IEOR 4404 Simulation **Fall 2024**

Homework 4

Instructor: Henry Lam

Problem 1 Let G be a cumulative distribution function and suppose that for constants $a < b$, we wish to generate a random variable X with cumulative distribution function

$$F(x) = \frac{G(x) - G(a)}{G(b) - G(a)}, \quad a \le x \le b.$$

(a) If random variable Y has cumulative distribution function G, then given what information is F the conditional distribution of Y? Prove your claim.

(b) Show that an acceptance-rejection method to generate X in this case is to generate a random variable Y having cumulative distribution function G and then accept it if it lies between a and b.

Problem 2 Derive an acceptance-rejection method to generate (X, Y) with joint density function

$$f(x, y) = \frac{1}{K}\exp(-x^2 - y^2 + x\sin(xy)),$$

where K is a normalizing constant. Implement your method in a computer to generate 100 pairs of (X, Y), and plot the distributions of X and Y respectively.

Problem 3 Derive a procedure to sample uniformly from the region

$$A = \left\{(x, y) : x \in [-1, 1], |y| \le |x|^{-1/2}\right\}.$$

Implement your procedure in a computer to generate 100 pairs of (X, Y), and plot their positions on a two-dimensional plane to check that they indeed appear uniformly distributed over the region A.

Problem 4 Consider the discrete random variable X where

$$\mathbb{P}(X = j) = \left(\frac{1}{2}\right)^{j+1} + \left(\frac{1}{2}\right)\frac{2^{j-1}}{3^j}, j = 1, 2, \ldots$$

which appears in HW2 Problem 3. Instead of using the inverse transform method as there, now present a composition method to generate X. Then implement your procedure in a computer to generate 100 copies of X, and plot their distribution.

Problem 4 Present a method to generate a random variable X having cumulative distribution function

$$F(x) = \int_0^\infty x^y e^{-y} dy, 0 \le x \le 1.$$

Implement your procedure in a computer to generate 100 copies of X, and plot their distribution.

Alexandre
Duhamel
afd2153
MSOR

HW4 : Simulation

Problem 1:

a) If Y has cdf G, then for $a \leq x \leq b$:

$$P(a \leq Y \leq x \mid a \leq Y \leq b) = \frac{P(a \leq Y \leq x \cap a \leq Y \leq b)}{P(a \leq Y \leq b)} = \frac{P(a \leq Y \leq x)}{P(a \leq Y \leq b)} = \frac{G(x) - G(a)}{G(b) - G(a)} = F(x)$$

So $F(x)$ is the conditional distribution of Y, given that $Y \in [a,b]$.

b) The acceptance-rejection method here is:

1. Generate $Y \sim G(x)$
2. Accept Y if it is in $[a,b]$, otherwise repeat step 1.

This generates exactly X because, here, we are generating Y then accepting it only if it is in $[a,b]$, so we generated Y conditioned on $[a,b]$ (which is X : $P(a \leq Y \leq x \mid a \leq Y \leq b) = F(x)$)

Problem 2:

Let's take $g(x,y) = \frac{1}{2\pi} e^{\frac{-x^2-y^2}{2}}$ and define $h(x,y) = e^{\frac{-x^2-y^2}{2} + x\sin(xy)}$ so that $f(x,y) = \frac{h(x,y)}{K}$

We find:

$$\tilde{C} = \max \frac{h(x,y)}{g(x,y)} = 2\pi \max\left(e^{-x^2-y^2+x\sin(xy)}\right) = 2\pi$$

for $x = y = 0$

The ARM is:

$$\begin{cases} \text{Generate } (X,Y) \sim g(x,y) \\ \text{Generate } U \sim \text{Unif}(0,1) \\ \text{If } U \leq \dfrac{h(X,Y)}{\tilde{C} g(X,Y)} = \dfrac{e^{-X^2-Y^2+X\sin(XY)}}{\frac{2\pi}{2\pi} e^{\frac{-X^2-Y^2}{2}}} = e^{\frac{-X^2-Y^2}{2} + X\sin(XY)} \end{cases}$$

