ME3180

FEM & CFD Theory

Assignment 2

Abhishek Ghosh

ME21BTECH11001

2D Steady State Heat Conduction Equation

Consider a case of steady heat conduction in a long square slab in which heat is generated at a uniform rate of q ''' W/m3 as shown in Fig 1. The top and right sides are maintained at T = T∞, the temperature of the surrounding fluid. The other two sides are insulated. Solve the steady state 2D heat conduction equation numerically and plot the steady state contour of the temperature distribution within the slab. Use a second-order central difference scheme to discretize your governing partial differential equation.

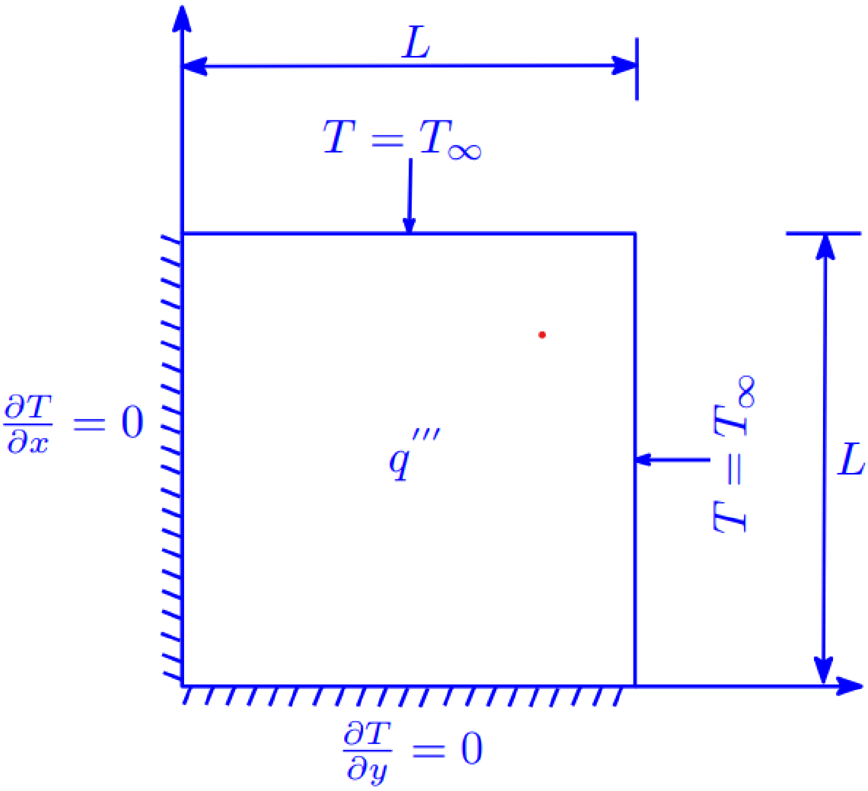
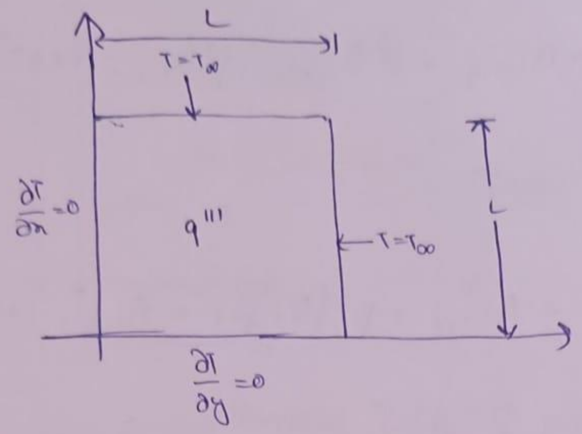


Figure 1: Computational Domain

ME 3180 Assignment 2

Abhishek Ghosh

ME21BTECH11001



Governing eqn: $\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{q'''}{k} = 0$

