Interactive Pipe Flow Simulation: Report

1. Introduction

This report presents an interactive pipe flow simulation designed to help users visualize and analyze fluid behavior in pipes. The simulation demonstrates key fluid dynamics principles, focusing on how velocity and pressure change as a fluid moves through pipes of different cross-sectional areas.

2. Understanding the Science

2.1. The Continuity Equation

The continuity equation states that the mass flow rate in a pipe remains constant. This means:

- If a pipe narrows, the fluid must speed up.
- If a pipe widens, the fluid slows down.

Mathematically, this is expressed as:

A1v1=A2v2A_1 v_1 = A_2 v_2

where AA represents the pipe’s cross-sectional area and vv is the velocity of the fluid.

2.2. Bernoulli’s Principle

Bernoulli’s principle explains how velocity and pressure are related in a fluid. When velocity increases in a narrower section of a pipe, pressure decreases. The simplified equation is:

P1+12ρv12=P2+12ρv22P_1 + \frac{1}{2}\rho v_1^2 = P_2 + \frac{1}{2}\rho v_2^2

where PP represents pressure, ρ\rho is fluid density, and vv is velocity.

3. How the Simulation Works

3.1. Key Features

The simulation is structured around an **InteractivePipeFlowSimulation** class, which:

- Defines the pipe structure and sections.
- Uses physics equations to calculate velocity and pressure.
- Animates fluid movement with particles.
- Provides an interactive interface with real-time updates.

3.2. Important Functions

- **add_pipe_section()** – Adds a new segment to the pipe.
- **calculate_velocities()** – Computes fluid velocity in each section.

- **calculate_pressures()** – Determines pressure using Bernoulli's equation.
- **update_particles()** – Animates fluid motion.
- **create_interactive_visualization()** – Launches the user interface.

#Code:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import matplotlib.patches as patches
from matplotlib.widgets import Slider, Button
from matplotlib.gridspec import GridSpec

class InteractivePipeFlowSimulation:
    def __init__(self, initial_velocity=2.0, density=1000):
        """
        Initialize pipe flow simulation with interactive features

        Args:
            initial_velocity (float): Initial fluid velocity in m/s
            density (float): Fluid density in kg/m³ (default: water)
        """
        self.initial_velocity = initial_velocity
        self.density = density
        self.sections = []
        self.particles = None
        self.p0 = 101325  # Reference pressure (Pa)

    def add_pipe_section(self, length, diameter, position=None):
        """
        Add a pipe section with specified dimensions

        Args:
            length (float): Length of pipe section in meters
            diameter (float): Diameter of pipe section in meters
            position (float): Starting x-position (None for auto-placement)
        """
        if not self.sections:
            start_pos = 0
        elif position is None:
            start_pos = self.sections[-1]['start_pos'] + self.sections[-1]['length']
        else:
            start_pos = position

        self.sections.append({
            'length': length,
```

```python
            'diameter': diameter,
            'start_pos': start_pos,
            'area': np.pi * (diameter/2)**2
        })

    def calculate_velocities(self):
        """Calculate velocities in each pipe section using continuity equation"""
        if not self.sections:
            return

        initial_area = self.sections[0]['area']
        initial_velocity = self.initial_velocity

        for section in self.sections:
            # A₁v₁ = A₂v₂
            section['velocity'] = initial_velocity * (initial_area / section['area'])

    def calculate_pressures(self):
        """Calculate pressures in each pipe section using Bernoulli's equation"""
        if not self.sections:
            return

        self.calculate_velocities()

        # Start with reference pressure at first section
        initial_pressure = self.p0
        initial_velocity = self.sections[0]['velocity']

        for section in self.sections:
            # Bernoulli's equation: P₁ + ½ρv₁² = P₂ + ½ρv₂²
            dynamic_pressure = 0.5 * self.density * (initial_velocity**2 - section['velocity']**2)
            section['pressure'] = initial_pressure + dynamic_pressure

    def initialize_particles(self, n_particles=100):
        """Initialize particles for animation"""
        if not self.sections:
            return

        total_length = self.sections[-1]['start_pos'] + self.sections[-1]['length']
        max_diameter = max(section['diameter'] for section in self.sections)

        # Initialize random positions
        self.particles = {
            'x': np.random.uniform(0, total_length, n_particles),
            'y': np.random.uniform(-max_diameter/2.5, max_diameter/2.5, n_particles),
            'colors': np.random.uniform(0.3, 0.9, n_particles)
```

```python
    }

    def update_particles(self, dt=0.05):
        """Update particle positions based on local velocity"""
        if self.particles is None or not self.sections:
            return

        for i in range(len(self.particles['x'])):
            x = self.particles['x'][i]

            # Find which section the particle is in
            section_idx = 0
            for j, section in enumerate(self.sections):
                if section['start_pos'] <= x < section['start_pos'] + section['length']:
                    section_idx = j
                    break

            # Update position based on local velocity
            velocity = self.sections[section_idx]['velocity']
            self.particles['x'][i] += velocity * dt

