Mohammadreza Aliyari, SBU ID:115908077

Ph.D. Student in the Civil Engineering Department

AMS-595 Final Project Report

Part II- Current Numerical Simulation

The numerical simulation of currents serves as a powerful tool for comprehending and predicting the behavior of fluid flow in various environments, such as oceans, rivers, or even within engineered systems like pipelines. By employing computational models and algorithms, this simulation aims to replicate real-world current dynamics, accounting for factors like terrain, obstructions, and varying conditions.

This simulation is crucial for several reasons:

- Understanding Environmental Impact: It helps in understanding how currents transport materials like nutrients, pollutants, or sediments, affecting ecosystems and environmental health.
- Engineering Design and Safety: Predicting currents is vital for designing infrastructure like offshore platforms, coastal constructions, or pipelines, ensuring they can withstand and function effectively in different flow conditions.
- Resource Management: In marine industries, such as fisheries or shipping, knowing current patterns aids in optimizing routes, predicting potential hazards, and efficiently utilizing resources.
- Climate Studies: Current simulations contribute to climate research by modeling ocean circulation, which influences weather patterns, heat distribution, and overall climate dynamics.

Overall, this simulation provides insights into fluid behavior, enabling better decision-making in various fields while enhancing our understanding of natural processes and their implications.

2-1: Governing equations

The assumption asserting the flow's irrotational nature is expressed by the equation

$$\vec{\zeta} = \vec{\nabla} \times \vec{V} \cong 0 \qquad (1)$$

$$\zeta_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0 \qquad (2)$$

And flow velocity components regarding the stream function are written as:

$$u = \frac{\partial \psi}{\partial y} \qquad (3)$$

$$v = -\frac{\partial \psi}{\partial x} \qquad (4)$$

Now by plugging equation 3 and 4 in 2 we have:

$$\frac{\partial}{\partial x}(-\frac{\partial \psi}{\partial x})-\frac{\partial}{\partial y}(-\frac{\partial \psi}{\partial y})=-\frac{\partial^2 \psi}{\partial x^2}-\frac{\partial^2 \psi}{\partial y^2}=0 \tag{5}$$

The 2D Laplace equation simplifies to one unknown variable, Ψ, instead of the two variables (u and v). The approach involves solving the equation for Ψ and then deriving u and v from it. Concerning the discretized computational domain, we compute the first and second derivatives of the stream function for each grid by leveraging information from neighboring grids. Employing the Finite Difference Method, the estimation of the first and second derivatives in the x-direction is depicted by equations (6) and (7) respectively.

$$\frac{\partial \psi}{\partial x}\approx\frac{1}{\Delta x}(\psi_{i+1,j}-\psi_{i,j}) \tag{6}$$

$$\frac{\partial^2 \psi}{\partial x^2}\approx\frac{1}{\Delta x^2}(\psi_{i+1,j}-2\psi_{i,j}+\psi_{i-1,j}) \tag{7}$$

And similarly in the y direction:

$$\frac{\partial \psi}{\partial y}\approx\frac{1}{\Delta y}(\psi_{i,j+1}-\psi_{i,j}) \tag{8}$$

$$\frac{\partial^2 \psi}{\partial y^2}\approx\frac{1}{\Delta y^2}(\psi_{i,j+1}-2\psi_{i,j}+\psi_{i,j-1}) \tag{9}$$

By plugging equation (9) and (7) into equation (5) we have:

$$2(1+\beta)\psi_{i,j}=\psi_{i-1,j}+\psi_{i+1,j}+\beta(\psi_{i,j-1}+\psi_{i,j+1}) \tag{10}$$

Where $\beta=(\Delta x/\Delta y)^2$ and it depends on computational mesh. Equation (10) will be solved for entire domain and then using equations (3) and (4) the corresponding velocity field will be calculated.

