Duke University, Pratt School of Engineering

ME 524 — Multiscale Thermal Analysis of Composite Material

Kyle Abrahm (kwa11)

Date Performed ..... Dec 13, 2024
Instructor ..... Johann Guilleminot

Abstract

This report presents a numerical approach to solving steady-state heat conduction in a 2D domain, focusing on the implementation of the finite element method (FEM) to model the heat transfer process. The report explores the theory behind the strong and weak forms of the governing equations, the process of h-convergence, and the effective conductivity tensor used in the problem. A numerical integration scheme is introduced and coded, with a discussion on the isotropic vs non-isotropic nature of the material properties. The report also examines how changes in the radius of the inner circular region affect the heat conduction process.

1 Introduction

In this project, I aim to compute the effective thermal conductivity of a long-fiber-reinforced material using the Finite Element Method (FEM). The effective conductivity is defined as the thermal conductivity that, when associated with a homogeneous material, would lead to the same thermal behavior as the composite material. This effective conductivity allows for the substitution of a complex material description with a simpler, homogeneous model involving a single thermal property. This concept is central to multiscale methods, which are used to design composite structures without the need for fine-scale descriptions of the material.

To simplify the analysis, we consider a two-dimensional setting with a unit cell as shown in Fig. 1. Material 1 corresponds to the matrix phase while material 2 represents the fiber.

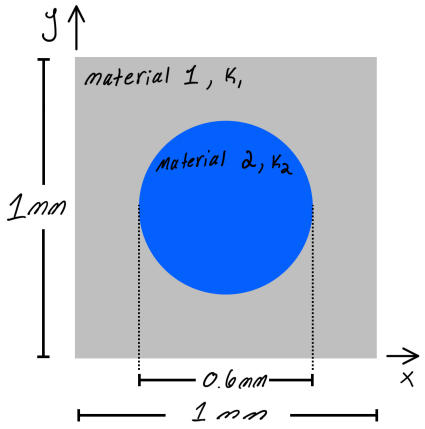


Figure 1: Unit Cell of the Composite Material

2 Theoretical Background

Edge Length (L)	0.001 m
Fiber Radius (R)	0.0003 m
k_1	1 [W/(mK)]
k_2	5 [W/(mK)]

Table 1: Problem Variables

Problem 1	$T(\mathbf{x}) = x_1, \forall \mathbf{x} \in \partial\Omega$
Problem 2	$T(\mathbf{x}) = x_2, \forall \mathbf{x} \in \partial\Omega$

Table 2: Boundary Conditions

2.1 Strong and Weak Forms of the Heat Conduction Equation

The strong form steady-state heat conduction equation in 2D is given by:

$$\nabla \cdot (k \nabla T) + s = 0 \tag{1}$$

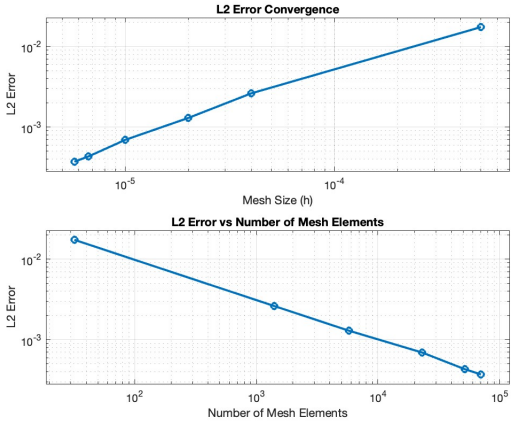
where T is the temperature, k is the thermal conductivity, and s is the heat source. For our problem, there is no heat source and thus, $s = 0$. In the strong form, this equation is enforced at every point in the domain. However, to apply numerical methods such as the finite element method, the strong form must be converted into its weak form by multiplying both sides of the equation by a test function v and integrating over the domain. The weak form is given by:

$$\int_{\Gamma_N} v \cdot \bar{q} d\mathbf{a} + \int_{\Omega} v \cdot s d\mathbf{x} - \int_{\Omega} \langle \nabla v, k \nabla T \rangle \, d\mathbf{x} = 0, \forall v \in V \tag{2}$$

This formulation allows the problem to be solved with finite element methods, where the solution is approximated in a function space.

