \nu \Delta u = -8\pi^2 \nu e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y) - \nu(-8\pi^2 e^{-8\pi^2 \nu t} \sin(2\pi x) \sin(2\pi y)) = 0$$

So our forcing term is:

$$f(x, y, t) = 0$$

and we are working with the homogeneous heat equation.

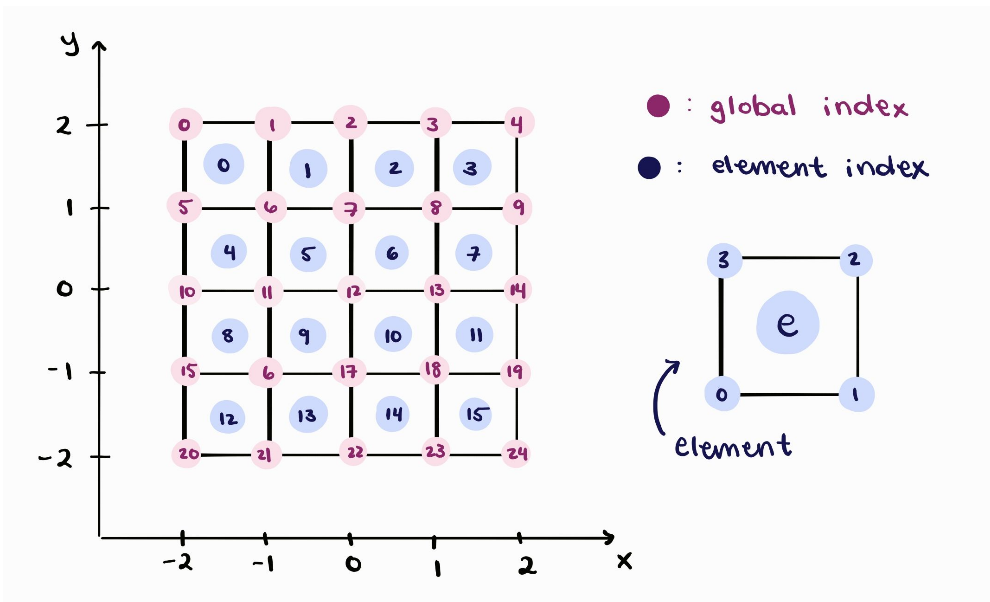
# Question 2

Discretize the spacial domain $\Omega$ into $16$ equal square elements arranged in a $4 \times 4$ grid, with the node coordinates:

$$(x, y) \in \{-2, -1, 0, 1, 2\} \times \{-2, -1, 0, 1, 2\}$$

Draw this mesh, define your own global numbering, label all global node numbers, and generate corresponding elemental connectivities.

*Solution.* All indexing used will start from $0$. The mesh is as follows:




Globally, we have $25$ nodes numbered from $0$ to $24$. Numbering starts from the top-left corner and goes row-wise. The elements are numbered from $0$ to $15$, also row-wise in the same fashion. For each element, its indicies $(0, 1, 2, 3)$ start from the bottom-left and go counter-clockwise to match the definition of the bilinear $Q1$ element. The folowing code was used to generate the connectivity matrix:

Listing 1: Question 2 Code

```
import numpy as np
import sympy as sp

def global_indexing(width, height=None, include_boundary=False):
    if height is None:
        height = width
    if include_boundary:
        return np.arange(width * height).reshape((width, height))
    return np.arange((width-2)*(height-2)).reshape((width-2, height-2))

def generate_connectivity_matrix(global_indices):
```

```python
    total_elements = (global_indices.shape[0] - 1) * (global_indices.shape[1] - 1)
    connectivity_matrix = np.zeros((total_elements, 4), dtype=int)
    element = 0
    for i in range(global_indices.shape[0] - 1):
        for j in range(global_indices.shape[1] - 1):
            connectivity_matrix[element, 0] = global_indices[i+1, j]
            connectivity_matrix[element, 1] = global_indices[i+1, j+1]
            connectivity_matrix[element, 2] = global_indices[i, j+1]
            connectivity_matrix[element, 3] = global_indices[i, j]
            element += 1
    return connectivity_matrix

if __name__ == "__main__":
    width = 5  # Number of nodes along one dimension
    global_with_boundary = global_indexing(width, include_boundary=True)
    connectivity_matrix_with_boundary = generate_connectivity_matrix(
    global_with_boundary)

    with open("./outputs_4/matrices.txt", "w") as f:
        latex_matrix = sp.latex(sp.Matrix(connectivity_matrix_with_boundary))
        f.write("Connectivity Matrix with Boundary:\n")
        f.write(latex_matrix + "\n\n")
```

The elemental connectivities are as follows:

| Element | Node 0 | Node 1 | Node 2 | Node 3 |
|---|---|---|---|---|
| 0 | 5 | 6 | 1 | 0 |
| 1 | 6 | 7 | 2 | 1 |
| 2 | 7 | 8 | 3 | 2 |
| 3 | 8 | 9 | 4 | 3 |
| 4 | 10 | 11 | 6 | 5 |
| 5 | 11 | 12 | 7 | 6 |
| 6 | 12 | 13 | 8 | 7 |
| 7 | 13 | 14 | 9 | 8 |
| 8 | 15 | 16 | 11 | 10 |
| 9 | 16 | 17 | 12 | 11 |
| 10 | 17 | 18 | 13 | 12 |
| 11 | 18 | 19 | 14 | 13 |
| 12 | 20 | 21 | 16 | 15 |
| 13 | 21 | 22 | 17 | 16 |
| 14 | 22 | 23 | 18 | 17 |
| 15 | 23 | 24 | 19 | 18 |

# Question 3

For one physical element $u^{(e)}$, write the mapping from the reference element $(\xi, \eta) \in (-1,1) \times (-1,1)$ to the physical coordinates $(x, y)$ in terms of the nodal coordinates $(x_n, y_n)$ and the shape functions $\Phi_n(\xi, \eta)$. Also, derive the Jacobian matrix $J(\xi, \eta)$ of this mapping.

*Solution.* Reindexing the four shape functions to fit our indexing scheme for the element nodes, we have:

$$\Phi_0(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad \Phi_1(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta),$$
$$\Phi_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad \Phi_3(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

On a physical element $e$, it is a rectangle of the region $[x_0, x_1] \times [y_0, y_1]$ where $(x_0, y_0)$ is the bottom-left corner and $(x_1, y_1)$ is the top-right corner. The mapping from $(\xi, \eta) \mapsto (x, y)$ should be given by the standard change of variables:

$$\begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(x_1 - x_0)\xi + \frac{1}{2}(x_1 + x_0) \\ \frac{1}{2}(y_1 - y_0)\eta + \frac{1}{2}(y_1 + y_0) \end{bmatrix} \tag{1}$$

We will verify this using the shape functions. We assume that:

$$x(\xi,\eta) = \sum_{n=0}^{3} x_n^{(e)} \Phi_n^{(e)}(\xi,\eta), \quad y(\xi,\eta) = \sum_{n=0}^{3} y_n^{(e)} \Phi_n^{(e)}(\xi,\eta)$$

where $(x_n^{(e)}, y_n^{(e)})$ are the nodal coordinates of element $e$. Note that by our indexing scheme:

$$(x_0^{(e)}, y_0^{(e)}) = (x_0, y_0), \quad (x_1^{(e)}, y_1^{(e)}) = (x_1, y_0),$$
$$(x_2^{(e)}, y_2^{(e)}) = (x_1, y_1), \quad (x_3^{(e)}, y_3^{(e)}) = (x_0, y_1)$$