2-2 Methodology

2-2-1: Boundary Condition

Given the 2D computational domain and the assumption of irrotational flow, solving the Laplace equation for the stream function allows us to derive the velocity field in this section. Figure 1 displays the boundary conditions pertinent to the field of interest

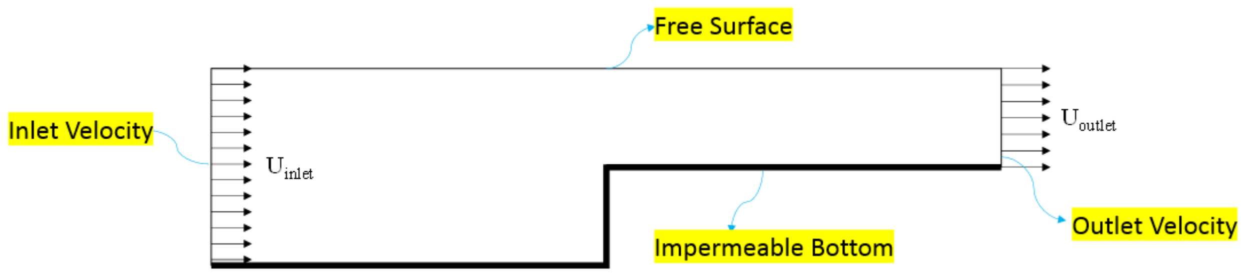

Fig. 1: Boundary Condition for Current Numerical Simulation

The boundary conditions applied to the streamline Laplace equation are represented by equations 11 to 14 for the inlet velocity, outlet velocity, impermeable bottom, and free surface boundaries, respectively.

$$\frac{Q}{(H_1+H_2)} = U_{inlet} = \frac{\partial \psi}{\partial y} \tag{11}$$

$$\frac{Q}{H_2} = U_{outlet} = \frac{\partial \psi}{\partial y} \tag{12}$$

$$\frac{\partial \psi}{\partial n} = 0, n \text{ is normal direction} \tag{13}$$

$$-\frac{\partial \psi}{\partial x} = v = 0 \tag{14}$$

2-2-2 : Solution Algorithm

The algorithmic solution, developed in both MATLAB and Python, encompasses several crucial steps:

- Defining computational domain specifications, such as grid count and spacing.
- Assigning initial values to variables.
- Implementing computational grids within the domain.
- Applying boundary conditions to the grid.
- Solving the discretized Laplace equation using the Gauss-Seidel method.
- Calculating horizontal and vertical velocities at specified grid points within the domain.
- Finally, visualizing the results through plotting.

