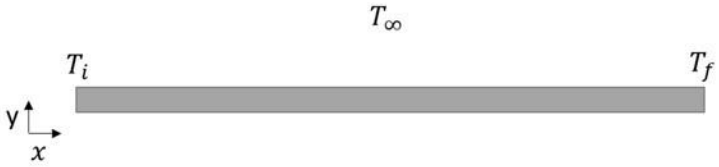
MEEN 644 – Numerical Heat Transfer and Fluid Flow
Spring 2020
HOMEWORK #5

Name: Yatharth Vaishnani Instructor: N. K. Anand

Due Date: April 9, 2020 Maximum Points: 100

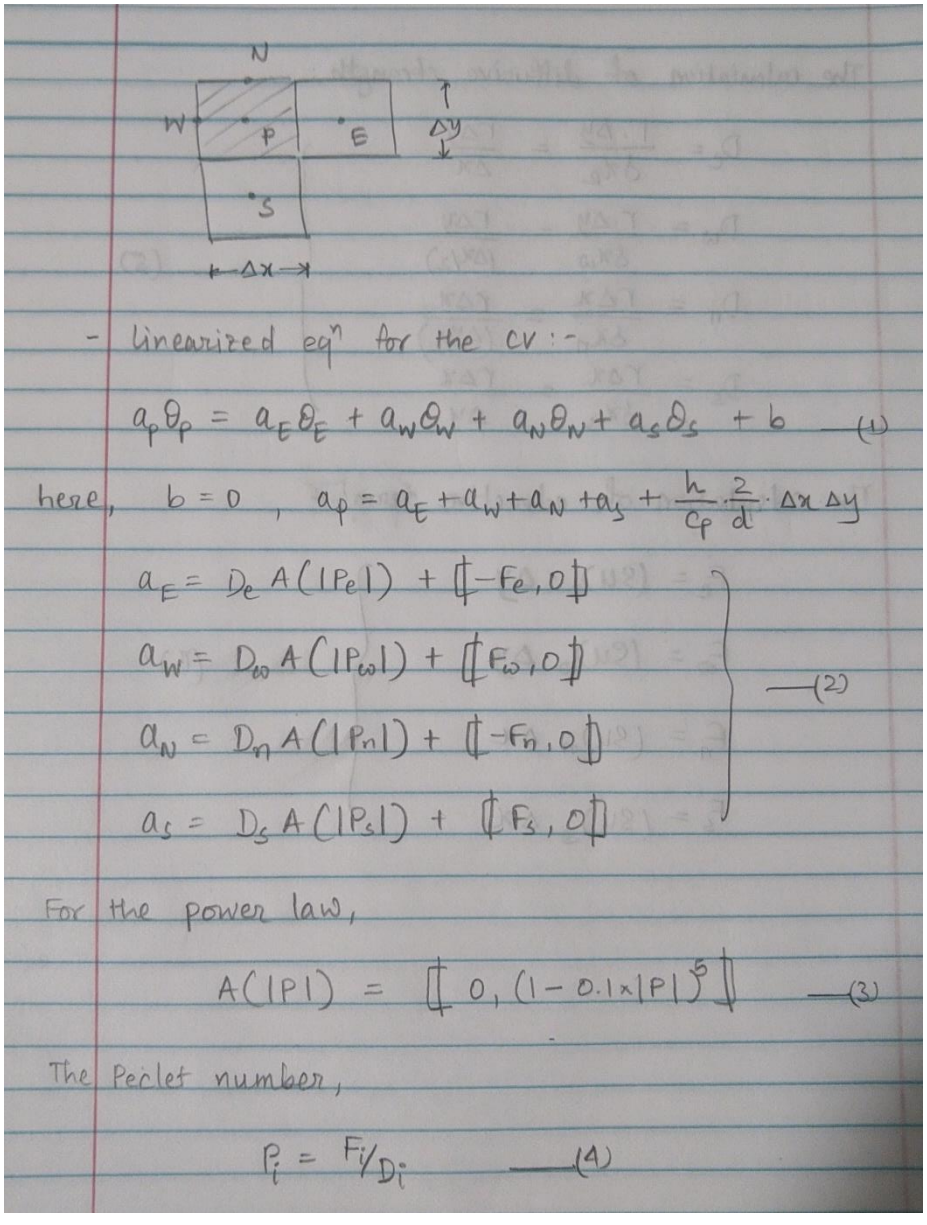


Consider a flow of liquid Lead in a 2D channel with length of 2 m and height of 0.05 m. The temperature at the entrance of the channel is held at (T_i) at 1,000 °C and the temperature at the exit (T_f) is held at exit at 700 °C. The surface of the channel is exposed to ambient temperature (T_∞) at 27 °C and convective heat transfer coefficient (h) is 2,000 W/m^2 °C. Density (ρ), specific heat capacity (c), thermal conductivity (k) of Lead is 10,000 kg/m^3, 140 J/kg °C, 21 W/m °C, respectively. Assume flow has constant velocity of 0.2 m/s in x direction.

Write a finite volume code to predict temperature distribution along the length of the channel and Nusselt number along the interior of the channel using power law. Use 100 uniformly sized CVs in stream-wise direction and 41 uniformly sized CVs in cross-stream direction. Plot and discuss your results. Include your code on the report.

Solution:

- Write down linearized equation for control volume at northwest corner surrounded by inlet and wall – 30 **Points**



The calculation of diffusive strength:

$$D_e = \frac{\Gamma \cdot \Delta y}{\delta x_e} = \frac{\Gamma \Delta y}{\Delta x}$$

$$D_w = \frac{\Gamma \cdot \Delta y}{\delta x_w} = \frac{\Gamma \Delta y}{(\Delta x/2)}$$

$$D_n = \frac{\Gamma \Delta x}{\delta x_n} = \frac{\Gamma \Delta x}{(\Delta y/2)}$$

$$D_s = \frac{\Gamma \Delta x}{\delta x_s} = \frac{\Gamma \Delta x}{\Delta y}$$

——— (5)

The calculation of advective terms:

$$F_e = (\rho u)_e \cdot \Delta y$$

$$F_w = (\rho u)_w \Delta y$$

$$F_n = (\rho u)_n \Delta x$$

$$F_s = (\rho u)_s \Delta x$$

——— (6)

- Build code
    - Please refer the appendix.

- Plot temperature at center, wall, and average along the channel.