Expanding $x(\xi, \eta)$:

$$\begin{aligned}
x(\xi,\eta) &= x_0^{(e)}\Phi_0^{(e)} + x_1^{(e)}\Phi_1^{(e)} + x_2^{(e)}\Phi_2^{(e)} + x_3^{(e)}\Phi_3^{(e)} \\
&= x_0\frac{1}{4}(1-\xi)(1-\eta) + x_1\frac{1}{4}(1+\xi)(1-\eta) + x_1\frac{1}{4}(1+\xi)(1+\eta) + x_0\frac{1}{4}(1-\xi)(1+\eta) \\
&= \frac{1}{4}\left[x_0(1-\xi)(1-\eta+1+\eta) + x_1(1+\xi)(1-\eta+1+\eta)\right] \\
&= \frac{1}{4}\left[2x_0(1-\xi) + 2x_1(1+\xi)\right] \\
&= \frac{x_1 - x_0}{2}\xi + \frac{x_1 + x_0}{2}
\end{aligned}$$

Similarly for $y(\xi, \eta)$, we get:

$$y(\xi,\eta) = \frac{y_1 - y_0}{2}\eta + \frac{y_1 + y_0}{2}$$

Therefore, the mapping from reference to physical coordinates is given by equation 1

Now we can derive the Jacobian matrix, $J(\xi, \eta)$, of this mapping, which is defined as:

$$J(\xi, \eta) = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

Let us compute each partial derivative:

$$\frac{\partial x}{\partial \xi} = \frac{x_1 - x_0}{2}, \quad \frac{\partial x}{\partial \eta} = 0$$

$$\frac{\partial y}{\partial \xi} = 0, \quad \frac{\partial y}{\partial \eta} = \frac{y_1 - y_0}{2}$$

Therefore, our Jacobian is:

$$J(\xi, \eta) = \begin{bmatrix} \frac{x_1 - x_0}{2} & 0 \\ 0 & \frac{y_1 - y_0}{2} \end{bmatrix}$$

Note that if our elements were all equally sized, the Jacobian would be identical for all elements.

# Question 4

Using the basis functions from the reference element, compute the following derivatives:

$$\frac{\partial \Phi_i}{\partial \xi}, \frac{\partial \Phi_i}{\partial \eta} \quad \text{for} \quad i = 0, 1, 2, 3$$

Then express the physical gradients $\nabla \Phi_i = \left[\frac{\partial \Phi_i}{\partial x}, \frac{\partial \Phi_i}{\partial y}\right]^T$ using the Jacobian.

*Solution.* Note that:

$$\Phi_i(x, y) = \Phi_i(\xi(x, y), \eta(x, y))$$

We know that:

$$\nabla \Phi_i(\xi, \eta) = J(\xi, \eta)\ \nabla \Phi_i(x, y)$$

where $J$ is the Jacobian matrix derived in the previous question. Therefore, we can express the physical gradients as:

$$\nabla \Phi_i(x, y) = J^{-1}(\xi, \eta)\ \nabla \Phi_i(\xi, \eta)$$

Given the Jacobian from before:

$$J(\xi, \eta) = \begin{bmatrix} \frac{x_1 - x_0}{2} & 0 \\ 0 & \frac{y_1 - y_0}{2} \end{bmatrix}$$

Its inverse is given by:

$$J^{-1}(x, y) = \begin{bmatrix} \frac{2}{x_1 - x_0} & 0 \\ 0 & \frac{2}{y_1 - y_0} \end{bmatrix}$$

Then for each $\Phi_i(x, y)$, we have:

$$\begin{aligned} \nabla \Phi_i(x, y) &= J^{-1}(\xi, \eta)\ \nabla \Phi_i(\xi, \eta) \\ &= \begin{bmatrix} \frac{2}{x_1 - x_0} & 0 \\ 0 & \frac{2}{y_1 - y_0} \end{bmatrix} \begin{bmatrix} \frac{\partial \Phi_i}{\partial \xi} \\ \frac{\partial \Phi_i}{\partial \eta} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{x_1 - x_0} & 0 \\ 0 & \frac{2}{y_1 - y_0} \end{bmatrix} \begin{bmatrix} \frac{\partial \Phi_i}{\partial \xi} \\ \frac{\partial \Phi_i}{\partial \eta} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{x_1 - x_0} \frac{\partial \Phi_i}{\partial \xi} \\ \frac{2}{y_1 - y_0} \frac{\partial \Phi_i}{\partial \eta} \end{bmatrix} \end{aligned}$$

Since we have square elements of length $1$, we have $x_1 - x_0 = 1$ and $y_1 - y_0 = 1$. Therefore, the physical gradients simplify to:

$$\nabla \Phi_i(x, y) = 2 \nabla \Phi_i(\xi, \eta)$$

Let us first calculate $\nabla \Phi_i(\xi, \eta)$. For reference, the shape functions are:

$$\Phi_0(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 - \eta), \quad \Phi_1(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 - \eta),$$

$$\Phi_2(\xi,\eta)=\frac{1}{4}(1+\xi)(1+\eta), \quad \Phi_3(\xi,\eta)=\frac{1}{4}(1-\xi)(1+\eta)$$

Starting with derivatives with respect to $\xi$:

$$\frac{\partial\Phi_0}{\partial\xi}=-\frac{1}{4}(1-\eta), \quad \frac{\partial\Phi_1}{\partial\xi}=\frac{1}{4}(1-\eta),$$

$$\frac{\partial\Phi_2}{\partial\xi}=\frac{1}{4}(1+\eta), \quad \frac{\partial\Phi_3}{\partial\xi}=-\frac{1}{4}(1+\eta)$$

Then for derivatives with respect to $\eta$:

$$\frac{\partial\Phi_0}{\partial\eta}=-\frac{1}{4}(1-\xi), \quad \frac{\partial\Phi_1}{\partial\eta}=-\frac{1}{4}(1+\xi),$$

$$\frac{\partial\Phi_2}{\partial\eta}=\frac{1}{4}(1+\xi), \quad \frac{\partial\Phi_3}{\partial\eta}=\frac{1}{4}(1-\xi)$$

Therefore, the gradients in reference coordinates are:

$$\nabla\Phi_0(\xi,\eta)=\frac{1}{4}\begin{bmatrix}-(1-\eta)\\-(1-\xi)\end{bmatrix}, \quad \nabla\Phi_1(\xi,\eta)=\frac{1}{4}\begin{bmatrix}(1-\eta)\\-(1+\xi)\end{bmatrix}$$

$$\nabla\Phi_2(\xi,\eta)=\frac{1}{4}\begin{bmatrix}(1+\eta)\\(1+\xi)\end{bmatrix}, \quad \nabla\Phi_3(\xi,\eta)=\frac{1}{4}\begin{bmatrix}-(1+\eta)\\(1-\xi)\end{bmatrix}$$