            # If particle leaves the pipe, reset to the beginning with new y-position
            if self.particles['x'][i] > self.sections[-1]['start_pos'] + self.sections[-1]
['length']:
                self.particles['x'][i] = 0
                diameter = self.sections[0]['diameter']
                self.particles['y'][i] = np.random.uniform(-diameter/2.5, diameter/2.5)

            # Keep particles within vertical bounds of current section
            for section in self.sections:
                if section['start_pos'] <= x < section['start_pos'] + section['length']:
                    max_y = section['diameter'] / 2.5
                    if abs(self.particles['y'][i]) > max_y:
                        self.particles['y'][i] = np.sign(self.particles['y'][i]) * max_y
                    break

    def create_interactive_visualization(self):
        """Create an interactive visualization with sliders and dynamic graphs"""
        self.calculate_pressures()

        # Create figure with GridSpec for better layout control
        fig = plt.figure(figsize=(15, 12))
        gs = GridSpec(4, 1, height_ratios=[2, 1, 0.1, 0.1])

        # Pipe view subplot
        ax_pipe = fig.add_subplot(gs[0])
        ax_pipe.set_title("Pipe Flow Visualization", fontsize=14)
```

```python

    # Pressure/velocity graph subplot
    ax_graph = fig.add_subplot(gs[1])

    # Slider axes
    ax_velocity_slider = fig.add_subplot(gs[2])
    ax_density_slider = fig.add_subplot(gs[3])

    # Plot pipe sections
    max_velocity = 0
    x_coordinates = []
    velocities = []
    pressures = []

    for section in self.sections:
        # Create pipe section rectangle
        rect = patches.Rectangle(
            (section['start_pos'], -section['diameter']/2),
            section['length'],
            section['diameter'],
            facecolor='lightgray',
            edgecolor='black'
        )
        ax_pipe.add_patch(rect)

        # Store data for graphs
        x_pos = section['start_pos'] + section['length']/2
        x_coordinates.append(x_pos)
        velocities.append(section['velocity'])
        pressures.append(section['pressure'])

        if section['velocity'] > max_velocity:
            max_velocity = section['velocity']

    # Initialize particles for animation if not already done
    if self.particles is None:
        self.initialize_particles()

    # Create scatter plot for particles
    scatter = ax_pipe.scatter(
        self.particles['x'],
        self.particles['y'],
        c=self.particles['colors'],
        cmap='Blues',
        s=10,
        alpha=0.8
    )
```

```
        # Set axis limits for pipe view
        ax_pipe.set_xlim(0, self.sections[-1]['start_pos'] + self.sections[-1]['length'])
        max_diameter = max(section['diameter'] for section in self.sections)
        ax_pipe.set_ylim(-max_diameter, max_diameter)
        ax_pipe.set_aspect('equal')
        ax_pipe.set_xlabel('Position (m)')
        ax_pipe.set_ylabel('Diameter (m)')

        # Plot velocity and pressure graphs
        velocity_line, = ax_graph.plot(x_coordinates, velocities, 'b-', label='Velocity 
(m/s)')
        ax_graph_twin = ax_graph.twinx()
        pressure_line, = ax_graph_twin.plot(x_coordinates, np.array(pressures)/1000, 'r-', 
label='Pressure (kPa)')

        ax_graph.set_xlabel('Position (m)')
        ax_graph.set_ylabel('Velocity (m/s)', color='b')
        ax_graph_twin.set_ylabel('Pressure (kPa)', color='r')

        # Add legends
        lines1, labels1 = ax_graph.get_legend_handles_labels()
        lines2, labels2 = ax_graph_twin.get_legend_handles_labels()
        ax_graph.legend(lines1 + lines2, labels1 + labels2, loc='upper right')

        # Create sliders
        velocity_slider = Slider(
            ax=ax_velocity_slider,
            label='Initial Velocity (m/s)',
            valmin=0.5,
            valmax=5.0,
            valinit=self.initial_velocity,
            valstep=0.1
        )

        density_slider = Slider(
            ax=ax_density_slider,
            label='Fluid Density (kg/m³)',
            valmin=800,
            valmax=1200,
            valinit=self.density,
            valstep=10
        )

        # Update function for parameters
        def update_simulation(val=None):
            self.initial_velocity = velocity_slider.val
```

```
            self.density = density_slider.val
            self.calculate_pressures()

            # Update velocity and pressure data
            velocities = [section['velocity'] for section in self.sections]
            pressures = [section['pressure'] for section in self.sections]

            # Update plots
            velocity_line.set_ydata(velocities)
            pressure_line.set_ydata(np.array(pressures)/1000)

            # Update y-axis limits for velocity and pressure
            ax_graph.set_ylim(0, max(velocities) * 1.1)
            ax_graph_twin.set_ylim(min(np.array(pressures)/1000) * 0.9,
                                   max(np.array(pressures)/1000) * 1.1)

            fig.canvas.draw_idle()

        # Connect sliders to update function
        velocity_slider.on_changed(update_simulation)
        density_slider.on_changed(update_simulation)

        plt.tight_layout()

