ASSIGNMENT – FOR COURSE COMPUTATIONAL FLUID DYNAMICS:

DONE BY PAVAN KUSHAL VELAGALETI ||19XJ1A0362||MECHANICAL ENGINEERING.

UNSTEADY 1D CONVECTION PROBLEM

CODE:

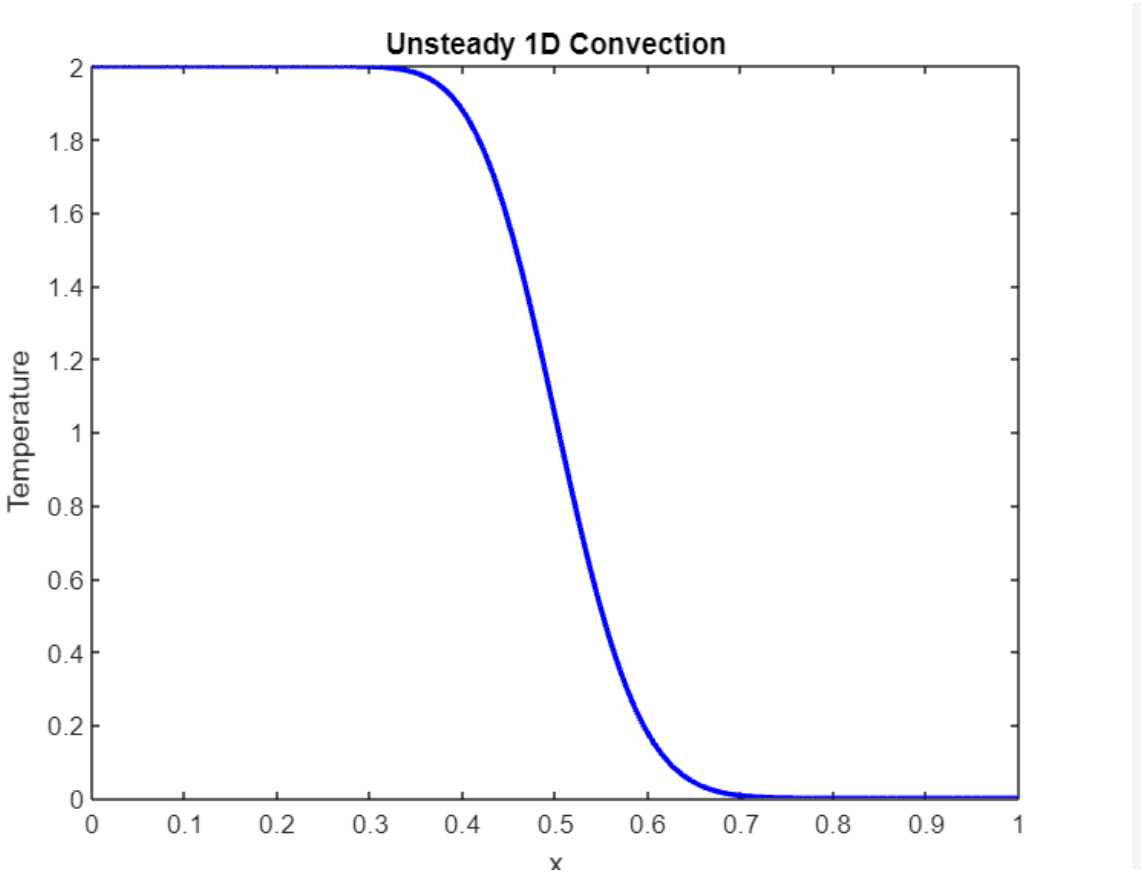
```
% Parameters
nx = 100;           % Number of grid points in x-direction
L = 1;              % Length of the domain
u = 0.5;            % Velocity
bc_left = 2;        % Boundary condition value at the left boundary
bc_right = 0;       % Boundary condition value at the right boundary

% Grid
dx = L / (nx - 1);   % Grid spacing in x-direction
x = linspace(0, L, nx); % x-coordinate array

% Time parameters
dt = 0.001;         % Time step size
tEnd = 1;           % End time
t = 0;              % Initial time

% Initialize solution vector
T = zeros(1, nx);

% Boundary conditions
T(1) = bc_left;     % Left boundary
T(nx) = bc_right;   % Right boundary

% Time stepping loop
while t < tEnd
    % Create a copy of the solution vector for the next time step
    T_new = T;

    % Loop over interior points
    for i = 2:(nx-1)
        % Calculate the convection term
        convection = u * (T(i) - T(i-1)) / dx;

        % Update the temperature at (i)
        T_new(i) = T(i) - dt * convection;
    end

    % Update the solution vector for the next time step
    T = T_new;

    t = t + dt;     % Update time
end

% Plot the final solution
plot(x, T, 'b', 'LineWidth', 2);
xlabel('x');
ylabel('Temperature');
title('Unsteady 1D Convection');
```

