



Numerical Methods

Project

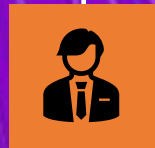
Computational Fluid Dynamics

Submitted to: LE. Rabia Basharat

Our Team



Muhammad Asim Umar

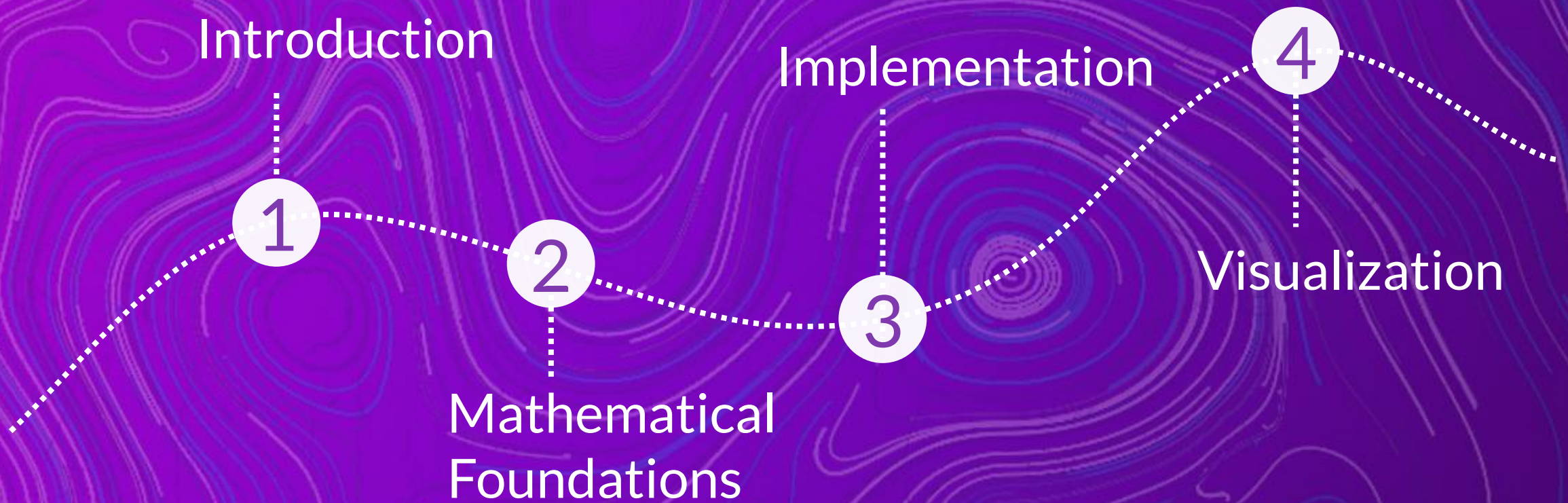


Muhammad Talha Kausar



Abdul Hannan Amir

Overview





PART 1

Introduction

Eulerian Fluid

Introduction

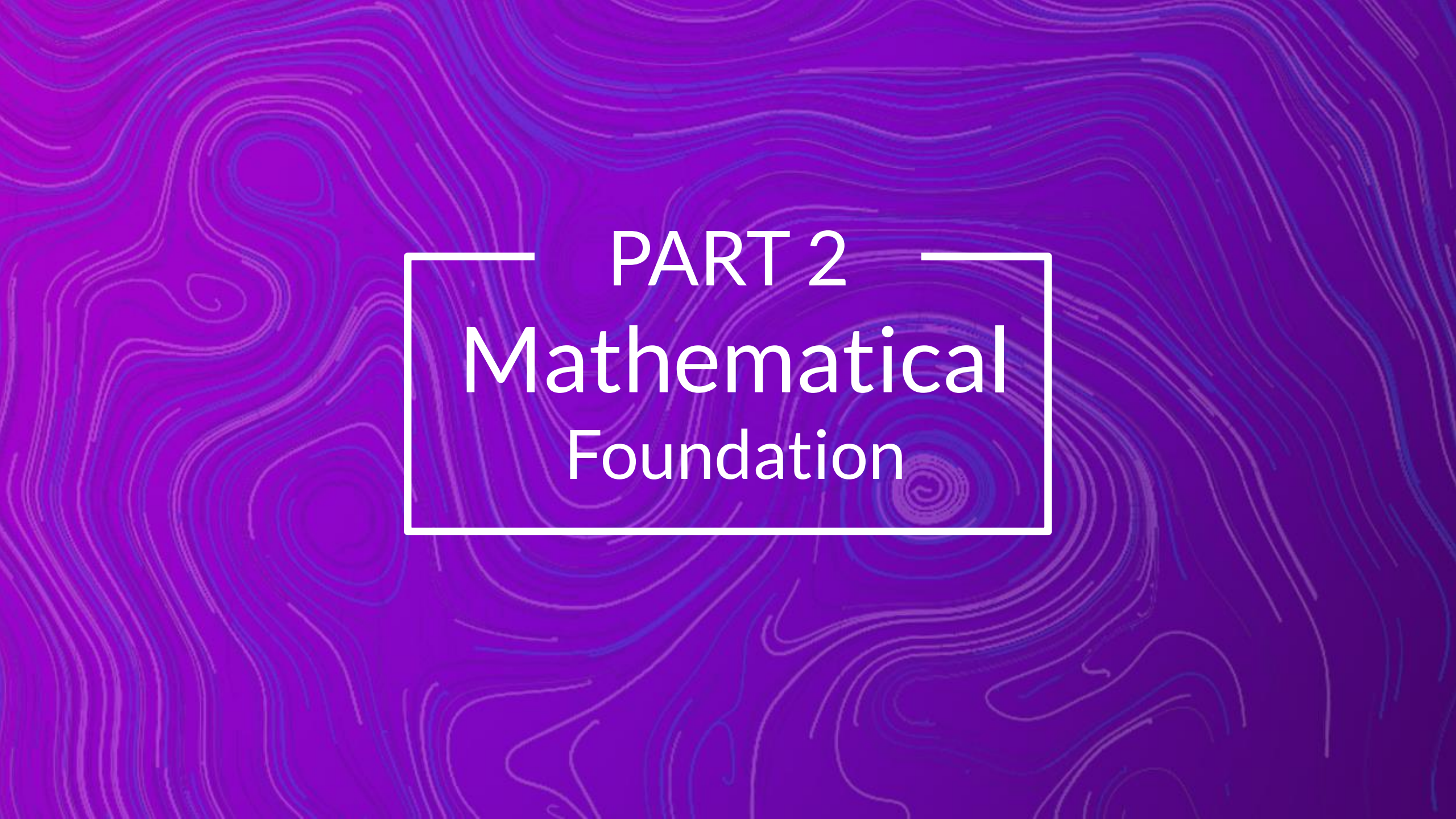
Objective

The goal of this project is to simulate fluid dynamics in a two-dimensional environment using MATLAB. By implementing numerical methods, we can visualize and analyze the behavior of fluids, which is essential for understanding various physical phenomena.

Importance

