Stochastic Simulation (MIE1613H) - Homework 4 (Solutions)

Problem 1. (15 Points) Consider the $M/G/1$ example of Section 4.3. Using a fixed budget of 200,000 samples, a single replication, and batching, provide a 95% confidence interval for the 0.85th and quantile of the steady-state waiting time Y. Determine the warmup using the conservative approach, i.e., by plotting the cumulative average

$$\bar{Y}(j) = \frac{1}{j}\sum_{i=1}^{j} Y_i,$$

and observing when the plot becomes flat. **HINT**: Remember that within-replication samples are not iid.

(15 Points) We will simulate the $M/G/1$ queue using Lindley's recursion. Figure 1 presents the cumulative average of the customer wait times $\bar{Y}(j)$ for $j = 1, \ldots, 50000$ customers. We observe that after 10,000 customers, the cumulative average is roughly constant. Therefore, we use $d = 10000$ as the length of the warm-up period. We then use $K = 25$ batches (any number of batches between 10 and 30 would be acceptable) which divide the remaining 190,000 samples evenly into batches of 7,600 samples. Let θ be the true 0.85-quantile of Y, i.e., $P(Y \leq \theta) = 0.85$. For each batch k, $k = 1, \ldots, K$, we derive the batch quantile as the $(0.85)(7600) = 6460$th sample in the order statistic $Y^k_{(1)} \leq Y^k_{(2)}, \ldots, Y^k_{(7600)}$. The batch quantiles $Y^k_{(6460)}$ are approximately independent and Normally distributed. Thus, we can estimate θ with the sample average of the batch quantiles

$$\hat{\theta} = \frac{1}{K}\sum_{k=1}^{K} Y^k_{(6460)} \approx 4.44,$$

whose 95% CI is given by

$$\hat{\theta} \pm t_{1-0.05/2,K-1}\frac{S(K)}{\sqrt{K}},$$

where

$$S^2(K) = \frac{1}{K-1}\sum_{k=1}^{K}\left(Y^k_{(6460)} - \hat{\theta}\right)^2.$$

Implementing this approach, we obtain the CI $[4.291, 4.590]$.

```
1 import numpy as np
2 import scipy.stats as stats
3 import matplotlib.pyplot as plt
4
5 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, "+/-", h

AllWait = []
AllCumAvg = []

m = 200000
MeanTBA = 1.0 # average interarrival time
MeanST = 0.8 # average service time
np.random.seed(2)

Y = 0
CumAvg = 0
for i in range(m):
    A = np.random.exponential(MeanTBA, 1)
    X = np.sum(np.random.exponential(MeanST/3,3))
    Y = float(max(0, Y + X - A))
    CumAvg = (CumAvg*i + Y ) / (i+1)
    AllCumAvg.append(CumAvg)
    AllWait.append(Y)

# Plot cumulative average
plt.plot(AllCumAvg[:50000])
plt.grid("minor",alpha=0.3)
plt.xlabel(r"$j$")
plt.ylabel(r"$\bar{Y}(j)$")
plt.savefig('Q1_cum_avg.pdf')
plt.show()

# Sample average of batch quantiles
K = 25 # number of batches
d = 10000 # warm-up period
AllWait = AllWait[d:]
b = int((m - d) / K) # batch size
q = int(np.ceil(0.85*b)) # 0.85-quantile index

Allestimates = []
for k in range(K):
    batch_quantile = np.sort(AllWait[b*k:b*(k+1)])[q-1]
    Allestimates.append(batch_quantile)

print(t_mean_confidence_interval(Allestimates, 0.05))
```

```
CI for the expected cost:  (4.440, '+/-', 0.149)
```

Problem 2. A company that sells a single product would like to decide how many items it should have in inventory for each of the next T months (T is a fixed input parameter). The times between demands are IID exponential random variables with a mean of 0.1 month. The sizes of the demands,

