ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

Programming Assignment

ECEN-303-200

Andrew Bond
UIN: 228003829

December 5, 2020

Part I

1. Write a code using the programming language of your interest (preferably Python, C++, or MATLAB) that takes a positive scalar λ as input, and returns **one** sample of an exponential random variable X with parameter λ (i.e., $X \sim \text{Exp}(\lambda)$). You can only use the RAND function (or its equivalent) that generates a random number in the interval [0,1].

```python
# Generate one sample of an exponential random variable with
# parameter y >= 0.
def sample(y):
    if y < 0:
        raise ValueError(
            "sample: parameter of an exponential distribution must be positive"
        )

    # Returns a sample of R~Unif[0,1].
    r = random.random()

    # Generate sample using inverse of exponential CDF.
    return -1 / y * math.log(1 - r)
```

2. Write a code that takes a positive integer n and a postive scalar λ as input, and generates n independent samples of $X \sim \text{Exp}(\lambda)$. You can use the function you have defined for problem 1 as part of the code for this problem.

```python
# Generate a list of n samples of an exponential random
# variable with parameter y >= 0.
def n_samples(y, n):
    return np.array([sample(y) for _ in range(n)])
```

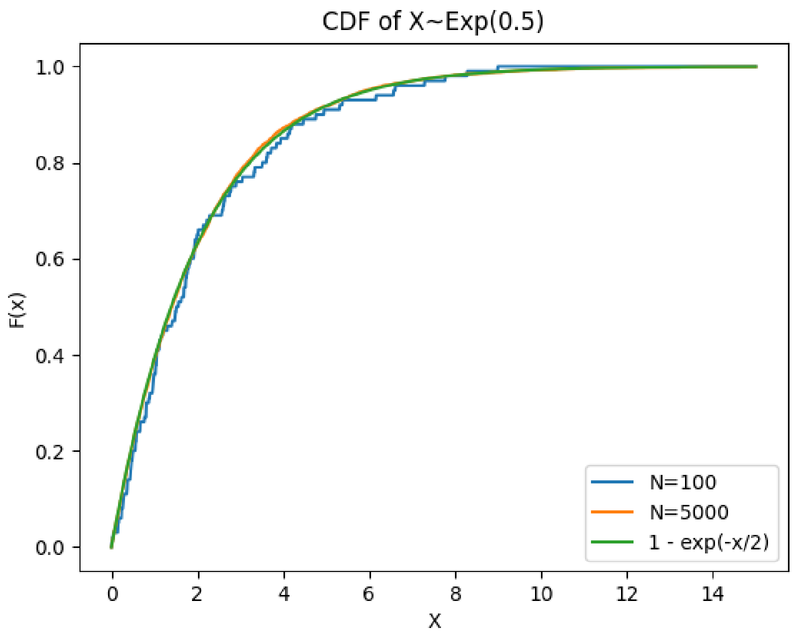
3. Write a code that takes n independent samples of a random variable X as input, and returns an approximation of the CDF of X.

```python
# Approximate the CDF of an exponential random variable
# using random samples and a set of x values.
def approximate_cdf(samples, xs):
    n = len(samples)

    # Estimate P(t<=a) for all t in samples and a in xs by
    # finding the fraction of ts less than each a.
    result = []
    for x in xs:
        s = 0
        for sample in samples:
            if sample <= x:
                s += 1
        result.append(s / n)

    return np.array(result)
```

4. Using your code from problem 2, generate two sets of samples of $X \sim \text{Exp}(\frac{1}{2})$, one for $n = 100$ and the other for $n = 5000$; and for each set of samples, compute an approximation of the CDF of X using your code for problem 3 (when $x_{min} = 0$, $x_{max} = 15$, and $\Delta = 0.01$).



