Stochastic Simulation (MIE1613H) - Homework 1 (Solutions)

Problem 1. **(20 Pts.)** Assume that X is continuous and uniformly distributed in $[2, 10]$ (X is a continuous random variable). We are interested in computing $\theta = E[(X-5)^+]$. (Note: $a^+ = \text{Max}(a, 0)$, i.e, if $a < 0$ then $a^+ = 0$ and if $a \geq 0$ then $a^+ = a$.)

(a) Compute θ exactly using the definition of expected value. **HINT:** Recall that a continuous Uniform random variable in $[a, b]$ has density function,

$$f(x) = \frac{1}{b-a},$$

for $x \in [a, b]$ and 0 otherwise.

(5 points) Using the definition of the expected value of a function of a continuous random variable we have,

$$\begin{aligned}
\theta = E[(X-5)^+] &= \int_a^b (x-5)^+ \frac{1}{b-a} dx \\
&= \int_5^{10} \frac{1}{8}(x-5) dx \\
&= \frac{x^2}{16} - \frac{5x}{8}\Big|_5^{10} = \left(\frac{100}{16} - \frac{50}{8}\right) - \left(\frac{25}{16} - \frac{25}{8}\right) = \frac{25}{16} = 1.5625.
\end{aligned}$$

(b) Estimate θ using Monte Carlo simulation and provide a 95% confidence interval for your estimate. **Note**: Use the `np.random.random()` method in Python and transform it to a sample of X. You may NOT use other built-in methods of Python to generate the samples.

(10 points) To estimate θ we generate $n = 10,000$ iid samples of the random variable $(X-5)^+$ and compute the sample average. The `np.random.random()` method returns a sample from a uniformly distributed random variable in $[0, 1]$ that should be first multiplied by $(b-a)$ and then added by a to generate a sample of a uniformly distributed random variable in $[a, b]$. The sample average estimate is 1.55 and the 95% CI is given by $[1.52, 1.58]$ which includes the exact value.

(c) Create a plot that demonstrates the convergence of the Monte Carlo estimate to the exact value as the number of samples increases.

(5 points) See the source code and the output below.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
np.random.seed(1)
```

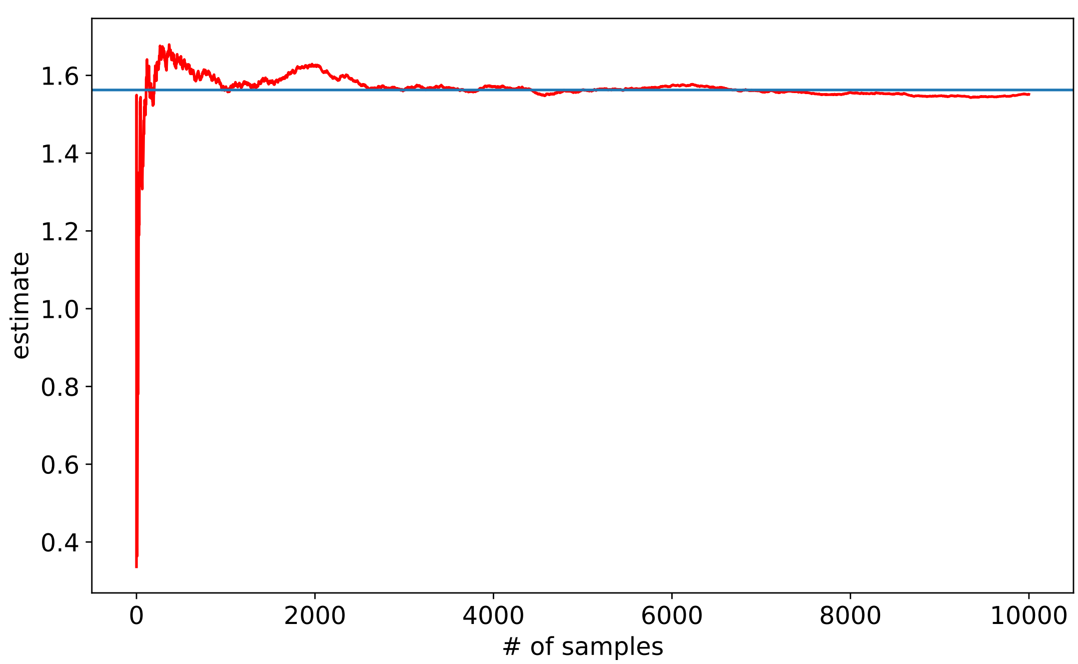
```python