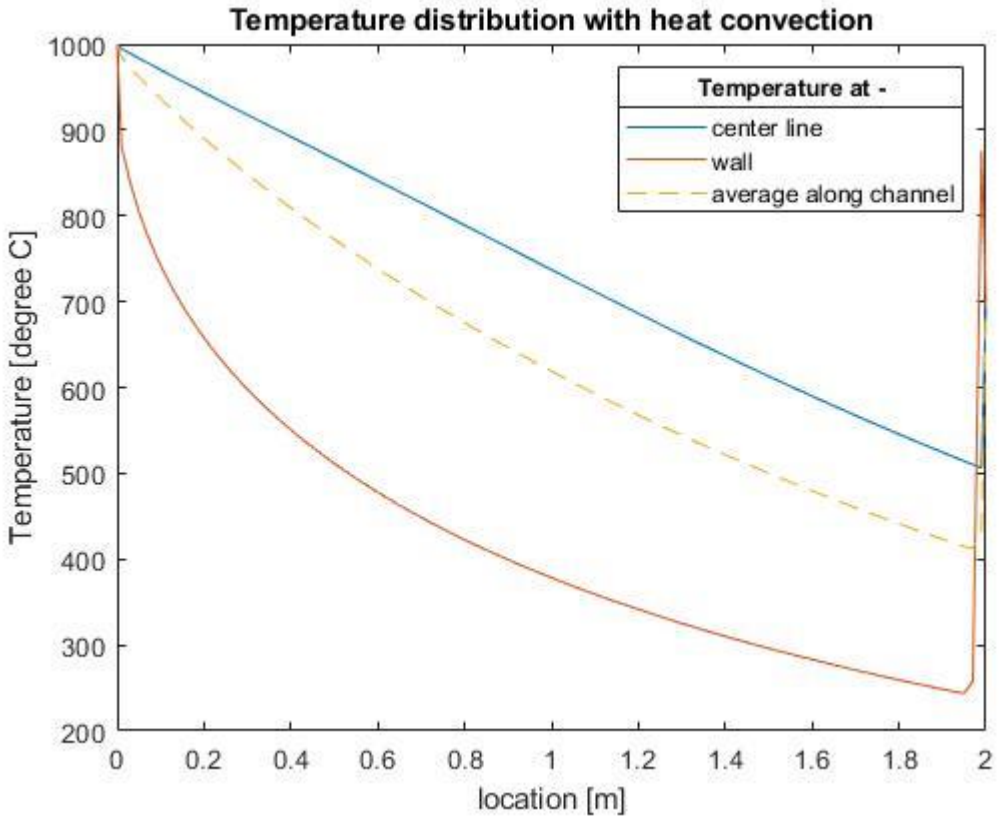


Figure 1 Temperature distribution across the length of the channel

- Plot Nusselt number along the interior