Stochastic Simulation (MIE1613H) - Homework 2 (Solutions)

Problem 1. (20 Pts.) Consider the $AR(1)$ model described in Section 3.3. of the textbook. Derive an expression for the asymptotic MSE of the sample mean $\bar{Y}(m)$. **HINT**: Start with the definition of asymptotic bias from Eq. (5.4) and asymptotic variance from Eq. (5.7).

(20 Points) The asymptotic bias is defined as

$$\beta = \sum_{i=1}^{\infty}(E(Y_i) - \mu),$$

and based on Eq. (3.9) for the $AR(1)$ model

$$E(Y_i) = \mu + \varphi^i(E(Y_0) - \mu).$$

Thus, we have

$$\beta = \sum_{i=1}^{\infty}(\mu + \varphi^i(E(Y_0) - \mu) - \mu) = \sum_{i=1}^{\infty}\varphi^i(E(Y_0) - \mu) = \frac{\varphi(E(Y_0) - \mu)}{1 - \varphi},$$

where we have used the fact that $\sum_{i=1}^{\infty}\varphi^i$ is a Geometric sum and converges to $\frac{\varphi}{1-\varphi}$. The asymptotic variance for a covariance stationary process is given by

$$\gamma^2 = \tilde{\sigma}^2\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right),$$

where

$$\tilde{\sigma}^2 = \lim_{m\to\infty} Var(Y_m) = \frac{\sigma^2}{1-\phi^2},$$
$$\rho_k = \lim_{m\to\infty} Corr(Y_m, Y_{m+k}) = \phi^k.$$

Here, we have used the notation $\tilde{\sigma}^2$ to avoid confusion with σ^2, the variance of X_i in the AR(1) model. Thus, substituting above and simplifying, we have for the AR(1) model

$$\begin{aligned}
\gamma^2 &= \frac{\sigma^2}{1-\phi^2}\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right) \\
&= \frac{\sigma^2}{1-\phi^2}\left(1 + 2\frac{\phi}{1-\phi}\right) \\
&= \frac{\sigma^2}{1-\phi^2}\frac{1+\phi}{1-\phi} \\
&= \frac{\sigma^2}{(1-\phi)^2}.
\end{aligned}$$

Finally, the asymptotic MSE is given by $\beta^2 + \gamma^2$.

Problem 2. (15 Pts.) Consider the $M(t)/M/\infty$ model of the parking lot example from the lecture.

Using simulation, you have generated n iid sample paths of the process $\{N(t); 0 \leq t \leq T = 24\}$, i.e., the number of cars in the parking lot at time t during a 24-h period, and starting with 0 cars in the parking lot, i.e., $N(0) = 0$. Denote the simulated sample paths by $N_1(t), N_2(t), ..., N_n(t)$.

For each of the performance measures of interest below, propose an unbiased (if possible) estimator using the simulation output.

(a) Expected average number of cars in the parking lot during a day

(b) Expected maximum number of cars in the parking lot during a day

(c) Probability of having more than 5 cars left in the parking lot at the end of the day

(d) 95th quantile of the maximum number of cars in the parking lot during the day

(a) **(3 Points)** Define

$$\bar{N}_i(24) = \frac{1}{24} \int_0^{24} N_i(t)dt, \quad \forall i = 1, \ldots, n,$$

which represents the average number of cars in the lot during a day for the ith sample path. Since $\bar{N}_i(24)$'s are iid random variables, their sample average, $\frac{1}{n}\sum_{i=1}^n \bar{N}_i(24)$, is an unbiased estimator of the expected average number of cars in the lot during a day.

(b) **(3 Points)** Let

$$M_i(24) = \max_{0 \leq t \leq 24} N_i(t), \quad \forall i = 1, \ldots, n,$$

be the maximum number of cars in the lot observed during a day in the ith sample path. Since $M_i(24)$'s are iid random variables, the sample average, $\frac{1}{n}\sum_{i=1}^n M_i(24)$, is an unbiased estimator of the expected maximum number of cars in the lot during a day.

(c) **(4 Points)** $N_i(24)$ represents the number of cars in the lot at the end of the day in the ith sample path. Define

$$I_i(N_i(24) > 5) = \begin{cases} 1, & \text{if } N_i(24) > 5, \\ 0, & \text{otherwise.} \end{cases}$$

Since $I_i(\cdot)$'s are iid random variables, again the sample average, $\frac{1}{n}\sum_{i=1}^n I_i(N_i(24) > 5)$, is an unbiased estimator of $P(N(24) > 5)$, i.e., the probability that there are more than 5 cars in the lot at the end of the day.