Fluid simulations are very important in our lives:

- **Physics and Engineering:** Used to model weather patterns, ocean currents, and aerodynamics.
- **Computer Graphics:** Essential for creating realistic animations in movies and video games.
- **Medical Research:** Helps in simulating blood flow in the human body.



PART 2

Mathematical Foundation

Mathematical Foundation

Fluid Motion

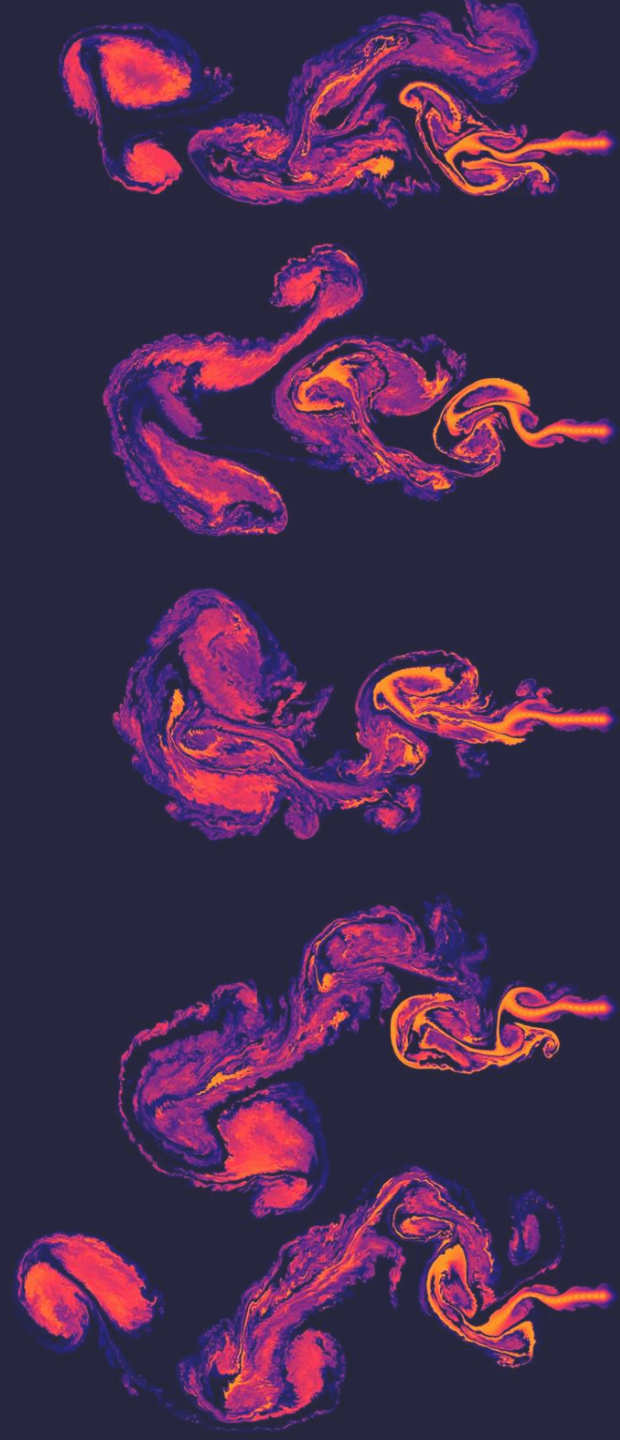
The project Simulates Eulerian fluid motion, the flow field is described from a fixed point in space. We observe how the fluid properties (such as velocity and pressure) change over time at each point in the grid.