Non-dimensional → $\theta = \frac{T - T_\infty}{q''' L^2 / k}$

@ $x = L$ $\theta = 0$ & @ $y = L$ $\theta = 0$

@ $x = 0$ $\frac{\partial \theta}{\partial x} = 0$ & @ $y = 0$ $\frac{\partial \theta}{\partial y} = 0$

eqn: $\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} + 1 = 0$

Discretizing using central diff scheme :—

$$\frac{\theta_{i+1,j} - 2\theta_{i,j} + \theta_{i-1,j}}{\Delta x^2} + \frac{\theta_{i,j+1} - 2\theta_{i,j} + \theta_{i,j-1}}{\Delta y^2} + 1 = 0$$

$$\beta = \frac{\Delta x^2}{\Delta y^2}$$

$$\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j} + \beta^2\left(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}\right) + \Delta x^2 = 0$$

$$\phi_{i,j} = \frac{1}{2(1+\beta^2)}\left[\phi_{i+1,j} + \phi_{i-1,j} + \beta^2\phi_{i,j+1} + \beta^2\phi_{i,j-1} + \Delta x^2\right]$$

Gauss Siedel for k^{th} iteratn

$$\phi_{i,j}^{k} = \frac{1}{2(1+\beta^2)}\left[\phi_{i+1,j}^{k-1} + \phi_{i-1,j}^{k} + \beta^2\left(\phi_{i,j+1}^{k-1} + \phi_{i,j-1}^{k}\right) + \Delta x^2\right]$$

Gauss siedel with over & under relaxatn →

$$\phi_{i,j}^{k} = (1-\alpha)\,\phi_{i,j}^{k-1} + \alpha\frac{1}{2(1+\beta^2)}\left[\phi_{i+1,j}^{k-1} + \phi_{i-1,j}^{k} + \beta^2\left(\phi_{i,j+1}^{k-1} + \phi_{i,j-1}^{k}\right) + \Delta x^2\right]$$

chose $0 < \alpha < 1$ → under relaxatn

$1 < \alpha < 2$ → over relaxatn

Line by line

$$2(1+\beta^2)\,\phi_{i,j}^{k} - \phi_{i-1,j}^{k} + \phi_{i+1,j}^{k} = \Delta x^2 + \beta^2\phi_{i,j+1}^{k-1} + \beta^2\phi_{i,j-1}^{k-1}$$

TDMA →

where $a = 2(1+\beta^2)$

$b = 1$

$c = 1$

$d = \Delta x^2 + \beta^2\phi_{i,j+1} + \beta^2\phi_{i,j-1}$

Code & Results:

Initial Conditions

```
% Abhishek Ghosh
% ME21BTECH11001

% Given parameters
L = 1;
nx = 31;
ny = 31;

x = linspace(0, L, nx);
y = linspace(0, L, ny);

dx = x(2) - x(1);
dy = y(2) - y(1);
beta = dx / dy;
Tolerance = 1e-4;
```