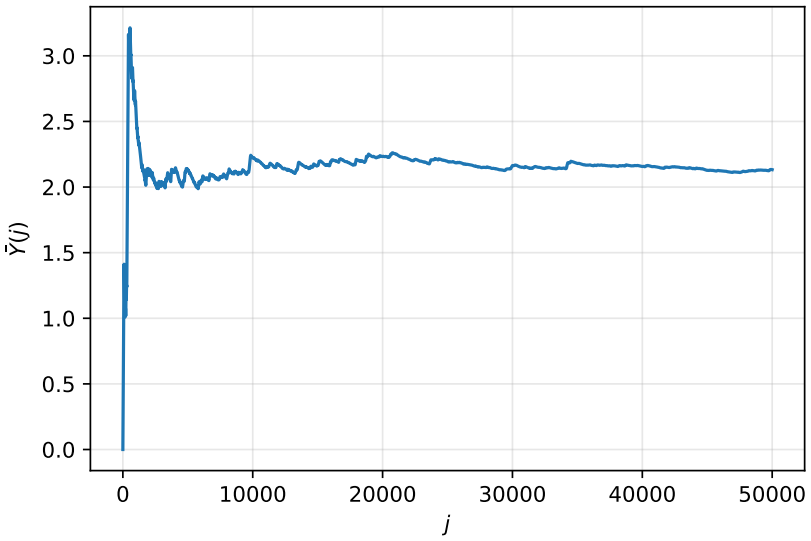

Figure 1: Cumulative average of the customer wait time.

D, are IID random variables (independent of when the demands occur), with

$$D = \begin{cases} 1 & \text{w.p.1/6} \\ 2 & \text{w.p.1/3} \\ 3 & \text{w.p.1/3} \\ 4 & \text{w.p.1/6} \end{cases}$$

where w.p. is short for "with probability".

At the beginning of each month, the company reviews the inventory level and decides how many items to order from its supplier. If the company orders Z items, it incurs a cost of $K + iZ$, where $K = \$32$ is the setup cost and $i = \$3$ is the incremental cost per item ordered. (If $Z = 0$, no cost is incurred.) When an order is placed, the time required for it to arrive (called the delivery lag or lead time) is a random variable that is distributed uniformly between 0.5 and 1 month.

The company uses a (s, S) policy to decide how many to order, i.e.,

$$Z = \begin{cases} S - I, & \text{if } I < s, \\ 0 & \text{if } I \geq s, \end{cases}$$

where I is the inventory level at the beginning of the month.

When a demand occurs, it is satisfied immediately if the inventory level is at least as large as the demand. If the demand exceeds the inventory level, the excess of demand over supply is backlogged and satisfied by future deliveries. (In this case, the new inventory level is equal to the old inventory level minus the demand size, resulting in a negative inventory level.) When an order arrives, it is first used to eliminate as much of the backlog (if any) as possible; the remainder of the order (if any) is added to the inventory.

Let $I(t)$ be the inventory level at time t (note that $I(t)$ could be positive, negative, or zero); let $I^+(t) = \max\{I(t), 0\}$ be the number of items physically on hand in the inventory at time t (note

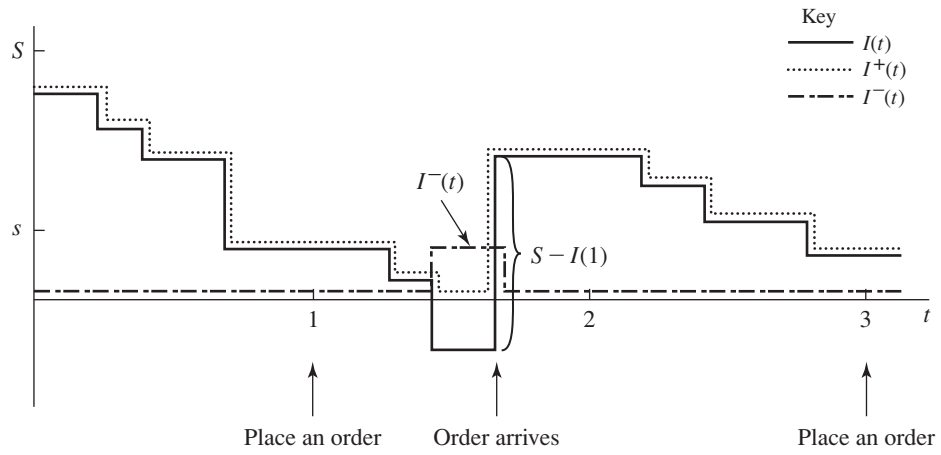

Figure 2: A sample path of $I(t), I^+(t)$, and $I^-(t)$.