Numerical Methods

Numerical Methods in Fluid motion is being used in following ways:

Navier-Stokes Equation: The foundation of fluid dynamics, describing the motion of fluid substances. For 2D incompressible flow, these equations are simplified into two main parts:

- **Momentum Equation:** Governs the velocity field.
- **Continuity Equation:** Ensures mass conservation and is used to compute the pressure field.



Numerical Methods

Jacobi Method

Solves the Pressure Equation to ensure that velocity is diverges based on pressure & iteratively updates the pressure field using Stencil operation:


$$p = \text{conv2}(p, J, \text{'same'}) + \text{rhs} / 4;$$

:: J = Stencil Equation

Runge-Kutta 4th Order

Accurately changes the velocity field and moves particles through the flow. It computes steps to update velocity and position.

$$[px_new, py_new] = \text{RK4}(X, Y, vx, vy, px, py, 1);$$



PART 3

Implementation Code

Implementation

Simulation Parameter & Grid Initialization

% Simulation parameters

s = 200; % Grid size

ar = 2; % Aspect ratio

J = [0 1 0; 1 0 1; 0 1 0]/4; % Stencil for Jacobi
method

% Create a grid

[X, Y] = meshgrid(1:s*ar, 1:s);

% Initialize pressure and velocity fields

[p, vx, vy] = deal(zeros(s, s*ar));

% Initial positions of particles (spread over a larger
area)

[px, py] = meshgrid(1:0.5:s*ar, 1:0.5:s);

px = reshape(px, numel(px), 1);

py = reshape(py, numel(py), 1);

% Save these initial positions for the inflow

pxo = px;

pyo = py;

Implementation

Jacobi Method

% Simulation parameters

s = 200; % Grid size

ar = 2; % Aspect ratio

J = [0 1 0; 1 0 1; 0 1 0]/4; % Stencil for Jacobi method

% Initialize pressure and velocity fields

[p, vx, vy] = deal(zeros(s, s*ar));

% Compute right-hand side for pressure equation

rhs = -computeDivergence(vx, vy);

% Jacobi iteration to solve for pressure

for i = 1:100

p = conv2(p, J, 'same') + rhs / 4;

end

% Function for computing divergence

function div = computeDivergence(vx, vy)

[dx_vx, ~] = gradient(vx);

[~, dy_vy] = gradient(vy);

div = dx_vx + dy_vy;

end

Implementation

RK4 Method

```
% Advect particles using Runge-Kutta 4th order  
method (1 = forward)
```

```
[px, py] = RK4(X, Y, vx, vy, px, py, 1);
```

```
% Function for Runge-Kutta 4th order method for  
advection
```

```
function [x_new, y_new] = RK4(X, Y, vx, vy, px, py, h)
```

```
    k1x = interp2(X, Y, vx, px, py, 'linear', 0);
```

```
    k1y = interp2(X, Y, vy, px, py, 'linear', 0);
```

```
    k2x = interp2(X, Y, vx, px + h/2 * k1x, py + h/2 * k1y,
```

```
    'linear', 0);
```

```
    k2y = interp2(X, Y, vy, px + h/2 * k1x, py + h/2 * k1y,
```

```
    'linear', 0);
```

```
    k3x = interp2(X, Y, vx, px + h/2 * k2x, py + h/2 * k2y,
```

```
    'linear', 0);
```

```
    k3y = interp2(X, Y, vy, px + h/2 * k2x, py + h/2 * k2y,
```

```
    'linear', 0);
```

```
    k4x = interp2(X, Y, vx, px + h * k3x, py + h * k3y,
```

```
    'linear', 0);
```


```
    k4y = interp2(X, Y, vy, px + h * k3x, py + h * k3y,
```

```
    'linear', 0);
```

```
    x_new = px + h/6 * (k1x + 2*k2x + 2*k3x + k4x);
```

```
    y_new = py + h/6 * (k1y + 2*k2y + 2*k3y + k4y);
```