5. Using the samples generated for both $n = 100$ and $n = 5000$, compute the sample mean $\bar{x}$ and the sample variance s^2, where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$.

n=100

$\bar{x} = 1.77$ Difference from $\mathbb{E}[X]$: 11.4%

$s^2 = 3.49$ Difference from $\mathrm{Var}[X]$: 12.8%

n=5000

$\bar{x} = 2.01$ Difference from $\mathbb{E}[X]$: 0.734%

$s^2 = 3.99$ Difference from $\mathrm{Var}[X]$: 0.0891%

Part II

6. Write a code that takes a positive scalar μ, and generates **one** sample of a Poisson random variable M with parameter μ. Use the function you wrote for problem 1 in part I.

```
# Generate one sample of a Poisson random variable with
# parameter u >= 0.
def sample(u):
    if u < 0:
        raise ValueError("sample: parameter of a Poisson distribution must be
    positive")

    # Sum independent samples of Y~Exp(1) until reaching u,
    # counting the m samples taken.
    s = m = 0
    y = exponential.sample(1)
    while True:
        s += y
        if s > u:
            break
        m += 1
        y = exponential.sample(1)

    return m
```

7. Using your code from problem 6, generate 5000 independent samples of $M \sim \text{Po}(5)$, and compute the sample mean and the sample variance using the generated samples.

$\bar{x} = 4.99$	Difference from $\mathbb{E}[M]$: 0.132%
$s^2 = 5.02$	Difference from $\text{Var}[M]$: 0.495%

Part III

8. Write a code to generate 1,000,000 independent samples of T. You can use the functions you wrote for the problems in parts I and II.

```
NUM_SAMPLES = 1_000_000
ts = []

# Generate NUM_SAMPLES samples of random variable T.
for _ in range(NUM_SAMPLES):
    n = poisson.sample(5)

    m = t = 0
    count_ms_gte_1 = 0
    for _ in range(n):
        m = poisson.sample(2)

        # Count ms>=1. Later, if this count == n,
        # then for this sample all ms>=1.
        if m >= 1:
            count_ms_gte_1 += 1

        for _ in range(m):
            t += exponential.sample(0.5)

    ts.append(t)
```

9. Compute an estimate of the probability of each of the following events, by computing the fraction of times (out of 1,000,000 generated samples) that the event of interest has occured:

   (i) The event that T is no more than 20 minutes.

   $$Pr(T \leq 20) \approx 0.56$$

   (ii) The event that T is more than 20 minutes *given that* Jane goes to the bank at least 5 times during a month.

   $$Pr(T \geq 20 | N \geq 5) \approx 0.66$$

   (iii) The event that T is more than 20 minutes *given that* each time there is at least 1 customer ahead of Jane.

   $$Pr(T \geq 20 | \min(M_1, M_2, \ldots, M_N) \geq 1) \approx 0.44$$

   (iv) The event that T is more than 20 minutes *given that* Jane goes to the bank at least 5 times during a month and each time there is at least 1 customer ahead of her.

   $$Pr(T \geq 20 | N \geq 5, \min(M_1, M_2, \ldots, M_N) \geq 1) \approx 0.76$$

10. Compute the sample mean and the sample variance of T.

$$\bar{x} = 19.99 \qquad s^2 = 159.7$$

Multiple runs of the simulation strongly suggest that

$$\mathbb{E}[T] = 20$$

$$\text{Var}[T] = 160$$

To confirm,

$$\begin{aligned}
\mathbb{E}[T] &= \mathbb{E}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_j] \\
&= \mathbb{E}[\mathbb{E}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_j|N]] && \text{(LTE)} \\
&= \mathbb{E}[\sum_{i=1}^{n}\mathbb{E}[\sum_{j=1}^{M_i} X_j] && (N \text{ is independent of } M \text{ and } X) \\
&= \mathbb{E}[\sum_{i=1}^{n}\mathbb{E}[\mathbb{E}[\sum_{j=1}^{M_i} X_j|M_i]]] \\
&= \mathbb{E}[\sum_{i=1}^{n}\mathbb{E}[\sum_{j=1}^{m_i} X_j]] && (\text{All } M_i \text{ are independent of } X \text{ and each other}) \\
&= \mathbb{E}[\sum_{i=1}^{n} 2M] \\
&= \mathbb{E}[2NM] \\
&= 20
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathrm{Var}[T] &= \mathrm{Var}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_k] \\
&= \mathbb{E}[\mathrm{Var}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_k|N] + \mathrm{Var}[\mathbb{E}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_k|N]] && \text{(LTV)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_k|N] &= \sum_{i=1}^{n}\mathrm{Var}[\sum_{j=1}^{M_i} X_k] \\
&= \sum_{i=1}^{n}\{\mathbb{E}[\mathrm{Var}[\sum_{j=1}^{M_i} X_j|M_i]] + \mathrm{Var}[\mathbb{E}[\sum_{j=1}^{M_i} X_j|M_i]]\} \\
&= \sum_{i=1}^{n}\{\mathbb{E}[4M] + \mathrm{Var}[2M]\} \\
&= \sum_{i=1}^{n}\{4\mathbb{E}[M] + 4\mathrm{Var}[M]\} \\
&= 16N
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}[\sum_{i=1}^{N}\sum_{j=1}^{M_i} X_k|N] &= \sum_{i=1}^{n}\mathbb{E}[\sum_{j=1}^{M_i} X_j] \\
&= \sum_{i=1}^{n}\mathbb{E}[\mathbb{E}[\sum_{j=1}^{M_i} X_j|M_i]] \\
&= \sum_{i=1}^{n}\mathbb{E}[2M] \\
&= 4N
\end{aligned}$$