that $I^+(t) \geq 0$) and let $I^-(t) = \max\{-I(t), 0\}$ be the backlog at time t (note that $I^-(t) \geq 0$). A sample path of $I(t), I^+(t)$ and $I^-(t)$ is show in Figure 2.

Assume that the company incurs a holding cost of $h = \$1$ per item per month held in (positive) inventory. The holding cost includes such costs as warehouse rental, insurance, taxes, and maintenance, as well as the opportunity cost of having capital tied up in inventory rather than invested elsewhere. Since $I^+(t)$ is the number of items held in inventory at time t, the time-average (per month) number of items held in inventory for the T-month period is

$$\bar{I}^+ = \frac{\int_0^T I^+(t)dt}{T},$$

so the average holding cost per month is $h\bar{I}^+$.

Similarly, suppose that the company incurs a backlog cost of $\pi = \$5$ per item per month in backlog; this accounts for the cost of extra record keeping when a backlog exists, as well as loss of customers' goodwill. The time-average number of items in backlog is

$$\bar{I}^- = \frac{\int_0^T I^-(t)dt}{T},$$

so the average backlog cost per month is $\pi\bar{I}^-$.

Assume that the initial inventory level is $I(0) = 60$ and that no order is out-standing. We are interested in comparing the following nine inventory policies with respect to the total expected costs over $T = 120$ months (which is the sum of the average ordering cost per month, the average holding cost per month, and the average backlog cost per month):

s	20	20	20	20	40	40	40	60	60
S	40	60	80	100	60	80	100	80	100

(a) (20 Points) Develop a simulation model to estimate the expected cost of a given (s, S) policy.

(b) (5 Points) Provide a 95% confidence interval for the expected cost of a policy with $(s, S) = (20, 40)$ using 500 replications.

(c) (15 Points) Implement the subset selection method to identify a subset of "good" policies among the above 9 policies. Use $\alpha = 0.05$ and 100 replications.

Note: Use must use Common Random Numbers (CRN) when comparing the designs (see Page 246 of the textbook). Note that re-calling the method SimRNG.InitializeRNSeed() in PythonSim does not re-initialize the random number generator. To implement CRN, you must either run the scenarios separately and save the outputs; generate the random numbers at the beginning of the simulation from different streams; or make you own modification to the SimRNG code to reset the random number generator.

(a) (15 Points) There are three types of event that can happen over $T = 120$ months. Demands arrive according to a Poisson distribution with the rate $\lambda = 10$, where the sizes of the demands are IID random variables with the probability mass function explained in the question. After determining the size of demand and satisfying it with the on hand inventory at time t, the inventory level is updated that could be positive, negative, or zero and $I^+(t)$ and $I^-(t)$ are recorded. Reviews for Inventory level are another type of events scheduled at the beginning of each month. During each review, if the inventory level is below s, an order with the size $S - I$ is placed and its delivery time is scheduled to the calendar and its ordering cost is recorded. When the order is delivered at time t, the inventory level is updated and $I^+(t)$ and $I^-(t)$ are recorded. The code is in part (b).