wall of the channel. Its value might be higher than you thought.

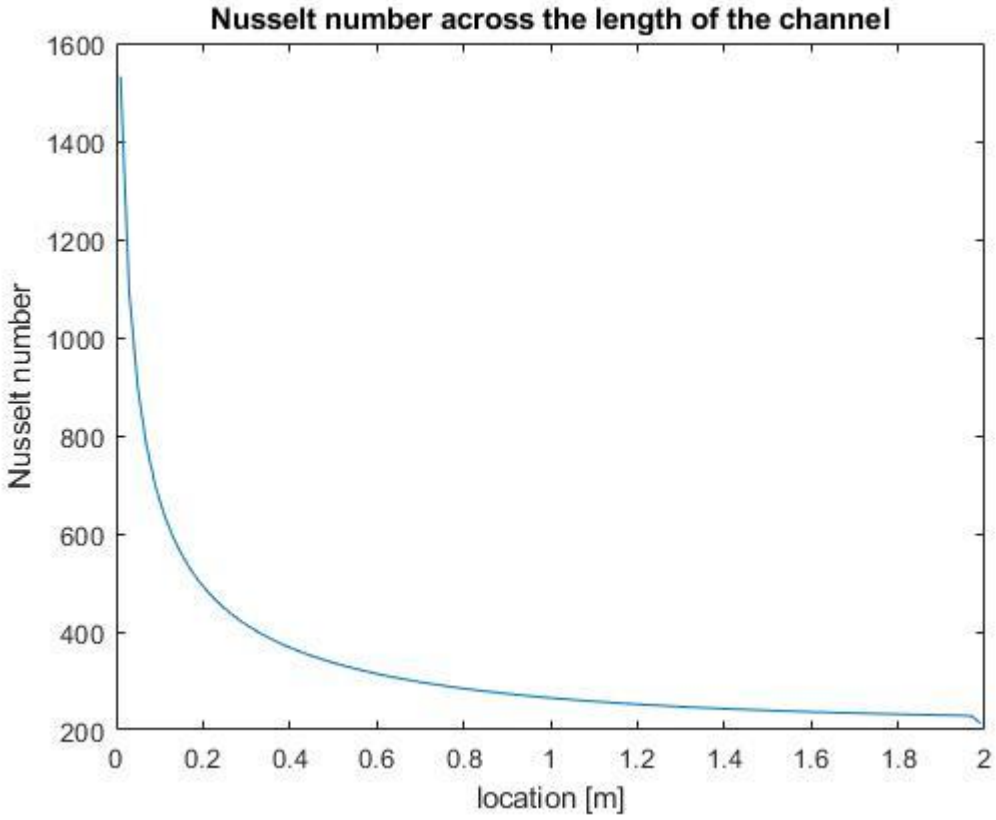
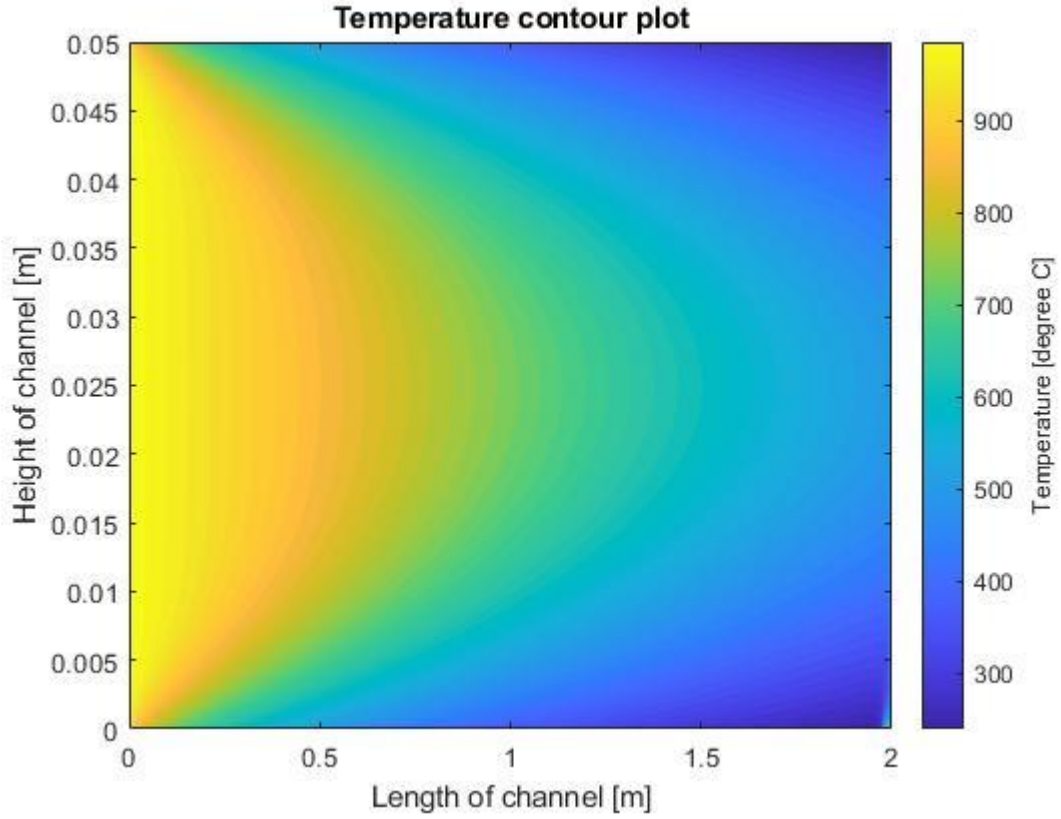