2.2 h-Convergence

In finite element analysis, h-convergence refers to the process of refining the mesh (that is, decreasing the element size h) to improve the precision of the solution. Using an incredibly fine mesh with $Hmax = L/200$, the model was solved and held as a baseline. Next, more suitable, larger $Hmax$ values were compared to the fine mesh in the L^2 sense. Below is a graph of the relative errors vs. mesh sizes and the precise values.



Mesh Sizes	L2 Errors
L/2	0.0174
L/25	0.0026
L/50	0.0013
L/100	0.0007
L/150	0.0004
L/175	0.0004

Figure 2: L2 Error Visualization

A mesh size of $L/50$ will be used for the finite element formulation because the L^2 error is sufficiently small at this resolution, indicating that the solution is accurate within an acceptable tolerance. Additionally, T3 (linear triangular) elements are chosen because they provide a balance between accuracy and computational efficiency. With three nodes per element, T3 elements are well-suited for problems where the solution is expected to vary linearly or smoothly across the domain, ensuring accurate results while maintaining fast computational run time.

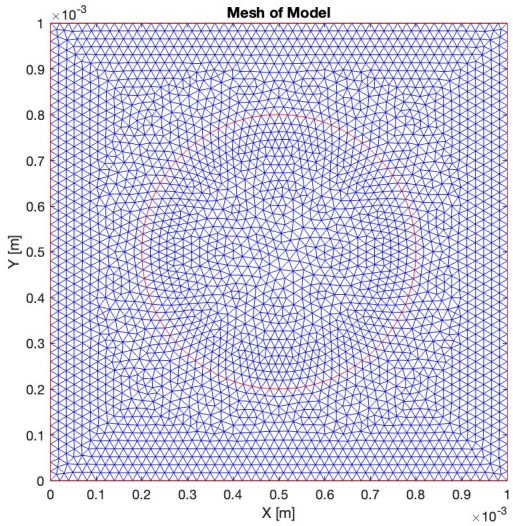


Figure 3: L/50 Mesh Size Visualization

3 Effective Conductivity Tensor

The tensor $[k]$ is the effective thermal conductivity tensor of the composite domain. In a heterogeneous domain composed of multiple materials (e.g., a square matrix with a circular inclusion), each material region may have its own thermal conductivity properties. For isotropic materials, thermal conductivity is uniform in all directions, and $[k]$ reduces to a scalar value k_0 often represented in tensor form as $k_0 I$ where I is the identity matrix. For anisotropic materials, the thermal conductivity varies with direction, making $[k]$ a full tensor that can have distinct diagonal entries and potentially off-diagonal terms:

$$k = \begin{bmatrix} k_x & k_{xy} \\ k_{yx} & k_y \end{bmatrix}$$

In this project, the focus is on a simplified scenario where the conductivity tensors are diagonal:

$$k = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

The final effective conductivity tensor $[\overline{k}]$ for the entire composite domain is computed by blending the contributions from all elements according to:

$$[\overline{k}] = \frac{1}{|\Omega|} \sum_{e=1}^{N} \int_{\Omega_e} [k_e] \, [B_e(\mathbf{x})] \, [T_e] \, d\mathbf{x} \tag{3}$$

where $|\Omega|$ is the area of the entire domain, and the summation and integral consider all elements and their respective local fields.

3.1 $[k_e], [B_e(\mathbf{x})], [T_e]$ calculation

1. $[k_e]$ – **Element Conductivity Tensor:** Each element e inherits its thermal conductivity from the material it occupies. If the element lies in the matrix region, $[k_e]$ may be a simple scalar or diagonal tensor with uniform conductivity. If the element is in the fiber/inclusion region, $[k_e]$ may have different values or directions. Thus, $[k_e]$ is determined based on the element's centroid or nodal coordinates that identify which material region it belongs to.