So we have the physical gradients:

$$\nabla\Phi_0(x,y)=\frac{1}{2}\begin{bmatrix}-(1-\eta)\\-(1-\xi)\end{bmatrix}, \quad \nabla\Phi_1(x,y)=\frac{1}{2}\begin{bmatrix}(1-\eta)\\-(1+\xi)\end{bmatrix}$$

$$\nabla\Phi_2(x,y)=\frac{1}{2}\begin{bmatrix}(1+\eta)\\(1+\xi)\end{bmatrix}, \quad \nabla\Phi_3(x,y)=\frac{1}{2}\begin{bmatrix}-(1+\eta)\\(1-\xi)\end{bmatrix}$$

# Question 5

Use the following formulas for the elemental mass matrix $M^{(e)}$, and stiffness matrix $K^{(e)}$:

$$M_{ij}^{(e)} = \int_{\Omega^{(e)}} \Phi_i^{(e)} \Phi_j^{(e)} \; dx, \quad K_{ij}^{(e)} = \int_{\Omega^{(e)}} \nabla \Phi_i^{(e)} \cdot \nabla \Phi_j^{(e)} \; dx$$

to evaluate these integrals explicitly for an arbitrary square element in this mesh. Your $M^{(e)}$ and $K^{(e)}$ should be $4 \times 4$ matrices.

*Solution.* For ease of notation let $\Phi_i^{(e)} = \Phi_i$.

Let us denote $\Phi = [\Phi_0, \Phi_1, \Phi_2, \Phi_3]^T$ as the vector of shape functions for element $e$. Note that $M^{(e)}$ can also be expressed a $M^{(e)} = \int_{\Omega_e} \Phi \cdot \Phi^T \; dx$. Using the change of variables from physical to reference coordinates, we have:

$$M^{(e)} = \int_{-1}^{1} \int_{-1}^{1} \Phi \cdot \Phi^T |\det J(\xi, \eta)| \; d\xi d\eta$$

where $|\det J(\xi, \eta)|$ is the absolute value of the determinant of our Jacobian. From question 5, we have:

$$|\det J(\xi, \eta)| = \left| \frac{(x_1 - x_0)}{2} \cdot \frac{(y_2 - y_1)}{2} \right| = \frac{|x_1 - x_0| \cdot |y_2 - y_1|}{4}$$

When corrected for our local indexing scheme.

Since this constant, we can factor it out of the integral:

$$M^{(e)} = \frac{|x_1 - x_0| \cdot |y_2 - y_1|}{4} \int_{-1}^{1} \int_{-1}^{1} \Phi \cdot \Phi^T \; d\xi d\eta$$

Note that:

$$\Phi \cdot \Phi^T = \begin{bmatrix} \Phi_0 \\ \Phi_1 \\ \Phi_2 \\ \Phi_3 \end{bmatrix} \begin{bmatrix} \Phi_0 & \Phi_1 & \Phi_2 & \Phi_3 \end{bmatrix}$$

$$= \begin{bmatrix} \Phi_0\Phi_0 & \Phi_0\Phi_1 & \Phi_0\Phi_2 & \Phi_0\Phi_3 \\ \Phi_1\Phi_0 & \Phi_1\Phi_1 & \Phi_1\Phi_2 & \Phi_1\Phi_3 \\ \Phi_2\Phi_0 & \Phi_2\Phi_1 & \Phi_2\Phi_2 & \Phi_2\Phi_3 \\ \Phi_3\Phi_0 & \Phi_3\Phi_1 & \Phi_3\Phi_2 & \Phi_3\Phi_3 \end{bmatrix}$$

We will compute some auxiliary integrals with dummy variables first:

$$\int_{-1}^{1} (1 \pm z)^2 \; dz = \pm \frac{(1 \pm z)^3}{3} \bigg|_{-1}^{1} = \frac{8}{3}$$

$$\int_{-1}^{1} (1 + z)(1 - z) \; dz = \int_{-1}^{1} (1 - z^2) \; dz = z - \frac{z^3}{3} \bigg|_{-1}^{1} = \frac{4}{3}$$

By the nature of the calculations, the integral matrix will be symmetric. We will show one sample calculation for each main-diagonal and off-diagonal entries. For reference, the shape functions are:

$$\Phi_0(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad \Phi_1(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta),$$

$$\Phi_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad \Phi_3(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

First note that the product of any pair of shape functions will always have a factor of $\frac{1}{16}$. Also note that each product pair will contain two factors in some combination of the forms shown in the auxiliary integrals above.

Case: Main-diagonal entry ($i = j = 0$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \Phi_0\Phi_0 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \frac{1}{16}(1-\xi)^2(1-\eta)^2 \; d\xi d\eta \\
&= \frac{1}{16}\int_{-1}^{1}(1-\eta)^2\int_{-1}^{1}(1-\xi)^2 \; d\xi d\eta \\
&= \frac{1}{16}\cdot\frac{8}{3}\int_{-1}^{1}(1-\eta)^2 \; d\eta \\
&= \frac{1}{16}\cdot\frac{8}{3}\cdot\frac{8}{3} \\
&= \frac{4}{9}
\end{aligned}$$

Case: 1st Off-diagonal entry ($i = 0, j = 1$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \Phi_0\Phi_1 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \frac{1}{16}(1-\xi)(1-\eta)(1+\xi)(1-\eta) \; d\xi d\eta \\
&= \frac{1}{16}\int_{-1}^{1}(1-\eta)^2\int_{-1}^{1}(1-\xi^2) \; d\xi d\eta \\
&= \frac{1}{16}\cdot\frac{4}{3}\int_{-1}^{1}(1-\eta)^2 \; d\eta \\
&= \frac{1}{16}\cdot\frac{4}{3}\cdot\frac{8}{3} \\
&= \frac{2}{9}
\end{aligned}$$

Case: 2nd Off-diagonal entry ($i = 0, j = 2$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \Phi_0\Phi_2 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \frac{1}{16}(1-\xi)(1-\eta)(1+\xi)(1+\eta) \; d\xi d\eta \\
&= \frac{1}{16}\int_{-1}^{1}(1-\eta^2)\int_{-1}^{1}(1-\xi^2) \; d\xi d\eta \\
&= \frac{1}{16}\cdot\frac{4}{3}\int_{-1}^{1}(1-\eta^2) \; d\eta \\
&= \frac{1}{16}\cdot\frac{4}{3}\cdot\frac{4}{3} \\
&= \frac{1}{9}
\end{aligned}$$

Case: 3rd Off-diagonal entry ($i = 0, j = 3$):

Same as 1st off-diagonal by symmetry, so the result is $\frac{2}{9}$.