Gauss Siedel

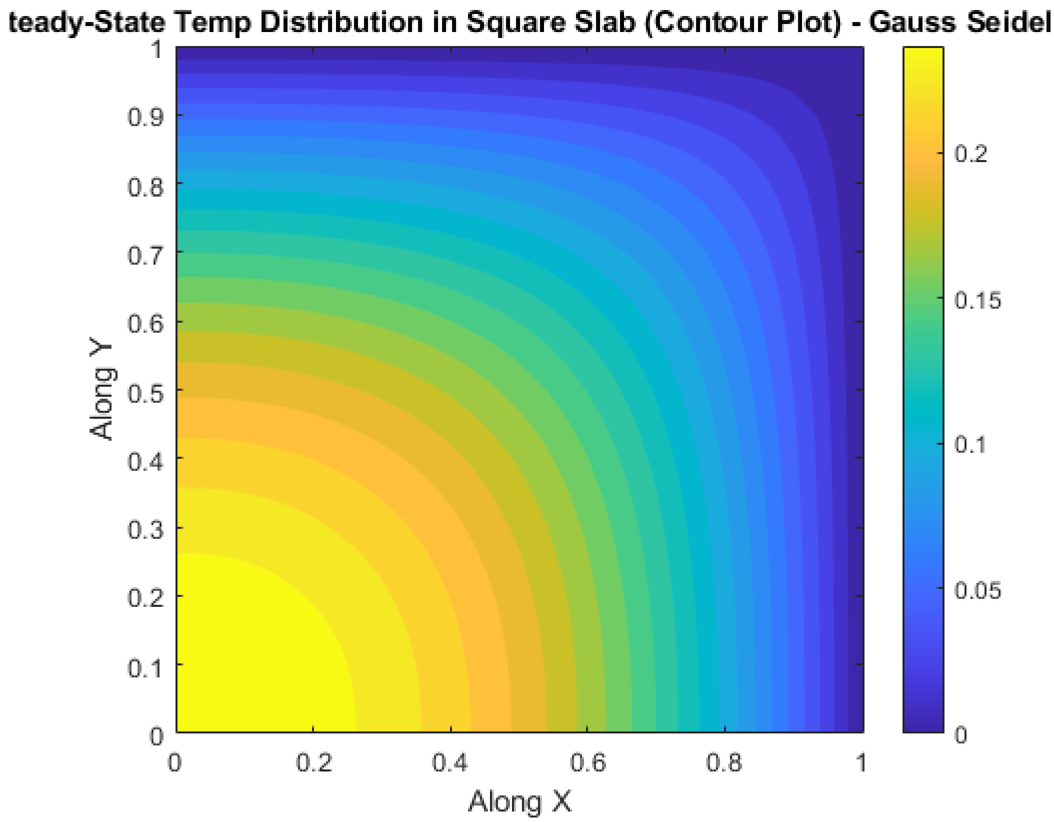
```
% Gauss Siedel Method

% Coefficient Matrix
theta = zeros(nx, ny);
theta_old = theta;

theta(nx, :) = 0;
theta(:, ny) = 0;

it_gs = 0;
err_gs = 1;
tol_gs = 1e-4;

while err_gs > tol_gs
    for i = 2:nx-1
        for j = 2:ny-1
            theta(i, j) = 0.5 * (dx^2 + theta_old(i+1,j) + theta(i-1, j) +
beta^2*theta_old(i, j+1) + beta^2*theta(i, j-1)) / (1 + beta^2);
        end
    end

    theta(2:end, 1) = theta(2:end, 2);
    theta(1, 1:end-1) = theta(2, 1:end-1);

    err_gs = max(max(abs(theta - theta_old)));
    it_gs = it_gs + 1;
    theta_old = theta;
end

fprintf('No. of iterations in Gauss-Seidel Method: %d\n', it_gs);
```

```
No. of iterations in Gauss-Seidel Method: 805
```

Contour

```
[X_gs, Y_gs] = meshgrid(x, y);

figure;
```

```
contourf(X_gs, Y_gs, theta, 20, 'LineColor', 'none');
colorbar();
title('Steady-State Temp Distribution in Square Slab (Contour Plot) - Gauss
Seidel');
xlabel('Along X');
ylabel('Along Y');
axis equal;
axis tight;
```



Gauss Siedel with Successive Over Relaxation

```
% Gauss Siedel with Successive Over-Relaxation (SOR)
theta = zeros(nx, ny);
theta_old = theta;

theta(1, :) = 0;
theta(:, ny) = 0;

it_sor = 0;
err_sor = 1;
tol_sor = 1e-4;
alpha_sor = 1.8;

while err_sor > tol_sor
    for i = 2:nx-1
        for j = 2:ny-1
            theta(i, j) = (1 - alpha_sor)*theta_old(i,j) + (alpha_sor*0.5*(dx^2 +
theta_old(i+1,j) + theta(i-1, j) + (beta^2)*theta_old(i, j+1) + (beta^2)*theta(i,
j-1))) / (1 + beta^2);
        end
    end

    theta(2:end, 1) = theta(2:end, 2);
```

```
    theta(1, 1:end-1) = theta(2, 1:end-1);

    err_sor = max(max(abs(theta - theta_old)));
    it_sor = it_sor + 1;
    theta_old = theta;
end

fprintf('No. of iterations in Gauss-Seidel Method with SOR: %d\n', it_sor);
```

```
No. of iterations in Gauss-Seidel Method with SOR: 225
```

