Computerized Simulation
Exercise No. 3

name : Seyed Mohammad Ghoreishy
teacher : Seyed Amirhossein Tabatabaei

Date: 1403.09.26

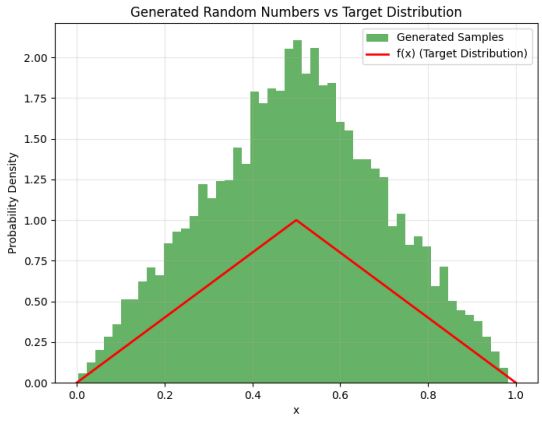
Contents

1 Exercise 1

Write a program using the **accept-reject method** to generate random numbers following a custom probability distribution defined by the piecewise function as follows:

$$f(x) = \begin{cases} 2x & 0 \leq x \leq 0.5 \\ 2(1-x) & 0.5 < x \leq 1. \end{cases}$$

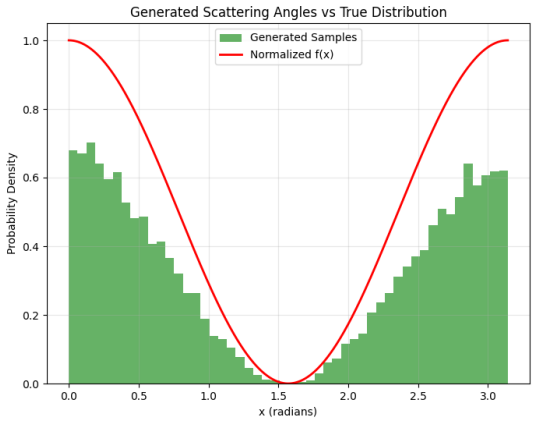
```python
import numpy as np
import matplotlib.pyplot as plt

def piecewise_function(x):
    return np.where((0 <= x) & (x <= 0.5), 2 * x,
                    np.where((0.5 < x) & (x <= 1), 2 * (1 - x), 0))

def accept_reject_sampling(target_func, domain, n_samples, max_val):
    samples = []
    x_min, x_max = domain
    while len(samples) < n_samples:
        x = np.random.uniform(x_min, x_max)
        u = np.random.uniform(0, 1)
        if u <= target_func(x) / max_val:
            samples.append(x)
    return np.array(samples)

def plot_results(samples, target_func, domain, bins=50):
    x_vals = np.linspace(domain[0], domain[1], 1000)
    f_vals = target_func(x_vals)
    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=bins, density=True, alpha=0.6, color='g', label='Generated Samples')
    plt.plot(x_vals, f_vals, 'r-', linewidth=2, label='f(x) (Target Distribution)')
    plt.xlabel('x')
    plt.ylabel('Probability Density')
    plt.title('Generated Random Numbers vs Target Distribution')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

n_samples = 10000
domain = (0, 1)
max_val = 1
np.random.seed(0)
random_numbers = accept_reject_sampling(piecewise_function, domain, n_samples, max_val)
plot_results(random_numbers, piecewise_function, domain)
```



2 Exercise 2

Simulate the distribution of angles at which particles scatter when the probability distribution of scattering angles is proportional to $\cos^2(x)$. Use the **accept-reject method** to generate the angles.

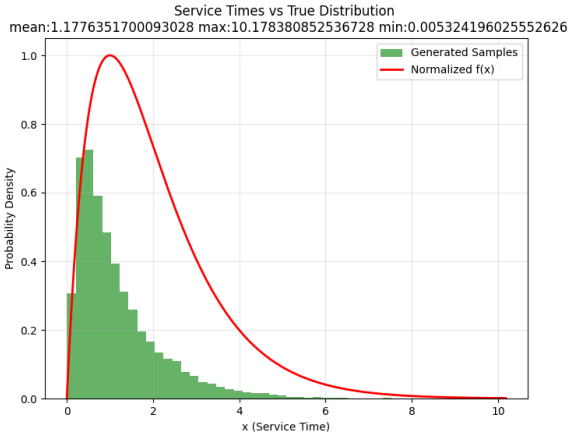
```python
import numpy as np
import matplotlib.pyplot as plt

def scattering_distribution(x):
    return np.cos(x)**2

def accept_reject_sampling(target_func, domain, n_samples, max_val):
    samples = []
    x_min, x_max = domain
    while len(samples) < n_samples:
        x = np.random.uniform(x_min, x_max)
        u = np.random.uniform(0, 1)
        if u <= target_func(x) / max_val:
            samples.append(x)
    return np.array(samples)

def plot_results(samples, target_func, domain, bins=50):
    x_vals = np.linspace(domain[0], domain[1], 1000)
    f_vals = target_func(x_vals)
    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=bins, density=True, alpha=0.6, color='g', label='Generated Samples')
    plt.plot(x_vals, f_vals / np.max(f_vals), 'r-', linewidth=2, label='Normalized f(x)')
    plt.xlabel('x (radians)')
    plt.ylabel('Probability Density')
    plt.title('Generated Scattering Angles vs True Distribution')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

n_samples = 10000
domain = (0, np.pi)
max_val = 1
np.random.seed(0)
scattering_angles = accept_reject_sampling(scattering_distribution, domain, n_samples, max_val)
plot_results(scattering_angles, scattering_distribution, domain)
```



3 Exercise 3

Simulate service times for customers in a queue. Assume service times follow a specific distribution (e.g., $f(x) = xe^{-x}$), and use the **accept-reject method** to generate the times. Analyze the average waiting time.