2. $[B_e]$ – **Shape Function Derivatives in Physical Space:** The matrix $[B_e]$ is formed from the spatial derivatives of the element's shape functions. Starting from standard reference-element shape functions, a coordinate transformation maps these functions to the actual (physical) element coordinates. The resulting $[B_e]$ matrix is 2×3 (for linear triangular elements) or $2 \times n$ (depending on the number of element nodes) matrix containing the partial derivatives of each shape function with respect to x and y. Essentially, for a 3-node triangular element:

$$[B_e] = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \dfrac{\partial N_3}{\partial x} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_3}{\partial y} \end{bmatrix}.$$

3. **[Te]** – **Element Temperature Values:** After solving the global finite element system, nodal temperatures $T(\mathbf{x}_i)$ at each node i are extracted for each element e to form the vector:

$$[T_e] = \begin{bmatrix} T(\mathbf{x}_1) \\ T(\mathbf{x}_2) \\ T(\mathbf{x}_3) \end{bmatrix}$$

   in the case of a triangular element. This vector provides the necessary local solution data to evaluate the integrand $[k_e][B_e][T_e]$.

4 Numerical Integration Scheme

To compute the integral

$$\int_{\Omega_e} [k_e][B_e(\mathbf{x})][T_e]\, d\mathbf{x},$$

Gaussian quadrature is often used as a numerical integration technique. Gaussian quadrature approximates the integral by evaluating the integrand at specific quadrature points and weighting the results. The typical process involves the following steps:

1. **Select Quadrature Points and Weights:** Choose two quadrature points (ξ_i, η_i) and their corresponding weights w_i in the reference element (e.g., the unit triangle). These points and weights are predefined to ensure accurate integration for a given polynomial degree.

2. **Map to Physical Coordinates:** Using the nodal coordinates of the element, the reference Gauss points are mapped into the element's physical domain. This mapping accounts for the element's geometry, which may not match the unit triangle's shape.

3. **Evaluate the Integrand:** At each quadrature point in the physical domain, compute the integrand $[k_e][B_e][T_e]$, which includes the local conductivity tensor $[k_e]$, the shape function derivatives $[B_e]$, and the temperature field $[T_e]$. For T3 elements, where $[B_e]$ and $[k_e]$ are constant within each element, this step is straightforward.

4. **Sum and Multiply by Weights:** The integral is approximated as:

$$\int_{\Omega_e} [k_e][B_e][T_e]\, d\mathbf{x} \approx \sum_{i=1}^{2} w_i \left([k_e][B_e][T_e]\right)_{\mathbf{x}_i} |\det(J)|$$

where $|\det(J)|$ is the determinant of the Jacobian matrix, which maps the reference element to the physical element.

However, in this study, the use of Gaussian quadrature and explicit mapping to the reference domain was not necessary because T3 elements (linear triangular elements) were employed. For these elements:

- The shape function derivatives $[B_e]$ are constant across the element, so the integrand $[k_e][B_e][T_e]$ is also constant.
- The integral simplifies to the product of the integrand and the element area:

$$\int_{\Omega_e} [k_e][B_e][T_e]\, d\mathbf{x} = [k_e][B_e][T_e] \cdot \text{Area}_e.$$

- The area of each element is computed directly from the nodal coordinates in the physical domain without requiring a reference-to-physical mapping.

This simplification reduces computational effort while maintaining accuracy, as the constant integrand allows for exact integration over each triangular element. The approach balances computational efficiency with numerical accuracy, making it well-suited for this finite element analysis.

