IEOR 4404 Simulation **Fall 2024**

Homework 3

Instructor: Henry Lam

Problem 1 Suppose that the loss L, due to the arrival of certain type of insurance claims, is distributed according to a Pareto distribution. Specifically, it has been observed that the distribution of L is satisfactorily modeled by the cumulative distribution function,

$$P(L \leq x) = \begin{cases} 1 - 1/x^3, & \text{if } x > 1, \\ 0, & \text{otherwise.} \end{cases}$$

Given access only to a procedure that generates Unif(0,1), write an algorithm to generate L. Implement the algorithm on a computer to simulate 100 copies of L and plot their distribution.

Problem 2 We consider generating a random variable having density function

$$f(x) = \frac{e^x}{e-1}, 0 < x < 1.$$

(a) Write the inverse transform method for generating this random variable.
(b) Write the acceptance-rejection method, with a proposal density Unif $[0,1]$.
(c) Which method in part (a) or (b) do you prefer? Explain you answer.
(d) Implement the algorithms in both parts (a) and (b) on a computer. Using each algorithm, simulate 100 copies of L and plot their distribution.

Problem 3 A Gamma distribution with parameters $(n,1)$ (denoted by $Gamma(n,1)$) has density function

$$f(x) = \begin{cases} e^{-x}\frac{x^{n-1}}{(n-1)!}, & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

(a) Derive the acceptance-rejection method to generate a random variable X with distribution $Gamma(n,1)$, by using an exponential density function with parameter $1/2$, i.e.,

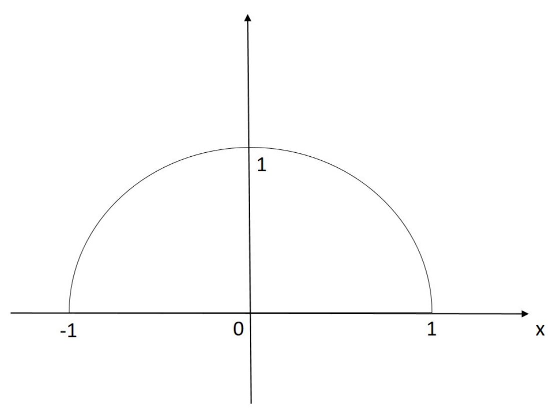
$$g(x) = \frac{1}{2}e^{-x/2}, \quad x \geq 0.$$

Assume you have access only to a procedure that can generate Unif(0,1).
(b) Do you think using an exponential density with parameter 1 as the proposal would work in part (a)? Explain your answer.

Problem 4 The random variable X has a *semi-circular* distribution, with the probability density function (p.d.f) $f(x)$ given by

$$f(x) = \begin{cases} \frac{2}{\pi}\sqrt{1-x^2} & \text{if } -1 \leq x \leq 1; \\ 0 & \text{otherwise.} \end{cases}$$

A plot of the p.d.f is given below.
(a) Derive the acceptance-rejection method to generate the random variable X, by using Unif(-1,1) as the proposal. Assume you have access only to a procedure that can generate Unif(0,1).
(b) What is the expected runtime of the method, in terms of the number of acceptance-rejection trials needed to get one output?


1
-1
0
1
x

Alexandre
Duhamel
afd2153
MSOR

HW3 : Simulation

Problem 1:

$$\mathbb{P}(L \leq x) = \begin{cases} 1 - 1/x^3, & \text{if } x > 1 \\ 0 & \text{otherwise} \end{cases}$$