Substituting all these results back into the integral matrix, we have:

$$\int_{-1}^{1}\int_{-1}^{1} \Phi \cdot \Phi^T \, d\xi d\eta = \frac{1}{9}\begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}$$

Therefore, the elemental mass matrix is:

$$M^{(e)} = \frac{|x_1 - x_0| \cdot |y_2 - y_1|}{4} \cdot \frac{1}{9}\begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}$$

For our mesh, all elements are squares of side length $1$, so $|x_1 - x_0| = |y_1 - y_0| = 1$. Therefore, we have:

$$M^{(e)} = \frac{1}{36}\begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}$$

Next, we compute the elemental stiffness matrix $K^{(e)}$. Denote $\nabla\Phi = [\nabla\Phi_0, \nabla\Phi_1, \nabla\Phi_2, \nabla\Phi_3]$. Note that $K^{(e)}$ can also be expressed as:

$$K^{(e)} = \int_{\Omega_e} \nabla\Phi(x, y) \cdot \nabla\Phi^T(x, y) \, dx$$

Using the change of variables, we have:

$$K^{(e)} = \int_{-1}^{1}\int_{-1}^{1} (J^{-1}\nabla\Phi(\xi, \eta)) \cdot (J^{-1}\nabla\Phi(\xi, \eta))^T |\det J(\xi, \eta)| \, d\xi d\eta$$

With the Jacobian determinant factored out, we have:

$$K^{(e)} = \frac{|x_1 - x_0| \cdot |y_2 - y_1|}{4} \int_{-1}^{1}\int_{-1}^{1} (J^{-1}\nabla\Phi(\xi, \eta)) \cdot (J^{-1}\nabla\Phi(\xi, \eta))^T \, d\xi d\eta$$

As reference from question 4, we have the gradients:

$$\nabla\Phi_0(x, y) = \frac{1}{2}\begin{bmatrix} -(1-\eta) \\ -(1-\xi) \end{bmatrix}, \quad \nabla\Phi_1(x, y) = \frac{1}{2}\begin{bmatrix} (1-\eta) \\ -(1+\xi) \end{bmatrix}$$

$$\nabla\Phi_2(x, y) = \frac{1}{2}\begin{bmatrix} (1+\eta) \\ (1+\xi) \end{bmatrix}, \quad \nabla\Phi_3(x, y) = \frac{1}{2}\begin{bmatrix} -(1+\eta) \\ (1-\xi) \end{bmatrix}$$

Note any product of the form $\nabla\Phi_i \cdot \nabla\Phi_j$ will have a factor of $\frac{1}{4}$.

Note that :

$$\nabla\Phi \cdot \nabla\Phi^T = \begin{bmatrix} \nabla\Phi_0 \cdot \nabla\Phi_0 & \nabla\Phi_0 \cdot \nabla\Phi_1 & \nabla\Phi_0 \cdot \nabla\Phi_2 & \nabla\Phi_0 \cdot \nabla\Phi_3 \\ \nabla\Phi_1 \cdot \nabla\Phi_0 & \nabla\Phi_1 \cdot \nabla\Phi_1 & \nabla\Phi_1 \cdot \nabla\Phi_2 & \nabla\Phi_1 \cdot \nabla\Phi_3 \\ \nabla\Phi_2 \cdot \nabla\Phi_0 & \nabla\Phi_2 \cdot \nabla\Phi_1 & \nabla\Phi_2 \cdot \nabla\Phi_2 & \nabla\Phi_2 \cdot \nabla\Phi_3 \\ \nabla\Phi_3 \cdot \nabla\Phi_0 & \nabla\Phi_3 \cdot \nabla\Phi_1 & \nabla\Phi_3 \cdot \nabla\Phi_2 & \nabla\Phi_3 \cdot \nabla\Phi_3 \end{bmatrix}$$

The stiffness matrix is also symmetric, so we will only show one sample calculation for each unique entry.

Case: Main-diagonal entry ($i = j = 0$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \nabla\Phi_0 \cdot \nabla\Phi_0 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \left(-\frac{1}{2}(1-\eta)\right)^2 + \left(-\frac{1}{2}(1-\xi)\right)^2 \; d\xi d\eta \\
&= \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1} (1-\eta)^2 + (1-\xi)^2 \; d\xi d\eta \\
&= \frac{1}{4}\left[\int_{-1}^{1} (1-\eta)^2 \; \int_{-1}^{1} \; d\xi \; d\eta + \int_{-1}^{1} (1-\xi)^2 \; \int_{-1}^{1} \; d\eta \; d\xi\right] \\
&= \frac{1}{4}\left[2\int_{-1}^{1} (1-\eta)^2 d\eta + 2\int_{-1}^{1} (1-\xi)^2 d\xi\right] \\
&= \frac{1}{2}\left[\frac{8}{3} + \frac{8}{3}\right] \\
&= \frac{1}{2} \cdot \frac{16}{3} \\
&= \frac{8}{3} \\
&= \frac{4}{6} \cdot 4
\end{aligned}$$

Case: First Off-Diagonal entry ($i = 0, j = 1$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \nabla\Phi_0 \cdot \nabla\Phi_1 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \left(-\frac{1}{2}(1-\eta)\right)\left(\frac{1}{2}(1-\eta)\right) + \left(-\frac{1}{2}(1-\xi)\right)\left(-\frac{1}{2}(1+\xi)\right) \; d\xi d\eta \\
&= \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1} -(1-\eta)^2 + (1-\xi^2) \; d\xi d\eta \\
&= \frac{1}{4}\left[\int_{-1}^{1} -(1-\eta)^2 \; \int_{-1}^{1} \; d\xi \; d\eta + \int_{-1}^{1} (1-\xi^2) \; \int_{-1}^{1} \; d\eta \; d\xi\right] \\
&= \frac{1}{4}\left[-2\int_{-1}^{1} (1-\eta)^2 d\eta + 2\int_{-1}^{1} (1-\xi^2) d\xi\right] \\
&= \frac{1}{2}\left[-\frac{8}{3} + \frac{4}{3}\right] \\
&= \frac{1}{2} \cdot \left(-\frac{4}{3}\right) \\
&= -\frac{2}{3} \\
&= \frac{4}{6} \cdot (-1)
\end{aligned}$$

Case: Second Off-Diagonal entry ($i = 0, j = 2$):

$$\begin{aligned}
\int_{-1}^{1}\int_{-1}^{1} \nabla\Phi_0 \cdot \nabla\Phi_2 \; d\xi d\eta &= \int_{-1}^{1}\int_{-1}^{1} \left(-\frac{1}{2}(1-\eta)\right)\left(\frac{1}{2}(1+\eta)\right) + \left(-\frac{1}{2}(1-\xi)\right)\left(\frac{1}{2}(1+\xi)\right) \; d\xi d\eta \\
&= -\frac{1}{4}\int_{-1}^{1}\int_{-1}^{1} (1-\eta^2) + (1-\xi^2) \; d\xi d\eta \\
&= \; -\frac{1}{4}\left[\int_{-1}^{1} (1-\eta^2) \; \int_{-1}^{1} \; d\xi \; d\eta + \int_{-1}^{1} (1-\xi^2) \; \int_{-1}^{1} \; d\eta \; d\xi\right] \\
&= -\frac{1}{4}\left[2\int_{-1}^{1} (1-\eta^2) d\eta + 2\int_{-1}^{1} (1-\xi^2) d\xi\right] \\
&= -\frac{1}{2}\left[\frac{4}{3} + \frac{4}{3}\right] \\
&= -\frac{1}{2} \cdot \frac{8}{3} \\
&= -\frac{4}{3} \\
&= \frac{4}{6} \cdot (-2)
\end{aligned}$$

Case: Third Off-Diagonal entry ($i = 0, j = 3$):

Same as First Off-Diagonal, so the result is $-\frac{4}{6}$.