5 Discussion: Isotropic vs. Anisotropic Material

A material is **isotropic** if its thermal conductivity is the same in every direction. In such a case:

$$[k] = k_0 I = \begin{bmatrix} k_0 & 0 \\ 0 & k_0 \end{bmatrix}.$$

An **anisotropic** material, however, has direction-dependent thermal conductivity:

$$[k] = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad \text{with} \quad k_x \neq k_y.$$

To determine whether this unit cell is isotropic vs. anisotropic, the temperature distribution field can be plotted and the output k_x and ky investigated. The results are as follows:

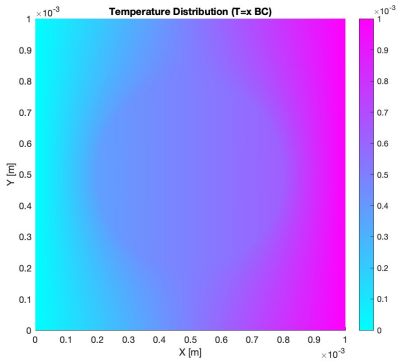


Figure 4: Problem 1 Temperature Field

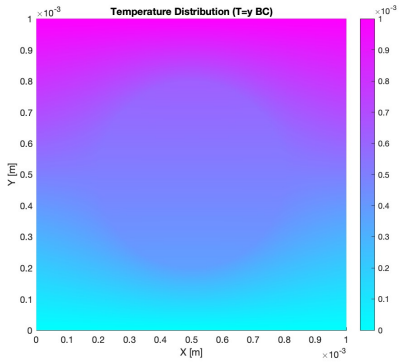


Figure 5: Problem 2 Temperature Field

$$[\overline{k}] = \begin{bmatrix} 1.4749 & 0 \\ 0 & 1.4749 \end{bmatrix}$$

The presence of a temperature gradient in the graph does not signify whether the unit cell is isotropic vs. anisotropic. The gradient just indicates that there is a driving force for heat flow (such as different boundary condition temperatures), not that the material itself has directionally dependent properties.

Our analysis must be based on the effective conductivity tensor $[\overline{k}]$ which collects the k_{xx} and the k_{yy} term. The result, $k_{xx} = k_{yy} = 1.4749$, proves that the **unit cell of material 1 and 2 is isotropic.** These k components provide a measure of how readily heat flows along each principal axis of the domain. Since k_{xx} and k_{yy} are equal, it suggests that the material does not favor one principal direction over the other, indicating isotropic behavior in terms of heat conduction. To add on, since material 1 and 2 have conductive heat transfer coefficients of 1 and 5 respectively, a final k of 1.4749 makes sense because $1 < 1.4749 < 5$.

Mathematics aside, since the problem is symmetric and the geometry, material distribution, and boundary conditions do not introduce any directional preference, the resulting effective thermal conductivity in the x and y directions should be the same. This symmetry means that there is no reason for heat to flow more easily in one direction versus the other so a this cell being isotropic makes logical sense.

6 Patch Test for Finite Element Verification

The patch test is a standard procedure used to verify the correctness of finite element implementations. It involves solving a simplified problem with a known analytical solution and comparing the numerical results to the exact solution. For a properly implemented FEM solver, the numerical solution should match the exact solution with negligible error.

In this project, the patch test was designed to verify the finite element solver for steady-state heat conduction. The exact solution chosen for the test was:

$$T(x, y) = T_0 + ax + by,$$

where T_0, a, and b are constants. This solution satisfies the Laplace equation:

$$\nabla^2 T = 0,$$

with no internal heat source. To implement this test:

- A square domain of size $1 \times 1\,\text{m}$ was defined, representing a uniform material with thermal conductivity $k = 1\,\text{W/mK}$.
- Boundary conditions were applied to match the exact solution:
  - Bottom edge ($y = 0$): $T = T_0 + ax$,
  - Right edge ($x = L$): $T = T_0 + aL + by$,
  - Top edge ($y = L$): $T = T_0 + ax + bL$,
  - Left edge ($x = 0$): $T = T_0 + by$.
- A linear finite element mesh was generated with triangular elements.
- The numerical solution was compared to the exact solution $T(x, y)$ at all nodes in the domain.

