

# Stochastic Simulation (MIE1613H) - Homework 4 (Solutions)

Due: April 10th, 2022

**Problem 1.** A company would like to decide how many items of a certain product it should keep 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,  $D$ , are IID random variables (independent of when the demands occur), where

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

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 according to a 2-stage Erlang distribution with mean 0.75 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 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 shown in Figure 1.

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

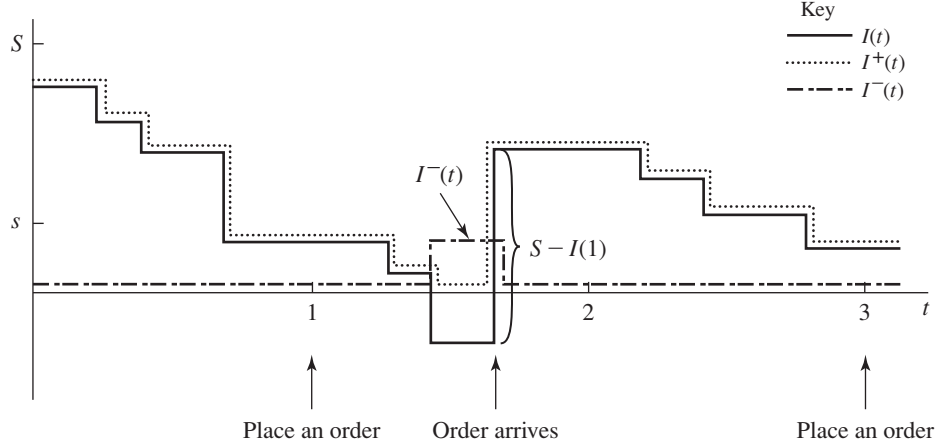


Figure 1: A sample path of  $I(t)$ ,  $I^+(t)$ , and  $I^-(t)$ .

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 six 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	30	30	30
$S$	50	60	70	60	80	100

**(a) (15 Pts.)** Develop a simulation model in PythonSim to estimate the expected cost of a given  $(s, S)$  policy.

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 Pts.)** Provide an estimate (with a 95% CI) of the expected average cost of an  $(s, S)$  policy with  $s = 20$  and  $S = 50$ .

---

```

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, 30, 30, 30]
35 S = [50, 60, 70, 60, 80, 100]
36 index = 0  # s = 20, S = 50
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.Erlang(2, 0.75,
65                                 2), Order)
66         OrderingCost.Record(K + i * Z)
67     else:
68         OrderingCost.Record(0)
69
70 def Delivery(Order):
71     global Inventory
72     Inventory = Inventory + Order.ClassNum
73     IP.Record(np.maximum(Inventory, 0))
74     IN.Record(np.maximum(-Inventory, 0))
75
76 for reps in range(0, n, 1):
77     Inventory = 60.0
78     SimFunctions.SimFunctionsInit(Calendar, [], TheCTStats, TheDTStats, [])
79     IP.Record(np.maximum(Inventory, 0))
80     IN.Record(np.maximum(-Inventory, 0))
81     SimFunctions.Schedule(Calendar, "Demand", SimRNG.Expon(MeanTBA, 1))
82     # scheduling the inventory level reviews at the beginning of each month
83     for r in range(T):
84         SimFunctions.Schedule(Calendar, "Review", r + 1)
85         SimFunctions.Schedule(Calendar, "EndSimulation", T)
86
87     NextEvent = Calendar.Remove()
88     SimClasses.Clock = NextEvent.EventTime
89
90     if NextEvent.EventType == "Demand":
91         Demand()
92
93     while NextEvent.EventType != "EndSimulation":
94         NextEvent = Calendar.Remove()
95         SimClasses.Clock = NextEvent.EventTime
96         if NextEvent.EventType == "Demand":
97             Demand()
98         elif NextEvent.EventType == "Review":
99             Review()
100         elif NextEvent.EventType == "Delivery":
101             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: (130.73036038641786, 130.29259904059072, 131.168121732245)

(c) (20 Pts.) Implement the subset selection method to identify a subset of “good” policies among the above 6 policies. Use  $\alpha = 0.05$  and report your results with (1) 100 and (2) 500 replications.

**Note:** Use must use Common Random Numbers (CRN) when comparing the six 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 code to reset the random number generator. This is possible but may require some programming experience.

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 first do the simulations for the 6 policies using 100 replications and calculate the sample means and sample variances as shown below and then do subset selection with  $K = 6$  using modified thresholds presented in page 246 of the textbook. The subset of “good” policies contains 3 policies that are  $(s, S) = (20, 50), (20, 60),$  and  $(30, 60)$ . We next do the subset selection with 500 replications and find  $(s, S) = (20, 50)$  as the only policy remaining in the subset of “good” policies.

$s$	$S$	Rep	Sample Mean	Sample Variance
20	50	100	130.76	25.81
20	50	500	130.73	24.82
20	60	100	131.61	26.67
20	60	500	131.75	29.37
20	70	100	134.76	32.57
20	70	500	134.80	31.12
30	60	100	131.56	14.40
30	60	500	131.75	17.54
30	80	100	138.68	24.93
30	80	500	138.59	25.24
30	100	100	147.54	30.62
30	100	500	148.38	36.59

---

