UQ

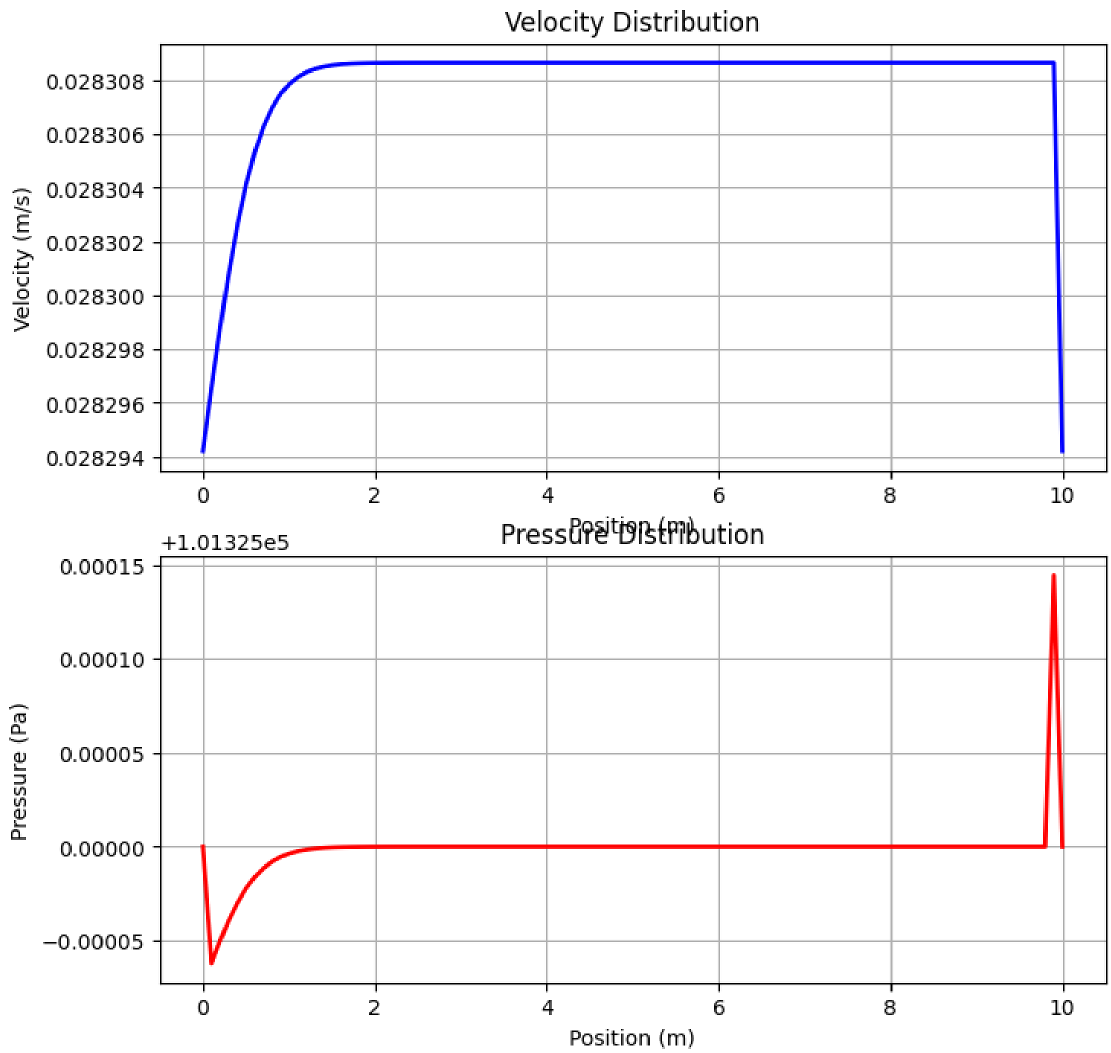
September 18, 2023

Fluid simulation

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     import math


     def calculate_friction_factor(Re, pipe_roughness, pipe_diameter):
         # Hagen-Poiseuille equaiton for luminar
         if Re < 2000:
             f = 64 / Re
         # Swamee-Jain equation for turbulance
         else:
             f = 0.25 / (math.log10((pipe_roughness / (3.7 * pipe_diameter)) + (5.74 /
      ↪ Re**0.9)))**2
         return f

     def simulate_1D_pipe_flow(pipe_length, pipe_diameter, pipe_roughness,␣
      ↪inlet_flow_rate, inlet_pressure, outlet_pressure, fluid_density,␣
      ↪fluid_viscosity, total_time, num_points):

         # Calculate pipe cross-sectional area
         area = np.pi * (pipe_diameter / 2)**2

         # Calculate the initial velocity
         initial_velocity = inlet_flow_rate / area
         velocity = np.ones(num_points) * initial_velocity
         pressure = np.linspace(inlet_pressure, outlet_pressure, num_points)

         dx = pipe_length / (num_points - 1)

         # Calculate the maximum velocity (for time step estimation)
         max_velocity = 1.2 * initial_velocity

         # Estimate the time step
         C = 0.1 # Courant number
         time_step = C * min(dx / max_velocity, dx**2 / fluid_viscosity)
         num_time_steps = int(total_time / time_step)
```

```python
    for t in range(num_time_steps):
        for i in range(1, num_points - 1):
            Re = fluid_density * abs(velocity[i]) * pipe_diameter / fluid_viscosity
            f = calculate_friction_factor(Re, pipe_roughness, pipe_diameter)

