Stochastic Simulation (MIE1613H) - Homework 2 (Solutions)

Due: February 24, 2022

Problem 1. (10 Pts.) Use the Lindely's recursion simulation of the $M/G/1$ queue to provide an estimate (including a 95% CI) for $E[Y_{10}]$, i.e., the expected waiting time of the 10th customer to enter the system. Write down a mathematical expression for your proposed estimator and state whether it is biased or not. Keep the parameters the same as in the example from the lecture.

Let $Y_{10}^1, Y_{10}^2, \ldots, Y_{10}^n$ be n i.i.d. samples of Y_{10}. We propose the sample average $\bar{Y}_{10}^n = \frac{1}{n}\sum_{i=1}^n Y_{10}^i$ to estimate $E[Y_{10}]$. This is an unbiased estimator of $E[Y_{10}]$ given that

$$E[\bar{Y}_{10}^n] = E\left[\frac{1}{n}\sum_{i=1}^n Y_{10}^i\right] = \frac{1}{n}E\left[\sum_{i=1}^n Y_{10}^i\right] = \frac{1}{n}nE[Y_{10}] = E[Y_{10}].$$

From the simulation code below, the estimate of the expected waiting time of the 10th customer is 1.157 with the 95% CI of [1.132, 1.182].

```
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), np.std(a, ddof=1)
    h = stats.t.ppf(1 - alpha / 2, n - 1) * se / np.sqrt(n)
    return m[0], "+/-", h[0]

m = 10  # the 10th customer
MeanTBA = 1.0  # average interarrival time
MeanST = 0.8  # average service time
np.random.seed(1)
Wait10th = []
for Rep in range(0, 10000):
    Y = 0
    for i in range(1, m, 1):
        A = np.random.exponential(MeanTBA, 1)
        X = np.sum(np.random.exponential(MeanST / 3, 3))
        Y = max(0, Y + X - A)
    # record the waiting time of the 10th customer
    Wait10th.append(Y)
print("CI for the average waiting time of the 10th customer:",
      t_mean_confidence_interval(Wait10th, 0.05))
```

```
CI for the average waiting time of the 10th customer:  (1.1568, '+/-', 0.0252)
```

Problem 2. (20 Pts.) 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). An up-and-out call option with barrier B and strike price K has payoff

$$I\left\{\max_{0\le t\le T} X(t) < B\right\}(X(T)-K)^+,$$

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

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

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 $B = 60, 65, 70$ 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.

To estimate the expected payoff, we define a binary variable 'active' initialized with value 1 and check whether the price has gone above the barrier every time 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 $B = 60, 65, 70$ is 0.049 , 0.318, and 0.828 with the corresponding 95% CIs of [0.046, 0.053], [0.305, 0.330], and [0.805, 0.851], respectively. Increasing barrier results in higher estimated expected payoff since it can generate positive payoff if the price does not go above the barrier which in turn increases 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
```

```
B = 70

TotalValue = []

for i in range(0,Replications,1):
    X = InitialValue
    # binary variable; set to 0 if the value of asset falls below barrier
    active = 1
    for j in range(0,Steps,1):
        Z = SimRNG.Normal(0,1,12)
        X = X * math.exp((InterestRate - Sigma2) * Interval + Sigma * math.
            sqrt(Interval) * Z)
        if X >= B:
            active = 0
    Value = math.exp(-InterestRate * Maturity) * max(X - StrikePrice, 0)
    if active == 1:
        TotalValue.append(Value)
    else:
        TotalValue.append(0)

print ("Barrier option:",CI_95(TotalValue))
```

```
Barrier option:  (0.8282352227504817, [0.8050533856155289, 0.8514170598854345])
```

Problem 3. (20 Pts.) 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 average service time at $\tau = 0.8$, and set the arrival rate to the number of servers, i.e., $\lambda = s$. Simulate the system for $s = 10, 20, 30$. Use 10 replications with run-length of 5500, and set the warmup to 500.

(a) Report the estimated steady-state expected system time, and expected utilization (average number of busy servers divided by the number of servers) in each case.

(b) Compare the results and state clearly what you observe. What you're doing is investigating the impact of system size on performance.

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) The estimates are summarized below:

	$s = 10$	$s = 20$	$s = 30$
Expected average wait	0.91	0.84	0.82
Expected utilization	0.80	0.80	0.80

(b) We observe that as the system size increases, the expected system time decreases while the utilization remains unchanged. The total time in system appears to be converging to the average

service time, which means that as the size of the system grows to infinity the queueing time goes to zero. This illustrates the economies of scale for large systems: a large system can provide short waiting times while maintaining a high utilization of servers.