```

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 = 500    # number of replications
16
17 def Demand():
18     global Inventory
19     SimFunctions.Schedule(Calendar, "Demand", SimRNG_Modified.Expon(ZSimRNG,
20                               MeanTBA, 1))
21     U = SimRNG_Modified.Uniform(ZSimRNG, 0, 1, 1)
22     if U <= 1/6:
23         D = 1
24     elif U > 1/6 and U <= 1/2:
25         D = 2
26     elif U > 1/2 and U <= 5/6:
27         D = 3
28     else:
29         D = 4
30     Inventory = Inventory - D
31     IP.Record(np.maximum(Inventory, 0))
32     IN.Record(np.maximum(-Inventory, 0))
33
34 def Review():
35     if Inventory < s[index]:
36         Order = SimClasses.Entity()
37         Z = S[index] - Inventory
38         Order.ClassNum = Z
39         SimFunctions.SchedulePlus(Calendar, "Delivery", SimRNG_Modified.Erlang
40                                   (ZSimRNG, 2, 0.75, 2), Order)
41         OrderingCost.Record(K + i * Z)
42     else:
43         OrderingCost.Record(0)
44
45 def Delivery(Order):
46     global Inventory
47     Inventory = Inventory + Order.ClassNum
48     IP.Record(np.maximum(Inventory, 0))
49     IN.Record(np.maximum(-Inventory, 0))
50
51 # first do the simulations for the 6 policies
52 ExpCost_list = []
53 for k in range(0, 6):
54     ZSimRNG = SimRNG_Modified.InitializeRNSeed()
55     Calendar = SimClasses.EventCalendar()
56
57     IP = SimClasses.CTStat()
58     IN = SimClasses.CTStat()
59     OrderingCost = SimClasses.DTStat()
60
61     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, 30, 30, 30]
69     S = [50, 60, 70, 60, 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                                         [])
76         IP.Record(np.maximum(Inventory, 0))
77         IN.Record(np.maximum(-Inventory, 0))
78         SimFunctions.Schedule( Calendar, "Demand", SimRNG.Modified.Expon(
79             ZSimRNG, MeanTBA, 1))
80         # scheduling the inventory level reviews at the beginning of each
81         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     ExpCost_list.append(ExpCost)
105
106     # Subset selection with k=6 designs
107     alpha = 0.05
108     tval = stats.t.ppf(1-alpha/5, n-1)
109     Subset = [1,2,3,4,5,6]
110     for i in range(0,6):
111         for j in range(0,6):
112             if np.mean(ExpCost_list[i]) > (np.mean(ExpCost_list[j])+tval*np.sqrt((n
113                 /(n-1))*np.var(np.array(ExpCost_list[i]) - np.array(ExpCost_list[j]))

```