def mean_confidence_interval_95(data):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = 1.96*se
    return m, m-h, m+h

samples = []
estimates = []
for n in range(0,10000):
    # generate a sample of the random variable and append to the list
    X = (10-2)*np.random.random() + 2
    samples.append(max(X-5, 0))
    estimates.append(np.mean(samples))
print("The estimate and 95% CI:", mean_confidence_interval_95(samples))

plt.plot(estimates,'r')
plt.xlabel('# of samples')
plt.ylabel('estimate')
plt.axhline(y = 1.5625)
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams.update({'font.size': 14})
plt.show()
```

```
The estimate and 95% CI: (1.5514013347811288, 1.5190310747453017, 1.583771594816956)
```



Problem 2. (25 Pts.) In the original TTF example we simulated the system until the time of first failure. Modify the simulation model to simulate the system for a given fixed number of days denoted by T. Assume that all other inputs and assumptions are the same as in the original example.

(a) We say that the system is fully functional provided that both components (active and spare) are functional. Denote by $A(t)$ a process that takes value 1 if the system is fully functional at time t and 0 otherwise. Then,

$$\bar{A}(T) = \frac{1}{T}\int_0^T A(t)dt,$$

is the fraction of time the system is fully functional between 0 and T. Modify your simulation model to estimate $\bar{A}(T)$ until $T = 1000$ based on one replication of the simulation.

(15 points) The logic is modified as follows. We start with $A = 1$ as the system is initially fully functional. Then, when one component fails, i.e., $S = 1$, we need to set $A = 0$, schedule the completion of a repair, and schedule the failure of the component that just started working. When both components fail, i.e., $S = 0$, the NextFailure is set to ∞. In addition, if $S = 2$ after the completion of a repair, we need to set $A = 1$ and set NextRepair to ∞. Simulating one sample path of the process $A(t)$ for $T = 1000$ time units we have

$$\frac{1}{1000}\int_0^{1000} A(t)dt = 0.339.$$

```
import numpy as np

# start with 2 functioning components at time 0
clock = 0
S = 2
A = 1  # system is fully functional at time 0

# fix random number seed
np.random.seed(1)

# initialize the time of events
NextRepair = float('inf')
NextFailure = np.ceil(6 * np.random.random())
EndSimulation = 1000

# Define variables to keep the area under the sample path
# and the time and state of the last event
Area = 0.0
Tlast = 0
Alast = 1

while clock != EndSimulation:
    # advance the time
    clock = min(NextRepair, NextFailure, EndSimulation)

    if NextFailure < NextRepair and NextFailure < EndSimulation:
        # next event is a failure
        S = S - 1
```

```
        A = 0
        if S == 1:
            NextRepair = clock + 2.5
            NextFailure = clock + np.ceil(6 * np.random.random())
        else:  # S == 0
            NextFailure = float('inf')

    elif NextRepair < NextFailure and NextRepair < EndSimulation:
        # next event is completion of a repair
        S = S + 1
        if S == 1:
            NextRepair = clock + 2.5
            NextFailure = clock + np.ceil(6 * np.random.random())
        else:  # S==2
            A = 1
            NextRepair = float('inf')

    else:
        # next event is simulation end
        continue

    # Update the area under the sample path and the
    # time and state of the last event
    Area = Area + (clock - Tlast) * Alast
    Tlast = clock
    Alast = A