(d) **(5 Points)** Define $M_i(24), \forall i = 1, \ldots, n$, in the same way as part (b), and let $M_{(1)}(24), \ldots, M_{(n)}(24)$ be the sorted sequence in an increasing order. The estimator for the 95th quantile of the maximum number of cars in the lot during the day is thus given by the value in the $\lceil 0.95n \rceil$th position, i.e., $M_{(\lceil 0.95n \rceil)}(24)$.

Problem 3. (15 Pts.) (Down-and-out call option) Another variation of European options are barrier options. Denote the stock price at time t by $X(t)$ and assume that it is modelled as a Geometric Brownian Motion (GBM). A down-and-out call option with barrier B and strike price K has payoff

$$I\left\{\min_{0 \leq t \leq T} X(t) > B\right\} (X(T) - K)^+,$$

where $I\{A\}$ is the indicator function of event A. This means that if the value of the asset falls below $\$B$ before the option matures then the option is worthless. Hence, the value of the option is

$$E\left[e^{-rT} I\left\{\min_{0\le t\le T} X(t) > B\right\}(X(T)-K)^+\right].$$

Using the same parameters as in the Asian option example, i.e., $T = 1$, $X(0) = \$50$; $K = \$55$; $r = 0.05$ and $\sigma^2 = (0.3)^2$, estimate the value of this option for barriers $B = 35, 40, 45$ and provide an intuitive reason for the effect of increasing the barrier on the value of the option. Use $m = 64$ steps when discretizing the GBM and use $n = 40,000$ replications. Report a 95% confidence interval for your estimates.

(15 Points) To estimate the expected payoff, we define a binary variable 'active' initialized with value 1 and check whether the price has fallen below the barrier everytime we generate a new point on the sample path. If the barrier is crossed, we set the value of the variable 'active' to 0. When all the points are generated we compute the payoff depending on the value of 'active'. The estimated value of this option, which is the sample average of the payoff across all replications, for barriers $B = 35, 40, 45$ is 5.03, 4.88, and 3.96 with the corresponding 95% CIs of [4.94, 5.13], [4.78, 4.97], and [3.87, 4.06], respectively. Increasing barrier results in lower (or cheaper) estimated expected payoff since it can only generate a positive payoff if the price does not fall below the barrier which in turn reduces the probability of having a positive final payoff.

```
1  import SimRNG
2  import math
3  import pandas
4  import numpy as np
5
6  def CI_95(data):
7      a = np.array(data)
8      n = len(a)
9      m = np.mean(a)
10     var = ((np.std(a))**2)*(n/(n-1))
11     hw = 1.96*np.sqrt(var/n)
12     return m, [m-hw,m+hw]
13
14 ZRNG = SimRNG.InitializeRNSeed()
15 Replications = 40000
16 Maturity = 1.0
17 Steps = 64
18 Sigma = 0.3
19 InterestRate = 0.05
20 InitialValue = 50.0
21 StrikePrice = 55.0
22 Interval = Maturity / Steps
23 Sigma2 = Sigma * Sigma / 2
24 # barrier threshold
25 B = 35.0
26
27 TotalValue = []
28
29 for i in range(0,Replications,1):
30     X = InitialValue
31     # binary variable; set to 0 if the value of asset falls below barrier
```

```
32      active = 1
33      for j in range(0,Steps,1):
34          Z = SimRNG.Normal(0,1,12)
35          X = X * math.exp((InterestRate - Sigma2) * Interval + Sigma * math.
                sqrt(Interval) * Z)
36          if X <= B:
37                  active = 0
38      Value = math.exp(-InterestRate * Maturity) * max(X - StrikePrice, 0)
39      if active == 1:
40          TotalValue.append(Value)
41      else:
42          TotalValue.append(0)
43
44  print ("Barrier option:",CI_95(TotalValue))
```

```
Barrier option:  (5.03446922842732, [4.93761728190281, 5.131321174951831])
```

Problem 4. (15 Pts.) (Chapter 4, Exercise 4) Beginning with the PythonSim event-based $M/G/1$ simulation, implement the changes necessary to make it an $M/G/s$ simulation (a single queue with s servers). Keep the arrival rate at $\lambda = 1$ and use average service time $\tau = 0.8 \times s$, and simulate the system for $s = 1, 2, 3$.