Gauss Siedel with Under Relaxation

```
% Gauss Siedel with Under-Relaxation (UR)
theta = zeros(nx, ny);
theta_old = theta;

theta(1, :) = 0;
theta(:, ny) = 0;

it_ur = 0;
err_ur = 1;
tol_ur = 1e-4;
alpha_ur = 0.6;

while err_ur > tol_ur
    for i = 2:nx-1
        for j = 2:ny-1
            theta(i, j) = (1 - alpha_ur)*theta_old(i,j) + (alpha_ur*0.5*(dx^2 +
theta_old(i+1,j) + theta(i-1, j) + (beta^2)*theta_old(i, j+1) + (beta^2)*theta(i,
j-1))) / (1 + beta^2);
        end
    end

    theta(2:end, 1) = theta(2:end, 2);
    theta(1, 1:end-1) = theta(2, 1:end-1);

    err_ur = max(max(abs(theta - theta_old)));
    it_ur = it_ur + 1;
    theta_old = theta;
end

fprintf('No. of iterations in Gauss-Seidel Method with UR: %d\n', it_ur);
```

```
No. of iterations in Gauss-Seidel Method with UR: 1128
```

Line by Line Gauss Siedel Method

```
% We are sweeping in the x direction
% Assume two known in y-direction
% Use Line by Line Gauss Seidel to solve the generated tridiagonal matrix
% Initializing the grid of temperature
T_ll = zeros(nx, ny);

% Initializing Dirichlet boundary conditions
T_ll(nx, :) = 0;
T_ll(:, ny) = 0;
```