(b) (5 Points)

```
1  import SimFunctions
2  import SimRNG
3  import SimClasses
4  import scipy.stats as sp
5  import numpy as np
6  import pandas as pd
7
8  def mean_confidence_interval(data, confidence=0.95):
9      a = 1.0*np.array(data)
10     n = len(a)
11     m, se = np.mean(a), sp.sem(a)
12     h = se * sp.t._ppf((1+confidence)/2., n-1)
13     return m, m-h, m+h
14
15 ZSimRNG = SimRNG.InitializeRNSeed()
16 Calendar = SimClasses.EventCalendar()
17
18 IP = SimClasses.CTStat()
19 IN = SimClasses.CTStat()
20 OrderingCost = SimClasses.DTStat()
21
22 TheCTStats = []
23 TheDTStats = []
24
25 TheCTStats.append(IP)
26 TheCTStats.append(IN)
27 TheDTStats.append(OrderingCost)
28
29 MeanTBA = 0.1
30 h = 1.0    # holding cost
31 b = 5.0    # backlog cost
```

```
32 K = 32.0  # setup cost
33 i = 3.0   # incremental cost
34 s = [20, 20, 20, 20, 40, 40, 40, 60, 60]
35 S = [40, 60, 80, 100, 60, 80, 100, 80, 100]
36 index = 0  # s = 20,  S = 40
37
38 T = 120    # months
39 n = 500    # number of replications
40
41 ExpCost = []
42
43 def Demand():
44     global Inventory
45     SimFunctions.Schedule(Calendar, "Demand", SimRNG.Expon(MeanTBA, 1))
46     U = SimRNG.Uniform(0, 1, 1)
47     if U <= 1/6:
48         D = 1
49     elif U > 1/6 and U <= 1/2:
50         D = 2
51     elif U > 1/2 and U <= 5/6:
52         D = 3
53     else:
54         D = 4
55     Inventory = Inventory - D
56     IP.Record(np.maximum(Inventory, 0))
57     IN.Record(np.maximum(-Inventory, 0))
58
59 def Review():
60     if Inventory < s[index]:
61         Order = SimClasses.Entity()
62         Z = S[index] - Inventory
63         Order.ClassNum = Z
64         SimFunctions.SchedulePlus(Calendar, "Delivery", SimRNG.Uniform(0.5, 1,
               2), Order)
65         OrderingCost.Record(K + i * Z)
66     else:
67         OrderingCost.Record(0)
68
69 def Delivery(Order):
70     global Inventory
71     Inventory = Inventory + Order.ClassNum
72     IP.Record(np.maximum(Inventory, 0))
73     IN.Record(np.maximum(-Inventory, 0))
74
75 for reps in range(0, n, 1):
76     Inventory = 60.0
77     SimFunctions.SimFunctionsInit(Calendar, [], TheCTStats, TheDTStats, [])
78     IP.Record(np.maximum(Inventory, 0))
79     IN.Record(np.maximum(-Inventory, 0))
80     SimFunctions.Schedule(Calendar, "Demand", SimRNG.Expon(MeanTBA, 1))
81     # scheduling the inventory level reviews at the beginning of each month
82     for r in range(T):
83         SimFunctions.Schedule(Calendar, "Review", r + 1)
84     SimFunctions.Schedule(Calendar, "EndSimulation", T)
```

```
85
86      NextEvent = Calendar.Remove()
87      SimClasses.Clock = NextEvent.EventTime
88
89      if NextEvent.EventType == "Demand":
90          Demand()
91
92      while NextEvent.EventType != "EndSimulation":
93          NextEvent = Calendar.Remove()
94          SimClasses.Clock = NextEvent.EventTime
95          if NextEvent.EventType == "Demand":
96              Demand()
97          elif NextEvent.EventType == "Review":
98              Review()
99          elif NextEvent.EventType == "Delivery":
100             Delivery(NextEvent.WhichObject)
101
102     ExpCost.append(IP.Mean() * h + IN.Mean() * b + OrderingCost.Mean())
103
104 Results = pd.DataFrame(ExpCost, columns=["Expected cost of a given policy"])
105
106 print ("CI for the expected cost:", mean_confidence_interval(Results.loc[:,"
    Expected cost of a given policy"]))
```

```
CI for the expected cost:  (125.7388750168428, 125.36557885633417, 126.11217117735143)
```

(c) (15 Points) We modify the code in SimRNG.py by removing the line "ZRNG = InitializeRNSeed()" and adding "ZRNG" as a new input argument to the relevant functions in SimRNG.py, which will be initialized when we run the scenarios. We then simulate for the 9 policies using 100 replications and calculate the sample means and sample variances as shown below and then do subset selection with $K = 9$ using modified thresholds presented in page 246 of the textbook. The subset of "good" policies contains only one policy, which is $(s, S) = (20, 60)$.