The maximum absolute error between the numerical and exact solutions was found to be:

$$\text{Max Absolute Error} = \mathbf{0}.$$

A negligible error demonstrates that the finite element solver correctly implements the governing equation, boundary conditions, and interpolation functions. This verification gives confidence in the solver's ability to accurately model more complex problems involving multiple materials, non-uniform geometries, and non-linear solutions. With this result, the FEM solver is deemed reliable for further analysis.

7 Impact of Changing the Radius

The radius of the circular region affects the heat distribution and conductivity in the domain. As the radius of the circular region increases, it alters the relative sizes of the two materials (square and circle), changing the heat flow dynamics. The graph below shows the exponential increase between radius and the effective conductivity of components, proving a positive correlation. To add on, no matter the radius size, $k_{xx} = k_{yy}$ meaning the unit cell stays isotropic (expected).

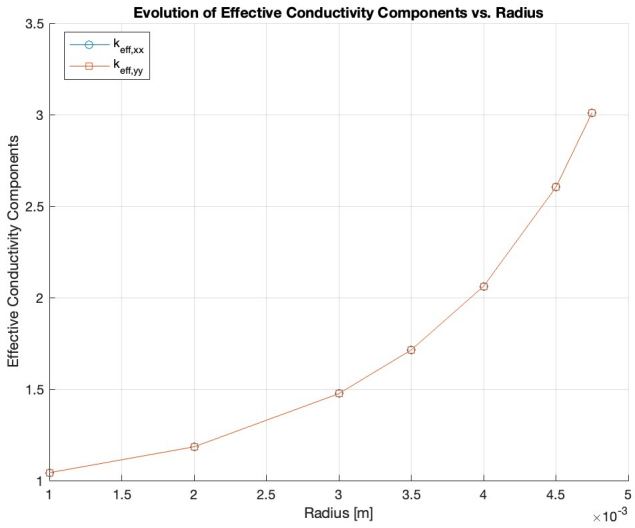


Figure 6: Evolution of the Effective Conductivity vs. Radius

8 Conclusion

In this project, I successfully implemented a numerical solution to the steady-state heat conduction problem using the finite element method. I examined the role of the effective conductivity tensor, implemented a Gaussian quadrature scheme for numerical integration, and studied the impact of material heterogeneity and mesh refinement on the solution accuracy. The results show that the material is isotropic. An investigation into how heat distribution evolves with increases of the circular region radius yielded expected results of increasing conductivity components. This data provides insights into a more complex material description and can be used to design structures with composite materials without resorting to a fine-scale description of the material.