```
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 = 30
Server.SetUnits(ServerNum)
MeanTBA = 1/30.0
MeanST = 0.8
Phases = 3
RunLength = 5500.0
WarmUp = 500.0

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:
```

```
50          Server.Seize(1)
51          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()/ServerNum)
93      print (reps+1, Wait.Mean(), Queue.Mean(), Queue.NumQueue(), Server.Mean()/
            ServerNum)
94
95  # output results
96  print("Estimated␣Expected␣Average␣wait:",t_mean_confidence_interval(
        AllWaitMean,0.05))
97  print("Estimated␣Expected␣Average␣#␣of␣servers␣busy:",
        t_mean_confidence_interval(AllServerBusyMean,0.05))
```

Problem 4. (20 Pts.) Modify the PythonSim event-based simulation of the $M/G/1$ queue to allow for customer returns. This means that after finishing service, a customer may return to the system with probability p and after a random amount of time which is exponentially distributed with mean $1/\gamma$. Assume that return probability is $p = 0.1$ and expected time to return is $1/\gamma = 2$ and provide an estimate for the expected steady-state number of customers in system (waiting in queue or in service). For other parameters use the same values as in the $M/G/1$ example.

To model returns, we create another event called "Return" with logic similar to the "Arrival" event except that unlike in the Arrival event we do not schedule the next arrival every time a Return event occurs. For departing customers, we schedule a Return event with probability p in the calendar, regardless of whether they are original arrivals or returns. The estimate for the expected number of customers in system is 5.86 with the 95% CI of [5.58, 6.15].

```
# M/G/1 queue with customer returns
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
MeanTR = 2.0
MeanST = 0.8
Phases = 3
p = 0.1
RunLength = 55000.0
```

```
39 WarmUp = 5000.0
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     Queue.Add(Customer)
49     if Server.Busy == 0:
50         Server.Seize(1)
51         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
52
53 def Return():
54     Customer = SimClasses.Entity()
55     Queue.Add(Customer)
56     if Server.Busy == 0:
57         Server.Seize(1)
58         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
59
60 def EndOfService():
61     DepartingCustomer = Queue.Remove()
62     Wait.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
63     # schedule the return of the departing customer with probability p
64     if SimRNG.Uniform(0, 1, 3) <= p:
65         SimFunctions.Schedule(Calendar, "Return", SimRNG.Expon(MeanTR, 1))
66
67     if Queue.NumQueue() > 0:
68         SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,
               MeanST,2))
69     else:
70         Server.Free(1)
71
72 for reps in range(0,10,1):
73     SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,
           TheResources)
74
75     SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
76     SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
77     SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)
78
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     while NextEvent.EventType != "EndSimulation":
```

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