s	S	Rep	Sample Mean	Sample Variance
20	40	100	126.09	17.76
20	60	100	119.74	14.78
20	80	100	122.17	10.77
20	100	100	127.84	8.86
40	60	100	126.70	7.39
40	80	100	126.22	5.67
40	100	100	132.51	6.38
60	80	100	144.82	6.07
60	100	100	144.96	4.38

```
1 import SimFunctions
2 import SimRNG_Modified
3 import SimClasses
4 import scipy.stats as stats
5 import numpy as np
6 import pandas as pd
```

```
7
8  MeanTBA = 0.1
9  h = 1.0    # holding cost
10 b = 5.0    # backlog cost
11 K = 32.0   # setup cost
12 i = 3.0    # incremental cost
13
14 T = 120     # months
15 n = 100     # number of replications
16
17 def Demand():
18     global Inventory
19     SimFunctions.Schedule(Calendar, "Demand", SimRNG_Modified.Expon(ZSimRNG,
           MeanTBA, 1))
20     U = SimRNG_Modified.Uniform(ZSimRNG, 0, 1, 1)
21     if U <= 1/6:
22         D = 1
23     elif U > 1/6 and U <= 1/2:
24         D = 2
25     elif U > 1/2 and U <= 5/6:
26         D = 3
27     else:
28         D = 4
29     Inventory = Inventory - D
30     IP.Record(np.maximum(Inventory, 0))
31     IN.Record(np.maximum(-Inventory, 0))
32
33 def Review():
34     if Inventory < s[index]:
35         Order = SimClasses.Entity()
36         Z = S[index] - Inventory
37         Order.ClassNum = Z
38         SimFunctions.SchedulePlus(Calendar, "Delivery", SimRNG_Modified.
               Uniform(ZSimRNG, 0.5, 1, 2), Order)
39         OrderingCost.Record(K + i * Z)
40     else:
41         OrderingCost.Record(0)
42
43 def Delivery(Order):
44     global Inventory
45     Inventory = Inventory + Order.ClassNum
46     IP.Record(np.maximum(Inventory, 0))
47     IN.Record(np.maximum(-Inventory, 0))
48
49 # first do the simulations for the 9 policies
50 ExpCost_list = []
51 for k in range(0, 9):
52     ZSimRNG = SimRNG_Modified.InitializeRNSeed()
53     Calendar = SimClasses.EventCalendar()
54
55     IP = SimClasses.CTStat()
56     IN = SimClasses.CTStat()
57     OrderingCost = SimClasses.DTStat()
58
```

```
59      TheCTStats = []
60      TheDTStats = []
61
62      TheCTStats.append(IP)
63      TheCTStats.append(IN)
64      TheDTStats.append(OrderingCost)
65
66      ExpCost = []
67
68      s = [20, 20, 20, 20, 40, 40, 40, 60, 60]
69      S = [40, 60, 80, 100, 60, 80, 100, 80, 100]
70      index = k  # policy s[k], S[k]
71
72      for reps in range(0, n, 1):
73          Inventory = 60.0
74          SimFunctions.SimFunctionsInit(Calendar, [], TheCTStats, TheDTStats,
                []))
75          IP.Record(np.maximum(Inventory, 0))
76          IN.Record(np.maximum(-Inventory, 0))
77          SimFunctions.Schedule(Calendar, "Demand", SimRNG_Modified.Expon(
                ZSimRNG, MeanTBA, 1))
78          # scheduling the inventory level reviews at the beginning of each
                month
79          for r in range(T):
80              SimFunctions.Schedule(Calendar, "Review", r + 1)
81          SimFunctions.Schedule(Calendar, "EndSimulation", T)
82
83          NextEvent = Calendar.Remove()
84          SimClasses.Clock = NextEvent.EventTime
85
86          if NextEvent.EventType == "Demand":
87              Demand()
88
89          while NextEvent.EventType != "EndSimulation":
90              NextEvent = Calendar.Remove()
91              SimClasses.Clock = NextEvent.EventTime
92              if NextEvent.EventType == "Demand":
93                  Demand()
94              elif NextEvent.EventType == "Review":
95                  Review()
96              elif NextEvent.EventType == "Delivery":
97                  Delivery(NextEvent.WhichObject)
98
99          ExpCost.append(IP.Mean() * h + IN.Mean() * b + OrderingCost.Mean())
100
101     ExpCost_list.append(ExpCost)
102
103 # Subset selection with K=9 designs
104 alpha = 0.05
105 tval = stats.t.ppf(1-alpha/8, n-1)
106 Subset = [1,2,3,4,5,6,7,8,9]
107 for i in range(0,9):
108     for j in range(0,9):
109         if np.mean(ExpCost_list[i]) > (np.mean(ExpCost_list[j])+tval*np.sqrt((
```

```
            n/(n-1))*np.var(np.array(ExpCost_list[i]) - np.array(ExpCost_list[
            j]))/n)):
110             Subset[i] = 0 # exclude from subset
111 print(Subset)
```