print("Average system full functionality till time T = 1000:", Area / clock)
```

```
Average system full functionality till time T = 1000:  0.361
```

(b) Estimate $\bar{A}(T)$ for $T = 2000$ and $T = 4000$ (again using a single replication) and compare the values with the estimate from part (a). Summarize your observation in one sentence.

(10 points) For $T = 2000$ and $T = 4000$, respectively, we get

$$\bar{A}(T) = \frac{1}{2000}\int_0^{2000} A(t)dt = 0.357,$$

and

$$\bar{A}(T) = \frac{1}{4000}\int_0^{4000} A(t)dt = 0.349.$$

We observe that the estimates under different T are roughly the same, suggesting that the time averages are converging to some constant θ, which is the long-run proportion of time that the system is fully functional:

$$\theta = \lim_{T\to\infty}\frac{1}{T}\int_0^T A(t)dt.$$

Problem 3. **(25 Pts.)** Modify the TTF simulation from the lecture so that instead of 2 components there can be any number of components. The number of components N should be an input of the simulation model. As before, assume that one component is active and the rest are kept as spares.

Also, only one component can be repaired at a time. Run your simulation for 1000 replications and report a 95% confidence interval for the expected time to failure of the system for $N = 2, 3$, and 4.

(25 points) With N components the logic is modified as follows. When a failure happens, we have $S = N - 1$ or $S < N - 1$. If $S = N - 1$, then a new component starts working and a new repair begins. Therefore, we need to schedule both the next failure and repair events. If $S < N - 1$ and $S \neq 0$, then a repair is already in progress and therefore we only schedule the next failure for the component that just started working. If $S = 0$, then the system fails. When a repair is completed we have $S = N$ or $S < N$. If $S = N$, then a failure is still pending and all components are functioning. Therefore we only need to set the next repair time to ∞. If $S < N$, again a failure is still pending but a new repair starts which we need to schedule.

Based on 1000 replications the 95% CI for the expected time to failure of the system when $N = 2, 3$, and 4 is $[13.75, 15.15]$, $[104.75, 117.68]$ and $[994.30, 1117.36]$, respectively.

```python
# The TTF example with multiple components; still 1 repair at a time
import numpy as np
from scipy import stats

comp = 4 # number of components available
Ylist = [] #  keeps samples of time to failure
Avelist = [] # keeps samples of average # of func. components
np.random.seed(1)

def mean_confidence_interval_95(data):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = 1.96*se
    return m, m-h, m+h

for reps in range(1000):
        # initialize clock, next events, state
        clock = 0
        S = comp
        NextRepair = float('inf')
        NextFailure = np.random.choice([1,2,3,4,5,6], 1)
        # define variables to keep the last state and time, and the area under the sample path
        Slast = S
        Tlast = clock
        Area = 0

        while S > 0: # While system is functional
                # Determine the next event and advance time
                clock = np.min([NextRepair,NextFailure])
                event = np.argmin([NextRepair,NextFailure])
                if event == 0: # Repair
                        S += 1
                        if S == 1: # this would never be the case for the current while loop
                                NextRepair = clock + 2.5
                                NextFailure = clock + np.random.choice([1,2,3,4,5,6], 1)
```

```python
            if S < comp: # a new repair starts
                NextRepair = clock + 2.5
            if S == comp: # all components are functional
                NextRepair = float('inf')

        else: # Failure
            S -= 1
            if S == comp - 1: # A new component starts working; a new repair starts
                NextRepair = clock + 2.5
                NextFailure = clock + np.random.choice([1,2,3,4,5,6], 1)
            else: # a new component starts working
                NextFailure = clock + np.random.choice([1,2,3,4,5,6], 1)

        # update the area udner the sample path
        Area = Area + Slast * (clock - Tlast)
        # record the current time and state
        Slast = S
        Tlast = clock
    # add samples to the lists
    Ylist.append(clock)
    Avelist.append(Area/clock)