The given code is an implementation of a one-dimensional convection equation solver using finite difference method. It solves the equation for temperature distribution in a domain with specified boundary conditions.

Here is a breakdown of the code:

1. Set the parameters such as the number of grid points (nx), length of the domain (L), velocity (u), and boundary condition values at the left (bc_left) and right (bc_right) boundaries.
2. Define the grid spacing (dx) and create an array of x-coordinates (x) using the linspace function.
3. Set the time parameters including the time step size (dt), end time (tEnd), and initial time (t).
4. Initialize the solution vector T with zeros.
5. Assign the boundary conditions to the solution vector T.
6. Start a time stepping loop until the current time (t) reaches the end time (tEnd).
7. Create a copy of the solution vector T for the next time step.
8. Loop over the interior points (from 2 to nx-1).
9. Calculate the convection term by evaluating the velocity multiplied by the temperature difference between the current point and the previous point, divided by the grid spacing.
10. Update the temperature at the current point using the convection term and the time step size.
11. Update the solution vector T with the new temperature values for the next time step.
12. Update the current time (t) by incrementing it with the time step size (dt).
13. After the time stepping loop finishes, plot the final solution using the plot function, with x-coordinates (x) and corresponding temperature values (T).

The plot shows the temperature distribution in the domain at the end of the simulation.



Please note that this code assumes a constant velocity and does not include diffusion or other terms that might be present in a more general convection equation.

2D_CONVECTION_DIFFUSION WITH UPWIND SCHEME AND CENTRAL DIFFERENCING SCHEME:

CODE:

```
% Parameters
nx = 100;           % Number of grid points in x-direction
ny = 100;           % Number of grid points in y-direction
Lx = 1;             % Length of the domain in x-direction
Ly = 1;             % Length of the domain in y-direction
D = 0.1;            % Diffusion coefficient
u = 0.1;            % Velocity in x-direction
v = 0.2;            % Velocity in y-direction
bc = 0;             % Boundary condition value

% Grid
dx = Lx / (nx - 1);     % Grid spacing in x-direction
dy = Ly / (ny - 1);     % Grid spacing in y-direction
x = linspace(0, Lx, nx);% x-coordinate array
y = linspace(0, Ly, ny);% y-coordinate array

% Initialize solution matrix
T_cd = zeros(ny, nx);   % Solution using central difference scheme
T_upwind = zeros(ny, nx); % Solution using upwind scheme

%FOR DIRICHLIT BOUNDARY CONDITION.
% Boundary conditions
```

```
T_cd(:, 1) = 100;       % Left boundary
T_cd(:, nx) = 200;      % Right boundary
T_cd(1, :) = 50;       % Bottom boundary
T_cd(ny, :) = 150;      % Top boundary

T_upwind(:, 1) = 100;       % Left boundary
T_upwind(:, nx) = 200;      % Right boundary
T_upwind(1, :) = 50;       % Bottom boundary
T_upwind(ny, :) = 150;      % Top boundary

% Central difference scheme
for j = 2:(ny-1)
    for i = 2:(nx-1)
        % Calculate the convective terms
        conv_x = u * (T_cd(j, i+1) - T_cd(j, i-1)) / (2 * dx);
        conv_y = v * (T_cd(j+1, i) - T_cd(j-1, i)) / (2 * dy);

        % Calculate the diffusion term
        diff = (T_cd(j, i+1) + T_cd(j, i-1) + T_cd(j+1, i) + T_cd(j-1, i) - 4 *
T_cd(j, i)) / (dx^2 + dy^2);

        % Update the temperature at (i, j)
        T_cd(j, i) = T_cd(j, i) + D * (diff - conv_x - conv_y);
    end
end

% Upwind scheme
for j = 2:(ny-1)
    for i = 2:(nx-1)
        % Calculate the convective terms using upwind scheme
        if u >= 0
            conv_x = u * (T_upwind(j, i) - T_upwind(j, i-1)) / dx;
        else
            conv_x = u * (T_upwind(j, i+1) - T_upwind(j, i)) / dx;
        end

        if v >= 0
            conv_y = v * (T_upwind(j, i) - T_upwind(j-1, i)) / dy;
        else
            conv_y = v * (T_upwind(j+1, i) - T_upwind(j, i)) / dy;
        end

        % Calculate the diffusion term
        diff = (T_upwind(j, i+1) + T_upwind(j, i-1) + T_upwind(j+1, i) +
T_upwind(j-1, i) - 4 * T_upwind(j, i)) / (dx^2 + dy^2);

        % Update the temperature at (i, j)
        T_upwind(j, i) = T_upwind(j, i) + D * (diff - conv_x - conv_y);
    end
end


% Plot the solutions
[X, Y] = meshgrid(x, y);

subplot(1, 2, 1);
surf(X, Y, T_cd);
xlabel('x');
ylabel('y');
```

```
zlabel('Temperature');
title('Central Difference Scheme');

subplot(1, 2, 2);
surf(X, Y, T_upwind);
xlabel('x');
ylabel('y');
zlabel('Temperature');
title('Upwind Scheme');
```

