DSO-570 Exam 1 Solutions (Spring 2019)

1. Probability (10 points)

For the following question, you are not expected to compute the numerical answer, but should leave your answer either as mathematical expressions that can be easily calculated, or as Python code that can be executed. For clarity, you should draw a box around the expression containing your final answer.

Suppose that the DSO-570 Midterm is either EASY or HARD. Based on past data, the professor estimates that the probability the exam is EASY is 0.6 and the probability it is HARD is 0.4. Conditional on the difficulty level of the exam, the score of each student is independently and normally distributed with the following parameters.

Exam difficulty	Mean (μ)	Standard Deviation (σ)
EASY	25	4
HARD	18	8

Assume that there are exactly 100 students taking the exam, and that the same difficulty level applies to all students. (If the exam is EASY, then every student's score is $Normal(25, 4)$ distributed; if the exam is HARD, then every student's score is $Normal(18, 8)$ distributed.) For this question, assume that a student's score can take any decimal number, and ignore the issue of negative scores and scores higher than the total number of points.

A) (3 points) Suppose you know that the exam difficulty is EASY, what is the expected number of students who get at least 20 points?

Let F_E be the CDF of a $Normal(25, 4)$ distribution. If the exam is easy, then each student will get at least 20 points with probability $1 - F_E(20)$. Hence, out of 100 students, the expected number who get at least 20 points is $100(1 - F_E(20))$.

```
[1]: # Alternative expression of the answer Python code.
     from scipy.stats import norm
     100*(1-norm(25,4).cdf(20))
```

```
89.43502263331446
```

B) (3 points) Suppose you don't know whether the exam is EASY or HARD, but only the professor's probabilistic estimate as given in the question, what is the expected number of students who get at least 20 points?

Let X denote the number of students who get at least 20 points. Let E and H denote the events that the exam is easy and hard respectively. Based on part A), we know that

$$\mathbb{E}[X|E] = 100(1 - F_E(20)).$$

Simlarly,

$$\mathbb{E}[X|H] = 100(1 - F_H(20)),$$

where F_E is the CDF of a $Normal(25, 4)$ distribution and F_H is the CDF of a $Normal(18, 8)$ distribution. Thus, the desired expectation is

$$\mathbb{E}[X] = P(E)\mathbb{E}[X|E] + P(H)\mathbb{E}[X|H] = 60(1 - F_E(20)) + 40(1 - F_H(20)).$$

```
[2]: # Alternative expression of the answer Python code.
     from scipy.stats import norm
```

```
distE=norm(25,4)
distH=norm(18,8)
60*(1-distE.cdf(20))+40*(1-distH.cdf(20))
```

```
69.71276055267172
```

C) (4 points) After grading the exam, suppose that the professor tells you that **at least** 50 students got 20 points or above, what is your best estimate of the probability that the exam was indeed Easy?

Let X be the number of students who get at least 20. Conditional on the exam being Easy, X is binomial distributed with $n = 100$ and $p = 1 - F_E(20)$. Similarly, conditional on the exam being Hard, X is binomial distributed with $n = 100$ and $p = 1 - F_H(20)$. Let G_E and G_H be the CDFs of these two distributions. Using Bayes rule,

$$P(E|X \geq 50) = \frac{P(X \geq 50|E)P(E)}{P(X \geq 50|E)P(E) + P(X \geq 50|H)P(H)} = \frac{0.6(1 - G_E(49))}{0.6(1 - G_E(49)) + 0.4(1 - G_H(49))}.$$

```
[3]: # Alternative expression of the answer Python code.
     from scipy.stats import binom
     E=binom(100,1-distE.cdf(20))
     H=binom(100,1-distH.cdf(20))
     top=0.6*(1-E.cdf(49))
     bottom=0.6*(1-E.cdf(49))+0.4*(1-H.cdf(49))
     top/bottom
```

```
0.9811528842777348
```

2. Generating Simulated Data using Python (10 points)

Suppose that Ralphs offers promotions on Fritos with probability 0.3 every month, independent from other months. Conditional on whether there is a promotion in a month and on the promotion pattern of previous months, the monthly demand for Fritos is independently and normally distributed with the following parameters. (Assume that demand can take any decimal number and ignore the issue of negative demand.)

Month	Mean (μ)	Standard Deviation (σ)
No promotion	500	100
Promotion	800 + 100t	150

In the above table, t is the number of months without a promotion since the last promotion. For example, if last month had a promotion, then $t = 0$. If the neither of the last two months had a promotion but the month before had one, then $t = 2$.

