Stochastic Simulation (MIE1613H) - Homework 1 (Solutions)

Due: Jan 29, 2019

Problem 1. Assume that X is exponentially distributed with rate $\lambda = 0.2$. We are intreated in computing $E[(X-3)^+]$. (Note: $a^+ = \mathrm{Max}(a, 0)$, i.e, if $a < 0$ then $a^+ = 0$ and if $a \geq 0$ then $a^+ = a$.)

(a) Compute the expected value exactly.

(5 points) Recalling that the pdf of an exponential random variable is $\lambda e^{-\lambda x}$ and using the defenition of the expected value of a function of a continuous random variable we have,

$$
\begin{aligned}
E[(X-3)^+] &= \int_0^{+\infty} (x-3)^+ \lambda e^{-\lambda x} dx \\
&= \int_3^{+\infty} (x-3) \lambda e^{-\lambda x} dx.
\end{aligned}
$$

Define $u = (x-3)$. The integral can be written as

$$
E[(X-3)^+] = \int_0^{+\infty} u \lambda e^{-\lambda(u+3)} du = e^{-3\lambda} \int_0^{+\infty} u \lambda e^{-\lambda u} du
$$

Note that $\int_0^{+\infty} u \lambda e^{-\lambda u} du$ is the expected value of an exponential r.v. with rate λ and is hence equal to $1/\lambda$. It follows that

$$
E[(X-3)^+] = e^{-3\lambda}(1/\lambda) = e^{-0.6}(5) \approx 2.744.
$$

(b) Estimate the expected value using Monte Carlo simulation and provide a 95% confidence interval for your estimate. **Note**: You can generate a random sample of an exponentially distributed random variable with rate λ in Numpy using `np.random.exponential(1/λ)`.

(10 points) To estimate the expected value we generate $n = 10,000$ iid samples of the random variable $(X-3)^+$ and compute the sample average. The estimate is 2.69 and the 95% CI is given by $[2.60, 2.78]$ which includes the exact value.

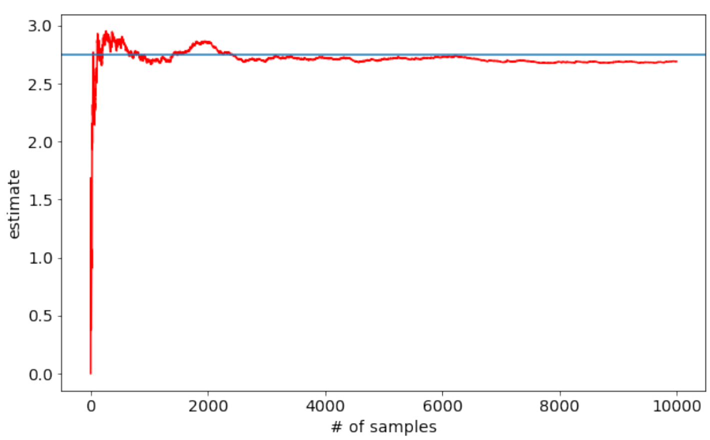
(c) Create a plot that demonstrates the convergence of the Monte Carlo estimate to the exact value as the number of samples increases.

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

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

def mean_confidence_interval_95(data):
    a = 1.0*np.array(data)
```

```python
    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
    samples.append(max(np.random.exponential(1/0.2)-3,0))
    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=2.744)
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams.update({'font.size': 14})
plt.show()
```

```
The estimate and 95% CI:, (2.690673712038912, 2.605194278819423, 2.776153145258401)
```



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

(a) What is the average number of functional components until time $T = 1000$ based on one replication of the simulation?

(20 points) The logic is modified as follows. When the system fails, i.e., $S = 0$, one repair is still pending. Therefore, we do not reschdule another repair but set the NextFailure to ∞. In addition, if $S = 1$ after the completion of a repair, we need to schedule the repair of the other component, and the failure of the component that just started working. Simulating one sample path of the process

$S(t)$ for $T = 1000$ time units we have

$$\frac{1}{1000}\int_0^{1000} S(t)dt = 1.2645.$$

(b) What is the average number of functional components until time $T = 2000$ based on one replication of the simulation? Compare the results from part (a) and (b) and summerize your observation in one sentence.

(10 points) For $T = 2000$ we get

$$\frac{1}{2000}\int_0^{2000} S(t)dt = 1.2685.$$

We observe that the results of part (a) and (b) are approximately the same suggesting that the time-average is converging to some constant θ, which is the long-run average number of functioning components:

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

```python
import numpy as np

def EndSim ():
    global Slast
    global Tlast
    global Area

    Area = Area + (clock - Tlast)* Slast
    Tlast = clock
    Slast = S

def Failure ():
    global S
    global Slast
    global Tlast
    global Area
    global NextFailure
    global NextRepair

    S = S - 1
    if S == 0:
        NextFailure = float('inf')
        # repair already in progress
    if S == 1:
        NextRepair = clock + 2.5
        NextFailure = clock + np.ceil(6*np.random.random())
    # Update the area under the sample path and the time and state at the last event
    Area = Area + (clock - Tlast)* Slast
    Tlast = clock
    Slast = S
```

```
def Repair():
    global S
    global Slast
    global Tlast
    global Area
    global NextFailure
    global NextRepair