(a) Report the estimated expected number of customers in the system (including customers in the queue and service), expected system time, and expected number of busy servers in each case.

(b) Compare the results and state clearly what you observe. What you're doing is comparing queues with the same service capacity, but with 1 fast server as compared to 2 or more slow servers.

HINT: You need to modify the logic, not just set the number of available servers to s. The attribute "NumberOfUnits" of the Resource object returns the number of available units for any instance of the object.

The modifications are as follows: The number of servers is set using variable 'ServerNum'. Upon arrival of a customer we schedule a departure if there is an idle server (Server.Busy < ServerNum) in which case the customer can immediatly start service. When an EndofService event occurs, we schedule the next EndofService if (Queue.NumQueue() >= ServerNum), i.e., if there is a customer waiting. Note that Queue here includes both the waiting customers and those in service.

(a) **(10 Points)** The estimates are summarized below:

	$s = 1$	$s = 2$	$s = 3$
Expected Average wait	2.93	3.51	4.15
Expected Average queue-length	2.93	3.51	4.15
Expected Average # of servers busy	0.80	1.60	2.40

(b) **(5 Points)** We observe that as the number of servers increases and the service rate decreases, the expected number of customers and the total time in system increases. The average number of busy servers is equal to $0.8s$ so the average utilization of the servers remains constant. Note that with $s = 1$ the output rate of the system is $1/0.8s = 1/0.8$ as long as there are customers in the system, while when $s > 1$ the maximum output rate of $1/0.8$ is only achieved when all servers are busy. In other words, when there are less than s customers in the system, the system *slows down* and hence on average there are more customers in the system.

```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats

def t_mean_confidence_interval(data,alpha):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1-alpha/2, n-1)*se
    return m, m-h, m+h

ZSimRNG = SimRNG.InitializeRNSeed()

Queue = SimClasses.FIFOQueue()
Wait = SimClasses.DTStat()
Server = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheDTStats.append(Wait)
TheQueues.append(Queue)
TheResources.append(Server)

ServerNum = 1
Server.SetUnits(ServerNum)
MeanTBA = 1.0
MeanST = 0.8 * ServerNum
Phases = 3
RunLength = 55000.0
WarmUp = 5000.0

AllWaitMean = []
AllQueueMean = []
AllQueueNum = []
AllServerBusyMean = []
print ("Rep", "Average Wait", "Average Number in Queue", "Number Remaining in
    Queue", "Average Server Busy")

def Arrival():
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    Customer = SimClasses.Entity()
    Queue.Add(Customer)

    if Server.Busy < ServerNum:
        Server.Seize(1)
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
            MeanST,2))
```

```
52
53 def EndOfService():
54     DepartingCustomer = Queue.Remove()
55     Wait.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
56     # if there are customers waiting
57     if Queue.NumQueue() >= ServerNum:
58         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
59     else:
60         Server.Free(1)
61
62 for reps in range(0,10,1):
63
64     SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,
           TheResources)
65     SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
66     SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
67     SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)
68
69     NextEvent = Calendar.Remove()
70     SimClasses.Clock = NextEvent.EventTime
71     if NextEvent.EventType == "Arrival":
72         Arrival()
73     elif NextEvent.EventType == "EndOfService":
74         EndOfService()
75     elif NextEvent.EventType == "ClearIt":
76         SimFunctions.ClearStats(TheCTStats,TheDTStats)
77
78     while NextEvent.EventType != "EndSimulation":
79         NextEvent = Calendar.Remove()
80         SimClasses.Clock = NextEvent.EventTime
81         if NextEvent.EventType == "Arrival":
82             Arrival()
83         elif NextEvent.EventType == "EndOfService":
84             EndOfService()
85         elif NextEvent.EventType == "ClearIt":
86             SimFunctions.ClearStats(TheCTStats,TheDTStats)
87
88
89     AllWaitMean.append(Wait.Mean())
90     AllQueueMean.append(Queue.Mean())
91     AllQueueNum.append(Queue.NumQueue())
92     AllServerBusyMean.append(Server.Mean())
93     print (reps+1, Wait.Mean(), Queue.Mean(), Queue.NumQueue(), Server.Mean())
94
95 # output results
96 print("Estimated␣Expected␣Average␣wait:",t_mean_confidence_interval(
       AllWaitMean,0.05))
97 print("Estimated␣Expected␣Average␣queue-length:", t_mean_confidence_interval(
       AllQueueMean,0.05))
98 print("Estimated␣Expected␣Average␣#␣of␣servers␣busy:",
       t_mean_confidence_interval(AllServerBusyMean,0.05))
```

Problem 5. (15 Pts.) (Chapter 4, Exercise 5) Modify the PythonSim event-based simulation of the $M/G/1$ queue to simulate a $M/G/1/c$ retrial queue. This means that customers who arrive to find $c < \infty$ customers in the system (including the customer in service) leave immediately, but arrive again after an exponentially distributed amount of time with mean MeanTR. Do we need the arrival rate λ to be smaller than service rate $1/\tau$ for the system to reach steady-state? Explain your answer using numerical evidence from the simulation model.