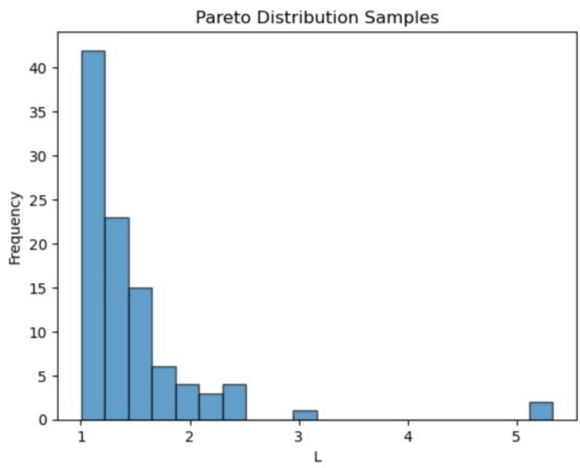
$\mathbb{P}(L \leq x) \in [0,1]$ so $\mathbb{P}(L \leq x) = U$ where $U = \text{Unif}(0,1)$

$$\Rightarrow 1 - 1/x^3 = U$$

$$\Rightarrow x = \sqrt[3]{\frac{1}{1-U}}$$

Thus if we generate $U \sim \text{Unif}(0,1)$, we'll have L by computing $L = \sqrt[3]{\frac{1}{1-U}}$

```python
# Inverse CDF for Pareto distribution
def inverse_pb1(num_samples):
    samples = []

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

        # Compute the L we established
        L = 1 / (1 - U)**(1/3)

        samples.append(L)
    return samples

# Generate 100 samples
num_samples = 100
pareto_samples = inverse_pb1(num_samples)

# Plot the distribution
plt.hist(pareto_samples, bins=20, edgecolor='k', alpha=0.7)
plt.title('Pareto Distribution Samples')
plt.xlabel('L')
plt.ylabel('Frequency')
plt.show()
```



Problem 2:

a) We have $f(x) = \frac{e^x}{e^1 - 1}$ for $0 < x < 1$

So $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u) \mathbb{1}_{[0,1]}(u)\,du = \int_0^x \frac{e^u}{e-1}\,du = \frac{1}{e-1}\left[e^u\right]_0^x = \frac{e^x - 1}{e-1}$

Inverse transform method with $U \sim Unif(0,1)$:

$F(x) = U \Rightarrow \frac{e^x - 1}{e-1} = U$

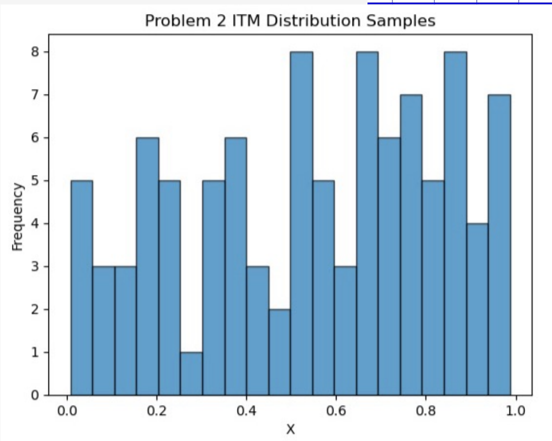
$\Rightarrow e^x - 1 = U(e-1)$

$\Rightarrow e^x = U(e-1)+1$

$\Rightarrow x = \ln\left(U(e-1)+1\right)$

Thus : $X = \ln(U(e-1)+1)$

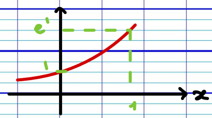
```python
def inverse_pb2(num_samples):
    samples = []

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

        # Compute the X we established
        X = np.log(U*(np.exp(1)-1)+1)

        samples.append(X)
    return samples

# Generate 100 samples
num_samples = 100
pareto_samples = inverse_pb2(num_samples)

# Plot the distribution
plt.hist(pareto_samples, bins=20, edgecolor='k', alpha=0.7)
plt.title('Problem 2 ITM Distribution Samples')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```



b) We are searching for g so that : $\frac{f(x)}{g(x)} \leq C$

let's try $g(x) = 1$

$$\frac{f(x)}{g(x)} = \frac{\frac{e^x}{e-1}}{1} = \frac{e^x}{e-1} \leq \frac{e}{e-1} = C$$

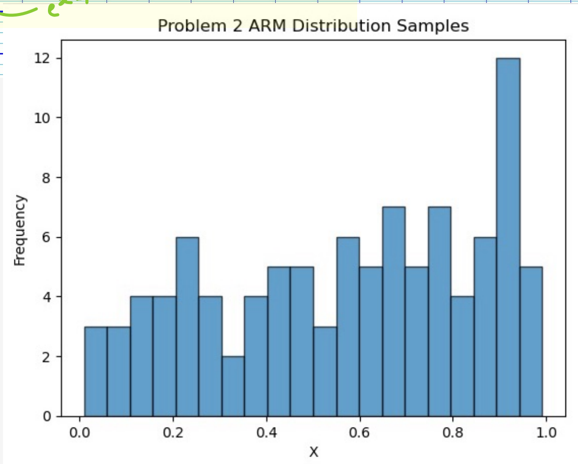


So generate $Y \sim g(y) = \mathrm{Unif}(0,1)$ $\quad G(x) = \int_{-\infty}^{x} g(y)\,dy = \int_0^x 1\,dy = x$ so $Y = \mathcal{U}$

generate $\mathcal{U} \sim \mathrm{Unif}(0,1)$

if $\mathcal{U} \leq \frac{f(Y)}{Cg(Y)}$ then $X = Y$, otherwise try again

return X $\quad e^{x-1}$




```python
def f(x) :
    return np.exp(x)/(np.exp(1) - 1)

def g(x) :
    return 1

def rejection_pb2(num_samples):
    samples = []

    # Set C constant
    C = np.exp(1)/(np.exp(1) - 1)

    # Loop for samples
    while len(samples) < num_samples:

        # Generate Y ~ g(y)
        Y = np.random.uniform()

        # Generate U ~ Unif(0,1)
        U = np.random.uniform()

        # Accept Y if U <= f(Y) / (C * g(Y))
        if U <= f(Y) / (C * g(Y)):  # Since g(Y) = 1, we just use f(Y) / C but makes the code reusable for future questions...
            samples.append(Y)  # Append the accepted sample

    return samples

# Generate 100 samples
num_samples = 100
distrib_samples = inverse_pb2(num_samples)

# Plot the distribution
plt.hist(distrib_samples, bins=20, edgecolor='k', alpha=0.7)
plt.title('Problem 2 ARM Distribution Samples')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```

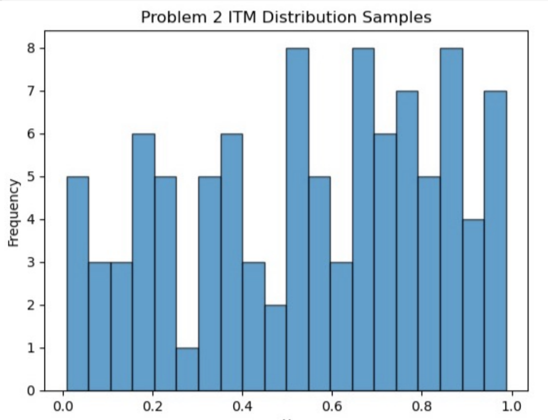
c) I mainly prefer the ITM as it is easier to establish and implement (simple function inversion). On top of that, it is way more efficient as it is direct computation whereas the ARM may involve several rejections before a sample is accepted. Moreover, if we want to improve the ARM algorithm, we have to find a small C to make it faster, not just any C... (if g is close to f, then C is small, so p is large and the expected number of trials is small).

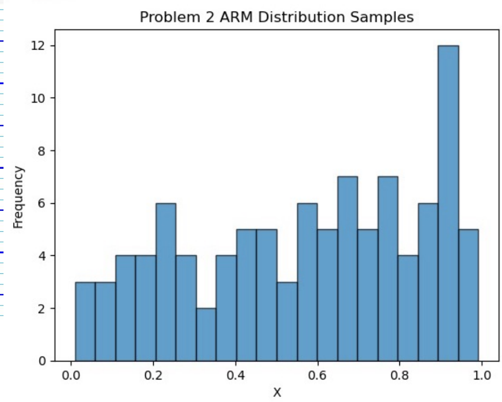
$C = 1/p_{accept}$

d)

```python
def inverse_pb2(num_samples):
    samples = []

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

        # Compute the X we established
        X = np.log(U*(np.exp(1)-1)+1)

        samples.append(X)
    return samples

# Generate 100 samples
num_samples = 100
pareto_samples = inverse_pb2(num_samples)

# Plot the distribution
plt.hist(pareto_samples, bins=20, edgecolor='k'
plt.title('Problem 2 ITM Distribution Samples')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```