2-2-2-1: MATLAB Code:

```
clearvars
clc
close all
% Solving The Laplace Equation For Obtaining Streamline Field in a 2D Forward Step
% Specifying parameters
X1 = 95.0 ;
X2 = 57.0 ;
H1 = 45 ;
H2 = 27 ;
Q = 45.0 ;
U_INLET = Q/(H1*1.0) ;
U_OUTLET = Q/(H2*1.0) ;

NX = 153;
NY = 46 ;


DEL_Y = H1/(NY-1) ;
DEL_X = (X1+X2)/(NX-1) ;
BETA = (DEL_X/DEL_Y).^2 ;
clearvars
clc
close all
% Solving The Laplace Equation For  Obtaining Streamline Field in a 2D Forward Step
% Specifying parameters
X1 = 95.0 ;
X2 = 57.0 ;
H1 = 45 ;
H2 = 27 ;
Q = 45.0 ;
U_INLET = Q/(H1*1.0) ;
U_OUTLET = Q/(H2*1.0) ;

NX = 153;
NY = 46 ;


DEL_Y = H1/(NY-1) ;
DEL_X = (X1+X2)/(NX-1) ;
BETA = (DEL_X/DEL_Y).^2 ;

% Applying Initial Value to Variables
ERROR = 1.0 ;

INITIAL_S = zeros(NY,NX) ;
FINAL_S = zeros(NY,NX) ;
EKHTELAF_S = zeros(NY,NX) ;
U = zeros(NY,NX) ;
V = zeros(NY,NX) ;

% Grid
```

```
Y = 0 : DEL_Y : H1 ;
Y = Y' ;
Y = repmat (Y ,1 , NX);

X = 0 : DEL_X : (X1+X2) ;
X = repmat(X,NY,1) ;

% Inlet Boundary Condition
for I = 2 : NY
    INITIAL_S(I , 1) = INITIAL_S(I-1,1)+(U_INLET*DEL_Y) ;
    FINAL_S(I,1) = FINAL_S(I-1,1)+(U_INLET*DEL_Y) ;
end
% Top Boundary Condition
INITIAL_S(NY , :) = INITIAL_S(NY , 1) ;
FINAL_S(NY , :) = FINAL_S(NY , 1) ;


% Outlet Boundary Condition
for I = NY : -1 : ((NY -(H2/DEL_Y))+1)

    INITIAL_S(I-1 , NX) = INITIAL_S(I , NX)-(U_OUTLET*DEL_Y) ;
    FINAL_S(I-1 , NX) = FINAL_S(I , NX)-(U_OUTLET*DEL_Y) ;
end

% Iterative Solving the Streamline Field
while ERROR > 1e-10

    INITIAL_S = FINAL_S ;

    for I = 2 : NY-1
        for J = 2 : NX-1

            if X(I,J) < X1
                FINAL_S(I,J) = (INITIAL_S(I,J+1)+INITIAL_S(I,J-1)+...
                    (BETA*(INITIAL_S(I-1,J)+INITIAL_S(I+1,J)))))/(2*(1+BETA)) ;

            else if (X(I,J) >= X1 && Y(I,J) > (H1-H2))

                    FINAL_S(I,J) = (INITIAL_S(I,J+1)+INITIAL_S(I,J-1)+...
                        (BETA*(INITIAL_S(I-1,J)+INITIAL_S(I+1,J)))))/(2*(1+BETA)) ;
                end
            end
        end
end


EKHTELAF_S = FINAL_S - INITIAL_S ;

ERROR = max(max(abs(EKHTELAF_S))) ;

end
% Calculating the Velocity Field
 U(2:NY,1) = U_INLET ;

for I = 2 : NY
```

```
   for J = 2 : NX

    U(I,J) = (FINAL_S(I,J)-FINAL_S(I-1,J))/DEL_Y ;
    V(I,J) = -(FINAL_S(I,J)-FINAL_S(I,J-1))/DEL_X ;

   end
end
% Exporting Output for Tecplot
for i = 1 : (NX)*(NY)

            a(i,1)=X(i);
            a(i,2)=Y(i);
            a(i,3)=U(i);
            a(i,4)=V(i);

end
% Find indices where U is zero
zero_indices = U == 0;

% Set the corresponding values in X, Y, and U to NaN
X(zero_indices) = NaN;
Y(zero_indices) = NaN;
U(zero_indices) = NaN;

% Assuming X, Y, and U are already defined as 46x153 matrices

figure(1)
% Create a 3D surface plot
surf(X, Y, U, 'EdgeColor', 'none');

% Set colormap to jet
colormap(turbo(256));

% Add labels and title
xlabel('X (m)');
ylabel('Y (m)');
zlabel('Velocity (U)');
title('Vertical Velocity');

xticks(0:4:152)

% Add colorbar for reference
clim([0 3])
colorbar;
axis equal tight;

% Adjust the view for better visualization (optional)
view(2); % 2D view from the top

grid off;

figure(2)
[startX,startY] = meshgrid(0,0:1:45);
h = streamline(0:1:152, 0:1:45, U, V, startX, startY);
set(h, 'Color', 'black', 'LineWidth', 1.5); % Customize streamline appearance
```

```
axis equal tight;
xticks(0:4:152)
title('Streamlines')
xlabel('X (m)')
ylabel('Y (m)')
```