Figure 2 Nusselt number across the length of the channel

- Plot contour graph for temperature



- *Figure 3 Contour plot of temperature through the channel*

- Short discussion
  - The results are obtained using successive under-relaxation, with the value of the relaxation factor as 0.95.
  - The calculation of the Nusselt number for the flow of molten metal through horizontal channel is 190 (h = 2000 W/m^{2}K, k = 21 W/mK, L = 2 m, Nu = hL/k), which is similar to what obtained through the calculation of the temperature gradient at surface for conduction and the same for the convection.
  - The temperature profile obtained through the power law scheme is parabolic in nature across the height of the channel.
  - A perfect symmetry can be observed in the temperature profile with reference to the axis of the channel. This supports that the results are converged.

Appendix (MATLAB codes)

1. homework.m

```
% homework 5

% given data
Nx = 100;       % number of control volumes in stream-wise direction
Ny = 41;        % number of control volumes in cross-stream direction
L = 2;          % length of the channel (m)
d = 0.05;       % height of the channel (m)
rho = 10000;    % density of liquid lead (kg/m^3)
Cp = 140;       % specific heat capacity of liquid lead (J/kg C)
k = 21;         % thermal conductivity of liquid lead (W/m C)
h = 2000;       % convective heat transfer coefficient (W/m^2 C)
T_i = 1000;     % temperature at entrance (Celcius)
T_f = 700;      % temperature at exit (Celcius)
T_amb = 27;     % ambient temperature (Celcius)
u = 0.2;        % velocity in positive x direction (m/s)
v = 0;          % velocity in positive y direction (m/s)

omega = 0.95;   % over-relaxation factor

% solution:

x_axis = linspace(-L/(2*Nx), L + (L/(2*Nx)), Nx+2);
x_axis(1) = 0;
x_axis(end) = L;
y_axis = linspace(-d/(2*Ny), d + (d/(2*Ny)), Ny+2);
y_axis(1) = 0;
y_axis(end) = d;

% Power law scheme
theta = solve_adv_diff(Nx, Ny, L, d, h, k, Cp, rho, u, v, T_i-T_amb, T_f-T_amb,
"power_law", omega);
Temperature = theta + T_amb;
num = sum(rho*Cp*u.*Temperature(2:end-1, 2:end-1), 2);
denom = rho*Cp*u*Ny;
avg_temp = num / denom;

% plot temperature distribution
figure (1)
plot(x_axis, Temperature(:, (Ny+3)/2)')
hold on
plot(x_axis, Temperature(:, 1)')
hold on
plot(x_axis(2:end-1), avg_temp, '--')
```

```
lgd1 = legend('center line', 'wall', 'average along channel');
title(lgd1, 'Temperature at -')
xlabel('location [m]')
ylabel('Temperature [degree C]')
title('Temperature distribution with heat convection')
hold off

% plot Nusselt number
dy = d / Ny;
numerator = (Temperature(2:end-1, end) - Temperature(2:end-1, end-1)) / (dy/2);
denominator = (Temperature(2:end-1, end) - avg_temp) / (L);
nu = numerator./ denominator;
figure (2)
plot(x_axis, nu)
title('Nusselt number across the length of the channel')
xlabel('location [m]')
ylabel('Nusselt number')

% plot contour graph
figure (3)
contourf(x_axis, y_axis, Temperature', 50, 'edgecolor', 'none')
title('Temperature contour plot')
xlabel('Length of channel [m]')
ylabel('Height of channel [m]')
c = colorbar;
c.Label.String = 'Temperature [degree C]';
```

