Simulating Random Variables with the Metropolis Algorithm

Given the following probability densities, we aim to simulate random variables that approximate each density using the Metropolis algorithm. By simulating these distributions, we can create histograms of the samples, allowing us to visually verify that the shapes of the histograms are consistent with the specified probability densities.

1. **Cauchy Distribution**:

$$\rho(x) = \frac{1}{1+(x-4)^2}, \quad x \in [1, 7]$$

2. **Normal Distribution**:

$$\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-4)^2}{2}}, \quad x \in [-10, 10]$$

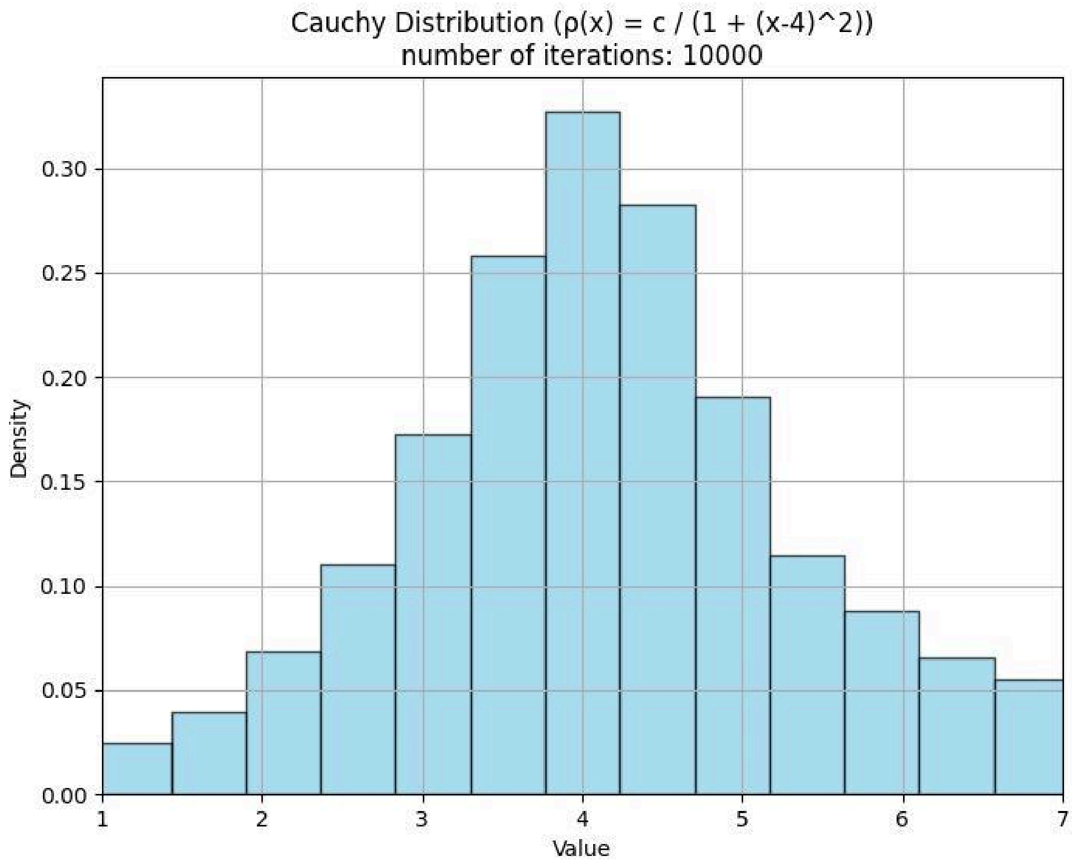
3. **Logarithmic Distribution**:

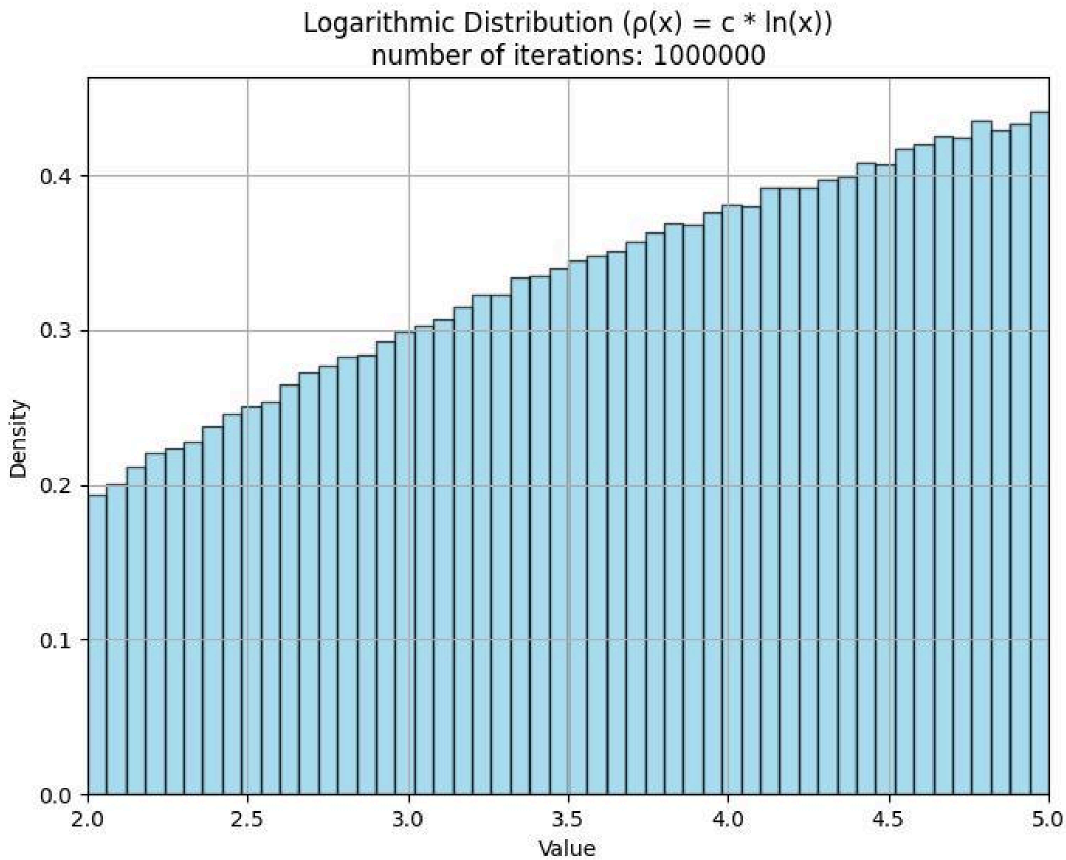
$$\rho(x) = \ln(x), \quad x \in [2, 5]$$