2-2-2-2: Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Solving the Laplace Equation for Obtaining Streamline Field in a 2D
Forward Step
# Specifying parameters
x1 = 95.0
x2 = 57.0
h1 = 45
h2 = 27
time = 200


q = 45.0
u_inlet = q / (h1 * 1.0)
u_outlet = q / (h2 * 1.0)

nx = 153
ny = 46

del_y = h1 / (ny - 1)
del_x = (x1 + x2) / (nx - 1)
beta = (del_x / del_y) ** 2

# Grid
y = np.linspace(0, h1, ny).reshape(-1, 1)
y = np.tile(y, (1, nx))

x = np.linspace(0, x1 + x2, nx).reshape(1, -1)
x = np.tile(x, (ny, 1))

# Applying initial value to variables
error = 1.0

initial_s = np.zeros((ny, nx))
final_s = np.zeros((ny, nx))
ekhtelaf_s = np.zeros((ny, nx))
u = np.zeros((ny, nx))
v = np.zeros((ny, nx))
```

```python
# Inlet Boundary Condition
for i in range(1, ny):
    initial_s[i, 0] = initial_s[i - 1, 0] + (u_inlet * del_y)
    final_s[i, 0] = final_s[i - 1, 0] + (u_inlet * del_y)

initial_s[-1, :] = initial_s[-1, 0]
final_s[-1, :] = final_s[-1, 0]

# Outlet Boundary Condition
for i in range(ny - 1, int(ny - (h2 / del_y)) - 1, -1):
    initial_s[i - 1, -1] = initial_s[i, -1] - (u_outlet * del_y)
    final_s[i - 1, -1] = final_s[i, -1] - (u_outlet * del_y)

# Iterative solving the streamline field
while error > 1e-10:
    initial_s = final_s.copy()

    for i in range(1, ny - 1):
        for j in range(1, nx - 1):
            if x[i, j] < x1:
                final_s[i, j] = (
                    initial_s[i, j + 1]
                    + initial_s[i, j - 1]
                    + beta * (initial_s[i - 1, j] + initial_s[i + 1, j])
                ) / (2 * (1 + beta))
            elif x[i, j] >= x1 and y[i, j] > (h1 - h2):
                final_s[i, j] = (
                    initial_s[i, j + 1]
                    + initial_s[i, j - 1]
                    + beta * (initial_s[i - 1, j] + initial_s[i + 1, j])
                ) / (2 * (1 + beta))

    ekhtelaf_s = final_s - initial_s
    error = np.max(np.abs(ekhtelaf_s))

# Calculating the velocity field
u = np.zeros((ny, nx))
v = np.zeros((ny, nx))
u[1:, 0] = u_inlet

for i in range(1, ny):
    for j in range(1, nx):
        u[i, j] = (final_s[i, j] - final_s[i - 1, j]) / del_y
        v[i, j] = -(final_s[i, j] - final_s[i, j - 1]) / del_x

# Exporting output for Tecplot
a = np.zeros(((nx) * (ny), 4))

k = 0
for i in range(ny):
    for j in range(nx):
        a[k, 0] = x[i, j]
        a[k, 1] = y[i, j]
        a[k, 2] = u[i, j]
        a[k, 3] = v[i, j]
        k += 1
```

```
# Handle non-finite values in x, y, and u
x = np.ma.masked_invalid(x).filled(np.nan)
y = np.ma.masked_invalid(y).filled(np.nan)
u = np.ma.masked_invalid(u).filled(np.nan)

# Create a pseudocolor plot
pcm = plt.pcolormesh(x, y, u, cmap='jet')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Velocity Field')