            # Crank-Nicolson method with 2nd-order upwind scheme
            a = fluid_viscosity * time_step / (2 * dx**2)
            b = f * velocity[i] * abs(velocity[i]) * time_step / (4 * pipe_diameter)
            c = fluid_density * dx / (2 * time_step)

            if velocity[i] >= 0:
                A = c - a + 0.5 * fluid_density * velocity[i] * dx
                B = c + a
                C = -a - b
                D = -a + b
            else:
                A = c - a
                B = c + a + 0.5 * fluid_density * velocity[i] * dx
                C = -a - b
                D = -a + b

            velocity[i] = (A * velocity[i - 1] + B * velocity[i] + C * velocity[i + 1]) / (A + B + C + D)

        pressure[0] = inlet_pressure
        pressure[-1] = outlet_pressure
        pressure[1:-1] = pressure[1:-1] - fluid_density * velocity[1:-1] * (velocity[2:] - velocity[:-2]) * dx

    return velocity, pressure

# Define pipe parameters
pipe_length = 10.0 #m
pipe_diameter = 0.03 #m
pipe_roughness = 0.0001
inlet_flow_rate = 0.00002  # m^3/s
inlet_pressure = 101325.0  # Pa
outlet_pressure = 101325.0   # Pa
fluid_density = 1000.0  # kg/m^3
fluid_viscosity = 0.001  # Pa*s
total_time = 2.0  # s
num_points = 100

# Run the simulation
```

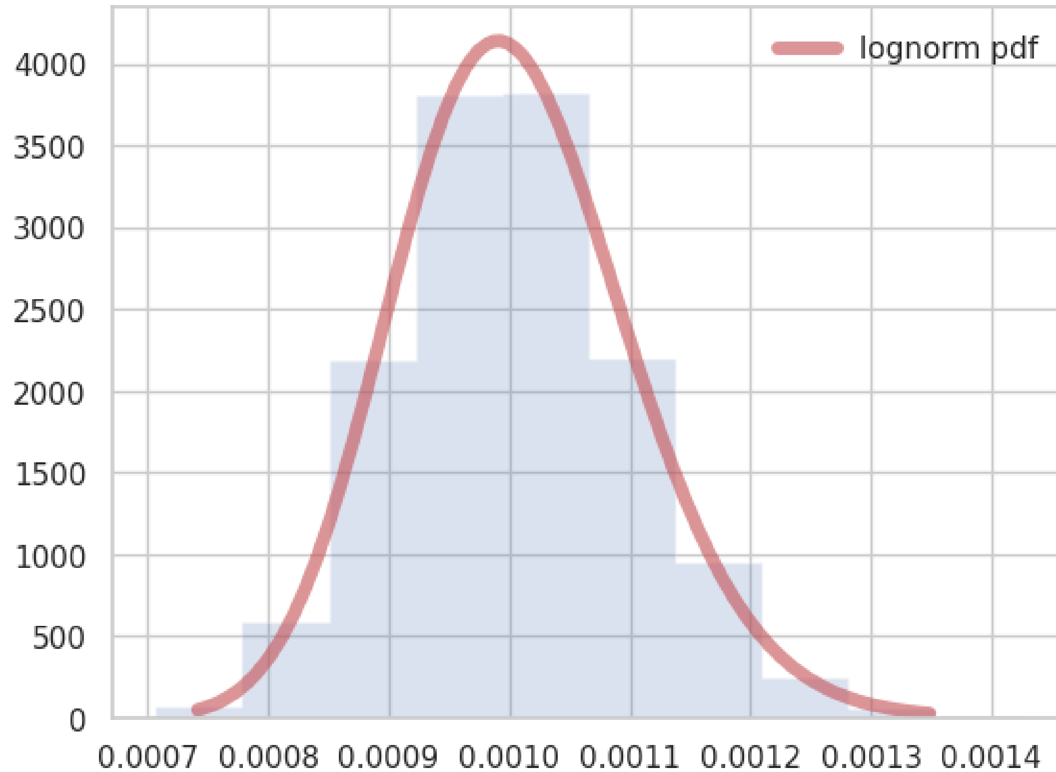
```
velocity, pressure = simulate_1D_pipe_flow(pipe_length, pipe_diameter,␣
↪pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure,␣
↪fluid_density, fluid_viscosity, total_time, num_points)

# Calculate the position of each point
x = np.linspace(0, pipe_length, num_points)

# Calculate the flow rate at each point
flow_rate = velocity * np.pi * (pipe_diameter / 2)**2

# Plot the results
fig, axs = plt.subplots(2, figsize=(8, 8))
axs[0].plot(x, velocity, 'b-', linewidth=2)
axs[0].set_xlabel('Position (m)')
axs[0].set_ylabel('Velocity (m/s)')
axs[0].set_title('Velocity Distribution')
axs[0].grid(True)

axs[1].plot(x, pressure, 'r-', linewidth=2)
axs[1].set_xlabel('Position (m)')
axs[1].set_ylabel('Pressure (Pa)')
axs[1].set_title('Pressure Distribution')
axs[1].grid(True)

plt.show()
avg_velocity = np.mean(velocity)
print("Average Velocity: {:.8f} m/s".format(avg_velocity))
```