```

110         )/n)):
Subset[i] = 0
111 print(Subset)

```

---

**Problem 2.** 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) (20 Pts.)** Estimate the gradient of  $E[Y(\mathbf{x})]$  at  $\mathbf{x} = (0.5, 1, 0.7, 1, 1)$  using the Finite Difference (FD) method. Compare your estimate (together with a 95% confidence interval) for  $\Delta x = 0.1, 0.05, 0.01$  to illustrate the Bias-Variance tradeoff. **Note:** Remember that you must always use Common Random Numbers (CRN) when estimating the gradient using the FD approach.

We use 1000 replications and common random numbers to calculate the finite-difference estimator, i.e.,

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

The table below presents the estimates for different values of  $\Delta x$ . As  $\Delta x$  decreases the bias of the estimator decreases but the variability of the estimator increases, although the magnitude of the increase is not large for the values of  $\Delta x$  selected here.

$\Delta x$	Sample Mean	95% Half-Width
0.1	(0.794, 0.572, 0.625, 0.460, 0.880)	(0.064, 0.066, 0.065, 0.062, 0.066)
0.05	(0.863, 0.569, 0.642, 0.478, 0.869)	(0.068, 0.062, 0.063, 0.064, 0.065)
0.01	(0.840, 0.577, 0.667, 0.444, 0.904)	(0.070, 0.067, 0.068, 0.061, 0.064)

---

```

1 import SimRNG
2 import math
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 ZRNG = SimRNG.InitializeRNSeed()
8
9 Replications = 1000
10
11 x = [0.5, 1, 0.7, 1, 1]
12 DeltaList = [0.1, 0.05, 0.01]
13
14 for delta in DeltaList:
15     for d in range(5):
16         DList = [] # List to keep the difference
17         x_delta = [0.5, 1, 0.7, 1, 1]
18         x_delta[d] += delta
19         for i in range(0, Replications):
20             U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
21             Y1 = max(-np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3],
22                     -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U
23                     [4])*x[4],
24                     -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4])
25             Y2 = max(-np.log(1-U[0])*x_delta[0] - np.log(1-U[3])*x_delta[3],
26                     -np.log(1-U[0])*x_delta[0] - np.log(1-U[2])*x_delta[2] -
27                     np.log(1-U[4])*x_delta[4],

```



```

26         -np.log(1-U[1])*x_delta[1] - np.log(1-U[4])*x_delta[4])
27     DList.append(Y2-Y1)
28     DList = np.array(DList)
29     mean = DList.mean()/delta
30     std = DList.std()/delta
31     upper = mean + 1.96 * std / math.sqrt(Replications)
32     lower = mean - 1.96 * std /math.sqrt(Replications)
33     print ("Mean", mean)
34     print ('Half-Width: _', (upper-lower)/2)

```

---

**(b) (15 Pts.)** Estimate the same gradient from part **(a)** using the Infinitesimal Perturbation Analysis (IPA) approach.

We use 1000 replications to estimate  $E\left[\frac{\partial Y(\mathbf{x})}{\partial x_i}\right]$ , where  $\frac{\partial Y(\mathbf{x})}{\partial x_i} = -\ln(1-U_i)1\{X_i \text{ is on the longest path}\}$ . The estimated sample mean is (0.782, 0.554, 0.605, 0.439, 0.877) with the 95% half-width of (0.064, 0.066, 0.065, 0.061, 0.066).

---

```

1  import SimRNG
2  import math
3  import pandas as pd
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  ZRNG = SimRNG.InitializeRNSeed()
8
9  Replications = 1000
10
11 x = [0.5, 1, 0.7, 1, 1]
12
13 for d in range(5):
14     IPA_List = []
15     for i in range(0, Replications):
16         U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
17         Y1 = -np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3]
18         Y2 = -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U[4])*x[4]
19         Y3 = -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4]
20         Y = max(Y1, Y2, Y3)
21         if Y == Y1:
22             Path = (0, 3)
23         elif Y == Y2:
24             Path = (0, 2, 4)
25         elif Y == Y3:
26             Path = (1, 4)
27
28         I = 0
29         if d in Path:
30             I = -np.log(1-U[d])
31         IPA_List.append(I)
32
33     IPA_List = np.array(IPA_List)
34     mean = IPA_List.mean()
35     std = IPA_List.std()
36     upper = mean + 1.96 * std / math.sqrt(Replications)

```