```
Estimated Expected Average wait:  (5.2779, 5.0357, 5.5201)
Estimated Expected Average queue-length:  (5.8638, 5.5796, 6.1480)
Estimated Expected Average # of servers busy:  (0.8893, 0.8857, 0.8928)
```

Problem 5. **(20 Pts.)** (Chapter 4, Exercise 15) The phone desk for a small office is staffed from 8 a.m. to 4 p.m. by a single operator. Calls arrive according to a Poisson process with rate 6 per hour, and the time to serve a call is uniformly distributed between 5 and 12 min. Callers who find the operator busy are placed on hold, if there is space available; otherwise, they receive a busy signal and the call is considered "lost." In addition, 10% of callers who do not immediately get the operator decide to hang up rather than go on hold; they are not considered lost, since it was their choice. Because the hold queue occupies resources, the company would like to know the smallest capacity (the number of callers) for the hold queue that keeps the daily fraction of lost calls under 5%. In addition, they would like to know the long-run utilization of the operator to make sure he or she will not be too busy. Use PythonSim to simulate this system and find the required capacity for the hold queue. Model the callers as class Entity, the hold queue as class FIFOQueue and the operator as class Resource. Use the PythonSim functions Expon and Uniform for random-variate generation. Use class DTStat to estimate the fraction of calls lost (record a 0 for calls not lost and a 1 for those that are lost so that the sample mean is the fraction lost). Use the statistics collected by class Resource to estimate the utilization.

The simulation logic is as follows: The queue only includes the calls that are placed on hold. When a call arrives, we schedule the end of service if the operator is not busy and record a 0 for Lost. Otherwise, we check the number of callers in the hold queue and if there is space available, we record a 0 for Lost and put the call in the hold queue with probability 0.9. If the capacity is reached, we record a 1 for Lost as the call is lost. To control the capacity in the hold queue, we define the *cap* parameter and gradually increase its value until we find the smallest number of callers for the hold queue that keeps the daily fraction of lost calls under 5%. Using 10,000 replications, we observe

with capacity 2 for the hold queue, the expected daily fraction of lost calls is 4.77% with the 95% CI of [4.68, 4.86]. To estimate the long-run utilization of the operator, we add warm-up and run the simulation for a long time. The estimate of utilization is 0.74 with the 95% CI of [0.72, 0.76].

```
# The phone desk
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()
Lost = SimClasses.DTStat()
Server = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

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

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

Server.SetUnits(1)
# The arrival rate is 6 per hour or every 10 minutes
MeanTBA = 10.0
# The office is staffed for 8 hours (480 minutes)
RunLength = 480.0
# Control the capacity in the hold queue
cap = 2

AllLostMean = []

def Arrival():
    SimFunctions.Schedule(Calendar, "Arrival", SimRNG.Expon(MeanTBA, 1))
    if Server.Busy == 0:
        Server.Seize(1)
        SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Uniform(5, 12,
            2))
```

```
49          Lost.Record(0)
50      else:
51          # check to see if there is space available in the hold queue
52          if Queue.NumQueue() <= cap:
53              Lost.Record(0)
54              # 10% of the caller decide to hung up
55              if SimRNG.Uniform(0, 1, 3) < 0.9:
56                  Customer = SimClasses.Entity()
57                  Queue.Add(Customer)
58          else:
59              Lost.Record(1)
60
61  def EndOfService():
62      if Queue.NumQueue() > 0:
63          Queue.Remove()
64          SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Uniform(5, 12,
                2))
65      else:
66          Server.Free(1)
67
68
69  for reps in range(0, 10000, 1):
70      SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats,
            TheResources)
71
72      SimFunctions.Schedule(Calendar, "Arrival", SimRNG.Expon(MeanTBA, 1))
73      SimFunctions.Schedule(Calendar, "EndSimulation", RunLength)
74
75      NextEvent = Calendar.Remove()
76      SimClasses.Clock = NextEvent.EventTime
77      if NextEvent.EventType == "Arrival":
78          Arrival()
79      elif NextEvent.EventType == "EndOfService":
80          EndOfService()
81
82      while NextEvent.EventType != "EndSimulation":
83          NextEvent = Calendar.Remove()
84          SimClasses.Clock = NextEvent.EventTime
85          if NextEvent.EventType == "Arrival":
86              Arrival()
87          elif NextEvent.EventType == "EndOfService":
88              EndOfService()
89
90      AllLostMean.append(Lost.Mean())
91
92  # output results
93  print("Estimated␣Expected␣Lost:", t_mean_confidence_interval(AllLostMean,
        0.05))
```

```
Estimated Expected Lost:  (0.047669, 0.046778, 0.048559)
```

```
1  # The long-run utilization of the operator
2  import SimFunctions
3  import SimRNG
```