```
Average Velocity: 0.02830785 m/s
```

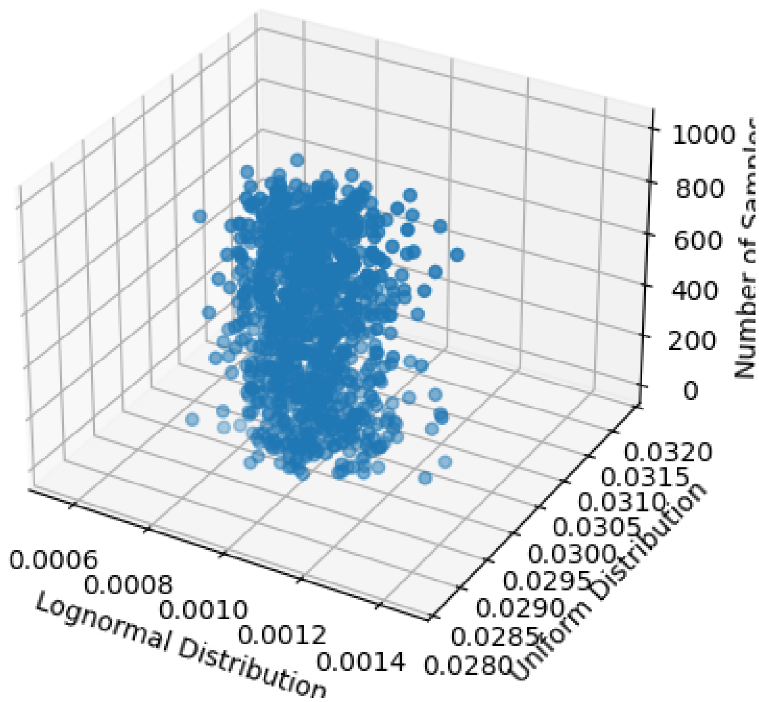
[ ]:
```python
import numpy as np
from scipy.stats import lognorm
import matplotlib.pyplot as plt

# Paremeters setting
mu = -6.907755  # mean value
sigma = 0.096809  # standard deviation

lognorm_dist = lognorm(s=sigma, scale=np.exp(mu))

samples = lognorm_dist.rvs(size=10000)

# Draw pdf
```

```python
fig, ax = plt.subplots(1, 1)
x = np.linspace(lognorm_dist.ppf(0.001), lognorm_dist.ppf(0.999), 100)
ax.plot(x, lognorm_dist.pdf(x), 'r-', lw=5, alpha=0.6, label='lognorm pdf')
ax.hist(samples, density=True, histtype='stepfilled', alpha=0.2)
ax.legend(loc='best', frameon=False)
plt.show()
```



[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Setting
mu = -6.907755  # Mean value
sigma = 0.096809  # Standard deviation
num_samples = 1000  # Number of samples

# Produce the samples
fluid_viscosity_samples = np.random.lognormal(mu, sigma, num_samples)
pipe_diameter_samples = np.random.uniform(0.029, 0.03, num_samples)

# Draw
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(fluid_viscosity_samples, pipe_diameter_samples, np.
 ↪arange(num_samples))
ax.set_xlabel('Lognormal Distribution')
ax.set_ylabel('Uniform Distribution')
ax.set_zlabel('Number of Samples')

# Adjust x,y axiss
ax.set_xlim([0.0005, 0.0015])
ax.set_ylim([0.028, 0.032])

plt.show()
```



Monte Carlo Solver

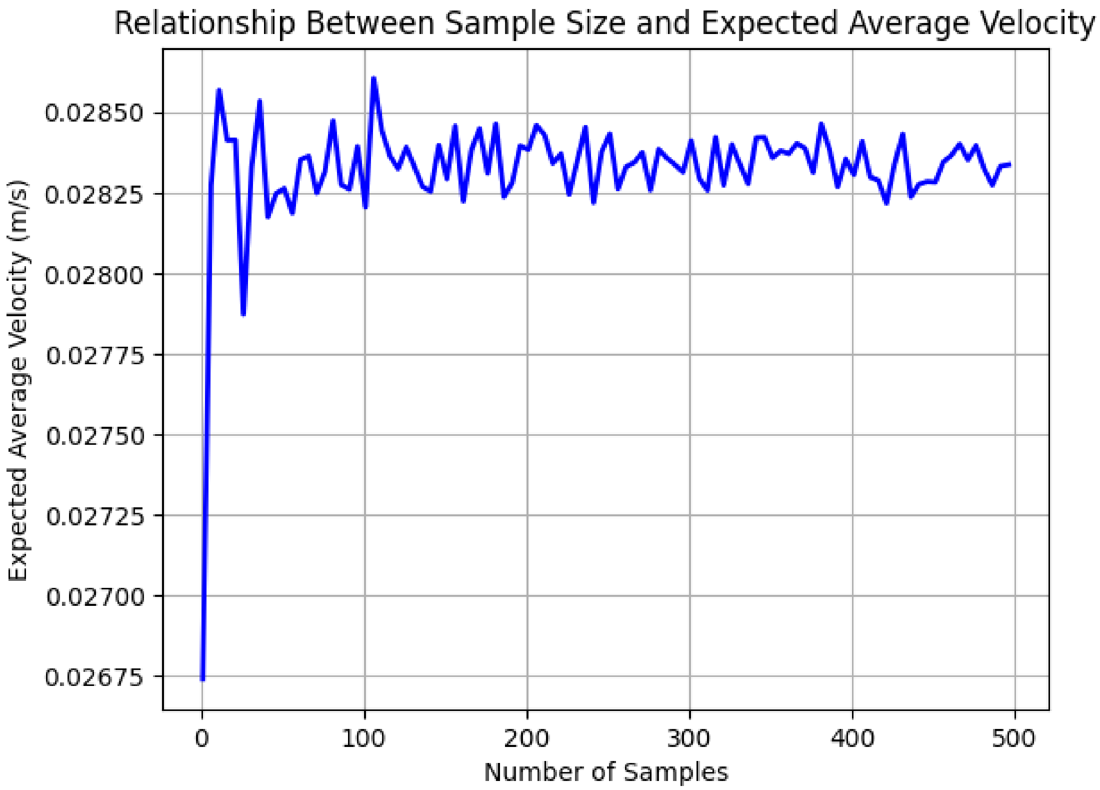
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import tqdm
import csv