HINT: The existence of retrial customers should not affect the arrival process for first-time arrivals.

(15 Points) To model retrials we create another event called "Retrial" with logic similar to the "Arrival" event except that unlike in the Arrival event we do not schedule the next arrival everytime a Retrial event occurs. Arriving customers who find the system with c or more customers, regardless of whether they are original arrivals or retrials, schedule a Retrial event in the calendar. We do not need the arrival rate λ to be smaller than service rate $1/\tau$ for the system to reach steady-state given that the number of customers in the system cannot exceed $c < \infty$. In the code below, we set $\lambda = \tau = 1$ and $c = 3$ and see that the estimated average queue length is close to 3. In fact, for $\lambda \geq 1/\tau$ in the retrial queue, the average queue length is at most c.

```
# M/G/1/c queue
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats

def t_mean_confidence_interval(data,alpha):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1-alpha/2, n-1)*se
    return m, m-h, m+h

Clock = 0.0
ZSimRNG = SimRNG.InitializeRNSeed()

Queue = SimClasses.FIFOQueue()
Wait = SimClasses.DTStat()
Server = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheDTStats.append(Wait)
TheQueues.append(Queue)
TheResources.append(Server)

Server.SetUnits (1)
MeanTBA = 1.0
MeanST = 1.0
Phases = 3
```

```
36 MeanTR = 2
37 RunLength = 55000.0
38 WarmUp = 5000.0
39 c = 3
40
41 AllWaitMean = []
42 AllQueueMean = []
43 AllServerMean = []
44
45 def Arrival():
46     SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
47     Customer = SimClasses.Entity()
48     if Queue.NumQueue() == c:
49         SimFunctions.Schedule(Calendar, "Retrial", SimRNG.Expon(MeanTR, 1))
50     elif Server.Busy == 0:
51         Queue.Add(Customer)
52         Server.Seize(1)
53         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
54     else:
55         Queue.Add(Customer)
56
57 def Retrial():
58     Customer = SimClasses.Entity()
59     if Queue.NumQueue() == c:
60         SimFunctions.Schedule(Calendar, "Retrial", SimRNG.Expon(MeanTR, 1))
61     elif Server.Busy == 0:
62         Queue.Add(Customer)
63         Server.Seize(1)
64         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
65     else:
66         Queue.Add(Customer)
67
68 def EndOfService():
69     DepartingCustomer = Queue.Remove()
70     Wait.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
71     if Queue.NumQueue() > 0:
72         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
73     else:
74         Server.Free(1)
75
76 for reps in range(0,10,1):
77     SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,
           TheResources)
78
79     SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
80     SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
81     SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)
82
83     NextEvent = Calendar.Remove()
84     SimClasses.Clock = NextEvent.EventTime
85     if NextEvent.EventType == "Arrival":
```

```
86              Arrival()
87          elif NextEvent.EventType == "EndOfService":
88              EndOfService()
89          elif NextEvent.EventType == "ClearIt":
90              SimFunctions.ClearStats(TheCTStats,TheDTStats)
91
92          while NextEvent.EventType != "EndSimulation":
93              NextEvent = Calendar.Remove()
94              SimClasses.Clock = NextEvent.EventTime
95              if NextEvent.EventType == "Arrival":
96                  Arrival()
97              if NextEvent.EventType == "Retrial":
98                  Retrial()
99              elif NextEvent.EventType == "EndOfService":
100                 EndOfService()
101             elif NextEvent.EventType == "ClearIt":
102                 SimFunctions.ClearStats(TheCTStats,TheDTStats)
103
104         AllWaitMean.append(Wait.Mean())
105         AllQueueMean.append(Queue.Mean())
106         AllServerMean.append(Server.Mean())
107         print (reps+1, Wait.Mean(), Queue.Mean(), Server.Mean())
108
109 # output results
110 print("Estimated␣Expected␣Average␣wait:",t_mean_confidence_interval(
        AllWaitMean,0.05))
111 print("Estimated␣Expected␣Average␣queue-length:", t_mean_confidence_interval(
        AllQueueMean,0.05))
112 print("Estimated␣Expected␣Average␣#␣of␣servers␣busy:",
        t_mean_confidence_interval(AllServerMean,0.05))
```

```
Estimated Expected Average wait:  (2.9591, 2.9363, 2.9819)
Estimated Expected Average queue-length:  (2.9520, 2.9285, 2.9755)
Estimated Expected Average # of servers busy:  (0.9975, 0.9957, 0.9993)
```