    S = S + 1
    if S == 1:
        NextRepair = clock + 2.5
        NextFailure = clock + np.ceil(6*np.random.random())
    else: #S==2
        NextRepair = float('inf')
    Area = Area + Slast * (clock - Tlast)
    Slast = S
    Tlast = clock

def Timer():
    global clock
    global NextRepair
    global NextFailure

    if NextEndSim < NextFailure and NextEndSim < NextRepair:
        result = "EndSim"
        clock = NextEndSim

    elif NextFailure < NextRepair:
        result = "Failure"
        clock = NextFailure

    else:
        result = "Repair"
        clock = NextRepair
    return result


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

clock = 0
S = 2
# initialize the time of events
NextRepair = float('inf')
NextFailure = np.ceil(6*np.random.random())
NextEndSim = 2000
# Define variables to keep the area under the sample path
# and the time and state of the last event
Area = 0.0
Tlast = 0
Slast = 2
NextEvent = Timer()

```

```
while NextEvent!="EndSim":
    NextEvent = Timer()
    if NextEvent == "Repair":
        Repair()
    elif NextEvent == "Failure":
        Failure()
    else:
        EndSim()

print('Average # of func. comp. till failure:', Area/clock)
```

```
Average # of func. comp. till failure: 1.2685
```

Problem 3. Modify the TTF simulation assuming that there are three components, one active and two spares, but still only one can be repaired at a time. Repair time is 3.5 days. Run your simulation for 1000 replications and report a 95% confidence interval for the expected time to failure of the system.

(20 points) With three components the logic is modified as follows. When a failure happens, we have $S = 2$ or $S = 1$. If $S = 2$, then a new component starts working and a new repair begins. Therefore, we need to schedule both the next failure and repair events. If $S = 1$, then a repair is already in progress and therefore we only schedule the next failure for the component that just started working. When a repair is completed we have $S = 3$ or $S = 2$. If $S = 3$, then a failure is still pending and all components are functioning. Therefore we only need to set the next failure time to ∞. If $S = 2$, again a failure is still pending but a new repair starts which we need to schedule.

Based on $n = 1000$ replications the 95% CI for the expected average number of functioning components untill failure is $[37.05, 41.43]$.

```
import numpy as np
import scipy.stats as stats

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

def Failure ():
    global S
    global Slast
    global Tlast
    global Area
    global NextFailure
    global NextRepair

    S = S - 1
    if S == 2:
        NextRepair = clock + 3.5
        NextFailure = clock + np.ceil(6*np.random.random())
    else: # S==1
        # one repair is already in progress
```

```
        NextFailure = clock + np.ceil(6*np.random.random())
    # Update the area under the sample path and the time and state at the last event
    Area = Area + (clock - Tlast)* Slast
    Tlast = clock
    Slast = S

def Repair():
    global S
    global Slast
    global Tlast
    global Area
    global NextFailure
    global NextRepair

    S = S + 1
    if S == 3:
        # a failure is still pending so no need to schedule a new one
        # all components functioning
        NextRepair = float('inf')
    else: # S == 2
        # a new repair starts; one failure pending
        NextRepair = clock + 3.5
    Area = Area + Slast * (clock - Tlast)
    Slast = S
    Tlast = clock

def Timer():
    global clock
    global NextRepair
    global NextFailure

    if NextFailure < NextRepair:
        result = "Failure"
        clock = NextFailure

    else:
        result = "Repair"
        clock = NextRepair
    return result


# Set number of replications
N = 1000
# Define lists to keep samples of the outputs across replications
TTF_list = []
Ave_list = []

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

# Replication loop
for reps in range(0,N):
    # start with 2 functioning components at time 0
```

```
    clock = 0
    S = 3
    # initialize the time of events
    NextRepair = float('inf')
    NextFailure = np.ceil(6*np.random.random())
    # Define variables to keep the area under the sample path
    # and the time and state of the last event
    Area = 0.0
    Tlast = 0
    Slast = 2

    while S > 0: # While system is functional
        NextEvent = Timer()

        if NextEvent == "Repair":
            Repair()
        else:
            Failure()

    # add samples to the lists
    TTF_list.append(clock)
    Ave_list.append(Area/clock)

print('95% CI for the expected time to failure:', mean_confidence_interval_95(
    TTF_list))
```

```
95% CI for the expected time to failure:, (39.239, 37.0480606471163, 41.42993935288369))
```

Problem 4. The standard error of an estimator is defined as the standard deviaition of that estimator. In class 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 statndard deviation of X is σ.

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

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

Therefore, the standard deviation is

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

Problem 5. If you buy a lottery ticket in 50 lotteries, in each of which your chance of winning a prize is 1/200, what is the (approximate) probability that you will win a prize (a) at least once, (b) exactly once, (c) at least twice?

(15 points) The number of lotteries in which you win a prize N has a Binomail distribution with parameters $n = 50$ and $p = 1/200$. Since, n is large and p is small we can approximate the Binomial distribution using a Poisson distribution with mean $np = 1/4$. Therefore,

$$P(N = n) \approx \frac{e^{-(1/4)}(1/4)^n}{n!}.$$

(a) $P(N \geq 1) = 1 - P(N = 0) \approx 1 - e^{-1/4} \approx 0.2212.$

(b) $P(N = 1) \approx (1/4)e^{-1/4} \approx 0.1947.$

(c) $P(N \geq 2) = 1 - P(N = 0) - P(N = 1) = 1 - e^{-1/4} - (1/4)e^{-1/4} \approx 0.0265.$