$$\therefore \mathrm{Var}[T] = \mathbb{E}[16N] + \mathrm{Var}[4N]$$

$$
\begin{aligned}
&= 16\mathbb{E}[N] + 16\mathrm{Var}[N] \\
&= 160
\end{aligned}
$$

Appendices

Listings


```python
#!/usr/bin/env python

import math
import random
import matplotlib.pyplot as plt
import numpy as np


# Generate one sample of an exponential random variable with
# parameter y >= 0.
def sample(y):
    if y < 0:
        raise ValueError(
            "sample: parameter of an exponential distribution must be positive"
        )

    # Returns a sample of R~Unif[0,1].
    r = random.random()

    # Generate sample using inverse of exponential CDF.
    return -1 / y * math.log(1 - r)


# Generate a list of n samples of an exponential random
# variable with parameter y >= 0.
def n_samples(y, n):
    return np.array([sample(y) for _ in range(n)])


# Approximate the CDF of an exponential random variable
# using random samples and a set of x values.
def approximate_cdf(samples, xs):
    n = len(samples)

    # Estimate P(t<=a) for all t in samples and a in xs by
    # finding the fraction of ts less than each a.
    result = []
    for x in xs:
        s = 0
        for sample in samples:
            if sample <= x:
                s += 1
        result.append(s / n)

    return np.array(result)


# Compute the mean of samples.
def mean(samples):
    n = s = 0
    for x in samples:
        s += x
        n += 1
    return s / n


# Compute the variance of samples.
def var(samples):
    m = mean(samples)
    n = s = 0
    for x in samples:
        s += (x - m) ** 2
        n += 1
    return s / (n - 1)
```

```python
if __name__ == "__main__":
    # Seed the random generator (system time is used by default).
    random.seed()

    # Generate two sample sets of n=100 and n=5000 with parameter 0.5.
    y = 0.5
    samples100 = n_samples(y, 100)
    samples5000 = n_samples(y, 5000)

    # Generate 1500 x values between 0 and 15 to plot
    # against the sample sets.
    xs = np.linspace(0, 15, num=1500)

    # Generate a graph of the approximate CDF of an
    # exponential random variable for values between 0 and
    # 15; one using samples100, another using samples5000,
    # and finally the actual cdf.
    fig, ax = plt.subplots()
    ax.set_title("CDF of X~Exp(0.5)")
    ax.set_xlabel("X")
    ax.set_ylabel("F(x)")
    ax.plot(xs, approximate_cdf(samples100, xs), label="N=100")
    ax.plot(xs, approximate_cdf(samples5000, xs), label="N=5000")
    ax.plot(xs, np.array([1 - math.exp(-y * x) for x in xs]), label="1 - exp(-x/2)")
    ax.legend()
    plt.savefig("cdf.png")

    # Compute and display the mean and variance, and the percent
    # difference from expected values, of both sample sets.
    u1 = mean(samples100)
    s1 = var(samples100)
    u2 = mean(samples5000)
    s2 = var(samples5000)
    print("n=100:")
    print("mean = ", format(u1, "g"))
    print("% diff =", format(abs(u1 - 2) / 2 * 100, "g"))
    print("variance =", format(s1, "g"))
    print("% diff =", format(abs(s1 - 4) / 4 * 100, "g"))
    print()
    print("n=5000:")
    print("mean =", format(u2, "g"))
    print("% diff =", format(abs(u2 - 2) / 2 * 100, "g"))
    print("variance =", format(s2, "g"))
    print("% diff =", format(abs(s2 - 4) / 4 * 100, "g"))
```