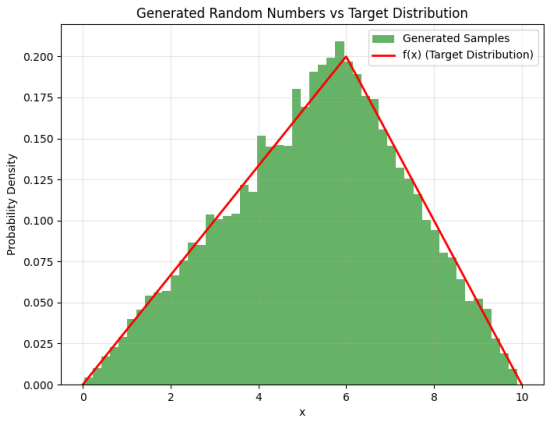
```python
import numpy as np
import matplotlib.pyplot as plt

def service_time_distribution(x):
    return x * np.exp(-x) if x > 0 else 0

def exponential_distribution(x):
    return np.exp(-x) if x > 0 else 0

def accept_reject_sampling(target_func, proposal_func, domain, n_samples, max_val):
    samples = []
    while len(samples) < n_samples:
        x = np.random.exponential(1)
        u = np.random.uniform(0, 1)
        if u <= target_func(x) / (max_val * proposal_func(x)):
            samples.append(x)
    return np.array(samples)

def plot_results(samples, target_func, bins=50):
    x_vals = np.linspace(0, max(samples), 1000)
    f_vals = np.array([target_func(x) for x in x_vals])
    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=bins, density=True, alpha=0.6, color='g', label='Generated Samples')
    plt.plot(x_vals, f_vals / f_vals.max(), 'r-', linewidth=2, label='Normalized f(x)')
    plt.xlabel('x (Service Time)')
    plt.ylabel('Probability Density')
    plt.title(f'Service Times vs True Distribution \n mean:{np.mean(samples)} max:{np.max(samples)}
min:{np.min(samples)}')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

n_samples = 10000
domain = (0, np.inf)
max_val = 1 / np.exp(1)
np.random.seed(0)
service_times = accept_reject_sampling(service_time_distribution, exponential_distribution, domain,
n_samples, max_val)
plot_results(service_times, service_time_distribution)
```



4 Exercise 4

Generate random numbers for a triangular distribution defined on $[a, b]$ with mode c.

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

def triangular_distribution_function(x, a, b, c):
    return np.where((a <= x) & (x < c), 2 * (x - a) / ((b - a) * (c - a)),
           np.where((c <= x) & (x <= b), 2 * (b - x) / ((b - a) * (b - c)), 0))

def accept_reject_sampling(target_func, domain, n_samples, max_val, *params):
    samples = []
    x_min, x_max = domain
    while len(samples) < n_samples:
        x = np.random.uniform(x_min, x_max)
        u = np.random.uniform(0, 1)
        if u <= target_func(x, *params) / max_val:
            samples.append(x)
    return np.array(samples)

def plot_results(samples, target_func, domain, params, bins=50):
    x_vals = np.linspace(domain[0], domain[1], 1000)
    f_vals = target_func(x_vals, *params)
    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=bins, density=True, alpha=0.6, color='g', label='Generated Samples')
    plt.plot(x_vals, f_vals, 'r-', linewidth=2, label='f(x) (Target Distribution)')
    plt.xlabel('x')
    plt.ylabel('Probability Density')
    plt.title('Generated Random Numbers vs Target Distribution')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

a, b, c = 0, 10, 6
n_samples = 10000
max_val = 2 / (b - a)
domain = (a, b)
np.random.seed(0)
random_numbers = accept_reject_sampling(triangular_distribution_function, domain, n_samples, max_val,
a, b, c)
plot_results(random_numbers, triangular_distribution_function, domain, (a, b, c))
```



5 Exercise 5

Implement a random number generator for the Rayleigh distribution with variance σ^2.

6 Exercise 6

Imagine a bank with multiple counters where customers arrive randomly. The arrival times of customers are independent and can be modeled using an exponential distribution. Similarly, the time taken to service a customer at a counter (service time) is also random and follows an exponential distribution.

Tasks:

- Simulate customer arrivals and service times over a period.
- Analyze key metrics such as:
  - Average waiting time,
  - Counter utilization (how busy they are),
  - Total time customers spend in the system.

7 Exercise 7

A logistics company operates a fleet of drones to deliver packages. Each package has a random destination within a defined area, and each drone has constraints like maximum payload and battery range. Packages arrive randomly over time, and drones are dispatched accordingly.

Inputs:

- Delivery area: A 2D space (e.g., a grid of 10×10 km).
- Drone characteristics: Speed, maximum payload, and battery range.
- Package data: Randomly generated delivery locations, weights, and arrival times.

Tasks:

1. Generate random package data.
2. Develop drone dispatch logic.
3. Simulate drone movements.

4. Analyze metrics such as:
    - Average delivery time,
    - Total distance traveled by all drones.