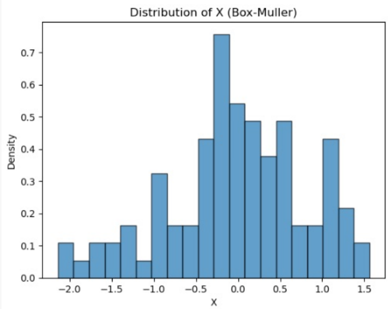
$$\begin{cases} \text{Generate } U_1 \sim \text{Unif}(0,1) \text{ and } U_2 \sim \text{Unif}(0,1) \\ \begin{cases} X = \sqrt{-2\log(U_1)}\cos(2\pi U_2) \\ Y = \sqrt{-2\log(U_2)}\cos(2\pi U_1) \end{cases} \end{cases}$$

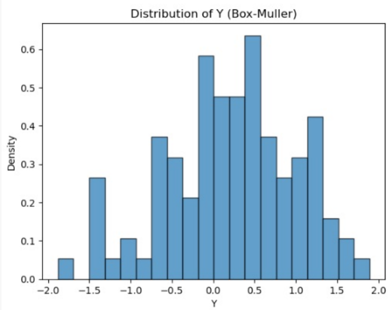
comes from polar coordinates

so $X, Y \sim \mathcal{N}(0,1)$

$$g_x(x,y) = \int_{\mathbb{R}} g(x,y)\,dx = \frac{1}{2\pi}\int_{\mathbb{R}} e^{\frac{-x^2-y^2}{2}}\,dx = \frac{e^{-\frac{y^2}{2}}}{2\pi}\int_{\mathbb{R}} e^{-\frac{x^2}{2}}\,dx = \frac{e^{-\frac{y^2}{2}}}{2\pi}\sqrt{2\pi} = \frac{e^{-y^2/2}}{\sqrt{2\pi}} \sim \mathcal{N}(0,1)$$

```python
def h(x, y):
    return np.exp(-x**2 - y**2 + x * np.sin(x * y))

def g(x, y):
    return (1 / (2 * np.pi)) * np.exp(-0.5 * (x**2 + y**2))

def box_muller():
    U1 = np.random.uniform(0, 1)
    U2 = np.random.uniform(0, 1)
    Z1 = np.sqrt(-2 * np.log(U1)) * np.cos(2 * np.pi * U2)
    Z2 = np.sqrt(-2 * np.log(U1)) * np.sin(2 * np.pi * U2)
    return Z1, Z2

def generate_samples(num_samples):
    samples = []
    C = 2 * np.pi

    while len(samples) < num_samples:
        X, Y = box_muller()

        U = np.random.uniform(0, 1)

        if U <= h(X, Y) / (C * g(X, Y)):
            samples.append((X, Y))

    return np.array(samples)

# Generate 100 pairs (X, Y)
num_samples = 100
samples = generate_samples(num_samples)

# Extract X and Y values
X_samples = samples[:, 0]
Y_samples = samples[:, 1]

# Plot the histogram of X samples
plt.hist(X_samples, bins=20, density=True, edgecolor='black', alpha=0.7)
plt.title('Distribution of X (Box-Muller)')
plt.xlabel('X')
plt.ylabel('Density')
plt.show()

# Plot the histogram of Y samples
plt.hist(Y_samples, bins=20, density=True, edgecolor='black', alpha=0.7)
plt.title('Distribution of Y (Box-Muller)')
plt.xlabel('Y')
plt.ylabel('Density')
plt.show()
```





Problem 3:

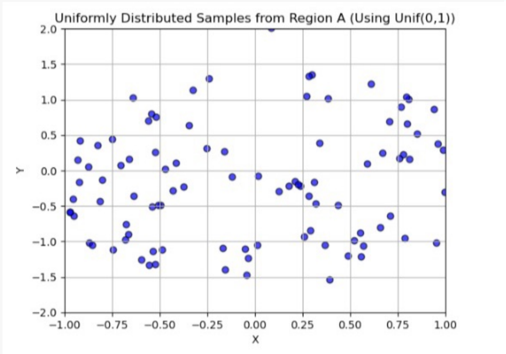
- We have $A = \{(x,y),\ x \in [-1,1],\ |y| \leq |x|^{-1/2}\}$