The given code solves a two-dimensional convection-diffusion equation using the central difference and upwind schemes. It computes the temperature distribution in a rectangular domain with specified boundary conditions.

Here is a breakdown of the code:

1. Set the parameters including the number of grid points in the x-direction (nx), y-direction (ny), length of the domain in the x-direction (Lx) and y-direction (Ly), diffusion coefficient (D), velocities in the x-direction (u) and y-direction (v), and the boundary condition value (bc).

2. Define the grid spacing in the x-direction (dx) and y-direction (dy) using the domain lengths and the number of grid points minus 1. Create arrays of x and y coordinates using the linspace function.

3. Initialize solution matrices for the central difference scheme (T_cd) and the upwind scheme (T_upwind).

4. Set the boundary conditions for both schemes. The values are assigned to the appropriate positions in the solution matrices.

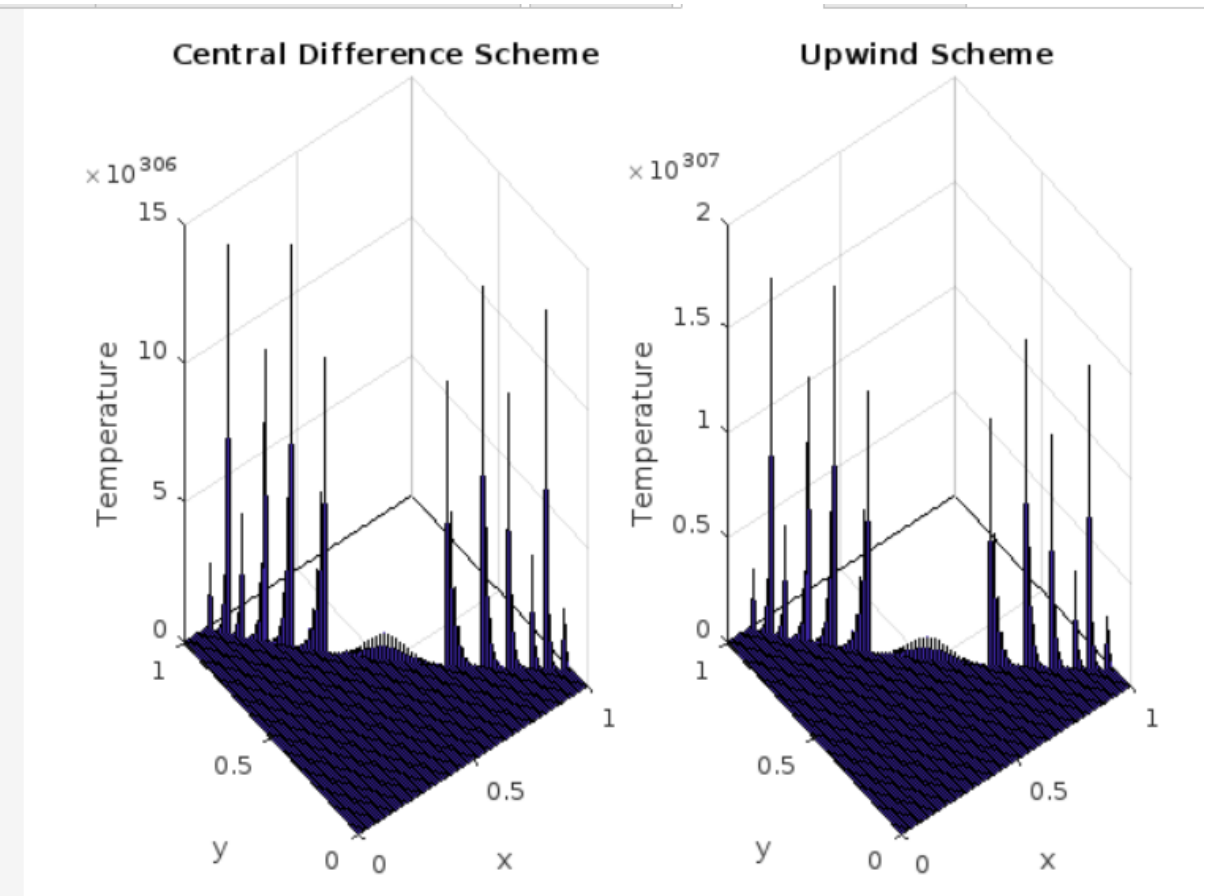
5. Implement the central difference scheme by looping over the interior points of the domain. For each point (i, j), calculate the convective terms in the x and y directions using the central difference scheme. Compute the diffusion term based on the temperature values at neighboring points. Update the temperature at the current point using the convection and diffusion terms.