Listing 1: exponential.py

```python
#!/usr/bin/env python

import math
import random
import numpy as np

import exponential

# Generate one sample of a Poisson random variable with
# parameter u >= 0.
def sample(u):
    if u < 0:
        raise ValueError("sample: parameter of a Poisson distribution must be
    positive")

    # Sum independent samples of Y~Exp(1) until reaching u,
    # counting the m samples taken.
    s = m = 0
    y = exponential.sample(1)
    while True:
        s += y
        if s > u:
            break
        m += 1
        y = exponential.sample(1)

    return m


# Generate n samples of a Poisson random variable with parameter u >= 0.
def n_samples(u, n):
    return np.array([sample(u) for _ in range(n)])


# Compute the mean of samples.
def mean(samples):
    n = s = 0
    for x in samples:
        s += x
        n += 1
    return s / n


# Compute the variance of samples.
def var(samples):
    m = mean(samples)
    n = s = 0
    for x in samples:
        s += (x - m) ** 2
        n += 1
    return s / (n - 1)


if __name__ == "__main__":
    # Seed the random generator (system time is used by default).
    random.seed()

    samples = n_samples(5, 5000)

    u = mean(samples)
    s = var(samples)

    print("mean =", format(u, "g"))
    print("% diff =", format(abs(u - 5) / 5 * 100, "g"))
    print("variance =", format(s, "g"))
    print("% diff =", format(abs(s - 5) / 5 * 100, "g"))
```

Listing 2: poisson.py

```python
#!/usr/bin/env python

import math
import os
import random
import numpy as np

import exponential
import poisson

# Seed the rng.
random.seed()

# Create variables to count the occurrences of each event
# we're asked to estimate the probability of.
count_n = 0
count_tn = 0
count_m = 0
count_tm = 0
count_nm = 0
count_tnm = 0
count_t = 0

NUM_SAMPLES = 1_000_000
ts = []

# Generate NUM_SAMPLES samples of random variable T.
for _ in range(NUM_SAMPLES):
    n = poisson.sample(5)

    m = t = 0
    count_ms_gte_1 = 0
    for _ in range(n):
        m = poisson.sample(2)

        # Count ms>=1. Later, if this count == n,
        # then for this sample all ms>=1.
        if m >= 1:
            count_ms_gte_1 += 1

        for _ in range(m):
            t += exponential.sample(0.5)

    ts.append(t)

    # Count all {T<=20}
    if t <= 20:
        count_t += 1

    # Count {T>20 and N>=5}
    if n >= 5:
        count_n += 1
        if t > 20:
            count_tn += 1

    # Count {T>20 and all Ms>=1}
    if count_ms_gte_1 == n:
        count_m += 1
        if t > 20:
            count_tm += 1

    # Count {T>20 and N>=5 and all Ms>=1}
    if n >= 5 and count_ms_gte_1 == n:
        count_nm += 1
        if t > 20:
            count_tnm += 1

# Divide each count by the number of samples to estimate the
# probability. In the case of conditional probabiliites,
# NUM_SAMPLES cancels and it suffices to divide the count of
# intersection with the count of the condition.
print("Pr(T<=20) =", format(count_t / NUM_SAMPLES, "g"))
print("Pr(T>=20|N>=5) =", format(count_tn / count_n, "g"))
print("Pr(T>=20|M>=1) =", format(count_tm / count_m, "g"))
```

```
print("Pr(T>=20|N>=5,M>=1) =", format(count_tnm / count_nm, "g"))

print("mean =", format(np.mean(ts), "g"))
print("variance =", format(np.var(ts), "g"))
```

Listing 3: part3.py