        # Animation update function
        def update_animation(frame):
            self.update_particles()
            scatter.set_offsets(np.column_stack((self.particles['x'], self.particles['y'])))
            return scatter,

        # Create animation
        ani = FuncAnimation(fig, update_animation, frames=100, interval=50, blit=True)

        return fig, ani

def main():
    # Create simulation instance
    sim = InteractivePipeFlowSimulation(initial_velocity=2.0)

    # Add pipe sections with different diameters
    sim.add_pipe_section(length=2, diameter=0.3)  # Initial section
    sim.add_pipe_section(length=1, diameter=0.15)  # Constriction
    sim.add_pipe_section(length=2, diameter=0.3)  # Back to original diameter
    sim.add_pipe_section(length=1, diameter=0.45)  # Expansion
    sim.add_pipe_section(length=2, diameter=0.3)  # Final section

    # Create and display interactive visualization
```

```python
    fig, ani = sim.create_interactive_visualization()

    plt.show()

if __name__ == "__main__":
    main()
```

4. Visualizing the Flow

4.1. What You See in the Simulation

- **Pipe Representation** – Sections of different diameters.
- **Particle Animation** – Moving particles show fluid velocity.
- **Dynamic Graphs** – Real-time velocity and pressure changes.

4.2. User Controls

- **Velocity Slider** – Adjusts initial flow speed.
- **Density Slider** – Changes fluid density, affecting pressure.

5. Example Setup

A sample pipe configuration includes:

- **Initial section:** 2m long, 0.3m diameter.
- **Constriction:** 1m long, 0.15m diameter (higher velocity).
- **Expansion:** 1m long, 0.45m diameter (lower velocity).
- **Final section:** 2m long, 0.3m diameter.

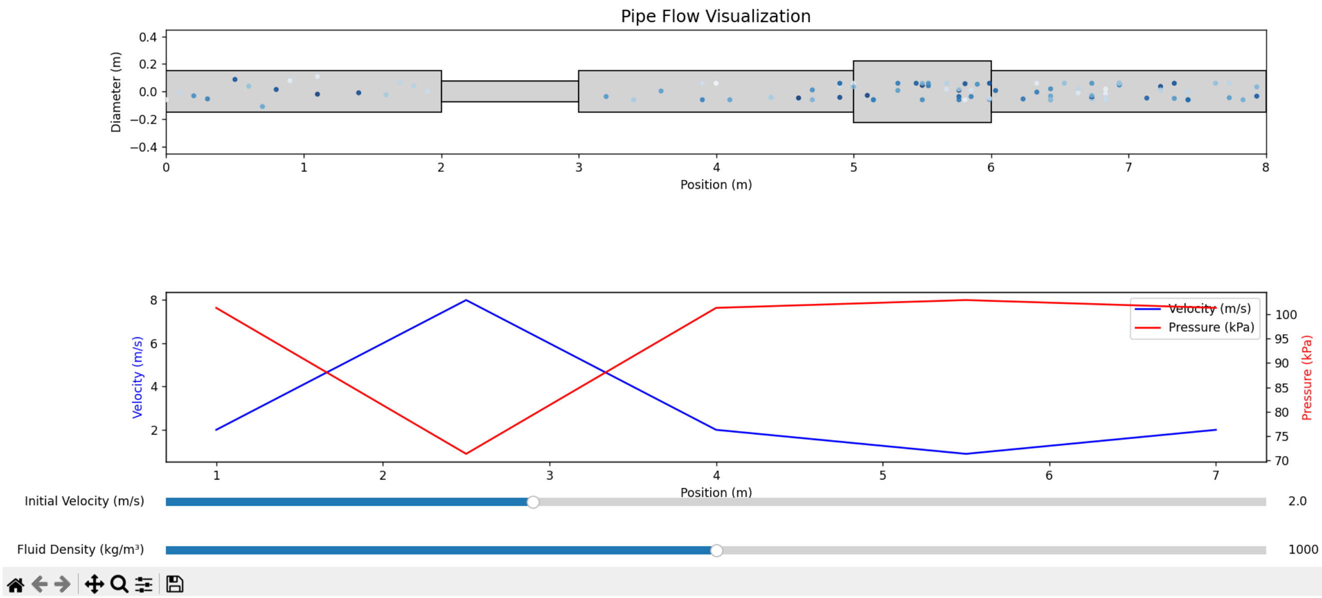
Expected Behavior:

- Velocity increases in narrow sections.
- Pressure decreases where velocity is high.
- Pressure recovers when the pipe widens.

6. Running the Simulation

1. Set the initial conditions (velocity, density).
2. Define the pipe sections.
3. Start the interactive visualization.
4. Adjust sliders and observe changes.

7. Output



8. Conclusion

This interactive simulation serves as a valuable educational tool for understanding fluid dynamics. By visualizing how velocity and pressure change in different pipe sections, users can intuitively grasp key principles like the continuity equation and Bernoulli's principle. Future updates could enhance realism by including friction, turbulence, and 3D modeling.