Substituting all these results back into the integral matrix, we have:

$$\int_{-1}^{1}\int_{-1}^{1} \nabla\Phi \cdot \nabla\Phi^T \; d\xi d\eta = \frac{4}{6}\begin{bmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{bmatrix}$$

Therefore, the elemental stiffness matrix is:

$$K^{(e)} = \frac{|x_1 - x_0| \cdot |y_2 - y_1|}{4} \cdot \frac{4}{6}\begin{bmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{bmatrix}$$

For our mesh, where all the elements are of side length $1$, we have:

$$K^{(e)} = \frac{1}{6}\begin{bmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{bmatrix}$$

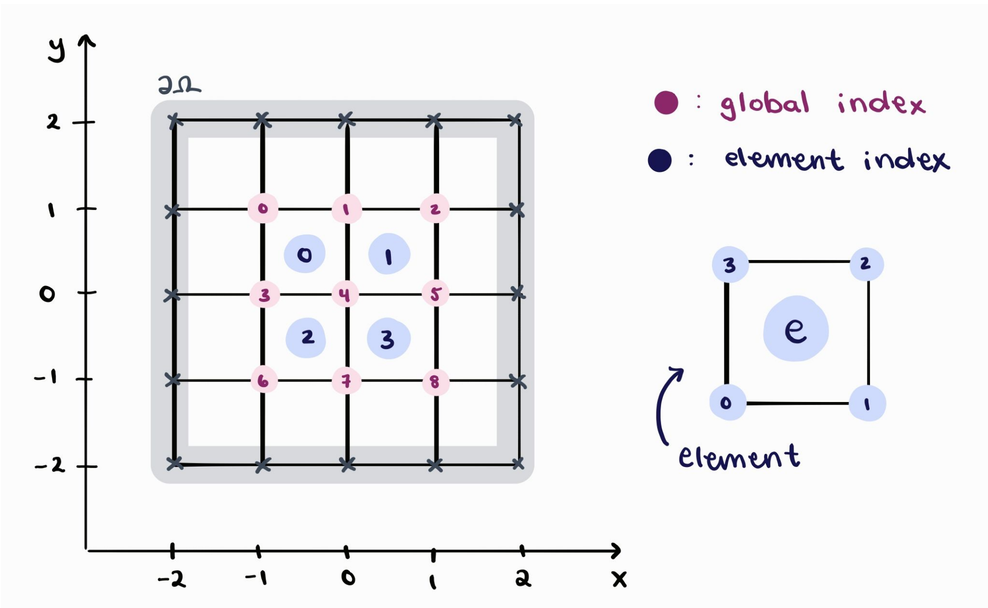
Note that our elemental matrices would differ based on the mesh construction due to definition of the Jacobian.

# Question 6

Assemble the global mass matrix $M$ and global stiffness matrix $K$ for the entire $2 \times 2$ mesh using the connectivity determined in question 2. Impose homogeneous Dirichlet boundary conditions on all boundary nodes. Write the resulting discretized ODE system in the following format:

$$M\frac{dU}{dt} + KU = F(t)$$

*Solution.* Given the Dirichlet boundary conditions on all boundary nodes, we only need to consider the interior nodes for our global matrices, therefore our mesh is reduced to $2 \times 2$ rectangles, for a total of $4$ rectangular elements. From our initial global indexing, we reduce the problem to have $9$ interior nodes, numbered from $0$ to $8$. We'll also number our elements from $0$ to $3$ in a row-wise manner:




Using the python method defined in question 2, the updated connectivities for each element are as follows:

| Element | Node 0 | Node 1 | Node 2 | Node 3 |
|---|---|---|---|---|
| 0 | 3 | 4 | 1 | 0 |
| 1 | 4 | 5 | 2 | 1 |
| 2 | 6 | 7 | 4 | 3 |
| 3 | 7 | 8 | 5 | 4 |

We will adjust the indexing in the preliminary setup to match this scenario. From before we had:

$$\sum_{i,j=1}^{N} \left( \int_\Omega \Phi_i \Phi_j \; dx \right) \frac{dU_i}{dt} + \nu \sum_{i,j=1}^{N} \left( \int_\Omega \nabla \Phi_i \cdot \nabla \Phi_j \; dx \right) U_i = \sum_{j=1}^{N} \int_\Omega f \Phi_j \; dx$$

Was our global discretized system, where $N$ is the total number of nodes. Rewriting this terms of the elemental components, we have:

$$\left(\sum_{e=1}^{E} M^{(e)}\right)\frac{dU}{dt} + \nu\left(\sum_{e=1}^{E} K^{(e)}\right)U = \sum_{e=1}^{E} F^{(e)}(t)$$

Where $E$ is the total number of elements.

Correcting for our indexing scheme, we have:

$$\left(\sum_{e=0}^{3} M^{(e)}\right)\frac{dU}{dt} + \nu\left(\sum_{e=0}^{3} K^{(e)}\right)U = \sum_{e=0}^{3} F^{(e)}(t)$$

Such that that $M = \sum_{e=0}^{3} M^{(e)}$ and $K = \sum_{e=0}^{3} K^{(e)}$. We could lump the $\nu$ into $K$ to get the form:

$$M\frac{dU}{dt} + KU = F(t)$$

Also, since we have the homogeneous heat equation, $F^{(e)}(t) = 0$ for all $e$.

In addition to methods defined in Question 2, the following code was used to assemble our global matrices:

Listing 2: Question 6 Python

```
import numpy as np
import sympy as sp

def det_jacobian(xe, ye):
    return (1/4) * abs(xe[1] - xe[0]) * abs(ye[2] - ye[1])

def element_mass_matrix(xe, ye):
    detJ = det_jacobian(xe, ye)
    Me = (detJ / 9) * np.array([[4, 2, 1, 2],
                                [2, 4, 2, 1],
                                [1, 2, 4, 2],
                                [2, 1, 2, 4]])
    return Me

def element_stiffness_matrix(xe, ye):
    detJ = det_jacobian(xe, ye)
    Ke = detJ *(4 / 6) * np.array([[4, -1, -2, -1],
                                [-1, 4, -1, -2],
                                [-2, -1, 4, -1],
                                [-1, -2, -1, 4]])
    return Ke

def generate_global_coordinates(width, height=None):
    if height is None:
        height = width
```

```python

    # Generate global coordinates for interior nodes
    # given Dirichlet BCs
    x_nodes = np.linspace(-2, 2, width)
    x_nodes = x_nodes[1:-1]
    y_nodes = np.linspace(-2, 2, height)
    y_nodes = y_nodes[1:-1]

    # Create meshgrid of coordinates
    x_mesh, y_mesh = np.array(np.meshgrid(x_nodes, y_nodes))
    # Reshape as list of (x, y) pairs
    global_coordinates = np.column_stack((x_mesh.ravel(), y_mesh.ravel()))
    return global_coordinates


def global_assembly(width, height=None, nu=0.05):
    if height is None:
        height = width

    global_coordinates = generate_global_coordinates(width, height)

    global_indices = global_indexing(width, height)
    connectivity_matrix = generate_connectivity_matrix(global_indices)

    num_nodes = (width - 2) * (height - 2)
    M_global = np.zeros((num_nodes, num_nodes))
    K_global = np.zeros((num_nodes, num_nodes))

    for element in range(connectivity_matrix.shape[0]):
        # Get the global node indices for this element
        element_coordinates = connectivity_matrix[element, :]

        # Extract x and y coordinates for element
        # x coordinates
        xe = global_coordinates[element_coordinates, 0]
        # y coordinates
        ye = global_coordinates[element_coordinates, 1]

        # Compute element matrices
        Me = element_mass_matrix(xe, ye)
        Ke = element_stiffness_matrix(xe, ye)

        # Add element contributions to global matrices
        for i_local in range(4):
            i_global = element_coordinates[i_local]
            for j_local in range(4):
                j_global = element_coordinates[j_local]
```

```
                M_global[i_global, j_global] += Me[i_local, j_local]
                K_global[i_global, j_global] += Ke[i_local, j_local]

    return M_global, K_global

if __name__ == "__main__":
    width = 5  # Number of nodes along one dimension
    global_indices = global_indexing(width)
    connectivity_matrix = generate_connectivity_matrix(global_indices)
    M, K = global_assembly(width)

    with open("./outputs_4/matrices.txt", "w") as f:
        print(global_indices)
        latex_matrix = sp.latex(sp.Matrix(global_indices))
        f.write("Global Indices Matrix:\n")
        f.write(latex_matrix + "\n\n")

        print(connectivity_matrix)
        latex_matrix = sp.latex(sp.Matrix(connectivity_matrix))
        f.write("Connectivity Matrix:\n")
        f.write(latex_matrix + "\n\n")

        M = np.round(M, 4)
        latex_matrix = sp.latex(sp.Matrix(M))
        f.write("Mass Matrix:\n")
        f.write(latex_matrix + "\n\n")

        K = np.round(K, 4)
        latex_matrix = sp.latex(sp.Matrix(K))
        f.write("Stiffness Matrix:\n")
        f.write(latex_matrix + "\n\n")
```