```

37     lower = mean - 1.96 * std / math.sqrt(Replications)
38     print ("Mean", mean)
39     print ('Half-Width: _', (upper-lower)/2)

```

---

**(c) (+8 Bonus 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.

We use IPA method in part (b) 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  $\alpha_i = 0.1/i$  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.

---

```

1  import SimRNG
2  import math
3  import pandas as pd
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  ZRNG = SimRNG.InitializeRNSeed()
8
9  Replications = 1000
10
11 def IPA (x):
12     mean = []
13     for d in range(5):
14         IPA_List = []
15         for i in range(0, Replications):
16             U = [SimRNG.Uniform(0, 1, 1) for j in range(5)]
17             Y1 = -np.log(1-U[0])*x[0] - np.log(1-U[3])*x[3]
18             Y2 = -np.log(1-U[0])*x[0] - np.log(1-U[2])*x[2] - np.log(1-U[4])*x
19                 [4]
20             Y3 = -np.log(1-U[1])*x[1] - np.log(1-U[4])*x[4]
21             Y = max(Y1, Y2, Y3)
22             if Y == Y1:
23                 Path = (0, 3)
24             elif Y == Y2:
25                 Path = (0, 2, 4)
26             elif Y == Y3:
27                 Path = (1, 4)
28
29             I = 0
30             if d in Path:
31                 I = -np.log(1-U[d])
32             IPA_List.append(I)
33
34         IPA_List = np.array(IPA_List)
35         mean.append(IPA_List.mean())
36     return mean
37
38 iteration = 1
39 x_old = [1, 1, 1, 1, 1]
40 x = x_old - (0.1/iteration)*np.array(IPA(x_old))

```

```

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

```

---

The estimate of the optimal activity mean times is [0.73837738 0.87226008 0.7855981 0.86991975 0.74321194]

**Problem 3.** 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 Pts.) 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.

We denote the inverse cdf by  $F^{-1}(\cdot)$ . 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 Pts.) Propose an Infinitesimal Perturbation Analysis (IPA) estimator for  $d\theta(x)/dx$ .

Let  $Y(x) = c_o \max(x - D, 0) + c_u \max(D - x, 0)$ .

$$d\theta(x)/dx = \lim_{\Delta x \rightarrow 0} E \left[ \frac{Y(x + \Delta x) - Y(x)}{\Delta x} \right] = E \left[ \lim_{\Delta x \rightarrow 0} \frac{Y(x + \Delta x) - Y(x)}{\Delta x} \right] = E \left[ dY(x)/dx \right],$$

where,

$$dY(x)/dx = c_o 1_{\{x > D\}} - c_u 1_{\{x < D\}}.$$

So,  $E[dY(x)/dx]$  can be estimated using  $n$  replications and with common random numbers.

(c) (10 Pts.) 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.

Let  $d_j, j \in \{1, \dots, n\}$  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 as follows:

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

$$s.t. \quad (2)$$

$$o_j \geq x - d_j, \quad \forall j \in \{1, \dots, n\} \quad (3)$$

$$u_j \geq d_j - x, \quad \forall j \in \{1, \dots, n\} \quad (4)$$

$$o_j \geq 0, \quad \forall j \in \{1, \dots, n\} \quad (5)$$

$$u_j \geq 0, \quad \forall j \in \{1, \dots, n\} \quad (6)$$

$$x \leq 50, \quad (7)$$

$$x \geq 0. \quad (8)$$

(d) (+ 8 Bonus Pts.) Solve the LP using a solver (e.g., Gurobi) and compare your solution with the exact solution  $F^{-1}(\frac{c_u}{c_o + c_u})$ .

We first generate 1000 samples of demand using the inversion algorithm proposed in part (a). Then, we assume  $c_o = c_u = 1$  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
32 co=1
33 cu=1
34 print("The_LP_solution_is_x=", Newsvendor(co, cu))
35 print("The_exact_solution_is_x=", 5/np.sqrt(1-(co/(co+cu))))

```

---

The LP solution is  $x = 7.121668440991481$

The exact solution is  $x = 7.071067811865475$