```python
def f(x) :
    return np.exp(x)/(np.exp(1) - 1)

def g(x) :
    return 1

def rejection_pb2(num_samples):
    samples = []

    # Set C constant
    C = np.exp(1)/(np.exp(1) - 1)

    # Loop for samples
    while len(samples) < num_samples:

        # Generate Y ~ g(y)
        Y = np.random.uniform()

        # Generate U ~ Unif(0,1)
        U = np.random.uniform()

        # Accept Y if U <= f(Y) / (C * g(Y))
        if U <= f(Y) / (C * g(Y)):  # Since g(Y) = 1, we just
            samples.append(Y)  # Append the accepted sample

    return samples

# Generate 100 samples
num_samples = 100
distrib_samples = inverse_pb2(num_samples)

# Plot the distribution
plt.hist(distrib_samples, bins=20, edgecolor='k', alpha=0.7)
plt.title('Problem 2 ARM Distribution Samples')
plt.xlabel('X')
plt.ylabel('Frequency')
plt.show()
```





Problem 3:

a) We have $f(x) = \begin{cases} e^{-x} \dfrac{x^{m-1}}{(m-1)!} & \text{if } x \geq 0 \\ 0 \text{ otherwise} \end{cases}$

Let's proceed to find the C of the ARM, for $x \geq 0$:

$$\frac{f(x)}{g(x)} = \frac{e^{-x} \frac{x^{m-1}}{(m-1)!}}{\frac{1}{2} e^{-x/2}} = 2 \frac{x^{m-1}}{(m-1)!} e^{-\frac{x}{2}}$$

$$C = \max_x \left\{ 2 \frac{x^{m-1}}{(m-1)!} e^{-\frac{x}{2}} \right\} = \frac{2}{(m-1)!} \max \left\{ x^{m-1} e^{-x/2} \right\} = \frac{2}{(m-1)!} \left(2(m-1)\right)^{m-1} e^{-(m-1)} = C$$

derive and $= 0$: $(m-1) x^{m-2} e^{-x/2} = \frac{x^{m-1}}{2} e^{-x/2}$

$\Rightarrow 2(m-1) x^{m-2} = x^{m-1}$

$\Rightarrow 2(m-1) = x^{m-1-m+2}$

$\Rightarrow x = 2(m-1)$

So the algorithm is:

Generate $Y \sim g(y)$

Generate $U \sim \text{Uniform}(0,1)$

if $U \leq \dfrac{f(Y)}{Cg(Y)}$ then $X = Y$, otherwise try again

return X

$G(y) = \int_0^y \frac{1}{2} e^{-x/2} dx = \frac{1}{2} \left[-2 e^{-x/2} \right]_0^y = 1 - e^{-y/2}$

$G(y) = \text{Unif} \Rightarrow -\text{Unif} + 1 = e^{-y/2} \Rightarrow y = -2\ln(1 - \text{Unif})$

Thus $Y = -2\ln(\text{Unif})$ 1- U and U have the same distribution

b) The pdf of $\text{Exp}(1)$ is: $e(x) = e^{-x}$ for $x \geq 0$

Note that in a) we used $\text{Exp}(1/2)$

$$\frac{f(x)}{e(x)} = \frac{e^{-x} \frac{x^{m-1}}{(m-1)!}}{e^{-x}} = \frac{x^{m-1}}{(m-1)!}$$

$$C = \max_x \left\{ \frac{x^{m-1}}{(m-1)!} \right\} = \frac{1}{(m-1)!} \max \left\{ x^{m-1} \right\} = +\infty$$

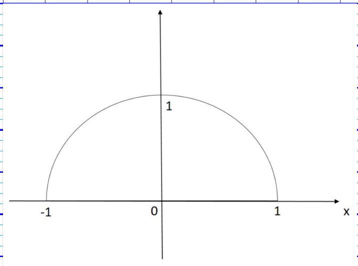