def calculate_friction_factor(Re, pipe_roughness, pipe_diameter):
```

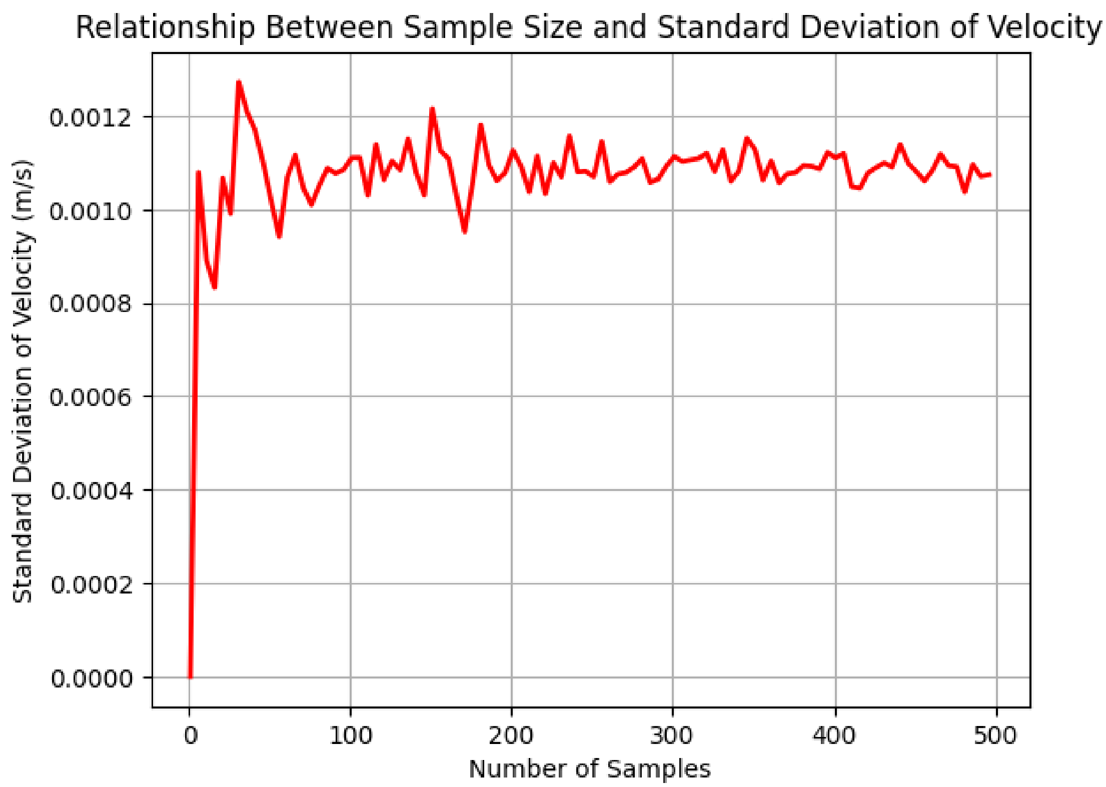
```python
    # Hagen-Poiseuille equaiton for luminar
    if Re < 2000:
        f = 64 / Re
    # Swamee-Jain equation for turbulance
    else:
        f = 0.25 / (math.log10((pipe_roughness / (3.7 * pipe_diameter)) + (5.74 / Re**0.9)))**2
    return f

def simulate_1D_pipe_flow(pipe_length, pipe_diameter, pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure, fluid_density, fluid_viscosity, total_time, num_points):

    # Calculate pipe cross-sectional area
    area = np.pi * (pipe_diameter / 2)**2

    # Calculate the initial velocity
    initial_velocity = inlet_flow_rate / area
    velocity = np.ones(num_points) * initial_velocity
    pressure = np.linspace(inlet_pressure, outlet_pressure, num_points)

    dx = pipe_length / (num_points - 1)

    # Calculate the maximum velocity (for time step estimation)
    max_velocity = 1.2 * initial_velocity

    # Estimate the time step
    C = 0.1 # Courant number
    time_step = C * min(dx / max_velocity, dx**2 / fluid_viscosity)
    num_time_steps = int(total_time / time_step)

    for t in range(num_time_steps):
        for i in range(1, num_points - 1):
            Re = fluid_density * abs(velocity[i]) * pipe_diameter / fluid_viscosity
            f = calculate_friction_factor(Re, pipe_roughness, pipe_diameter)