Our mass matrix $M$ will be of size $9 \times 9$ and was found to be:

$$\begin{bmatrix} 0.1111 & 0.0556 & 0.0 & 0.0556 & 0.0278 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0556 & 0.2222 & 0.0556 & 0.0278 & 0.1111 & 0.0278 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0556 & 0.1111 & 0.0 & 0.0278 & 0.0556 & 0.0 & 0.0 & 0.0 \\ 0.0556 & 0.0278 & 0.0 & 0.2222 & 0.1111 & 0.0 & 0.0556 & 0.0278 & 0.0 \\ 0.0278 & 0.1111 & 0.0278 & 0.1111 & 0.4444 & 0.1111 & 0.0278 & 0.1111 & 0.0278 \\ 0.0 & 0.0278 & 0.0556 & 0.0 & 0.1111 & 0.2222 & 0.0 & 0.0278 & 0.0556 \\ 0.0 & 0.0 & 0.0 & 0.0556 & 0.0278 & 0.0 & 0.1111 & 0.0556 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0278 & 0.1111 & 0.0278 & 0.0556 & 0.2222 & 0.0556 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0278 & 0.0556 & 0.0 & 0.0556 & 0.1111 \end{bmatrix}$$

Our stiffness matrix $K$ will also be of size $9 \times 9$ and was found to be (without $\nu$):

$$\begin{bmatrix} 0.6667 & -0.1667 & 0.0 & -0.1667 & -0.3333 & 0.0 & 0.0 & 0.0 & 0.0 \\ -0.1667 & 1.3333 & -0.1667 & -0.3333 & -0.3333 & -0.3333 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.1667 & 0.6667 & 0.0 & -0.3333 & -0.1667 & 0.0 & 0.0 & 0.0 \\ -0.1667 & -0.3333 & 0.0 & 1.3333 & -0.3333 & 0.0 & -0.1667 & -0.3333 & 0.0 \\ -0.3333 & -0.3333 & -0.3333 & -0.3333 & 2.6667 & -0.3333 & -0.3333 & -0.3333 & -0.3333 \\ 0.0 & -0.3333 & -0.1667 & 0.0 & -0.3333 & 1.3333 & 0.0 & -0.3333 & -0.1667 \\ 0.0 & 0.0 & 0.0 & -0.1667 & -0.3333 & 0.0 & 0.6667 & -0.1667 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.3333 & -0.3333 & -0.3333 & -0.1667 & 1.3333 & -0.1667 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.3333 & -0.1667 & 0.0 & -0.1667 & 0.6667 \end{bmatrix}$$

As expected, both $M$ and $K$ are symmetric, banded matrices.

# Question 7

Using the following initial condition

$$u(x, y, 0) = \sin(2\pi x)\sin(2\pi y)$$

to solve the reduced ODE system from $t = 0$ to $t = 1$. (Show Code)

*Solution.* Given our system:

$$M\frac{dU}{dt} + \nu KU = 0$$

We can rearrange this to get:

$$\frac{dU}{dt} = -\nu M^{-1}KU$$

## Explicit: Forward Euler

Let us find $U^{n+1}$ explicitly using forward difference in time:

$$\frac{U^{n+1} - U^n}{\Delta t} = -\nu M^{-1}KU^n$$

$$U^{n+1} = U^n - \nu\Delta t M^{-1}KU^n$$

Now let us derive the CFL condition for stability. Note that we may rewrite our scheme above as:

$$U^{n+1} = \left(I - \nu\Delta t M^{-1}K\right)U^n$$

Let $A = M^{-1}K$ and $B = I - \nu\Delta tA$. For stability, we require that the spectral radius of $B$ be less than or equal to $1$:

$$\rho(B) = \max|\lambda(B)| \leq 1$$

Where the maximum is taken over all eigenvalues $\lambda$ of $B$. Note that each eigenvalue of $B$ are related to a corresponding eigenvalue of $A$ as follows:

$$\lambda(B) = 1 - \nu\Delta t\lambda(A)$$

Therefore, we require:

$$\max|1 - \nu\Delta t\lambda(A)| \leq 1$$

Which implies that for all eigenvalues $\lambda(A)$:

$$|1 - \nu\Delta t\lambda(A)| \leq 1$$

We note that $A = M^{-1}K$ is positive definite (has real, positive eigenvalues). Through a quick check using Python, the list of eigenvalues of $A$ are:

$$\begin{bmatrix} 24.0 \\ 1.17981541490574 \cdot 10^{-15} \\ 3.0 \\ 6.0 \\ 15.0 \\ 12.0 \\ 3.0 \\ 12.0 \\ 15.0 \end{bmatrix}$$

Note that $\rho(A) = 24$.

So given that all eigenvalues of $A$ are non-negative, to satisfy our stability condition, we require:

$$|1 - \nu\Delta t\lambda(A)| \leq 1$$

Which implies:

$$-1 \leq 1 - \nu\Delta t\lambda(A) \leq 1$$

We may bound these inequalities separately. The right inequality simplifies to:

$$0 \leq \nu\Delta t\lambda(A)$$

Rearranging for $\Delta t$, we have:

$$\Delta t \geq 0$$

Which is vacuously true as heat is unstable in backwards time. The left inequality simplifies to:

$$-2 \leq -\nu\Delta t\lambda(A)$$

Rearranging for $\Delta t$, we have:

$$\Delta t \leq \frac{2}{\nu\lambda(A)}$$

To satisfy this for all eigenvalues of $A$, we use the maximum eigenvalue of A $\lambda_{max}(A) = 24$:

$$\Delta t \leq \frac{2}{\nu\lambda_{max}(A)} = \frac{2}{0.05 \cdot 24} = \frac{5}{3} \approx 1.6667$$

Therefore we may use the explicit scheme so long as $\Delta t \leq \frac{5}{3}$ for stability.