In the spaces below, write a function called "simulate" with one input argument:

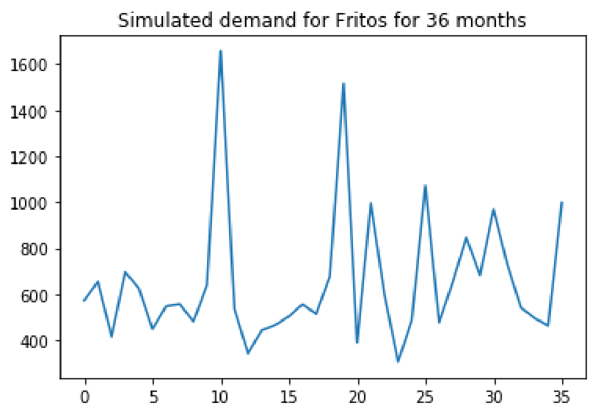
- **n**: the number of months to simulate. (Assume this is at least 1).

The function should return a list of n numbers corresponding to n consecutive months of simulated demand for Fritos. Assume that just prior to the months in the simualtion, there was a promotion, so $t = 0$ in the beginning. You need to import all packages and distributions you use.

```
[4]: from scipy.stats import norm
     import numpy as np
     def simulate(n):
         t=0
         demand=[]
         for i in range(n):
             decision=np.random.choice(['Promote','NoPromote'],p=[0.3,0.7])
             if decision=='Promote':
                 demand.append(norm(800+100*t,150).rvs())
                 t=0
             else:
                 demand.append(norm(500,100).rvs())
                 t+=1
         return demand
```

```
[8]: import pandas as pd
     np.random.seed(0)
     pd.Series(simulate(36)).plot(title='Simulated demand for Fritos for 36 months')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd37ae49710>
```



3. Analyzing Data using Python (10 points)

Suppose that a certain store replenishes its supply of Fritos as follows. There is a certain "base-stock" level z. At the end of every month, if the remaining inventory is x, and $x < z$, then the store orders exactly $z - x$ units, which will arrive at the beginning of the next month prior to store opening. However, assume that every unit of Fritos expires in 2 months, so that any supply that arrives at the beginning of l will need to be discarded at the beginning of month $l + 2$ if it is not sold in months l and $l + 1$. Assume that the store always sells units with the earliest expiration date first. Moreover, assume that in month 0, the initial inventory is equal to z and will last until the beginning of month 2.

Write a function called "analyze" with 2 input parameters:

- **demand**: a non-empty list of non-negative integers, with each element corresponding to the customer demand for Fritos in the corresponding month.
- **z**: the basestock level.

The function should return two numbers:

- **totalLost**: the total amount of customer demand that was not fulfilled from months 0 through $n-1$, due to insufficient inventory. Here, $n =$`len(demand)`.
- **totalDiscarded**: the total number of units that were discarded at the beginning of months 0 through $n-1$.

For example, after writing your function, the code

```
totalLost,totalDiscarded=analyze([30,40,20,100,120],100)
```

should set the variable "totalLost" to 30 and "totalDiscarded" to 40. The following table illustrates the inventory dynamics.

		Beginning			Lost	Remaining	Units
Month	Discarded	Inventory	Demand	Fulfilled	Sales	Inventory	Ordered
0	0	100	30	30	0	70	30
1	0	100	40	40	0	60	40
2	30	70	20	20	0	50	50
3	10	90	100	90	10	0	100
4	0	100	120	100	20	0	100
Total	**40**	-	-	-	**30**	-	-

(**Hint:** For months 0 and 1, the beginning inventory is always 100 because nothing has expired yet. Starting from month 2, the beginning inventory is always the minimum of 100 and the total number of units ordered in the previous two months. In each month, the units discarded is always 100 minus the beginning inventory.)

```
[6]: def analyze(demand,z):
         ordered=[]
         month=0
         totalLost=0
         totalDiscarded=0
         for curDemand in demand:
             if month<2:
                 beginInv=z
             else:
                 beginInv=min(z,ordered[month-1]+ordered[month-2])
             totalDiscarded+=z-beginInv
             if curDemand>beginInv:
                 totalLost+=curDemand-beginInv
                 endInv=0
             else:
                 endInv=beginInv-curDemand
             ordered.append(z-endInv)
             month+=1
         return totalLost,totalDiscarded
```

```
[7]: analyze([30,40,20,100,120],100)
```

```
(30, 40)
```