            # Crank-Nicolson method with 2nd-order upwind scheme
            a = fluid_viscosity * time_step / (2 * dx**2)
            b = f * velocity[i] * abs(velocity[i]) * time_step / (4 * pipe_diameter)
            c = fluid_density * dx / (2 * time_step)

            if velocity[i] >= 0:
                A = c - a + 0.5 * fluid_density * velocity[i] * dx
                B = c + a
                C = -a - b
```

```
                D = -a + b
            else:
                A = c - a
                B = c + a + 0.5 * fluid_density * velocity[i] * dx
                C = -a - b
                D = -a + b

            velocity[i] = (A * velocity[i - 1] + B * velocity[i] + C *␣
↪velocity[i + 1]) / (A + B + C + D)

        pressure[0] = inlet_pressure
        pressure[-1] = outlet_pressure
        pressure[1:-1] = pressure[1:-1] - fluid_density * velocity[1:-1] *␣
↪(velocity[2:] - velocity[:-2]) * dx

    return velocity, pressure

# Define pipe parameters
pipe_length = 10.0 #m
pipe_roughness = 0.0001
inlet_flow_rate = 0.00002  # m^3/s
inlet_pressure = 101325.0  # Pa
outlet_pressure = 101325.0   # Pa
fluid_density = 1000.0  # kg/m^3
total_time = 2.0  # s
num_points = 100

# Define the log-normal distribution of fluid viscosity
mu = -6.907755  # Log-normal distribution mean
sigma = 0.096809  # Log-normal distribution standard deviation

# Define the range of sample sizes to consider
min_samples = 1
max_samples = 500
step = 5

# Initialize arrays to store results
sample_sizes = np.arange(min_samples, max_samples + 1, step)
expected_avg_velocities = np.zeros(len(sample_sizes))
velocity_std_devs = np.zeros(len(sample_sizes))

# Loop over different sample sizes and calculate the expected average velocity␣
↪for each
for i, num_samples in enumerate(tqdm(sample_sizes, desc='Processing samples')):
    # Generate samples of fluid viscosity using Monte Carlo simulation
    fluid_viscosity_samples = np.random.lognormal(mu, sigma, num_samples)
    pipe_diameter_samples = np.random.uniform(0.029, 0.031, num_samples)
```

```
    # Calculate the average velocity for each sample of fluid viscosity
    avg_velocities = []
    for j in range(num_samples):
        _, pressure = simulate_1D_pipe_flow(pipe_length,␣
↪pipe_diameter_samples[j], pipe_roughness, inlet_flow_rate, inlet_pressure,␣
↪outlet_pressure, fluid_density, fluid_viscosity_samples[j], total_time,␣
↪num_points)
        avg_velocities.append(np.mean(_))

    # Calculate the expected value of the average velocity
    expected_avg_velocities[i] = np.mean(avg_velocities)
    velocity_std_devs[i] = np.std(avg_velocities)

# Save the results to a CSV file
with open('results.csv', 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(['Number of Samples', 'Expected Average Velocity (m/s)',␣
↪'Standard Deviation of Velocity (m/s)'])
    for i in range(len(sample_sizes)):
        csv_writer.writerow([sample_sizes[i], expected_avg_velocities[i],␣
↪velocity_std_devs[i]])

# Plot the relationship between sample size and expected average velocity
plt.figure(1)
plt.plot(sample_sizes, expected_avg_velocities, 'b-', linewidth=2)
plt.xlabel('Number of Samples')
plt.ylabel('Expected Average Velocity (m/s)')
plt.title('Relationship Between Sample Size and Expected Average Velocity')
plt.grid(True)

# Plot the relationship between sample size and standard deviation of velocity
plt.figure(2)
plt.plot(sample_sizes, velocity_std_devs, 'r-', linewidth=2)
plt.xlabel('Number of Samples')
plt.ylabel('Standard Deviation of Velocity (m/s)')
plt.title('Relationship Between Sample Size and Standard Deviation of Velocity')
plt.grid(True)

plt.show()
```

```
Processing samples: 100%|| 100/100 [00:57<00:00,  1.75it/s]
```


Relationship Between Sample Size and Expected Average Velocity
Expected Average Velocity (m/s)
Number of Samples
0.02850
0.02825
0.02800
0.02775
0.02750
0.02725
0.02700
0.02675
0
100
200
300
400
500



GMs(Polynominal Chaos Expansion)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import chaospy as cp
from tqdm import tqdm
import math
import csv

def calculate_friction_factor(Re, pipe_roughness, pipe_diameter):
    if Re < 2000:
        f = 64 / Re
    else:
        f = 0.25 / (math.log10((pipe_roughness / (3.7 * pipe_diameter)) + (5.74 /
 ↪ Re**0.9)))**2
    return f

def simulate_1D_pipe_flow(pipe_length, pipe_diameter, pipe_roughness,␣
 ↪inlet_flow_rate, inlet_pressure, outlet_pressure, fluid_density,␣
 ↪fluid_viscosity, total_time, num_points):
    area = np.pi * (pipe_diameter / 2)**2
    initial_velocity = inlet_flow_rate / area
```