# print the estimates with a 95% confidence interval
print('The estimate and 95% CI for the expected time to failure:',
        mean_confidence_interval_95(Ylist))
```

```
The estimate and 95% CI for the expected time to failure:
(1055.831, 994.3019475153425, 1117.3600524846572)
```

Problem 4. (5 Pts.) The standard error of an estimator is defined as the standard deviation of that estimator. In the lecture, we introduced the sample mean $\bar{X}_n = (1/n)\sum_{i=1}^{n} X_i$ as an estimator of $E[X]$ where X_i's are iid samples of the random variable X. What is the standard error of the estimator $\bar{X}_n$? Assume that the standard deviation of X is σ.

(5 points) The variance of the estimator is given by

$$Var\left(\bar{X}_n\right) = Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} n Var(X_1) = \frac{1}{n} Var(X_1).$$

Therefore, the standard deviation is

$$\sqrt{Var\left(\bar{X}_n\right)} = \sqrt{\frac{1}{n} Var(X_1)} = \frac{\sigma}{\sqrt{n}}.$$

Problem 5. (10 Pts.) Consider random variables X, Y and constants a, b. Using the properties of expected value and definitions of covariance and variance covered in the second lecture, show that the following statements are true (make sure to clearly explain your derivations and specify the properties you use in each step):

(a) $Var(aX + b) = a^2 Var(X)$

(5 points)

$$\begin{aligned}
Var(aX+b) &= E\left[(aX+b)^2\right] - E[aX+b]^2 \\
&= E[a^2X^2 + 2abX + b^2] - a^2E[X]^2 - 2abE[X] - b^2 \\
&= a^2\left(E\left[X^2\right] - E[X]^2\right) \\
&= a^2 Var(X).
\end{aligned}$$

The first equality follows from the definition of variance, the second inequality uses the linearity of expectation, and the last equality follows again from the definition of variance.

(b) $Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$.

(5 points)

$$\begin{aligned}
Var(X+Y) &= E\left[(X+Y)^2\right] - E[X+Y]^2 \\
&= E\left[X^2\right] + 2E[XY] + E\left[Y^2\right] - E[X]^2 - 2E[X]E[Y] - E[Y]^2 \\
&= \left(E\left[X^2\right] - E[X]^2\right) + \left(E\left[Y^2\right] - E[Y]^2\right) + 2(E[XY] - E[X]E[Y]) \\
&= Var(X) + Var(Y) + 2Cov(X,Y).
\end{aligned}$$

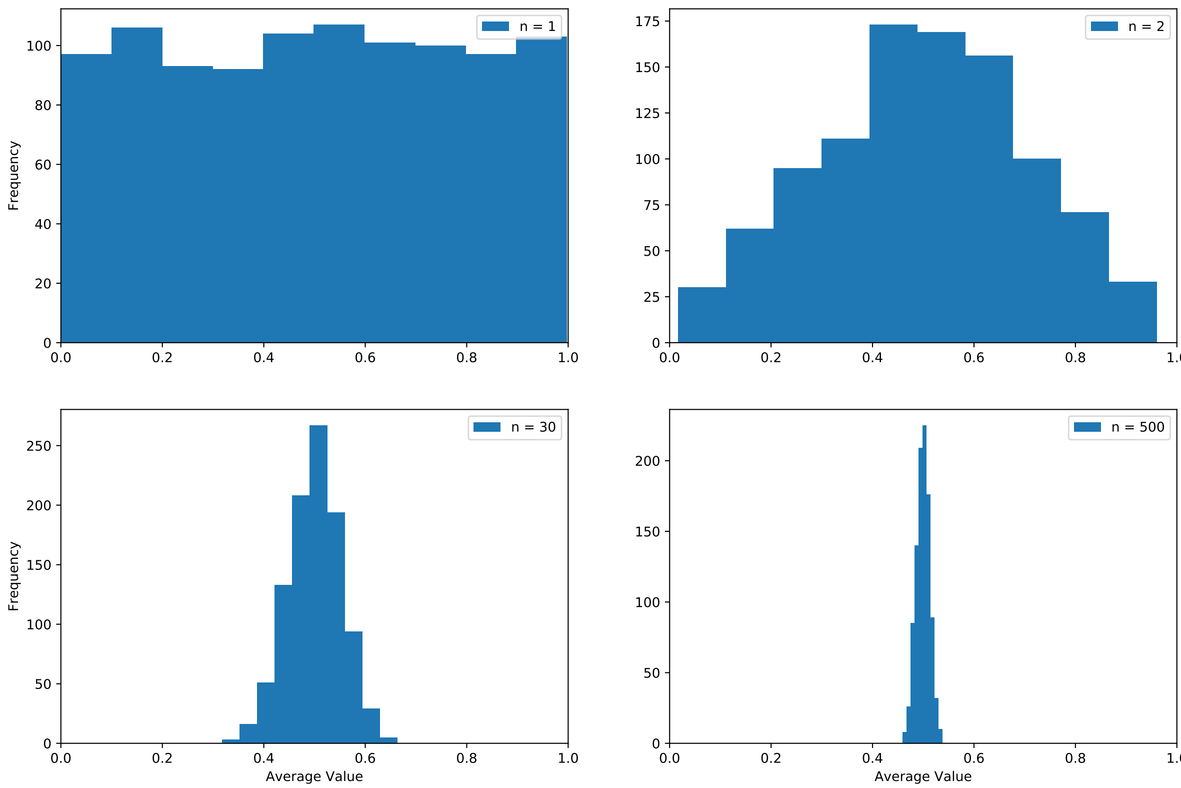