2. solve_adv_diff.m

```
% 2-D advection diffusion solution

% parameters required:
% N = number of CVs
% L = physical length of geometry (m)
% gamma = diffusion coefficient (kg/(m*s))
% rho = density of fluid (kg/m^3)
% u = velocity (m/s)
% phi_i = left-end boundary condition of quantity being solved (SI unit)
% phi_f = right-end boundary condition of quantity being solved (SI unit)
% solver_type = type of solver to be used for the calculations:
%   1) 'central_diff'
%   2) 'exponential'
%   3) 'power_law'

% return output:
```

```
% phi = distribution of the quantity over CVs

function [phi] = solve_adv_diff(Nx, Ny, L, d, h, k, Cp, rho, u, v, phi_i, phi_f,
solver_type, omega)
    % define phi for all nodes
    phi = zeros(Nx + 2, Ny + 2);

    % assign boundary conditions
    phi(1, :) = phi_i;
    phi(end, :) = phi_f;

    % calculate diffusion length array
    dx = L / Nx;
    dy = d / Ny;
    gamma = k / Cp;

    Dw = (dy * gamma / dx) * ones(Nx, Ny);
    Dw(1, :) = (dy * gamma /(dx / 2));
    De = (dy * gamma / dx) * ones(Nx, Ny);
    De(end, :) = (dy * gamma / (dx / 2));
    Ds = (dx * gamma / dy) * ones(Nx, Ny);
    Ds(:, 1) = (dx * gamma / (dy/2));
    Dn = (dx * gamma / dy) * ones(Nx, Ny);
    Dn(:, end) = (dx * gamma / (dy/2));

    % calculate flow strength
    Fw = rho * u * dy;
    Fe = rho * u * dy;
    Fs = rho * v * dx;
    Fn = rho * v * dx;

    % calculate Peclet number
    Pw = Fw * Dw.^-1;
    Pe = Fe * De.^-1;
    Ps = Fs * Ds.^-1;
    Pn = Fn * Dn.^-1;

    % calculate convective term
    q_conv = (h / Cp) * (2 * dx * dy / d);

    % compute coefficient arrays
    Aw = Dw.*peclet_function(abs(Pw), solver_type) + max(Fw, 0);
    Ae = De.*peclet_function(abs(Pe), solver_type) + max(-Fe, 0);
    As = Ds.*peclet_function(abs(Ps), solver_type) + max(Fs, 0);
    An = Dn.*peclet_function(abs(Pn), solver_type) + max(-Fn, 0);
```

```
    Ap = Aw + Ae + As + An + q_conv;

    % compute the value of quantity in steady state for each CV
    error = 1;
    while error > 10^-6
        phi_old = phi;

        % x-sweep
        A = zeros(Ny+2, 3);
        b = zeros(Ny+2, 1);

        % equations for CVs adjacent to left inlet (i = 2)
        % equation for bottom boundary node
        A(1, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        A(1, 3) = -(2*k/dy);
        b(1) = (k/dx) * phi(3, 1);
        b(1) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(2, 1) * (1-omega) / omega;

        % equation for interior cells
        A(2:end-1, 1) = -As(1, :);
        A(2:end-1, 2) = Ap(1, :) / omega;
        A(2:end-1, 3) = -An(1, :);
        b(2:end-1) = Aw(1, :).* phi(1, 2:end-1) + Ae(1, :).* phi(3, 2:end-1) +
Ap(1, :).* phi(2, 2:end-1) * ((1-omega)/omega);

        % equation for top boundary node
        A(end, 1) = -(2*k/dy);
        A(end, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        b(end) = (k/dx) * phi(3, end);
        b(end) = b(end) + ((k/dx) + (2*k/dy) + h) * phi(2, end) * (1-omega) /
omega;

        phi(2, :) = TDMA_solver(Ny+2, A, b);

        % equation for interior CVs (i = 3 to ITMAX-2)
        for i = 3:Nx
            % equation for bottom boundary node
            A(1, 2) = ((2*k/dx) + (2*k/dy) + h) / omega;
            A(1, 3) = -(2*k/dy);
            b(1) = (k/dx) * phi(i-1, 1) + (k/dx) * phi(i+1, 1);
            b(1) = b(1) + ((2*k/dx) + (2*k/dy) + h) * phi(i, 1) * (1-omega) /
omega;

            % equation for interior cells
            A(2:end-1, 1) = -As(i-1, :);
            A(2:end-1, 2) = Ap(i-1, :) / omega;
```

```
            A(2:end-1, 3) = -An(i-1, :);
            b(2:end-1) = Aw(i-1, :).* phi(i-1, 2:end-1) + Ae(i-1, :).* phi(i+1,
2:end-1) + Ap(i-1, :).* phi(i, 2:end-1) * ((1-omega)/omega);

            % equation for top boundary node
            A(end, 1) = -(2*k/dy);
            A(end, 2) = ((2*k/dx) + (2*k/dy) + h) / omega;