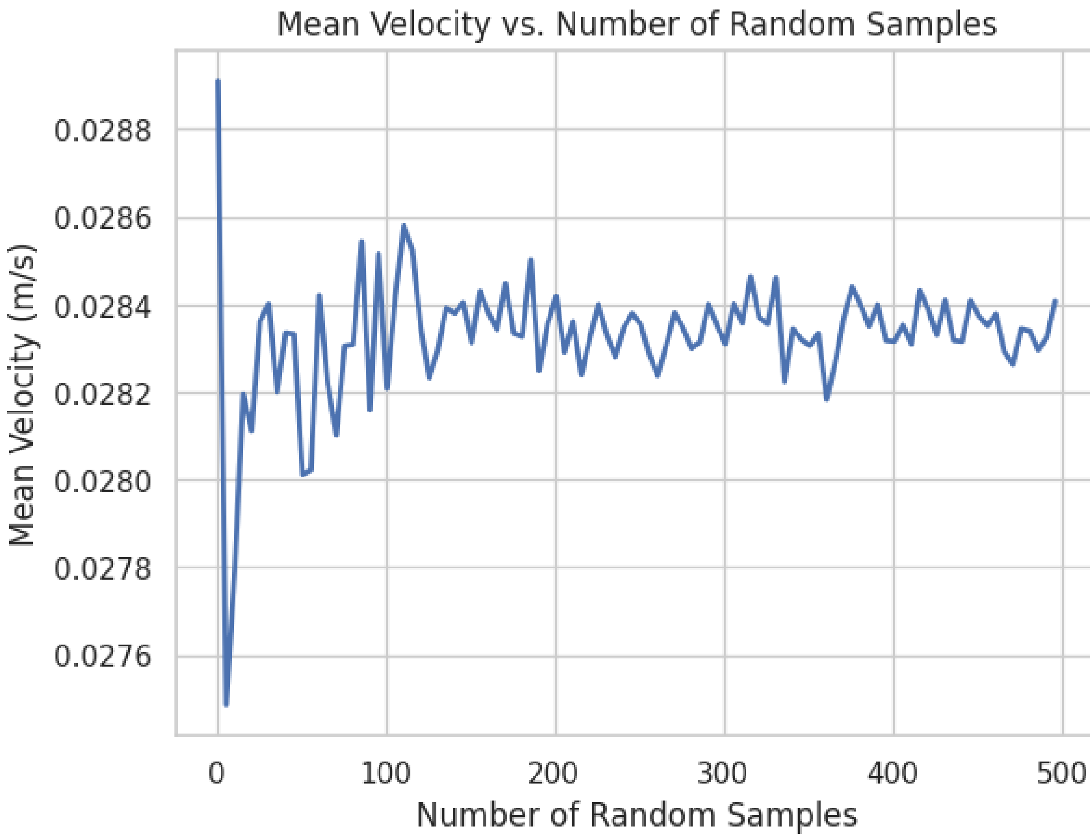
```
    velocity = np.ones(num_points) * initial_velocity
    pressure = np.linspace(inlet_pressure, outlet_pressure, num_points)

    dx = pipe_length / (num_points - 1)
    max_velocity = 1.2 * initial_velocity
    C = 0.1
    time_step = C * min(dx / max_velocity, dx**2 / fluid_viscosity)
    num_time_steps = int(total_time / time_step)

    for t in range(num_time_steps):
        for i in range(1, num_points - 1):
            Re = fluid_density * abs(velocity[i]) * pipe_diameter /
 ↪fluid_viscosity
            f = calculate_friction_factor(Re, pipe_roughness, pipe_diameter)

            a = fluid_viscosity * time_step / (2 * dx**2)
            b = f * velocity[i] * abs(velocity[i]) * time_step / (4 *
 ↪pipe_diameter)
            c = fluid_density * dx / (2 * time_step)

            if velocity[i] >= 0:
                A = c - a + 0.5 * fluid_density * velocity[i] * dx
                B = c + a
                C = -a - b
                D = -a + b
            else:
                A = c - a
                B = c + a + 0.5 * fluid_density * velocity[i] * dx
                C = -a - b
                D = -a + b

            velocity[i] = (A * velocity[i - 1] + B * velocity[i] + C *
 ↪velocity[i + 1]) / (A + B + C + D)

        pressure[0] = inlet_pressure
        pressure[-1] = outlet_pressure
        pressure[1:-1] = pressure[1:-1] - fluid_density * velocity[1:-1] *
 ↪(velocity[2:] - velocity[:-2]) * dx

    return velocity, pressure

# Pipe parameters
pipe_length = 10.0
pipe_roughness = 0.0001
inlet_flow_rate = 0.00002
inlet_pressure = 101325.0
outlet_pressure = 101325.0
```