# Add colorbar for reference
plt.colorbar(pcm)

# Adjust the aspect ratio of the axes for a better representation
plt.axis('equal')
plt.axis('tight')

# Display the grid (optional)
plt.grid(False)
```

2-3: Results and Discussions

As mentioned earlier, the lines with equal Ψ value indicate the streamline. Figure 2 shows the streamlines obtained in this section.

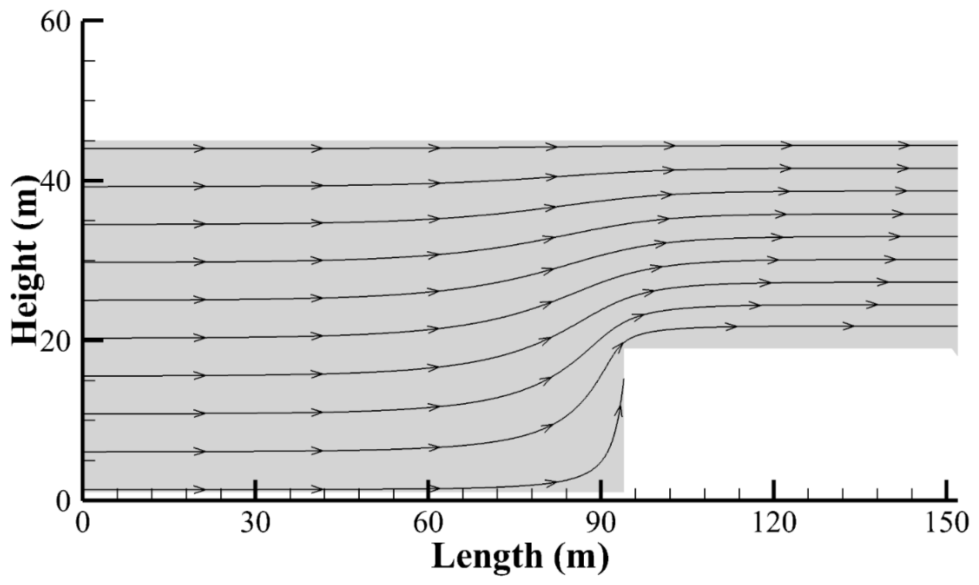

Fig. 2: Streamline conditions

Moreover, figure 3 shows the velocity vectors in the computational domain. It should be noted that the velocity vectors are tangent to the streamlines at any point.

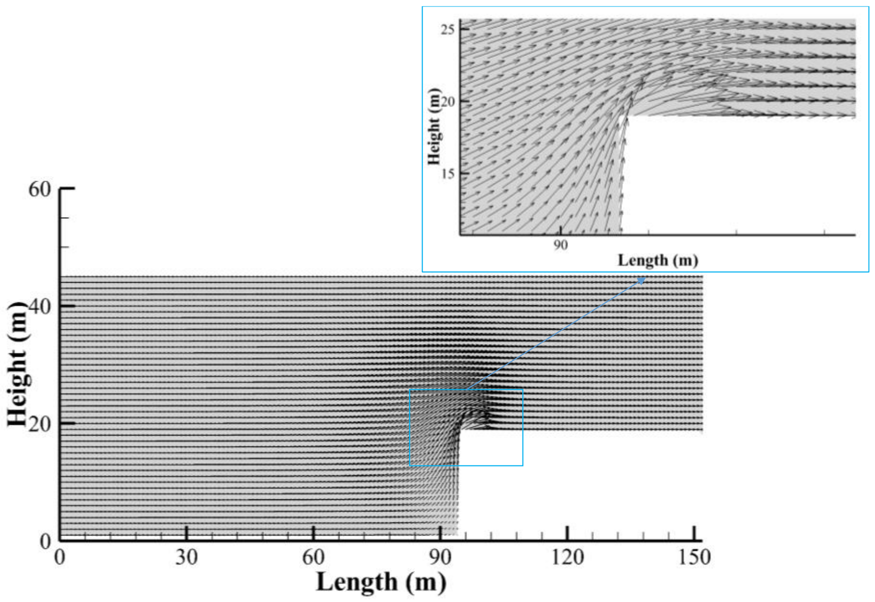

Fig.3: Velocity vectors in the domain

The fluid flow exhibits completely horizontal velocity vectors, as illustrated in Figure 3, until it reaches the vertical wall (elevated bottom). Upon approaching this wall, the impermeability condition induces an upward flow, resulting in the emergence of a vertical component in the velocity.

Figure 4 and figure 5 show the velocity contours in horizontal and vertical directions, respectively.

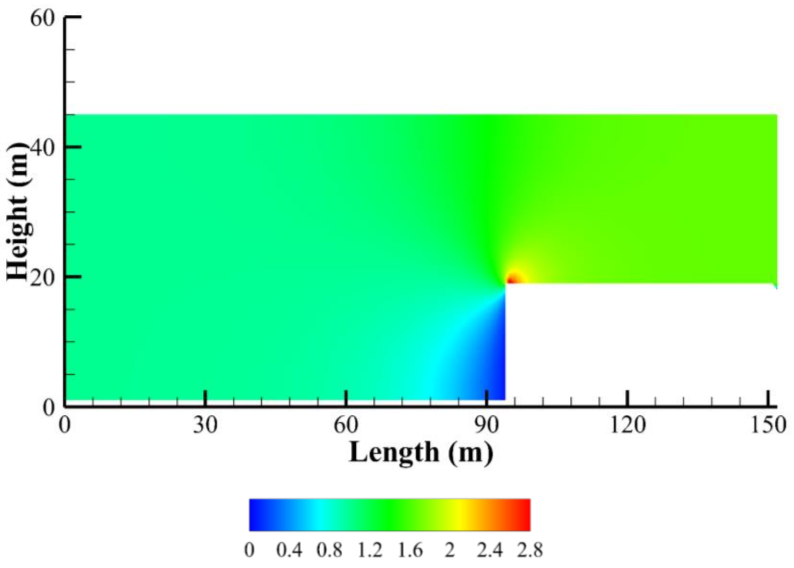

Fig. 4: Distribution of horizontal flow velocity in the domain

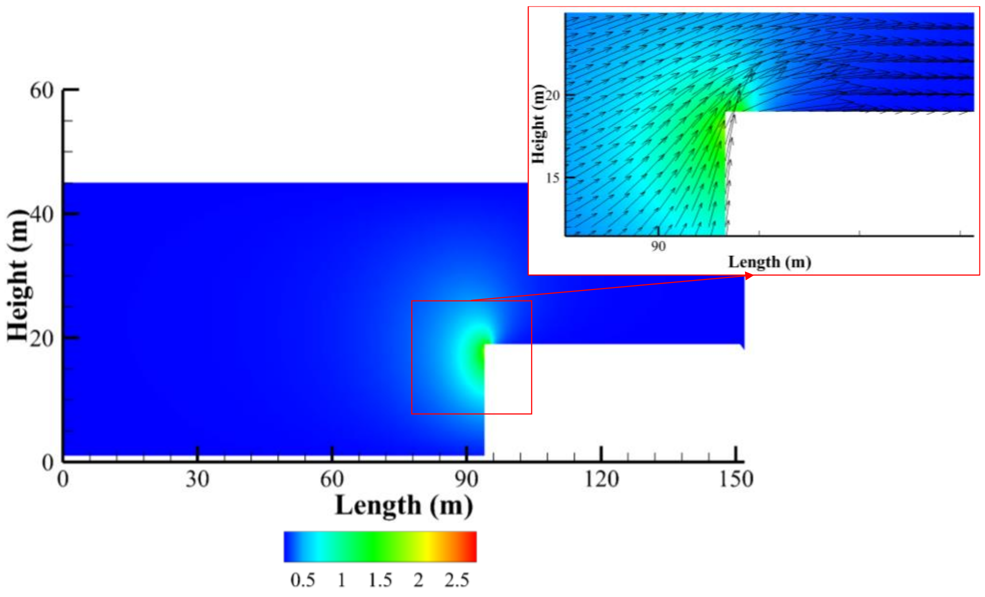

Fig. 5: Distribution of vertical flow velocity in the domain

In Figure 5, when the flow encounters the vertical wall (elevated bottom), the impermeable condition forces it upward. The emergence of vertical velocity components near the vertical wall can be attributed to this phenomenon. In the examined flow, an important observation is the rise in horizontal velocity following the vertical