Problem 3. Consider the continuous version of the SAN simulation optimization problem with $\tau_j = c_j = 1$, $l_j = 0.5$ for $j = 1, 2, 3, 4, 5$, and $b = 1$.

(a) (15 Pts.) Estimate the gradient of $E[Y(\mathbf{x})]$ at $\mathbf{x} = (0.5, 1, 0.7, 1, 1)$ using **(a)** the Finite Difference (FD) method with $\Delta x = 0.1$ and **(b)** using the Infinitesimal Perturbation Analysis (IPA) approach. Compare the two estimates and their variability using a 95% confidence interval. **Note:** Remember that you must always use Common Random Numbers (CRN) when estimating the gradient using the FD approach.

(c) (15 Pts.) Implement a stochastic approximation search to find the optimal activity mean times. Use $\mathbf{x} = (1, 1, 1, 1, 1)$ as the starting scenario and at least 50 replications for your estimations.

(a) (15 Points) We use 1000 replications and common random numbers to calculate the finite-difference estimator, i.e.,

$$\text{FD}(x_i) = \frac{Y(\mathbf{x} + \Delta x_i) - Y(\mathbf{x})}{\Delta x_i}, \quad \forall i \in \{1, \ldots, 5\}.$$

Similarly, we use 1000 replications to estimate $E\left[\frac{\partial Y(\mathbf{x})}{\partial x_i}\right]$, where $\frac{\partial Y(\mathbf{x})}{\partial x_i} = -\ln(1 - U_i)1_i$ and 1_i is 1 if X_i is on the longest path; 0 otherwise. The table below presents the estimates of the gradient under the two methods along with the 95% half width. We can see that the estimates are quite similar. The code for FD and IPA are below (in order).