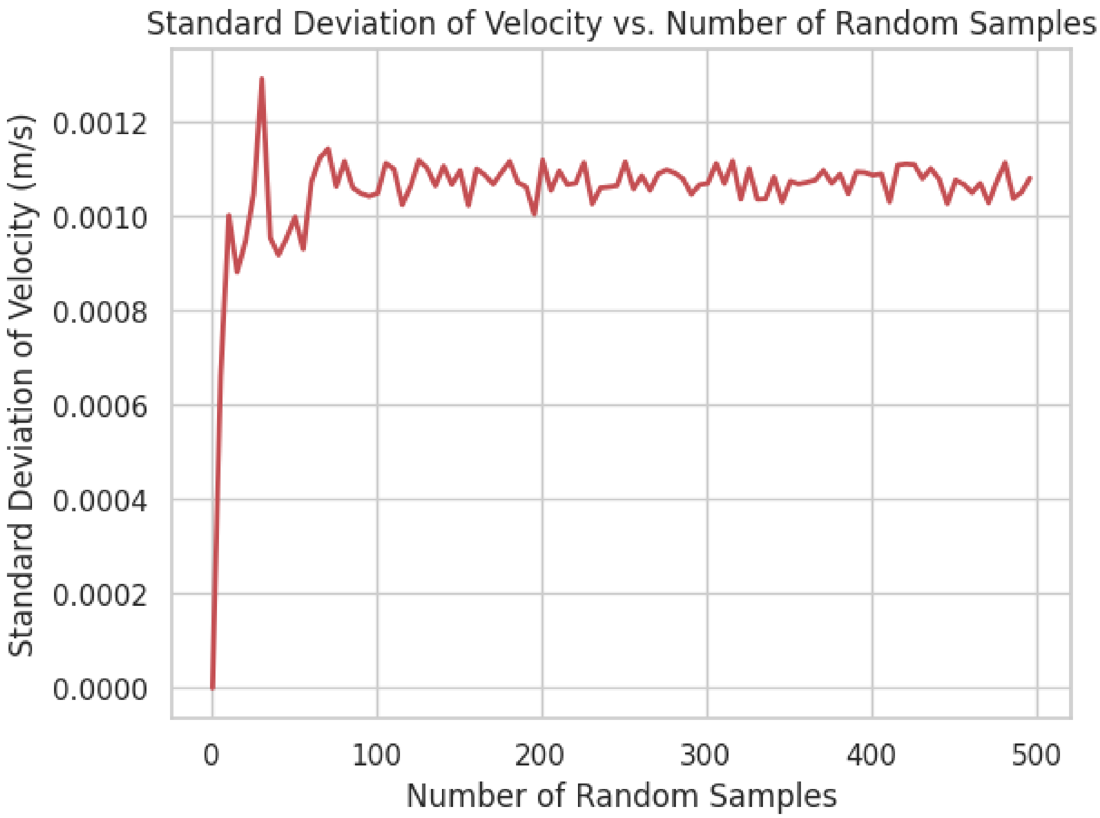
```
fluid_density = 1000.0
total_time = 2.0
num_points = 100


def uncertainty_analysis(fluid_viscosity_distribution,␣
↪pipe_diameter_distribution, num_samples):
    joint_distribution = cp.J(cp.LogNormal(mu=-6.907755, sigma=0.096809), cp.
↪Uniform(0.029, 0.031))

    samples = joint_distribution.sample(num_samples)
    avg_velocities = np.array([simulate_1D_pipe_flow(pipe_length, d,␣
↪pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure,␣
↪fluid_density, v, total_time, num_points)[0].mean() for v, d in samples.T])

    pce_order = 2
    basis_polynomials = cp.expansion.stieltjes(pce_order, joint_distribution)
    pce_coeffs = cp.fit_regression(basis_polynomials, samples, avg_velocities)

    return pce_coeffs, basis_polynomials

# Run the uncertainty analysis using 100 random samples
num_samples = 100
pce_coeffs, basis_polynomials = uncertainty_analysis(cp.LogNormal(mu=-6.907755,␣
↪sigma=0.096809), cp.Uniform(0.029, 0.031), num_samples)
pce_expansion = cp.sum(pce_coeffs * basis_polynomials)
print("PCE coefficients:", pce_coeffs)

# Define the range of sample sizes to consider
min_samples = 1
max_samples = 500
step = 5
sample_sizes = np.arange(min_samples, max_samples + 1, step)
pce_avg_velocities = []
pce_std_devs = []

# Create a CSV file to store the results
with open('pce_results.csv', mode='w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(['Number of Samples', 'Mean Velocity', 'Standard␣
↪Deviation'])

    # Run the uncertainty analysis for different sample sizes
    for num_random_samples in tqdm(sample_sizes, desc='Processing samples'):
        random_samples = cp.J(cp.LogNormal(mu=-6.907755, sigma=0.096809), cp.
↪Uniform(0.029, 0.031)).sample(num_random_samples)
```

```
        avg_velocity = np.mean([pce_expansion(*sample) for sample in␣
↪random_samples.T])
        std_dev = np.std([pce_expansion(*sample) for sample in random_samples.T])
        pce_avg_velocities.append(avg_velocity)
        pce_std_devs.append(std_dev)

        # Write the results to the CSV file
        csv_writer.writerow([num_random_samples, avg_velocity, std_dev])

# Plot the results
plt.figure(1)
plt.plot(sample_sizes, pce_avg_velocities, 'b-', linewidth=2)
plt.xlabel('Number of Random Samples')
plt.ylabel('Mean Velocity (m/s)')
plt.title('Mean Velocity vs. Number of Random Samples')
plt.grid(True)

plt.figure(2)
plt.plot(sample_sizes, pce_std_devs, 'r-', linewidth=2)
plt.xlabel('Number of Random Samples')
plt.ylabel('Standard Deviation of Velocity (m/s)')
plt.title('Standard Deviation of Velocity vs. Number of Random Samples')
plt.grid(True)

plt.show()
```

```
PCE coefficients: -5665.163709832621*q1**3+15.57300847153761*q0*q1**2+7184.42337
0636065*q0**2*q1-888.3674389939833*q0**3+605.9448976069962*q1**2-13.618935942079
386*q0*q1-214.32567903768657*q0**2-22.94086298272593*q1+0.40818645752662946*q0+0
.3241287090136312
```

```
Processing samples: 100%|| 100/100 [02:37<00:00,  1.58s/it]
```


Mean Velocity vs. Number of Random Samples
Mean Velocity (m/s)
0.0288
0.0286
0.0284
0.0282
0.0280
0.0278
0.0276
0
100
200
300
400
500
Number of Random Samples



Comparision Between The Monte Carlo Method and PCE(different order).