```
4  import SimClasses
5  import numpy as np
6  import scipy.stats as stats
7
8
9  def t_mean_confidence_interval(data,alpha):
10     a = 1.0*np.array(data)
11     n = len(a)
12     m, se = np.mean(a), stats.sem(a)
13     h = stats.t.ppf(1-alpha/2, n-1)*se
14     return m, m-h, m+h
15
16
17 Clock = 0.0
18 ZSimRNG = SimRNG.InitializeRNSeed()
19
20 Queue = SimClasses.FIFOQueue()
21 Lost = SimClasses.DTStat()
22 Server = SimClasses.Resource()
23 Calendar = SimClasses.EventCalendar()
24
25 TheCTStats = []
26 TheDTStats = []
27 TheQueues = []
28 TheResources = []
29
30 TheDTStats.append(Lost)
31 TheQueues.append(Queue)
32 TheResources.append(Server)
33
34 Server.SetUnits(1)
35 MeanTBA = 10.0
36 RunLength = 5500.0
37 WarmUp = 500.0
38 cap = 2
39
40 AllServerMean = []
41
42 def Arrival():
43     SimFunctions.Schedule(Calendar, "Arrival", SimRNG.Expon(MeanTBA, 1))
44     if Server.Busy == 0:
45         Server.Seize(1)
46         SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Uniform(5, 12,
               2))
47         Lost.Record(0)
48     else:
49         # check to see if there is space available in the hold queue
50         if Queue.NumQueue() <= cap:
51             Lost.Record(0)
52             # 10% of the caller decide to hung up
53             if SimRNG.Uniform(0, 1, 3) < 0.9:
54                 Customer = SimClasses.Entity()
55                 Queue.Add(Customer)
56         else:
```

```
57                 Lost.Record(1)
58
59 def EndOfService():
60     if Queue.NumQueue() > 0:
61         Queue.Remove()
62         SimFunctions.Schedule(Calendar, "EndOfService", SimRNG.Uniform(5, 12,
               2))
63     else:
64         Server.Free(1)
65
66
67 for reps in range(0, 10, 1):
68     SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats,
           TheResources)
69
70     SimFunctions.Schedule(Calendar, "Arrival", SimRNG.Expon(MeanTBA, 1))
71     SimFunctions.Schedule(Calendar, "EndSimulation", RunLength)
72     SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)
73
74     NextEvent = Calendar.Remove()
75     SimClasses.Clock = NextEvent.EventTime
76     if NextEvent.EventType == "Arrival":
77         Arrival()
78     elif NextEvent.EventType == "EndOfService":
79         EndOfService()
80     elif NextEvent.EventType == "ClearIt":
81         SimFunctions.ClearStats(TheCTStats,TheDTStats)
82
83     while NextEvent.EventType != "EndSimulation":
84         NextEvent = Calendar.Remove()
85         SimClasses.Clock = NextEvent.EventTime
86         if NextEvent.EventType == "Arrival":
87             Arrival()
88         elif NextEvent.EventType == "EndOfService":
89             EndOfService()
90         elif NextEvent.EventType == "ClearIt":
91             SimFunctions.ClearStats(TheCTStats, TheDTStats)
92
93     AllServerMean.append(Server.Mean())
94
95 # output results
96 print("Estimated Long-Run Utilization:", t_mean_confidence_interval(
       AllServerMean, 0.05))
```

```
Estimated Long-Run Utilization:  (0.739, 0.721, 0.758)
```

Problem 6. (10 Pts.) For the AR(1) model discussed in Section 3.3 of the textbook, obtain the asymptotic variance γ^2. **HINT:** Start with expression (5.7) on page 102 of the textbook.

The asymptotic variance for a covariance stationary process is given by (page 102 of the textbook):

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

where $\tilde{\sigma}^2 = \lim_{m\to\infty} Y_m$, and $\rho_k = \lim_{m\to\infty} \mathrm{Corr}(Y_m, Y_{m+k})$.

Note: Here we have used the notation $\tilde{\sigma}$ to avoid confusion with σ (variance of X_i) in the AR(1) model.

For AR(1) from page 27 of the textbook, we have

$$\lim_{m\to\infty} \mathrm{Var}(Y_m) = \frac{\sigma^2}{1-\phi^2},$$

and

$$\lim_{m\to\infty} \mathrm{Corr}(Y_m, Y_{m+k}) = \phi^k.$$

Substituting above and simplifying the asymptotic variance is,

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