wall. According to the continuity condition, the flow discharge rate remains constant along streamlines. As the streamlines converge, the flow velocity increases to maintain this condition. Figure 6 displays the contour of horizontal velocity and streamlines both before and after the vertical step.

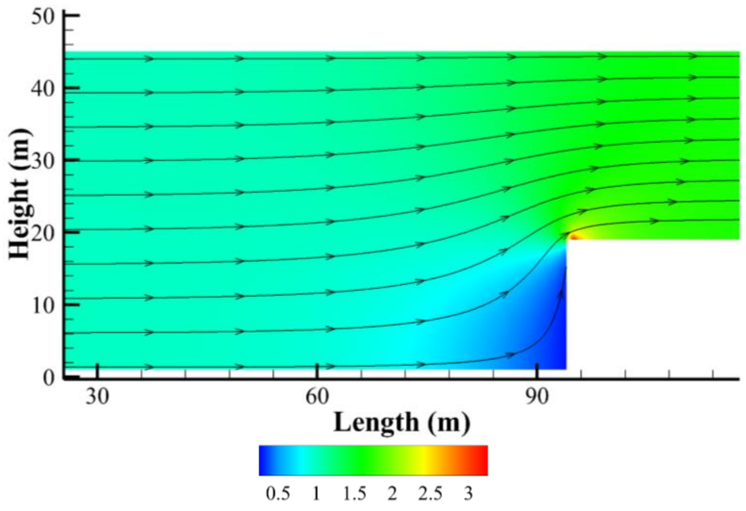

Fig. 6: Distribution of horizontal flow velocity and streamlines simultaneously in the domain

For improved visualization, Figures 2 to 6 were generated using Techplot. Moreover, Figures 7 to 9 depicting horizontal velocity, vertical velocity, and streamlines were directly generated within the programming languages used for implementation.

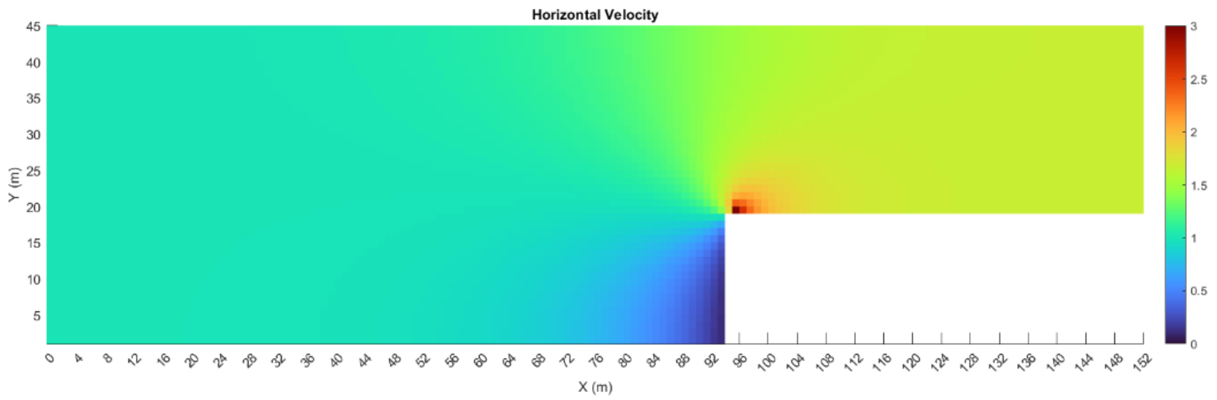

Figure 7: Horizontal Velocity

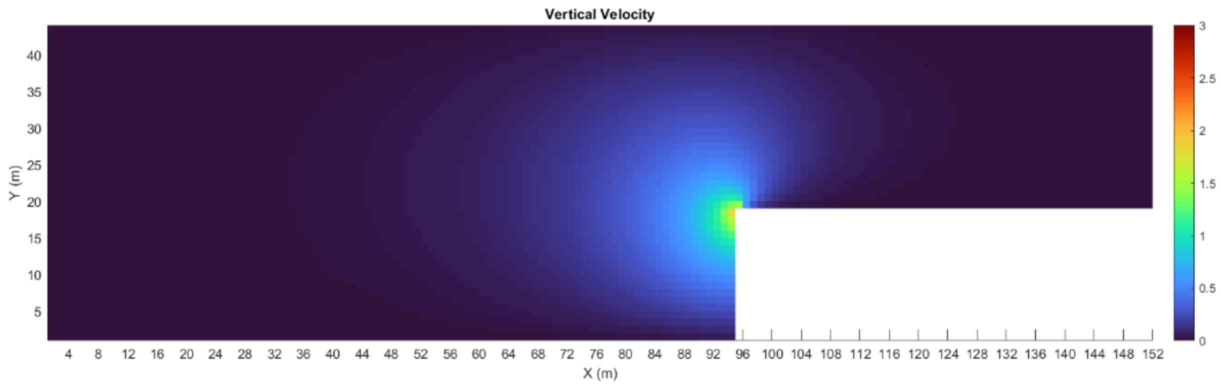

Figure 8: Vertical Velocity