            b(end) = (k/dx) * phi(i-1, end) + (k/dx) * phi(i+1, end);
            b(end) = b(end) + ((2*k/dx) + (2*k/dy) + h) * phi(i, end) * (1-
omega) / omega;

            phi(i, :) = TDMA_solver(Ny+2, A, b);
        end

        % equations for CVs adjacent to left inlet (i = ITMAX - 1)
        % equation for bottom boundary node
        A(1, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        A(1, 3) = -(2*k/dy);
        b(1) = (k/dx) * phi(end-2, 1);
        b(1) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(end-1, 1) * (1-omega) /
omega;

        % equation for interior cells
        A(2:end-1, 1) = -As(1, :);
        A(2:end-1, 2) = Ap(1, :) / omega;
        A(2:end-1, 3) = -An(1, :);
        b(2:end-1) = Aw(1, :).* phi(end-2, 2:end-1) + Ae(1, :).* phi(end, 2:end-
1) + Ap(1, :).* phi(end-1, 2:end-1) * ((1-omega)/omega);

        % equation for top boundary node
        A(end, 1) = -(2*k/dy);
        A(end, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        b(end) = (k/dx) * phi(end-2, end);
        b(end) = b(end) + ((k/dx) + (2*k/dy) + h) * phi(end-1, end) * (1-omega)
/ omega;

        phi(end-1, :) = TDMA_solver(Ny+2, A, b);

        % y-sweep
        A = zeros(Nx, 3);
        b = zeros(Nx, 1);

        % updating bottom boundary layer
        % equation for left most boundary node
        A(1, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        A(1, 3) = -(k/dx);
```

```
        b(1) = (2*k/dy) * phi(2, 2);
        b(1) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(2, 1) * (1-omega) / omega;

        % equation for interior boundary nodes
        A(2:end-1, 1) = -(k/dx);
        A(2:end-1, 2) = ((2*k/dx) + (2*k/dy) + h) / omega;
        A(2:end-1, 3) = -(k/dx);
        b(2:end-1) = (2*k/dy) * phi(3:end-2, 2);
        b(2:end-1) = b(2:end-1) + ((2*k/dx) + (2*k/dy) + h) * phi(3:end-2, 1)
* (1-omega) / omega;

        % equation for right most boundary node
        A(end, 1) = -(k/dx);
        A(end, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        b(end) = (2*k/dy) * phi(end-1, 2);
        b(end) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(end-1, 1) * (1-omega) /
omega;

        phi(2:end-1, 1) = TDMA_solver(Nx, A, b);

        % sweeping through interior cells
        for j = 2:Ny+1
            A(2:end, 1) = -Aw(2:end, j-1);
            A(:, 2) = Ap(:, j-1) / omega;
            A(1:end-1, 3) = -Ae(1:end-1, j-1);
            b = As(:, j-1).* phi(2:end-1, j-1);
            b = b + An(:, j-1).* phi(2:end-1, j+1);
            b = b + Ap(:, j-1).* phi(2:end-1, j) * ((1-omega)/omega);
            b(1) = b(1) + Aw(1, j-1) * phi(1, j);
            b(end) = b(end) + Ae(end, j-1) * phi(end, j);

            phi(2:end-1, j) = TDMA_solver(Nx, A, b);
        end

        % updating top boundary layer
        % equation for left most boundary node
        A(1, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        A(1, 3) = -(k/dx);
        b(1) = (2*k/dy) * phi(2, end-1);
        b(1) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(2, end) * (1-omega) / omega;

        % equation for interior boundary nodes
        A(2:end-1, 1) = -(k/dx);
        A(2:end-1, 2) = ((2*k/dx) + (2*k/dy) + h) / omega;
        A(2:end-1, 3) = -(k/dx);
        b(2:end-1) = (2*k/dy) * phi(3:end-2, end-1);
```

```
        b(2:end-1) = b(2:end-1) + ((2*k/dx) + (2*k/dy) + h) * phi(3:end-2, end)
* (1-omega) / omega;

        % equation for right most boundary node
        A(end, 1) = -(k/dx);
        A(end, 2) = ((k/dx) + (2*k/dy) + h) / omega;
        b(end) = (2*k/dy) * phi(end-1, end-1);
        b(end) = b(1) + ((k/dx) + (2*k/dy) + h) * phi(end-1, end) * (1-omega)
/ omega;

        phi(2:end-1, 1) = TDMA_solver(Nx, A, b);

        % calculate error
        error = sum(sum(abs(phi - phi_old)));
    end
end
```