Problem 6. (20 Pts.) A long-term care home in Toronto is planning to test all of its 170 staff (nurses and caregivers) before they start their shift using the new COVID-19 Rapid Test. Consider the 7AM shift and assume that staff arrive between 6:30AM and 7AM for their shift with arrival times uniformly distributed during the 30-minute period prior to the shift. Test times take on average 15 minutes and can be modeled using an Erlang distribution with 4 phases.

(a) Provide a 95% Confidence Interval for the expected average waiting time (excluding the test time) of staff when there are 10 parallel servers (testing stations) available. Assume that staff wait in a single First-Come, First-Served queue before being tested. Use 100 replications.

(b) Approximately how many servers are required to keep the average waiting time below 15 minutes?

HINT: You may want to schedule the arrival events for all 170 staff to the calendar at the beginning of each replication.

(a) **(15 Points)** We first schedule the arrival events for all 170 staff to the calendar at the beginning of each replication by generating uniformly distributed arrival times between 0 and 30. We also create

a variable (test) to count the number of tests completed and use it to terminate our simulation once all staff are tested. To compute the waiting times, if a staff arrives and a testing station is idle, then we record the waiting time of zero, make a testing station busy, and schedule the end of the test; if all testing stations are busy, the staff will be added to the queue. When a test is complete, we increase the 'test' variable by one and check the queue to see if it is empty or not. If not, the next staff in the queue is removed from the queue, its waiting time is recorded, and its departure time is scheduled. Otherwise, i.e., if no staff is in the queue, a testing station will become idle. Using 100 replications, the estimated expected average waiting time of staff is 104.58 minutes with the 95% CI of [103.48, 105.68]

(b) **(5 Points)** We repeat the simulation by gradually increasing the number of servers. We observe that at least 37 servers are needed to say with 95% confidence that the average waiting time of staff is below 15 minutes.

```
# COVID-19 Rapid Test
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats


def CI_95(data):
    a = np.array(data)
    n = len(a)
    m = np.mean(a)
    var = ((np.std(a))**2)*(n/(n-1))
    hw = 1.96*np.sqrt(var/n)
    return m, [m-hw,m+hw]


Clock = 0.0
ZSimRNG = SimRNG.InitializeRNSeed()

Queue = SimClasses.FIFOQueue()
Wait = SimClasses.DTStat()
Server = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheDTStats.append(Wait)
TheQueues.append(Queue)
TheResources.append(Server)

ServerNum = 37
Server.SetUnits(ServerNum)
MeanST = 15.0
Phases = 4
StaffNum = 170
```

```
AllWaitMean = []


def Arrival():
    if Server.Busy < ServerNum:
        Wait.Record(0)
        Server.Seize(1)
        SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Erlang(Phases,
            MeanST, 2))
    else:
        Staff = SimClasses.Entity()
        Queue.Add(Staff)


def EndOfService():
    global  test
    test += 1
    if Queue.NumQueue() > 0:
        NextStaff = Queue.Remove()
        Wait.Record(SimClasses.Clock - NextStaff.CreateTime)
        SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Erlang(Phases,
            MeanST, 2))
    else:
        Server.Free(1)


for reps in range(0, 100, 1):
    SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats,
        TheResources)

    # Schedule the arrival events for all 170 staff
    for i in range(StaffNum):
        SimFunctions.Schedule(Calendar, "Arrival", SimRNG.Uniform(0, 30, 1))

    # To keep track of the number of tests completed
    test = 0
    while test < StaffNum:
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "EndOfService":
            EndOfService()

    AllWaitMean.append(Wait.Mean())

# output results
print("Estimated␣expected␣average␣waiting␣time:", CI_95(AllWaitMean))
```

```
Estimated expected average waiting time:  (13.99539759361021, [13.740448137227974,
14.250347049992447])
```