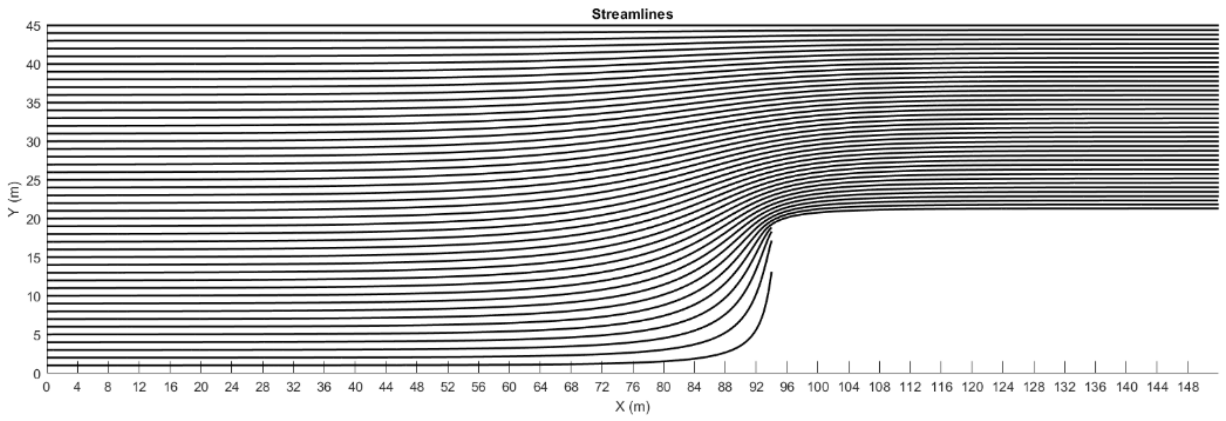

Figure 9: Streamlines

2-4 Conclusion

Investigating current simulation through the analysis of velocity fields encompassing horizontal and vertical flow velocities, along with streamlines, yielded insightful results. The observed outcomes demonstrated a commendable alignment with established theoretical equations. Notably, anticipated phenomena such as flow separation from sharp edges and channelizations were vividly visualized and substantiated within the simulation.

The overall findings affirm the capability of the employed numerical model in accurately replicating current simulations. This validation underscores the model's reliability and its potential utility across various practical applications in understanding and predicting current flow behaviors.