The following code was used to implement the explicit forward Euler method:

Listing 3: explicit_solver.py

```
from assembly import global_assembly
import numpy as np
import math
from copy import deepcopy
import matplotlib.pyplot as plt

def u_0(mesh):
    x = mesh[:, 0]
    y = mesh[:, 1]
    return np.sin(2*np.pi*x) * np.sin(2*np.pi*y)

def u_exact(x, y, t, nu=0.05):
    return np.exp(-8 * (np.pi**2) * nu * t) * np.sin(2*np.pi*x) * np.sin(2*np.pi*y)

def explicit_heat_solver(width, dt, t_final, height=None, nu=0.05):
    if dt > 5/3:
        raise ValueError("Time step dt is too large for stability. Pick dt <= 5/3.")
    if height is None:
        height = width

    global_coordinates, M, K = global_assembly(width, height)
    M_inverse = np.linalg.inv(M)
    Identity = np.eye(M.shape[0])
    A = np.dot(M_inverse, K)
    B = Identity - nu * dt * A

    U = []
    t=0
    current_U = u_0(global_coordinates)
    U.append((t, deepcopy(current_U)))

    time_steps = math.ceil(t_final / dt)
    for _ in range(time_steps-1):
        t += dt
        current_U = np.dot(B, current_U)
        U.append((t, deepcopy(current_U)))

    dt = t_final - (time_steps-1)*dt
    B= Identity - nu * dt * A
    current_U = np.dot(B, current_U)
    U.append((t_final, deepcopy(current_U)))

    return U

```

```python
if __name__ == "__main__":
    width = 5
    dt = 0.05
    t_final = 1.0
    U = explicit_heat_solver(width, dt, t_final)

    u_approx = U[-1][1].reshape((width-2, width-2))
    u_approx = np.pad(u_approx, (1, 1), 'constant', constant_values=(0,))
    x = np.linspace(-2, 2, width)
    x_interior = x[1:-1]
    y = np.linspace(-2, 2, width)
    y_interior = y[1:-1]
    x_mesh, y_mesh = np.meshgrid(x, y)
    x_interior_mesh, y_interior_mesh = np.meshgrid(x_interior, y_interior)
    u_exact_values = u_exact(x_interior_mesh, y_interior_mesh, t_final)
    u_exact_values = np.pad(u_exact_values, (1, 1), 'constant', constant_values
=(0,))

    fig = plt.figure()
    ax = fig.add_subplot( projection='3d')
    ax.plot_surface(x_mesh, y_mesh, u_approx, cmap='viridis', alpha=0.8, label=
'Approximate Solution')
    # ax.plot_surface(x_mesh, y_mesh, u_exact_values, cmap='plasma', alpha=0.3,
    label='Exact Solution')
    plt.show()
```

## Semi-Implicit: Crank-Nicolson Method

Let us use Crank-Nicolson method to solve this system.

The forward Euler step is:

$$M\frac{U^{n+1}-U^n}{\Delta t} = F^n(U) = -KU^n$$

The backward Euler step is:

$$M\frac{U^{n+1}-U^n}{\Delta t} = F^{n+1}(U) = -KU^{n+1}$$

Using Crank-Nicolson, we average these two steps:

$$M\frac{U^{n+1}-U^n}{\Delta t} = \frac{1}{2}\left(F^n(U)+F^{n+1}(U)\right) = -\frac{1}{2}K\left(U^n+U^{n+1}\right)$$

Rearranging, we have:

$$M(U^{n+1}-U^n) = -\frac{\Delta t}{2}K\left(U^n+U^{n+1}\right)$$

$$\left(M+\frac{\Delta t}{2}K\right)U^{n+1} = \left(M-\frac{\Delta t}{2}K\right)U^n$$

# Question 8

Write your own solver for solving linear systems. You can freely choose the methods from Gaussian, Jacobi, or Gauss-Seidel.

*Solution. Extra: See Appendix for past Maple implementations of Gaussian Elimination, Jacobi, and Gauss-Seidel.*

# Question 9

Perform a convergence study with different refinements on time steps. Plot the log-log plot of error vs time step size.

# Question 10

Plot $U(t)$ at $T = 1$.

# Appendix

# 1 Maple Implementations of Linear Solvers

These are implementations of Gaussian Elimination, Jacobi Method, and Gauss-Seidel Method in Maple I have done in the past.

These were the questions being answered:

3. Use $LU$ factorization to solve the system:

$$\begin{aligned} 6x_1 - 2x_2 + 2x_3 + 4x_4 &= 16 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 &= 26 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 &= -19 \\ -6x_1 + 4x_2 + x_3 - 18x_4 &= -34 \end{aligned}$$

   Be sure to state the matrices $L$ and $U$.

4. Use the Jacobi iterative method and the Gauss-Seidel iterative method to find the solution to the following set of equations within $10^{-4}$ in the $\ell_\infty$ norm using $\mathbf{x}^{(0)} = \mathbf{0}$ as your initial condition. Show theoretically whether or not both methods will converge in this case.

$$\begin{aligned} 4x_1 + x_2 + x_3 + x_4 &= -5 \\ x_1 + 8x_2 + 2x_3 + 3x_4 &= 23 \\ x_1 + 2x_2 - 5x_3 &= 9 \\ -x_1 + 2x_3 + 4x_4 &= 4 \end{aligned}$$

The following pages contain the Maple implementations and sample outputs.

Note: the Matrix Solver is Gaussian Elimination with addition of LU decomposition. No row exchanges were implemented.

# Matrix Solver

```
with(LinearAlgebra) :
MatrixSolve := proc(A, b)
   local i, j, n, L, U, v, const, det;
      n := RowDimension(A);
      L := Matrix(n);
      U := A;
      v := b;
      det := 1;
      print(A, b);
   for i from 1 to n − 1 do
      for j from i + 1 to n do

         if (U[i, i] = 0) then
            error "zero along main diagonal";
         end if;
         const := U[j, i] / U[i, i];
         if (const ≠ 0) then
            L[j, i] := const;
            U := RowOperation(U, [j, i], −const);
            v[j] := v[j] − const·v[i];
            print(R(j) − const·R(i), U, v);
         end if;

      end do;
   end do;

   for i from 1 to n do
      det := det·U[i, i];
      L[i, i] := 1;
   end do;

   print(L, U, v, det);

   end proc:
```

3 **Problem #3**

$A := Matrix([[6,-2,2,4],[12,-8,6,10],[3,-13,9,3],[-6,4,1,-18]]);$
$b := Vector([16,26,-19,-34]);$

$$A := \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}$$

$$b := \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix} \tag{6}$$

$MatrixSolve(A, b);$

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}, \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix}$$

$$R(2) - 2\,R(1), \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}, \begin{bmatrix} 16 \\ -6 \\ -19 \\ -34 \end{bmatrix}$$

$$R(3) - \frac{R(1)}{2}, \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ -6 & 4 & 1 & -18 \end{bmatrix}, \begin{bmatrix} 16 \\ -6 \\ -27 \\ -34 \end{bmatrix}$$

$$R(4) + R(1), \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix}, \begin{bmatrix} 16 \\ -6 \\ -27 \\ -18 \end{bmatrix}$$

$$R(3) - 3\,R(2), \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 2 & 3 & -14 \end{bmatrix}, \begin{bmatrix} 16 \\ -6 \\ -9 \\ -18 \end{bmatrix}$$

$$R(4)+\frac{R(2)}{2},\ \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix},\ \begin{bmatrix} 16 \\ -6 \\ -9 \\ -21 \end{bmatrix}$$

$$R(4)-2\,R(3),\ \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix},\ \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

$$\boxed{\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix}}_{L},\ \underbrace{\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix}}_{u}},\ \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix},\ 144 \tag{7}$$