9 Appendix

9.1 h-convergence code

```
clear
close all
clc

%% Generate the geometry
model = createpde('thermal', 'steadystate');
L = 0.001; % Length of the square domain in meters
rect = [3, 4, 0, L, L, 0, 0, 0, L, L]';
radius = 0.0003; % Radius of the circular region in meters
circle = [1, 0.5*L, 0.5*L, radius, 0, 0, 0, 0, 0, 0]';
gd = [rect, circle];
sf = 'R1+C1';
ns = char('R1', 'C1')';
g = decsg(gd, sf, ns);
geometryFromEdges(model, g);

%% Assign Thermal Properties
thermalProperties(model, 'ThermalConductivity', 1, 'Face', 1); % Square region
thermalProperties(model, 'ThermalConductivity', 5, 'Face', 2); % Circular region

%% Apply Boundary Conditions
thermalBC(model, 'Edge', 1:4, 'Temperature', @(location, state) location.x);

%% h-convergence and L2 norm
% Define a range of mesh sizes for convergence study
mesh_sizes = [L/2, L/25, L/50, L/100, L/150, L/175];

% Preallocate arrays to store results
L2_errors = zeros(size(mesh_sizes));
mesh_element_counts = zeros(size(mesh_sizes));

% Create Fine mesh for comparison
hmax_fine = L/200; % Fine mesh size
generateMesh(model, 'Hmax', hmax_fine);
results_fine = solve(model);
temperature_fine = results_fine.Temperature;
fine_coords = model.Mesh.Nodes;

% H-Convergence Loop
for i = 1:length(mesh_sizes)
    % Solve for Test Mesh
    hmax_test = mesh_sizes(i);
    generateMesh(model, 'Hmax', hmax_test);
    results_test = solve(model);
```

```
    temperature_test = results_test.Temperature;
    test_coords = model.Mesh.Nodes;

    % Interpolation of test mesh solution to fine mesh
    temperature_test_interp = zeros(size(temperature_fine));

    % Nearest neighbor interpolation
    for j = 1:length(temperature_fine)
        % Find the closest point in the test mesh
        [~, closest_idx] = min(sum((test_coords - fine_coords(:,j)).^2, 1));

        % Interpolate temperature from the closest point
        temperature_test_interp(j) = temperature_test(closest_idx);
    end

    % Compute the L2 norm of the error between the fine and interpolated test solutions
    error = temperature_fine - temperature_test_interp;
    L2_errors(i) = norm(error, 2); % L2 norm calculation

    % Store mesh element count
    mesh_element_counts(i) = size(model.Mesh.Elements, 2);
end

% Visualization of Convergence
figure;
subplot(2,1,1);
loglog(mesh_sizes, L2_errors, '-o', 'LineWidth', 2);
title('L2 Error Convergence');
xlabel('Mesh Size (h)');
ylabel('L2 Error');
grid on;
axis tight;  % First fit the axis to the data
axis padded; % Then add some padding

subplot(2,1,2);
loglog(mesh_element_counts, L2_errors, '-o', 'LineWidth', 2);
title('L2 Error vs Number of Mesh Elements');
xlabel('Number of Mesh Elements');
ylabel('L2 Error');
grid on;
axis tight;  % First fit the axis to the data
axis padded; % Then add some padding

% Display Errors
disp('Mesh Sizes:');
disp(mesh_sizes);
disp('L2 Errors:');
disp(L2_errors);
```