The first equality follows from the definition of variance, the second equality follows from linearity of expectation, and the last equality follows from the definition of variance and covariance.

Problem 6. (15 Pts.) Assume that $\{X_i; i \geq 1\}$ is a sequence of independent uniform random variables between $(0, 1)$, i.e., $X_i \sim \text{Unif}(0, 1)$ for all $i \geq 1$. Consider the sample average of n such random variables,

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

(a) Simulate 1000 samples of $\bar{X}_1, \bar{X}_2, \bar{X}_{30}, \bar{X}_{500}$ and plot a histogram of the samples for each case (i.e. 4 histograms in total). What happens to the mean and variability of the sample means as n increases? Relate your observations to the Central Limit Theorem discussed in the class. **Note**: You can plot a histogram in Python using the `plt.hist` method of the `matplotlib.pyplot` library.

(15 points) We observe that as n increases, the variability in sample means decreases and the samples become more concentrated around the common mean of $1/2$. In addition, the shape of the histogram becomes more symmetric and bell-shaped. This is consistent with Central Limit Theorem in the sense that the distribution of sample means is converging to a Normal distribution with mean equal to 0.5.




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

np.random.seed(1)

# n = [1, 2, 30, 500]
def Xbar (n):
    # Keep samples of average
    Avelist = []
    for i in range (1000):
        Avelist.append(np.mean(np.random.random(n)))
    return Avelist

plt.figure(figsize = (15, 12))

# For n = 1
plt.subplot(221)
plt.hist(Xbar(1), label = 'n = ' + str(1))
plt.xlim(xmin=0, xmax=1)
plt.ylabel('Frequency')
plt.legend()

# For n = 2
plt.subplot(222)
plt.hist(Xbar(2), label = 'n = ' + str(2))
plt.xlim(xmin=0, xmax=1)
plt.legend()
```

```

# For n = 30
plt.subplot(223)
plt.hist(Xbar(30), label = 'n = ' + str(30))
plt.xlim(xmin=0, xmax=1)
plt.xlabel('Average Value')
plt.ylabel('Frequency')
plt.legend()

# For n = 500
plt.subplot(224)
plt.hist(Xbar(500), label = 'n = ' + str(500))
plt.xlim(xmin=0, xmax=1)
plt.xlabel('Average Value')
plt.legend()

plt.show()
```