- We first sample $X \sim \text{Unif}(-1,1) = 2\,\text{Unif}(0,1) - 1$ $\qquad \text{Unif}(a,b) = (b-a)\,\text{Unif}(0,1) + a$

  Then sample $Y \sim \text{Unif}(-f(x), f(x))$ where $f(x) = |x|^{-1/2}$

  $= 2f(x)\,\text{Unif}(0,1) - f(x)$

```python
def sample_from_A(num_samples):
    samples = []

    for _ in range(num_samples):
        U1 = np.random.uniform(0, 1)
        X = -1 + 2 * U1

        if X == 0:
            Y = 0  # Avoid division by zero if X = 0...
        else:
            bound = np.abs(X)**(-1/2)
            U2 = np.random.uniform(0, 1)
            Y = -bound + 2 * bound * U2

        samples.append((X, Y))

    return np.array(samples)

# Generate 100 samples from the region A
num_samples = 100
samples = sample_from_A(num_samples)

# Extract X and Y values for plotting
X_samples = samples[:, 0]
Y_samples = samples[:, 1]

# Plot the sampled (X, Y) pairs on a 2D plane
plt.scatter(X_samples, Y_samples, color='blue', alpha=0.7, edgecolor='black')
plt.title('Uniformly Distributed Samples from Region A (Using Unif(0,1))')
plt.xlabel('X')
plt.ylabel('Y')
plt.xlim(-1, 1)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```



Problem 4:

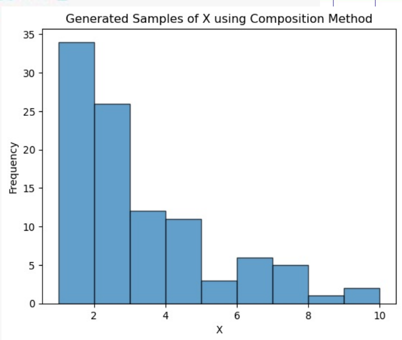
- We can represent $\mathbb{P}(X=j) =$ something + something else

$$= \frac{1}{2}\mathbb{P}_1(X=j) + \frac{1}{2}\mathbb{P}_2(X=j) \quad \text{where } \begin{cases} \mathbb{P}_1(X=j) = \left(\frac{1}{2}\right)^j \\ \mathbb{P}_2(X=j) = \frac{2^{j-1}}{3^j} \end{cases}$$

taking out the constants

- We can see it as a Bernoulli RV Y, where $Y = \begin{cases} 0 \text{ wp } \frac{1}{2}, \\ 1 \text{ wp } \frac{1}{2}, \end{cases}$ generate X using $\begin{cases} \mathbb{P}_1 \\ \mathbb{P}_2 \end{cases}$

```python
def generate_X(num_samples):
    samples = []

    for _ in range(num_samples):
        # Step 1: Generate a Bernoulli random variable with probabilit
        bernoulli = np.random.uniform(0, 1)

        if bernoulli <= 0.5:
            # Step 2: Use P_1 component (geometric-like distribution w
            X = 1
            while np.random.uniform(0, 1) > 0.5:
                X += 1
        else:
            # Step 2: Use P_2 component (with P(X=j) = 2^(j-1) / 3^j)
            U = np.random.uniform(0, 1)
            X = 1
            P = 1/3  # Initial probability for X = 1
            while U > P:
                X += 1
                P += (2 ** (X - 1)) / (3 ** X)

        samples.append(X)

    return np.array(samples)

# Generate 100 samples of Gib
num_samples = 100
samples_X = generate_X(num_samples)

# Plot the distribution of the generated samples
plt.hist(samples_X, bins=range(1, max(samples_X)+2), edgecolor='k', al
plt.title('Generated Samples of X using Composition Method')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```



This was my original approach to get 2 geo RV
but taking a larger n doesn't match the other result → I would like to understand/discuss why this doesn't work

Problem 4:

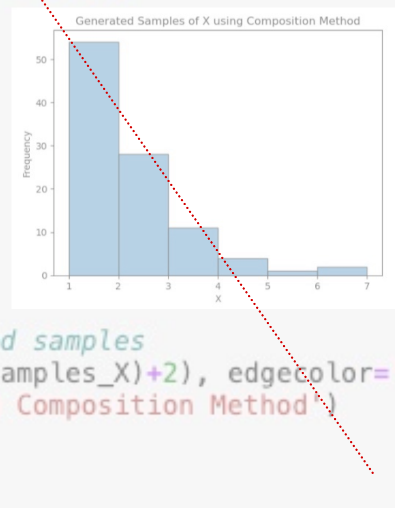
- We can represent $\mathbb{P}(X=j)$ = something + something else

$$= \underbrace{\frac{1}{2}}\mathbb{P}_1(X=j) + \underbrace{\frac{1}{4}}\mathbb{P}_2(X=j) \quad \text{where } \begin{cases} \mathbb{P}_1(X=j) = \left(\frac{1}{2}\right)^j \\ \mathbb{P}_2(X=j) = \left(\frac{2}{3}\right)^j \end{cases}$$

taking out the constants

- We can see it as a Bernoulli RV Y, where $Y = \begin{cases} 0 \text{ wp } \frac{2}{3}, \\ 1 \text{ wp } \frac{1}{3}, \end{cases}$ generate X using $\begin{cases} \mathbb{P}_1 \\ \mathbb{P}_2 \end{cases}$

```python
def generate_X(num_samples):
    samples = []

    for _ in range(num_samples):
        # Step 1: Generate a Bernoulli random variable with probabi
        bernoulli = np.random.uniform(0, 1)

        if bernoulli <= 2/3:
            # Step 2: Use P_1 component (geometric-like distributio
            X = 1
            while np.random.uniform(0, 1) > 0.5:
                X += 1
        else:
            # Step 2: Use P_2 component (geometric-like distributio
            X = 1
            while np.random.uniform(0, 1) > 2/3:
                X += 1

        samples.append(X)

    return np.array(samples)

# Generate 100 samples of X
num_samples = 100
samples_X = generate_X(num_samples)

# Plot the distribution of the generated samples
plt.hist(samples_X, bins=range(1, max(samples_X)+2), edgecolor='k',
plt.title('Generated Samples of X using Composition Method')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```



Problem 5:

- Let's use the continuous mixture: $F(x) = \mathbb{P}(X \le x) = \int_0^{+\infty} \mathbb{P}(Y_t \le x)\, f_T(t)\, dt$

We have: $F(x) = \int_0^{+\infty} x^y e^{-y}\, dy$ where $\begin{cases} \mathbb{P}(Y_t \le x) = x^t \\ f_T(t) = e^{-t} \text{ so } T \sim Exp(1) \end{cases}$

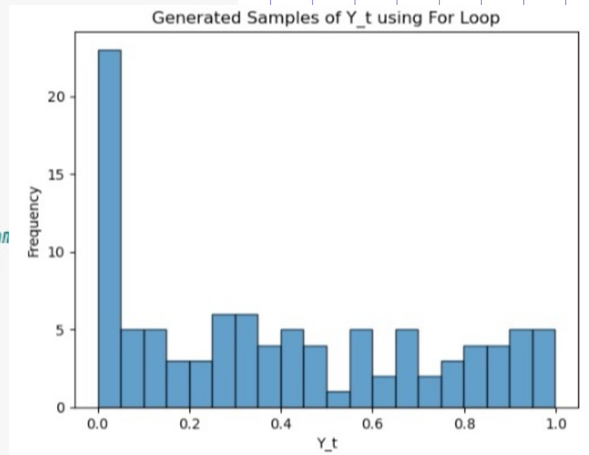
To generate T: $F(t) = \int_0^x f_T(t)\, dt = \int_0^x e^{-t}\, dt = \left[-e^{-t}\right]_0^x = 1 - e^{-x}$

$F(T) = \mathcal{U} \to 1 - e^{-T} = \mathcal{U} \to T = -\ln(1-\mathcal{U}) \Leftrightarrow T = -\ln(\mathcal{U})$ where $\mathcal{U} \sim Unif(0,1)$

To generate Y_t: $F(Y_t) = \mathcal{U} \to x = \mathcal{U}^{1/t}$ where $\mathcal{U} \sim Unif(0,1)$

Algorithm:
1. Generate $T \sim Exp(1)$ as $T = -\ln(\mathcal{U})$ where $\mathcal{U} \sim Unif(0,1)$
2. Given $T = t$, generate Y_t as $Y_t = V^{1/T}$ where $V \sim Unif(0,1)$