[27]:
```
import numpy as np
import matplotlib.pyplot as plt
import chaospy as cp
from tqdm import tqdm
import math
import csv
import seaborn as sns


def calculate_friction_factor(Re, pipe_roughness, pipe_diameter):
    if Re < 2000:
        f = 64 / Re
    else:
        f = 0.25 / (math.log10((pipe_roughness / (3.7 * pipe_diameter)) + (5.74 /
 ↪ Re**0.9)))**2
    return f
```

```python
def simulate_1D_pipe_flow(pipe_length, pipe_diameter, pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure, fluid_density, fluid_viscosity, total_time, num_points):
    area = np.pi * (pipe_diameter / 2)**2
    initial_velocity = inlet_flow_rate / area
    velocity = np.ones(num_points) * initial_velocity
    pressure = np.linspace(inlet_pressure, outlet_pressure, num_points)

    dx = pipe_length / (num_points - 1)
    max_velocity = 1.2 * initial_velocity
    C = 0.1
    time_step = C * min(dx / max_velocity, dx**2 / fluid_viscosity)
    num_time_steps = int(total_time / time_step)

    for t in range(num_time_steps):
        for i in range(1, num_points - 1):
            Re = fluid_density * abs(velocity[i]) * pipe_diameter / fluid_viscosity
            f = calculate_friction_factor(Re, pipe_roughness, pipe_diameter)

            a = fluid_viscosity * time_step / (2 * dx**2)
            b = f * velocity[i] * abs(velocity[i]) * time_step / (4 * pipe_diameter)
            c = fluid_density * dx / (2 * time_step)

            if velocity[i] >= 0:
                A = c - a + 0.5 * fluid_density * velocity[i] * dx
                B = c + a
                C = -a - b
                D = -a + b
            else:
                A = c - a
                B = c + a + 0.5 * fluid_density * velocity[i] * dx
                C = -a - b
                D = -a + b

            velocity[i] = (A * velocity[i - 1] + B * velocity[i] + C * velocity[i + 1]) / (A + B + C + D)

        pressure[0] = inlet_pressure
        pressure[-1] = outlet_pressure
        pressure[1:-1] = pressure[1:-1] - fluid_density * velocity[1:-1] * (velocity[2:] - velocity[:-2]) * dx

    return velocity, pressure

# Pipe parameters
```

```python
pipe_length = 10.0
pipe_roughness = 0.0001
inlet_flow_rate = 0.00002
inlet_pressure = 101325.0
outlet_pressure = 101325.0
fluid_density = 1000.0
total_time = 2.0
num_points = 100

def uncertainty_analysis(fluid_viscosity_distribution,␣
↪pipe_diameter_distribution, num_samples, pce_order):
    joint_distribution = cp.J(cp.LogNormal(mu=-6.907755, sigma=0.096809), cp.
↪Uniform(0.029, 0.031))

    samples = joint_distribution.sample(num_samples)
    avg_velocities = np.array([simulate_1D_pipe_flow(pipe_length, d,␣
↪pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure,␣
↪fluid_density, v, total_time, num_points)[0].mean() for v, d in samples.T])

    basis_polynomials = cp.expansion.stieltjes(pce_order, joint_distribution)
    pce_coeffs = cp.fit_regression(basis_polynomials, samples, avg_velocities)

    return pce_coeffs, basis_polynomials

def plot_pce_vs_mc(pce_order):
    pce_coeffs, basis_polynomials = uncertainty_analysis(cp.LogNormal(mu=-6.
↪907755, sigma=0.096809), cp.Uniform(0.029, 0.031), num_samples, pce_order)
    pce_expansion = cp.sum(pce_coeffs * basis_polynomials)

    # Calculate average velocities using PCE method
    avg_velocities_pce = pce_expansion(*samples_mc)

    # Plot the probability density functions
    sns.kdeplot(avg_velocities_pce, label=f'PCE Order {pce_order}', linewidth=2)

# Run the uncertainty analysis using 100 random samples
num_samples = 100

# Generate random samples
num_samples_mc = 10000
joint_distribution = cp.J(cp.LogNormal(mu=-6.907755, sigma=0.096809), cp.
↪Uniform(0.029, 0.031))
samples_mc = joint_distribution.sample(num_samples_mc)

# Calculate average velocities using Monte Carlo method
```

```python
avg_velocities_mc = np.array([simulate_1D_pipe_flow(pipe_length, d,␣
↪pipe_roughness, inlet_flow_rate, inlet_pressure, outlet_pressure,␣
↪fluid_density, v, total_time, num_points)[0].mean() for v, d in␣
↪tqdm(samples_mc.T, desc="Progress")])

# Plot the probability density functions
plt.figure()
sns.kdeplot(avg_velocities_mc, label='Monte Carlo', linewidth=2, color='black')

# Loop through pce_orders from 1 to 5
for pce_order in range(1, 5):
    plot_pce_vs_mc(pce_order)

plt.xlabel('Average Velocity (m/s)')
plt.ylabel('Probability Density')
plt.legend()
plt.show()
```

```
Progress: 100%|| 10000/10000 [00:37<00:00, 267.75it/s]
```