<u>Ly=b</u>

$y1 := 16:$

$y2 := -2\cdot y1 + 26:$

$y3 := -\frac{1}{2}\cdot y1 - 3\cdot y2 - 19:$

$y4 := y1 + \frac{1}{2}\cdot y2 - 2\cdot y3 - 34:$

$y := Vector([y1, y2, y3, y4]);$

$$y := \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix} \tag{8}$$

<u>Ux=y</u>

$x4 := -\frac{1}{3}(-3):$

$x3 := \frac{1}{2}(5\cdot x4 - 9):$

$x2 := -\frac{1}{4}(-2\cdot x3 - 2\cdot x4 - 6):$

$x1 := \frac{1}{6}(2\cdot x2 - 2\cdot x3 - 4\cdot x4 + 16):$

$x := Vector([x1, x2, x3, x4]);$

$$\boxed{x := \begin{bmatrix} 3 \\ 1 \\ -2 \\ 1 \end{bmatrix}} \tag{9}$$

## Jacobi

```
with(Student[LinearAlgebra]) :
Jacobi := proc(T, c, α, ε, N)
    local  i, x, err, temp;
        x := α;

    for i from 1 to N do
        temp := T · x + c;
        err := Norm(temp − x, infinity);
        x := temp;
        if (err < ε) then
            break;
        end if;
    end do;

 print(evalf(x), i);
 end proc:
```

$T := Matrix\left(\left[\left[0, -\frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}\right], \left[-\frac{1}{8}, 0, -\frac{1}{4}, -\frac{3}{8}\right], \left[\frac{1}{5}, \frac{2}{5}, 0, 0\right], \left[\frac{1}{4}, 0, -\frac{1}{2}, 0\right]\right]\right);$

$$T := \begin{bmatrix} 0 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{8} & 0 & -\frac{1}{4} & -\frac{3}{8} \\ \frac{1}{5} & \frac{2}{5} & 0 & 0 \\ \frac{1}{4} & 0 & -\frac{1}{2} & 0 \end{bmatrix} \tag{1}$$

$c := Vector\left(\left[-\frac{5}{4}, \frac{23}{8}, -\frac{9}{5}, 1\right]\right);$

$$c := \begin{bmatrix} -\frac{5}{4} \\ \frac{23}{8} \\ -\frac{9}{5} \\ 1 \end{bmatrix} \tag{2}$$

$\alpha := Vector([0, 0, 0, 0]);$

$$\alpha := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

$Jacobi(T, c, \alpha, 10^{-4}, 100);$

$$\boxed{\begin{bmatrix} -1.999988563 \\ 3.000005627 \\ -1.000036062 \\ 0.9999687134 \end{bmatrix}}, 20 \tag{4}$$

## Gauss-Seidel

```
with(LinearAlgebra) :
GaussSeidel :=proc(A, b, α, ε, N)
  local i, j, n, L, D, U, T, c, x, err, temp;
    n := RowDimension(A);
    L := Matrix(n);
    D := Matrix(n);
    U := Matrix(n);
    x := α;

  # Create L
  for j from 1 to n − 1 do
    for i from j + 1 to n do
      L[i, j] := A[i, j];
    end do;
  end do;

  # Create D
  for i from 1 to n do
    D[i, i] := A[i, i];
  end do;

  # Create U
  for i from 1 to n − 1 do
    for j from i + 1 to n do
      U[i, j] := A[i, j];
    end do;
  end do;

  T :=− (L + D)^(−1) · U;
  c := (L + D)^(−1) · b;

  for i from 1 to N do
    temp := T · x + c;
    err := Norm(temp − x, infinity);
    x := temp;
    if (err < ε) then
      break;
    end if;
  end do;

 print(evalf(x), i);
 return T;
 end proc:
```

$A := Matrix([[4, 1, 1, 1], [1, 8, 2, 3], [1, 2, -5, 0], [-1, 0, 2, 4]]);$

$$A := \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 8 & 2 & 3 \\ 1 & 2 & -5 & 0 \\ -1 & 0 & 2 & 4 \end{bmatrix} \tag{5}$$

$b := Vector([-5, 23, 9, 4]);$

$$b := \begin{bmatrix} -5 \\ 23 \\ 9 \\ 4 \end{bmatrix} \tag{6}$$

$T2 := GaussSeidel(A, b, \alpha, 10^{-4}, 100);$

$$\boxed{\begin{bmatrix} -2.000010143 \\ 2.999995446 \\ -1.000003850 \\ 0.9999993894 \end{bmatrix}}, 8$$

$$T2 := \begin{bmatrix} 0 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ 0 & \frac{1}{32} & -\frac{7}{32} & -\frac{11}{32} \\ 0 & -\frac{3}{80} & -\frac{11}{80} & -\frac{3}{16} \\ 0 & -\frac{7}{160} & \frac{1}{160} & \frac{1}{32} \end{bmatrix} \tag{7}$$

Jacobi

$evalf(Eigenvalues(T));$

$$\begin{bmatrix} 0.3637795831 \\ 0.0960484587 \\ -0.2299140209 + 0.5521692700\,\mathrm{I} \\ -0.2299140209 - 0.5521692700\,\mathrm{I} \end{bmatrix} \tag{8}$$

$eval3 := \mathrm{sqrt}(0.2299140209^2 + 0.552169270^2);$

$eval4 := \mathrm{sqrt}(0.2299140209^2 + 0.552169270^2);$

$$eval3 := 0.5981231978$$

$$eval4 := 0.5981231978 \tag{9}$$

ρ(T) =0.5981231978 $< 1$, therefore the sequence will converge.

Gauss-Seidel

$evalf(Eigenvalues(T2));$

$$\begin{bmatrix} 0. \\ 0. \\ 0.1388341998 \\ -0.2138341998 \end{bmatrix} \tag{10}$$

$$eval3 := 0.2053959591$$

$$eval4 := 0.2053959591 \tag{11}$$

ρ(T) =0.2138341998 $< 1$, therefore the sequence will converge.

**Actual**

$with(Student[NumericalAnalysis])$ :

$evalf\left(IterativeApproximate\left(A, b, method = jacobi, initialapprox = \alpha, tolerance = 10^{-4}, maxiterations = 100\right)\right);$

$evalf\left(IterativeApproximate\left(A, b, method = gaussseidel, initialapprox = \alpha, tolerance = 10^{-4}, maxiterations = 100\right)\right);$

$$\begin{bmatrix} -1.999988563 \\ 3.000005627 \\ -1.000036062 \\ 0.9999687134 \end{bmatrix}$$

$$\begin{bmatrix} -2.000010143 \\ 2.999995446 \\ -1.000003850 \\ 0.9999993894 \end{bmatrix} \qquad \textbf{(12)}$$

# 2 Assignment Code

The complete code used for this assignment is provided in the appendix for reference. Files can be accessed directly at this GitHub repository.