Code

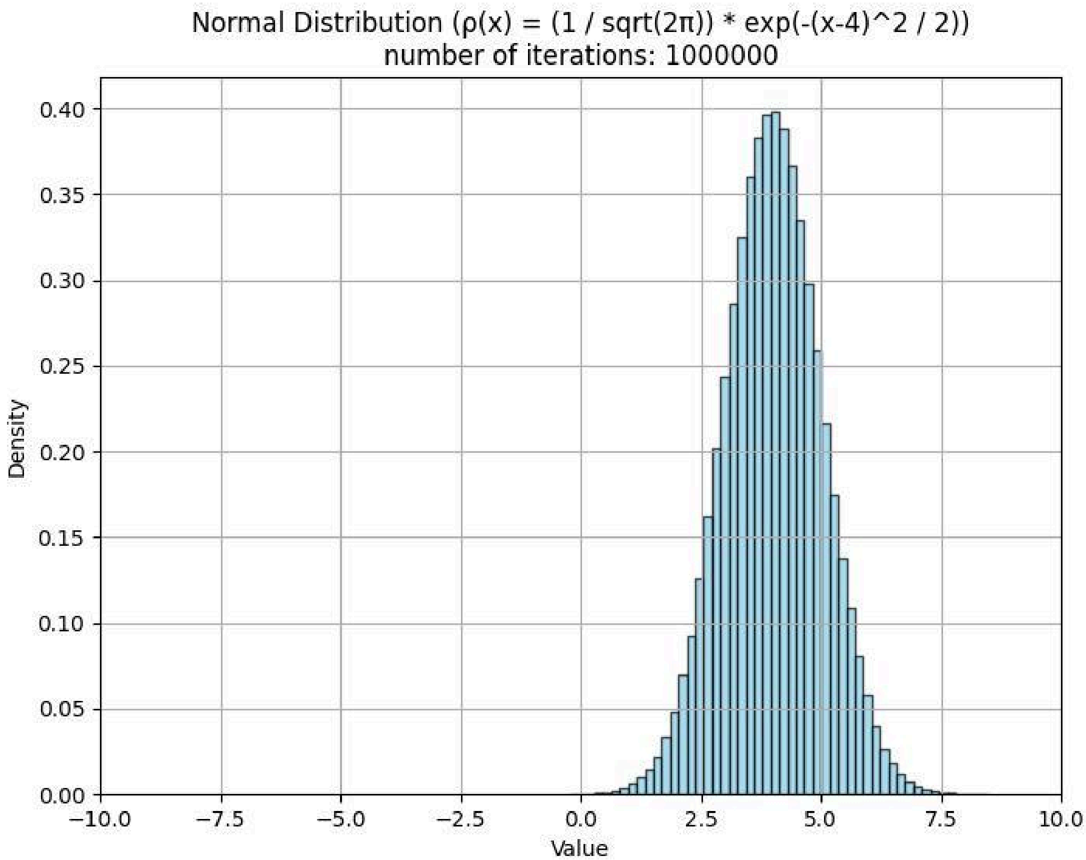
```python
import numpy as np
import matplotlib.pyplot as plt

# Metropolis algorithm for random variable simulation
def metropolis_algorithm(pdf, proposal_dist, proposal_pdf, start, iterations=10000):
    samples = []
    x = start
    for i in range(iterations):
        x_proposed = proposal_dist(x)
        acceptance_ratio = min(1, pdf(x_proposed) / pdf(x) * proposal_pdf(x, x_proposed) / proposal_pdf(x_proposed, x))
        if np.random.rand() < acceptance_ratio:
            x = x_proposed
        samples.append(x)
    return np.array(samples)


# 1. Cauchy distribution (ρ(x) = c / (1 + (x-4)^2))
def pdf_cauchy(x):
    return 1 / (1 + (x - 4)**2)

def proposal_cauchy(x):
    return np.random.normal(x, 0.5)  # Normal distribution as proposal

def proposal_pdf_cauchy(x, x_prime):
    return 1 / np.sqrt(2 * np.pi * 0.5**2) * np.exp(-(x_prime - x)**2 / (2 * 0.5**2))


# 2. Normal distribution (ρ(x) = (1 / sqrt(2π)) * exp(-(x-4)^2 / 2))
def pdf_normal(x):
    return np.exp(-(x - 4)**2 / 2) / np.sqrt(2 * np.pi)

def proposal_normal(x):
    return np.random.normal(x, 0.5)  # Normal distribution as proposal

def proposal_pdf_normal(x, x_prime):
    return 1 / np.sqrt(2 * np.pi * 0.5**2) * np.exp(-(x_prime - x)**2 / (2 * 0.5**2))


# 3. Logarithmic distribution (ρ(x) = c * ln(x))
def pdf_log(x):
    return np.log(x)

def proposal_log(x):
    return np.random.uniform(2, 5)  # Uniform distribution for log

def proposal_pdf_log(x, x_prime):
    return 1 / (5 - 2)  # Uniform PDF


num_of_iterations = 1000000
samples_cauchy = metropolis_algorithm(pdf_cauchy, proposal_cauchy, proposal_pdf_cauchy, 4, 10000)
samples_normal = metropolis_algorithm(pdf_normal, proposal_normal, proposal_pdf_normal, 4, num_of_iterations)
samples_log = metropolis_algorithm(pdf_log, proposal_log, proposal_pdf_log, 3, num_of_iterations)


def plot_histogram(samples, bins, density, title, xlim, filename):
    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=bins, density=density, alpha=0.7, color='skyblue', edgecolor='black')
    plt.title(title)
    plt.xlim(xlim)
    plt.xlabel("Value")
    plt.ylabel("Density")
    plt.grid(True)
    plt.savefig(filename, format='jpeg')
    plt.close()


plot_histogram(samples_cauchy, bins=50, density=True,
               title="Cauchy Distribution (ρ(x) = c / (1 + (x-4)^2))\nnumber of iterations: 10000",
               xlim=(1, 7), filename="cauchy_histogram.jpeg")

plot_histogram(samples_normal, bins=50, density=True,
               title=f"Normal Distribution (ρ(x) = (1 / sqrt(2π)) * exp(-(x-4)^2 / 2))\nnumber of iterations: {num_of_iterations}",
               xlim=(-10, 10), filename="normal_histogram.jpeg")

plot_histogram(samples_log, bins=50, density=True,
               title=f"Logarithmic Distribution (ρ(x) = c * ln(x))\nnumber of iterations: {num_of_iterations}",
               xlim=(2, 5), filename="log_histogram.jpeg")

print("Histograms saved as JPEG.")
```

Plots


Cauchy Distribution (ρ(x) = c / (1 + (x-4)^2))
number of iterations: 10000
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Value
1
2
3
4
5
6
7



Logarithmic Distribution (ρ(x) = c * ln(x))
number of iterations: 1000000
Density
0.0
0.1
0.2
0.3
0.4
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Value


Normal Distribution (ρ(x) = (1 / sqrt(2π)) * exp(-(x-4)^2 / 2))
number of iterations: 1000000
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Value