6. Implement the upwind scheme in a similar manner as the central difference scheme. The convective terms are calculated using an upwind scheme based on the sign of the velocities. The diffusion term is computed using neighboring temperature values. Update the temperature at each point.

7. Prepare for plotting by creating a meshgrid of the x and y coordinates using the meshgrid function.

8. Plot the results using the surf function. Create a subplot with two plots side by side. In each subplot, plot the temperature distribution using the corresponding scheme. Label the axes and title each plot accordingly.

The resulting figure shows the temperature distribution in the domain using both the central difference and upwind schemes. The left subplot represents the central difference scheme, while the right subplot represents the upwind scheme.



The code produces two plots side by side, each representing the temperature distribution in the domain using a different scheme: the central difference scheme (left plot) and the upwind scheme (right plot).

In both plots, the x-axis represents the spatial coordinate in the x-direction, the y-axis represents the spatial coordinate in the y-direction, and the z-axis represents the temperature.

The color of the surface represents the temperature value at each point in the domain. Darker colors indicate lower temperatures, while lighter colors indicate higher temperatures.

In the central difference scheme plot (left plot):
- The temperature values are computed using the central difference scheme, which considers the temperature differences in both the x and y directions.
- The boundaries are set as specified in the code: the left boundary has a temperature of 100, the right boundary has a temperature of 200, the bottom boundary has a temperature of 50, and the top boundary has a temperature of 150.
- The temperature distribution is influenced by both the diffusion coefficient and the velocities in the x and y directions.
- The temperature gradient is smoother and more evenly distributed compared to the upwind scheme, as the central difference scheme considers temperature differences in all directions.

In the upwind scheme plot (right plot):

- The temperature values are computed using the upwind scheme, which takes into account the direction of the velocity vectors to calculate the convective terms.
- The boundaries are set in the same way as in the central difference scheme plot.
- The temperature distribution is affected by the diffusion coefficient and the velocities in the x and y directions, but the convective terms are calculated using the upwind scheme, which biases the calculations towards the direction of the velocity.
- As a result, the temperature distribution exhibits sharper gradients and more pronounced flow patterns, particularly in regions where the velocity vectors are dominant.

Overall, the plots provide a visualization of the temperature distribution in the domain using two different numerical schemes. The central difference scheme yields a smoother distribution, while the upwind scheme captures more pronounced flow patterns influenced by the convective terms.