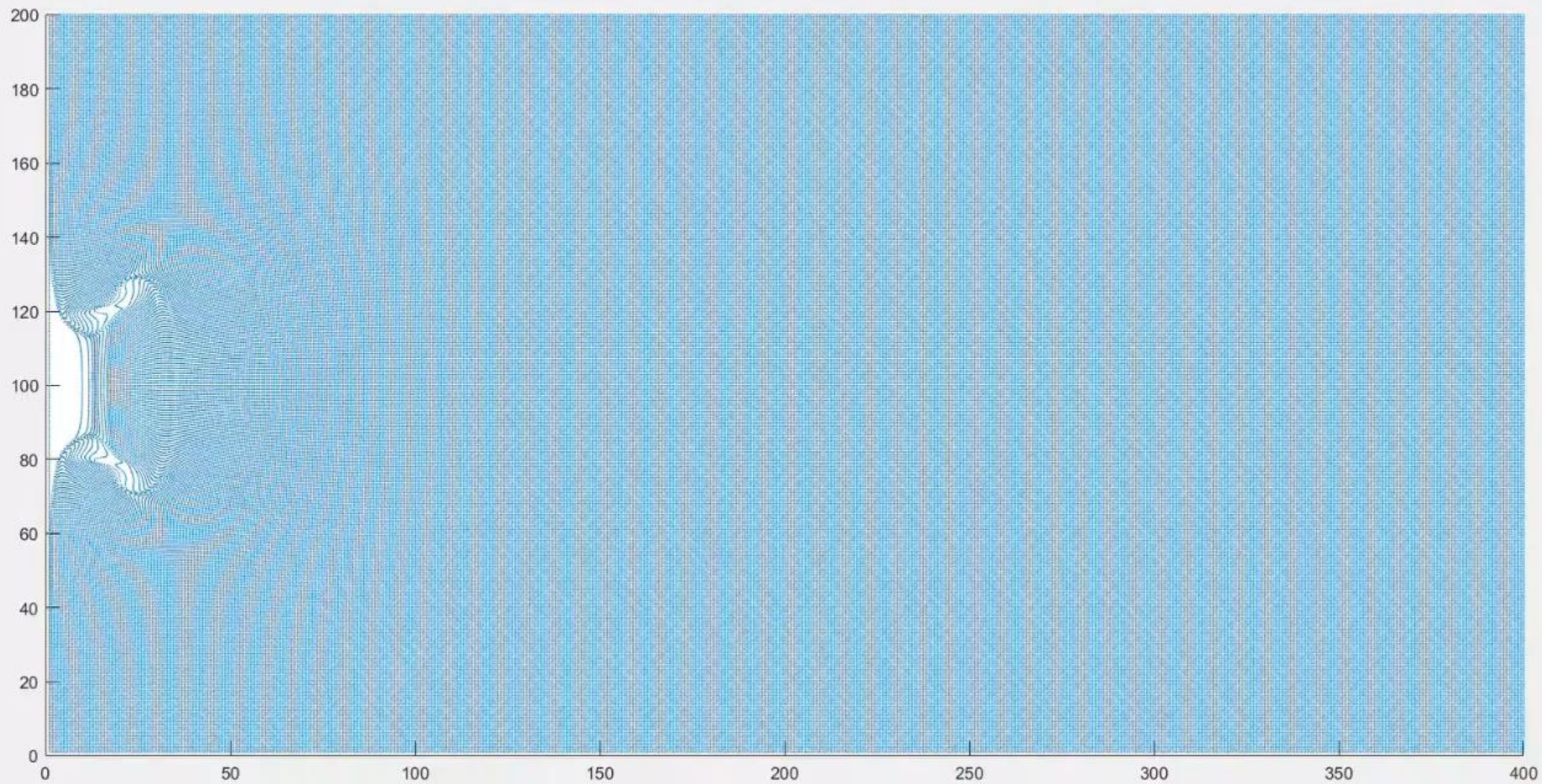
```
end
```

PART 4

Simulation

Fluid Motion



THE END !