```python
# Generate T ~ Exp(1)
def generate_T():
    U = np.random.uniform(0, 1)
    return -np.log(U)

# Generate P(Y_t <= x) = x^T
def generate_Y_t():
    T = generate_T()
    U = np.random.uniform(0, 1)
    Y_t = U**(1/T)
    return Y_t

# Function to generate multiple Y_t sam
def generate_Y_t_samples(num_samples):
    samples = []
    for _ in range(num_samples):
        Y_t = generate_Y_t()
        samples.append(Y_t)
    return np.array(samples)

# Set number of samples
num_samples = 100
Y_t_samples = generate_Y_t_samples(num_samples)

# Plot the histogram of the generated Y_t values
plt.hist(Y_t_samples, bins=20, edgecolor='k', alpha=0.7)
plt.title('Generated Samples of Y_t using For Loop')
plt.xlabel('Y_t')
plt.ylabel('Frequency')
plt.show()
```



Just research as I didn't see $P(X \leq x) = \int_0^{\infty} P(Y_t \leq x)\, f_T(t)dt$ at first.

I thought of approximation F(x) by computing many of its values, then inverse it by interpolation. I'm not sure if it works but the graphs look similar (completely the same for $n = 10000$) :)

Problem 5:

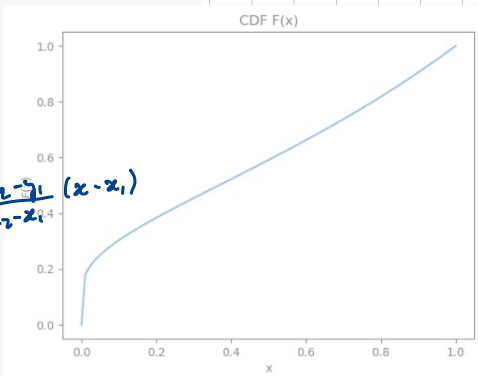
We have $F(x) = \int_0^{+\infty} x^y e^{-y} dy = \int_0^{+\infty} x^y \underbrace{(1 e^{-1y})}_{\exp(1)} dy = \int_0^{+\infty} x^y g(y)\, dy$

We want to inverse F so that we can have x.

We will start by computing numerically F(x) for various values of $x \in [0,1]$

We then use an interpolation to solve: $F(x) = U \Rightarrow x = F^{-1}(U)$ where $U \sim Unif(0,1)$

```
def F(x):
    result, _ = quad(lambda y: x**y * np.exp(-y), 0, np.inf)
    return result

# Precompute F(x) for a range of x values (0 to 1)
x_vals = np.linspace(0, 1, 100)
F_vals = np.array([F(x) for x in x_vals])

# Plot F(x) to visualize its shape
plt.plot(x_vals, F_vals)
plt.title('CDF F(x)')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.show()
```



can be done by hand if linear as: $y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$

```
# NumPy for linear interpolation of F(x)
def numpy_interpolate(F_vals, x_vals, U):
    return np.interp(U, F_vals, x_vals)

# Generate X using inverse transform sampling
def generate_X(num_samples):
    X_samples = []

    for _ in range(num_samples):
        # Generate U ~ Unif(0, 1)
        U = np.random.uniform(0, 1)

        # Find X using the interpolated CDF
        X = numpy_interpolate(F_vals, x_vals, U)
        X_samples.append(X)

    return np.array(X_samples)

# Generate 100 samples of X
num_samples = 100
samples_X = generate_X(num_samples)

# Plot the distribution of the generated samples
plt.hist(samples_X, bins=20, edgecolor='k', alpha=0.7)
plt.title('Generated Samples of X using NumPy Interpolation')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```