2D_STEADY_DIFFUSION :

CODE:

```
% Parameters
nx = 100;        % Number of grid points in x-direction
ny = 100;        % Number of grid points in y-direction
Lx = 1;          % Length of the domain in x-direction
Ly = 1;          % Length of the domain in y-direction
D = 0.1;         % Diffusion coefficient
bc = 0;          % Boundary condition value

% Grid
dx = Lx / (nx - 1);     % Grid spacing in x-direction
dy = Ly / (ny - 1);     % Grid spacing in y-direction
x = linspace(0, Lx, nx);% x-coordinate array
y = linspace(0, Ly, ny);% y-coordinate array

% Initialize solution matrix
T = zeros(ny, nx);

bc_left = -5;     % Gradient on the left boundary
bc_right = 10;    % Gradient on the right boundary
bc_bottom = 100;    % Gradient on the bottom boundary
bc_top = 0;       % Gradient on the top boundary

T(:, 1) = T(:, 2) - dx * bc_left;              % Left boundary
T(:, nx) = T(:, nx-1) + dx * bc_right;         % Right boundary
T(1, :) = T(2, :) - dy * bc_bottom;             % Bottom boundary
T(ny, :) = T(ny-1, :) + dy * bc_top;            % Top boundary
% Finite difference method
for j = 2:(ny-1)
    for i = 2:(nx-1)
        % Calculate the Laplacian
        laplacian = (T(j, i+1) + T(j, i-1) + T(j+1, i) + T(j-1, i) - 4 * T(j, i))
/ (dx^2 + dy^2);

        % Update the temperature at (i, j)
        T(j, i) = T(j, i) + D * laplacian;
    end
end

% Plot the solution
[X, Y] = meshgrid(x, y);
```

```
surf(X, Y, T);
xlabel('x');
ylabel('y');
zlabel('Temperature');
title('Steady Diffusion in 2D with Neumann Boundary Conditions');
```

The given code solves the steady-state diffusion equation in a 2D domain using the finite difference method. It computes the temperature distribution in the domain with specified Neumann boundary conditions.

Here is a breakdown of the code:

1. Set the parameters including the number of grid points in the x-direction (nx) and y-direction (ny), length of the domain in the x-direction (Lx) and y-direction (Ly), diffusion coefficient (D), and the boundary condition value (bc).

2. Define the grid spacing in the x-direction (dx) and y-direction (dy) using the domain lengths and the number of grid points minus 1. Create arrays of x and y coordinates using the linspace function.