Method	Sample Mean	95% Half-Width
FD	(0.811, 0.622, 0.656, 0.440, 0.917)	(0.066, 0.067, 0.066, 0.070, 0.058)
IPA	(0.780, 0.577, 0.711, 0.460, 0.922)	(0.066, 0.068, 0.070, 0.061, 0.069)

```
 1 import SimRNG
 2 import math
 3 import numpy as np
 4 import matplotlib.pyplot as plt
 5
 6 ZRNG = SimRNG.InitializeRNSeed()
 7
 8 Replications = 1000
 9
10 x = [0.5, 1, 0.7, 1, 1]
11 delta = 0.1
12
13 for d in range(5):
14     DList = []  # List to keep the difference
15     x_delta = [0.5, 1, 0.7, 1, 1]
16     x_delta[d] += delta
17     for i in range(0, Replications):
18         U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
19         Y1 = max(-np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3],
20                  -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U[4])*x
                     [4],
```

```
                    -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4])
        Y2 = max(-np.log(1-U[0])*x_delta[0] - np.log(1-U[3])*x_delta[3],
                    -np.log(1-U[0])*x_delta[0] - np.log(1-U[2])*x_delta[2] - np.
                        log(1-U[4])*x_delta[4],
                    -np.log(1-U[1])*x_delta[1] - np.log(1-U[4])*x_delta[4])
        DList.append(Y2-Y1)
    DList = np.array(DList)
    mean = DList.mean()/delta
    std = DList.std()/delta
    upper = mean + 1.96 * std / math.sqrt(Replications)
    lower = mean - 1.96 * std /math.sqrt(Replications)
    print ("Mean", mean)
    print ('Half-Width: ', 1.96*std/math.sqrt(Replications))
```

```
import SimRNG
import math
import numpy as np
import matplotlib.pyplot as plt

ZRNG = SimRNG.InitializeRNSeed()

Replications = 1000

x = [0.5, 1, 0.7, 1, 1]

for d in range(5):
    IPA_List = []
    for i in range(0, Replications):
        U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
        Y1 = -np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3]
        Y2 = -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U[4])*x[4]
        Y3 = -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4]
        Y = max(Y1, Y2, Y3)
        if Y == Y1:
            Path = (0, 3)
        elif Y == Y2:
            Path = (0, 2, 4)
        elif Y == Y3:
            Path = (1, 4)

        I = 0
        if d in Path:
            I = -np.log(1-U[d])
        IPA_List.append(I)

    IPA_List = np.array(IPA_List)
    mean = IPA_List.mean()
    std = IPA_List.std()
    upper = mean + 1.96 * std / math.sqrt(Replications)
    lower = mean - 1.96 * std /math.sqrt(Replications)
    print ("Mean", mean)
    print ('Half-Width: ',(upper-lower)/2)
```

(c) (15 Points) We use IPA method in part (c) to estimate the gradient and implement the recursion $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha_i \hat{\nabla}\theta(\mathbf{x}_i)$. We choose $\alpha_i = 0.05/i$ to ensure a sufficient number of iterations and at each iteration check to see if $\mathbf{x}_{i+1}$ satisfies the conditions, i.e., $\sum_{j=1}^{5} c_j(\tau_j - x_j) \leq b$ and $x_j \geq l_j$. If not, we report the last feasible $\mathbf{x}$ as the final solution of the stochastic approximation search.

```
import SimRNG
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

ZRNG = SimRNG.InitializeRNSeed()

Replications = 1000

def IPA (x):
    mean = []
    for d in range(5):
        IPA_List = []
        for i in range(0, Replications):
            U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
            Y1 = -np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3]
            Y2 = -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U[4])*x
                [4]
            Y3 = -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4]
            Y = max(Y1, Y2, Y3)
            if Y == Y1:
                Path = (0, 3)
            elif Y == Y2:
                Path = (0, 2, 4)
            elif Y == Y3:
                Path = (1, 4)

            I = 0
            if d in Path:
                I = -np.log(1-U[d])
            IPA_List.append(I)

        IPA_List = np.array(IPA_List)
        mean.append(IPA_List.mean())
    return mean

iteration = 1
x_old = [1, 1, 1, 1, 1]
x = x_old - (0.1/iteration)*np.array(IPA(x_old))
while (sum([1, 1, 1, 1, 1] - x) <= 1 and sum(x >= 0.5) == 5):
    iteration += 1
    x_old = x
    x = x_old - (0.05 / iteration) * np.array(IPA(x_old))
print("The estimate of the optimal activity mean times is ", x_old)
```

```
The estimate of the optimal activity mean times is [0.744, 0.860, 0.779, 0.873, 0.745].
```

Problem 4. In the single-period Newsvendor problem, a decision maker needs to decide on the

order size (denoted by x) to satisfy a random demand (denoted by D) with cdf $F(\cdot)$. For each unit of extra inventory left after satisfying the demand, the decision maker incurs a cost of c_o, and for each unit of lost demand (i.e., inventory less than demand) the decision maker incurs a cost of c_u. The objective is to find the order size that minimizes the expected total cost given by:

$$\theta(x) = E[c_o \ \max(x - D, 0) + c_u \ \max(D - x, 0)].$$

(a) (5 Points) Assume that the cdf of the demand D is given by,

$$F(x) = \begin{cases} 1 - (\frac{5}{x})^2, & x \geq 5, \\ 0, & x < 5. \end{cases}$$

Propose an inversion algorithm to generate samples from this distribution.