9.2 Effective Conductivity Code

```
clear; close all; clc;

%% Geometry and Material Parameters
L = 0.001; % [m]
radius = 0.0003; % [m]
domainArea = L^2;

%% Create PDE Models
model_x = createpde('thermal', 'steadystate');
model_y = createpde('thermal', 'steadystate');

% Define geometry: square + circle
rect = [3,4, 0, L, L, 0, 0,0, L,L]';
circle = [1, 0.5*L, 0.5*L, radius, 0,0,0,0,0,0]';
gd = [rect, circle];
ns = char('R1','C1')';
sf = 'R1+C1';
g = decsg(gd,sf,ns);

geometryFromEdges(model_x,g);
geometryFromEdges(model_y,g);

% Assign thermal properties
thermalProperties(model_x, 'ThermalConductivity', 1, 'Face',1);
thermalProperties(model_x, 'ThermalConductivity', 5, 'Face',2);

thermalProperties(model_y, 'ThermalConductivity', 1, 'Face',1);
thermalProperties(model_y, 'ThermalConductivity', 5, 'Face',2);

%% Boundary Conditions
% T = x on outer boundary for model_x
thermalBC(model_x,'Edge',1:4,'Temperature',@(loc,~)loc.x);

% T = y on outer boundary for model_y
thermalBC(model_y,'Edge',1:4,'Temperature',@(loc,~)loc.y);

%% Mesh
Hmax_value = L/50;
generateMesh(model_x, 'Hmax', Hmax_value, 'GeometricOrder', 'linear'); % Creates T3
generateMesh(model_y, 'Hmax', Hmax_value, 'GeometricOrder', 'linear'); % T6 by default

%% Plot the Mesh for Visual Reference
figure;
pdeplot(model_y, 'Mesh', 'on'); % Plots the mesh for model_y
axis equal tight;
xlabel('X [m]'); ylabel('Y [m]');
```

```
title('Mesh of Model');

%% Solve the Problems
results_x = solve(model_x);
results_y = solve(model_y);

T_x = results_x.Temperature; % Temperature field for T=x condition
T_y = results_y.Temperature; % Temperature field for T=y condition

nodes = model_x.Mesh.Nodes;      % 2xN
elements = model_x.Mesh.Elements; % 3xNe

numElements = size(elements,2);

%% Compute Effective Conductivity
% Initialize integral sum
integral_sum = zeros(2,2);

for e = 1:numElements
    % Get node indices and coordinates for element e
    elemNodes = elements(:,e);
    Xe = nodes(1,elemNodes);
    Ye = nodes(2,elemNodes);

    % Extract element temperatures from both solutions
    Te = [T_x(elemNodes), T_y(elemNodes)]; % 3x2 matrix

    % Compute area of the element
    x1 = Xe(1); y1 = Ye(1);
    x2 = Xe(2); y2 = Ye(2);
    x3 = Xe(3); y3 = Ye(3);
    detJ = (x2 - x1)*(y3 - y1) - (x3 - x1)*(y2 - y1);
    area_e = abs(detJ)/2;

    % Compute shape function derivatives (B_e)
    % For a linear triangle (N1, N2, N3):
    % dN/dx = [ (y2 - y3), (y3 - y1), (y1 - y2)]/(2*A)
    % dN/dy = [ (x3 - x2), (x1 - x3), (x2 - x1)]/(2*A)
    b1 = (y2 - y3)/(2*area_e); b2 = (y3 - y1)/(2*area_e); b3 = (y1 - y2)/(2*area_e);
    c1 = (x3 - x2)/(2*area_e); c2 = (x1 - x3)/(2*area_e); c3 = (x2 - x1)/(2*area_e);

    B_e = [b1 b2 b3; c1 c2 c3]; % 2x3 matrix

    % Determine local conductivity k_e based on element centroid
    centroidX = mean(Xe);
    centroidY = mean(Ye);
    if ( (centroidX - 0.5*L)^2 + (centroidY - 0.5*L)^2 <= radius^2 )
        k_e = [5 0; 0 5]; % Fiber
```

```
    else
        k_e = [1 0; 0 1]; % Matrix
    end

    % Compute integrand [k_e][B_e][T_e]
    % Dimensions: (2x2)*(2x3)*(3x2) = (2x2)
    integrand = k_e * B_e * Te;

    % Integrate over the element: integrand is constant, so integral = integrand * area_e
    integral_sum = integral_sum + integrand * area_e;
end

% Divide by total area to get effective conductivity
k_eff = (1/domainArea)*integral_sum;

disp('Effective Conductivity Tensor [k]:');
disp(k_eff);

% Check isotropy
if abs(k_eff(1,2))<1e-12 && abs(k_eff(2,1))<1e-12
    disp('Off-diagonal terms are negligible.');
end
disp(['k_xx = ', num2str(k_eff(1,1)), ', k_yy = ', num2str(k_eff(2,2))]);

%% Visualization of the Temperature Distributions

% Plot the temperature field for T=x BC case
figure;
pdeplot(model_x,'XYData',results_x.Temperature,'Title','Temperature Distribution (T=x BC)','Mesh
colorbar; axis equal tight;
xlabel('X [m]'); ylabel('Y [m]');

% Plot the temperature field for T=y BC case
figure;
pdeplot(model_y,'XYData',results_y.Temperature,'Title','Temperature Distribution (T=y BC)','Mesh
colorbar; axis equal tight;
xlabel('X [m]'); ylabel('Y [m]');

%% Interpreting Isotropy
disp('Effective Conductivity Tensor [k]:');
disp(k_eff);

% Grab values
k_xx = k_eff(1,1);
k_yy = k_eff(2,2);
k_xy = k_eff(1,2);
k_yx = k_eff(2,1);
```

```
disp('Check isotropy:');
disp(['k_xx = ', num2str(k_xx), ', k_yy = ', num2str(k_yy)]);
disp(['k_xy = ', num2str(k_xy), ', k_yx = ', num2str(k_yx)]);

if abs(k_xy) < 1e-12 && abs(k_yx) < 1e-12 && abs(k_xx - k_yy) < 1e-12
    disp('The material appears isotropic.');
else
    disp('The material appears anisotropic.');
end
```