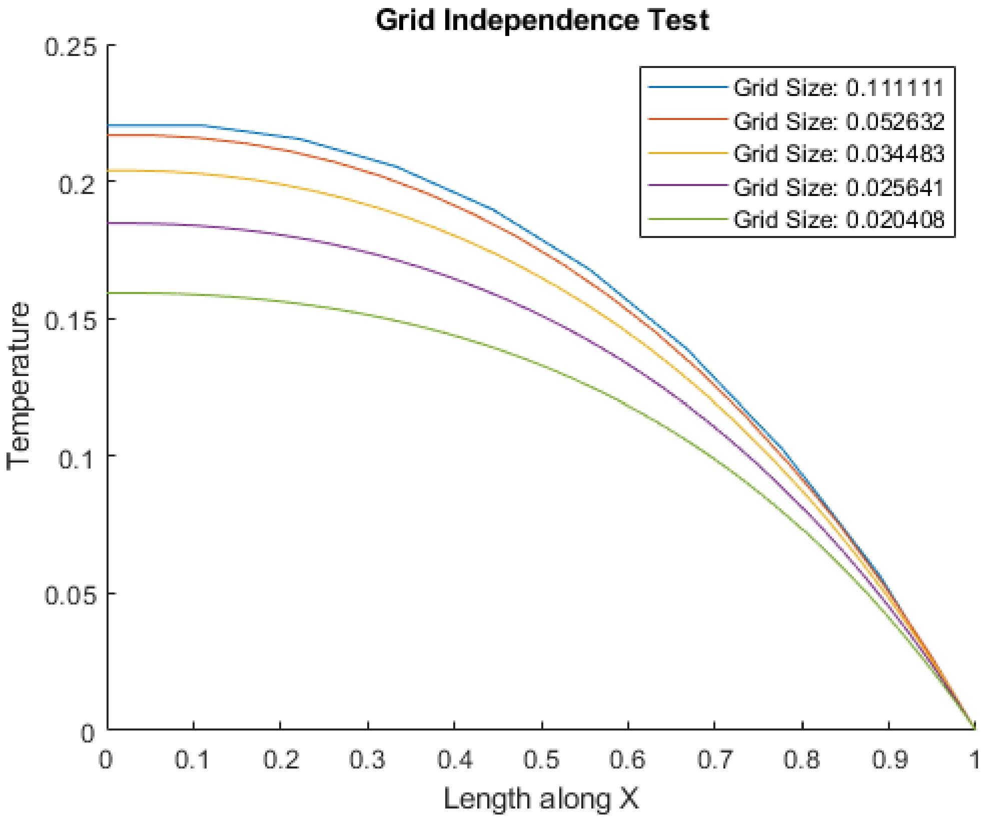
```
T_ll_old = T_ll;

iterations = 0;
Error = 1;

while Error > Tolerance
    for i = 2:nx-1
        % Using TDMA
        T_tdma = zeros(ny, 1);
        T_tdma(ny) = 0;

        P = zeros(ny, 1);
        Q = zeros(ny, 1);

        a = 2 * (1 + beta^2);
        b = 1;
        c = 1;

        P(1) = 1;
        Q(1) = 0;
        d = zeros(ny, 1);

        for k = 1:ny
            d(k) = dx^2 + beta^2 * T_ll(i-1, k) + beta^2 * T_ll(i+1, k);
        end

        for j = 2:ny-1
            P(j) = b / (a - c * P(j-1));
            Q(j) = (d(j) + c * Q(j-1)) / (a - c * P(j-1));
        end

        Q(ny) = T_tdma(ny);

        for j = ny-1:-1:1
            T_tdma(j) = T_tdma(j+1) * P(j) + Q(j);
        end

        T_ll(i, :) = T_tdma';
    end

    % Initializing Neumann boundary conditions
    T_ll(1, :) = T_ll(2, :);

    Error = max(max(abs(T_ll - T_ll_old)));
    iterations = iterations + 1;
    T_ll_old = T_ll;
end

disp(['No. of iterations in Line by Line Gauss-Seidel Method: ',
num2str(iterations)]);
```

```
No. of iterations in Line by Line Gauss-Seidel Method: 530
```

Centre Line Temperature Distribution along x axis

```
% center line plot
grid_points = [10,20,30,40,50];
figure;
hold on;

for gp = grid_points
    T_plot = gauss_siedel(gp);
    T_mid_x = T_plot(round(gp/2), :);
```

```
    x_t = linspace(0, L, gp);

    plot(x_t, T_mid_x, 'DisplayName', sprintf('Grid Size: %.6f', x_t(2) - x_t(1)));
end

hold off;
xlabel('Length along X');
ylabel('Temperature');
title('Grid Independence Test');
legend;
```