(b) (10 Points) Write down a linear program (LP) formulation of the sample average approximation method using n replications to find the optimal order size x assuming that x can take any real value between 0 and 50. **Hint**: First write down the sample average approximation formulation and then re-formulate it as a linear program.

(c) (+ 10 Bonus Points) Solve the LP using a solver (e.g., Gurobi) and compare your solution with the exact solution $F^{-1}(\frac{c_u}{c_o+c_u})$.

(a) (5 Points) We denote the inverse cdf by $F^{-1}(.)$. So,

$$F(x) = u \iff x = F^{-1}(u).$$

We have

$$1 - (\frac{5}{x})^2 = u \implies (\frac{5}{x})^2 = 1 - u \implies x = \frac{5}{\sqrt{1-u}}$$

Therefore,

$$F^{-1}(u) = \frac{5}{\sqrt{1-u}}$$

So, the inversion algorithm works as follows:

1. Generate $U \sim U[0, 1]$.
2. Set $X = \frac{5}{\sqrt{1-U}}$ and return X.

(b) (10 Points) Let d_j, $j \in \{1, \ldots, n\}$ be the realization of D in replication j. The SAA formulation is as below:

$$\min_{0 \leq x \leq 50} \frac{1}{n} \sum_{j=1}^{n} [c_o \ \max(x - d_j, 0) + c_u \ \max(d_j - x, 0)].$$

The above formulation can be linearized by introducing new decision variables o_j and u_j for each replication j and adding constraints to ensure they are equal to the max values in the original

objective:

$$\min_x \frac{1}{n}\sum_{j=1}^{n}[c_o\ o_j + c_u\ u_j], \tag{1}$$

$$s.t. \tag{2}$$

$$o_j \geq x - d_j, \quad \forall j \in \{1,\ldots,n\} \tag{3}$$

$$u_j \geq d_j - x, \quad \forall j \in \{1,\ldots,n\} \tag{4}$$

$$o_j \geq 0, \quad \forall j \in \{1,\ldots,n\} \tag{5}$$

$$u_j \geq 0, \quad \forall j \in \{1,\ldots,n\} \tag{6}$$

$$x \leq 50, \tag{7}$$

$$x \geq 0. \tag{8}$$

(c) (10 Points) We first generate 1000 samples of demand using the inversion algorithm proposed in part (a). Then, we assume $c_0 = c_u = 1$ and use Gurobi in Python to solve the corresponding LP model (1)-(8). As observed below, the exact solution is well approximated using the solution from the LP formulation.

```
1  import numpy as np
2  import gurobipy as gp
3  from gurobipy import GRB
4
5  n = 1000
6  D_list = []
7  np.random.seed(1)
8  for j in range(n):
9      U = np.random.random()
10     D = 5/np.sqrt(1-U)
11     D_list.append(D)
12
13
14 def Newsvendor(co, cu, d=D_list):
15     LP = gp.Model("Newsvendor")
16     x = LP.addVar(lb=0, ub=50, name="x")
17     o = LP.addVars(n, lb=0, name="o")
18     u = LP.addVars(n, lb=0, name="u")
19
20     LP.setObjective((1 / n) * sum(co * o[j] + cu * u[j] for j in range(n)),
           GRB.MINIMIZE)
21
22     LP.addConstrs(o[j] >= x - d[j] for j in range(n))
23     LP.addConstrs(u[j] >= d[j] - x for j in range(n))
24
25     LP.Params.OutputFlag = 0
26
27     # Optimize
28     LP.optimize()
29
30     return x.x
31
```

```
co=1
cu=1
print("The LP solution is x =", Newsvendor(co, cu))
print("The exact solution is x =", 5/np.sqrt(1-(co/(co+cu))))
```

The LP solution is $x = 7.121668440991481$
The exact solution is $x = 7.071067811865475$