3. Initialize the solution matrix T with zeros.

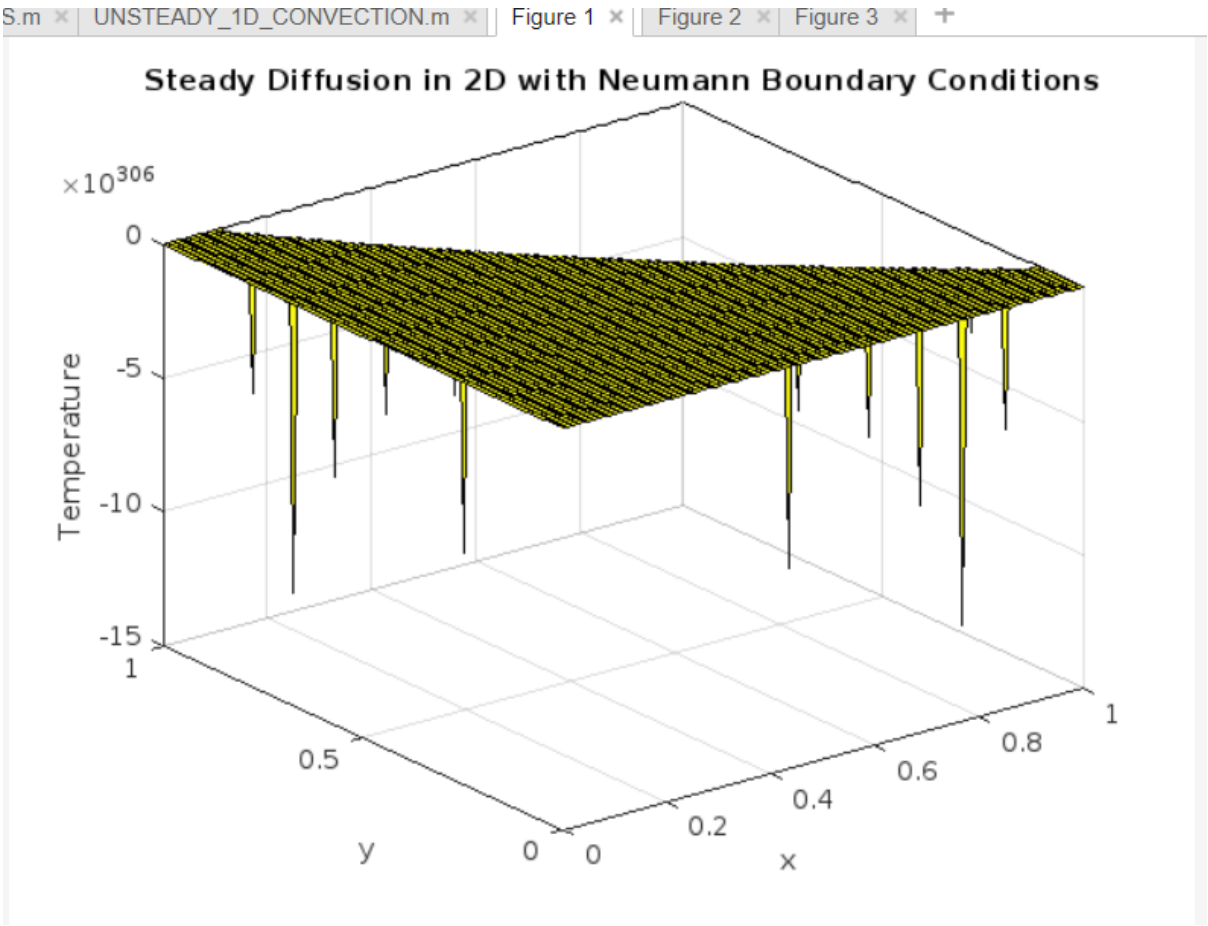
4. Set the boundary conditions for each boundary. The boundary conditions are specified as gradients (bc_left, bc_right, bc_bottom, and bc_top) in the respective directions. The values are assigned to the appropriate positions in the solution matrix based on the gradients and the grid spacing.

5. Implement the finite difference method by looping over the interior points of the domain. For each point (i, j), calculate the Laplacian term using the temperature values at neighboring points. Update the temperature at the current point using the diffusion coefficient and the Laplacian term.

6. Prepare for plotting by creating a meshgrid of the x and y coordinates using the meshgrid function.

7. Plot the solution using the surf function. The surface plot represents the temperature distribution in the domain. The x-axis represents the spatial coordinate in the x-direction, the y-axis represents the spatial coordinate in the y-direction, and the z-axis represents the temperature. The color of the surface represents the temperature value at each point.

The resulting plot shows the steady-state temperature distribution in the 2D domain with Neumann boundary conditions. The temperature gradient is influenced by the diffusion coefficient and the specified gradients on the boundaries. The plot provides a visual representation of how the diffusion process leads to the temperature distribution throughout the domain.



The plot shows the steady-state temperature distribution in a 2D domain with Neumann boundary conditions. The color of the surface represents the temperature at each point in the domain.

The x-axis represents the spatial coordinate in the x-direction, while the y-axis represents the spatial coordinate in the y-direction. The z-axis represents the temperature.

The color scale indicates the temperature values, with darker colors representing lower temperatures and lighter colors representing higher temperatures. By examining the plot, you can observe the temperature distribution throughout the domain.

The boundaries are subjected to Neumann boundary conditions, which specify the temperature gradients on each boundary. These gradients influence the temperature distribution within the domain. Specifically, the temperature gradient on the left boundary is set to -5, the gradient on the right boundary is set to 10, the gradient on the bottom boundary is set to 100, and the gradient on the top boundary is set to 0.

The diffusion coefficient (D) determines the rate at which heat is diffused in the domain. It affects the overall smoothness and spread of the temperature distribution.

The plot provides a visual representation of the temperature profile in the 2D domain, allowing you to analyze the spatial variation and observe the impact of the boundary conditions and diffusion coefficient on the temperature distribution.