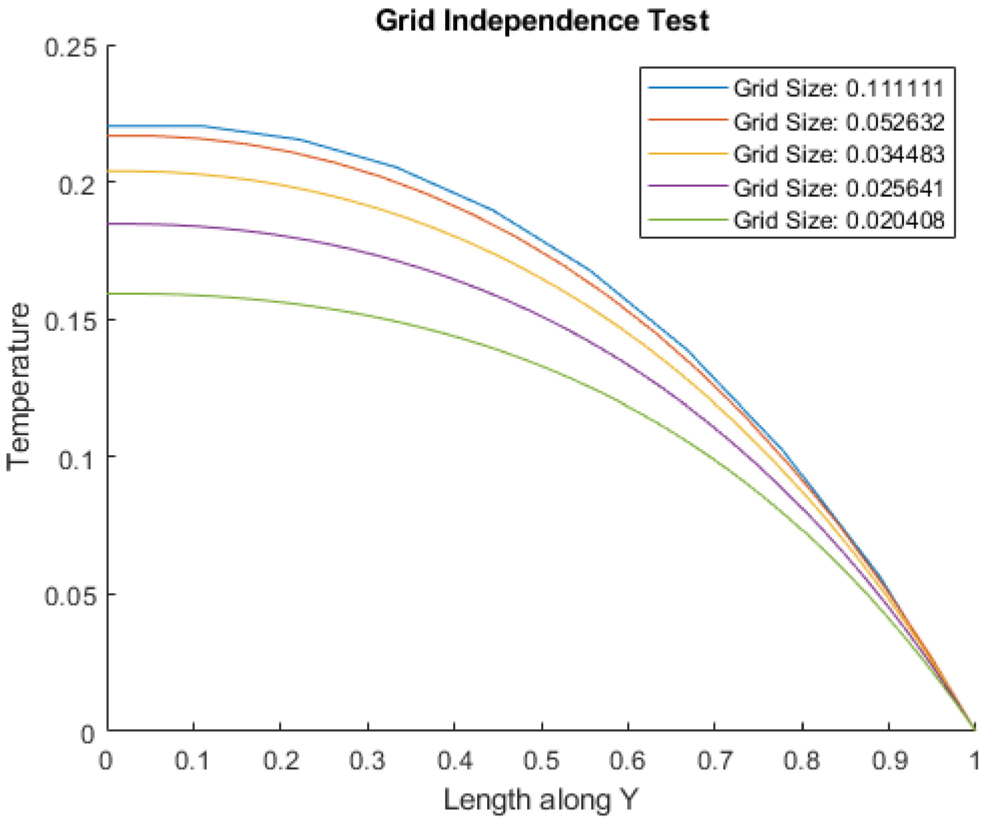

```
Grid Size: 0.111111, Iterations: 151

Grid Size: 0.052632, Iterations: 457

Grid Size: 0.034483, Iterations: 775

Grid Size: 0.025641, Iterations: 1035

Grid Size: 0.020408, Iterations: 1184
```

Centre Line Temperature Distribution along y axis

```
figure;

hold on;
for gp = grid_points
    T_plot = gauss_siedel(gp);
    T_mid_y = T_plot(:,round(gp/2));
    x_t = linspace(0, L, gp);
```

```
    plot(x_t, T_mid_y, 'DisplayName', sprintf('Grid Size: %.6f', x_t(2) - x_t(1)));
end

hold off;
xlabel('Length along Y');
ylabel('Temperature');
title('Grid Independence Test');
legend;
```

```
function T_gs = gauss_siedel(n)
    T_gs = zeros(n, n);
    dx = 1 / (n - 1);
    dy = 1 / (n- 1);
    beta = dx / dy;

    % Initializing Dirichlet boundary conditions
    T_gs(n, :) = 0;
    T_gs(:, n) = 0;

    T_gs_old = T_gs;

    % To keep track of the number of iterations
    iterations = 0;
    Error = 2;
    Tolerance = 1e-4;
    while Error > Tolerance
        for i = 2:n-1
            for j = 2:n-1
                T_gs(i, j) = 0.5 * (dx^2 + T_gs_old(i+1, j) + T_gs(i-1, j) + beta^2
* T_gs_old(i, j+1) + beta^2* T_gs(i, j-1)) / (1 + beta^2);
            end
        end

        % Initializing Neumann boundary conditions
        T_gs(2:end, 1) = T_gs(2:end, 2);
        T_gs(1, 1:n-1) = T_gs(2, 1:n-1);

        Error = max(max(abs(T_gs - T_gs_old)));
        iterations = iterations + 1;
        T_gs_old = T_gs;
    end

    fprintf('Grid Size: %.6f, Iterations: %d\n', dx, iterations);
end
```




```
Grid Size: 0.111111, Iterations: 151

Grid Size: 0.052632, Iterations: 457

Grid Size: 0.034483, Iterations: 775

Grid Size: 0.025641, Iterations: 1035

Grid Size: 0.020408, Iterations: 1184
```