9.3 k Evolution Plotting

```
% Initialize data arrays
radius = [0.001, 0.002, 0.003, 0.0035 ,0.004, 0.0045, 0.00475];
k_eff_xx = [1.0428, 1.1844, 1.4749, 1.7138, 2.0614, 2.6029, 3.0109];
k_eff_yy = [1.0428, 1.1844, 1.4749, 1.7138, 2.0614, 2.6029, 3.0109];
% Create the plot
figure;
hold on;
plot(radius, k_eff_xx, '-o', 'DisplayName', 'k_{eff,xx}'); % Plot for k_xx
plot(radius, k_eff_yy, '-s', 'DisplayName', 'k_{eff,yy}'); % Plot for k_yy

% Customize the plot
xlabel('Radius [m]');
ylabel('Effective Conductivity Components');
title('Evolution of Effective Conductivity Components vs. Radius');
legend('Location', 'best');
grid on;
hold off;
```

9.4 Patch Test Code

```
clear; close all; clc;

%% Domain Definition
L = 1.0;
rect = [3,4,0,L,L,0,0,0,L,L]';
gd = rect;
ns = char('R1')';
sf = 'R1';

% Thermal Conductivity
k = 1;

% Exact solution: T_exact = T0 + a*x + b*y
T0 = 100;
a = 10;
b = -5;

model_patch = createpde('thermal','steadystate');
g_patch = decsg(gd,sf,ns);
geometryFromEdges(model_patch,g_patch);

figure;
pdegplot(model_patch,'EdgeLabels','on'); axis equal;
title('Edge Labels');


% Set thermal properties (uniform conductivity)
thermalProperties(model_patch,'ThermalConductivity',k);

% Boundary conditions
% 1 = Left, 2 = Bottom, 3 = Right, 4 = Top
thermalBC(model_patch,'Edge',1,'Temperature',@(loc,~)T0 + a*loc.x);          % bottom
thermalBC(model_patch,'Edge',2,'Temperature',@(loc,~)T0 + a*L + b*loc.y);   % right
thermalBC(model_patch,'Edge',3,'Temperature',@(loc,~)T0 + a*loc.x + b*L);   % top
thermalBC(model_patch,'Edge',4,'Temperature',@(loc,~)T0 + b*loc.y);         % left


% Generate a linear mesh
generateMesh(model_patch,'GeometricOrder','linear','Hmax',L/50);

% Solve the patch test problem
results_patch = solve(model_patch);

% Extract node coordinates and computed temperature
X = model_patch.Mesh.Nodes(1,:)';
Y = model_patch.Mesh.Nodes(2,:)';
```

```
T_num_patch = results_patch.Temperature;

% Compute exact solution at node coordinates
T_ex_patch = T0 + a*X + b*Y;

% Compute error
error_patch = T_num_patch - T_ex_patch;
max_error_patch = max(abs(error_patch));

disp('PATCH TEST RESULTS');
disp(['Max absolute error in patch test: ', num2str(max_error_patch)]);

% Plot the numerical solution
figure;
pdeplot(model_patch,'XYData',results_patch.Temperature,'Title','Patch Test Temperature', 'mesh',
colorbar; axis equal tight;
xlabel('x'); ylabel('y');
```