3. peclet_function.m

```
% peclet function (A(|P|))

% parameters required:
% P: Peclet number (array)
% solver_type: type of solver ('central_diff', 'exponential', 'power_law')

% return output:
% P: A(|P|) depending upon the solver function

function [P] = peclet_function(P, solver_type)

    % central difference scheme
    if solver_type == "central_diff"
        P = 1 - 0.5 * abs(P);

    % exponential scheme
    elseif solver_type == "exponential"
        if abs(P) < 700
            P = exp(abs(P))./ (exp(abs(P)) - 1);
        else
            P = 1;
        end

    % power law
    elseif solver_type == "power_law"
        mult = (1 - 0.1 * abs(P));
        P = max(zeros(size(P, 1), 1), mult.*mult.*mult.*mult.*mult);
```

```
    else
        error("Please enter correct solver type")
    end
end
```

4. TDMA_solver.m

```
% Matrix solver (TDMA)

% parameters required:
% N: Number of unknowns
% A: Coefficient matrix
% b: RHS/constant array prior to calling TDMA

% return output:
% b: solution array after calling TDMA

function [b] = TDMA_solver(N, A, b)

    % forward substitution
    A(1,3) = -A(1,3)/A(1,2);
    b(1) = b(1)/A(1,2);
    for i = 2:N
        A(i,3) = -A(i,3)/(A(i,2) + A(i,1)*A(i-1,3));
        b(i) = (b(i) - A(i,1)*b(i-1))/(A(i,2) + A(i,1)*A(i-1,3));
    end

    % backward elimination
    for i = N-1:-1:1
        b(i) = A(i,3)*b(i+1) + b(i);
    end
end
```