Using the exponential distribution with a rate of 1 will not work for the ARM because $\frac{f(x)}{e(x)}$ increases indefinitely as $x \to +\infty$, thus there is no constant C that satisfies $\frac{f(x)}{e(x)} \leq C$. This violates the requirement of the ARM which needs a finite upper bound

$$G(y) = \int_{-1}^{y} \frac{1}{2}\,dx = \frac{y+1}{2}$$

because $G(y) = \mathcal{U} \Rightarrow Y = 2\mathcal{U} - 1 = \text{Unif}(-1,1)$

Problem 4:

a)



Let's take $g(x) = \frac{1}{2}$ for $-1 \leq x \leq 1$

So:

$$C = \max\left\{\frac{f(x)}{g(x)}\right\} = \max\left\{\frac{\sqrt{1-x^2}}{\frac{\pi}{4}}\right\} = \frac{4}{\pi}\max\sqrt{1-x^2}$$

$$= \frac{4}{\pi}$$

$$\frac{-2x}{2\sqrt{1-x^2}} = 0$$

ie $x = 0$ works

So generate $Y \sim g(y) = \text{Unif}(-1,1)$ see green top of page

generate $\mathcal{U} \sim \text{Unif}(0,1)$

if $\mathcal{U} \leq \frac{f(Y)}{Cg(Y)} = \frac{4/\pi\sqrt{1-Y^2}}{4/\pi} = \sqrt{1-Y^2}$ then $X = Y$, otherwise try again

return X

But if we only have access to $\text{Unif}(0,1)$, the ARM becomes:

generate $Y \sim g(y) = 2 \times \text{Unif}(0,1) - 1$

generate $\mathcal{U} \sim \text{Unif}(0,1)$

if $\mathcal{U} \leq \frac{f(Y)}{Cg(Y)}$ then $X = Y$, otherwise try again

return X

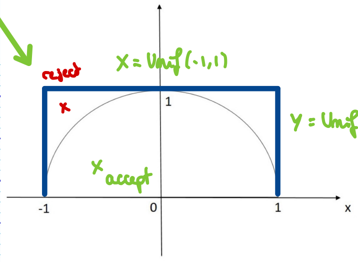
b) In class, we established that $\mathbb{P}(\text{accept}) = \frac{1}{C} = \frac{\pi}{4}$

Which is the probability of accepting a sample in one trial.

So trying until a success follows a Geometric variable of $p = \frac{\pi}{4}$

Thus $E[\text{trials}] = \frac{1}{p} = \frac{4}{\pi} \approx 1.2732$

So, the expected runtime of the ARM in terms of acceptance-rejection trials needed to get one output is $\frac{4}{\pi}$

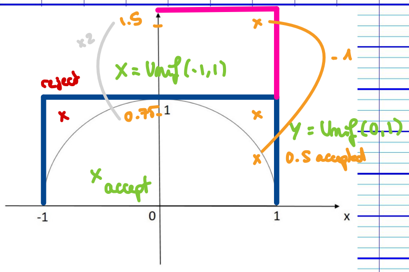


One easier way to look at it would be with the Monte-Carlo methode.
The black curve is $f(x)$ so the area under is $F(x) = \mathbb{P}(X \leq x)$ pdf
And holds the total probability of X

So we can generate points in the "box" that bounds the curve to generate the points

$X \sim$ Unif $(-1,1)$
$Y \sim$ Unif $(0,1)$ } we accept points in the semi-circle such that $Y \leq \sqrt{1-x^2}$

We successfully generated an X!



by symetry along the y axis, changing Unif$(-1,1)$ to 2Unif$(0,1)-1$ would imply choosing in the right part of the graph and making its value twice as big before cutting out 1 from it and see if it still lands within the boundaries of f (we're basically doing a vertical unif$(-1,1)$)