Practice Midterm B (Solutions)

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.

A certain warehouse makes ready 100 units of a certain product at the beginning of each week to fulfill customer demand. The manager estimates that the market condition each week is either Good or Bad, with the probability of each week being Good equal to 0.3, independent other other weeks. Given the market condition, the demand each week is independently and normally distributed according to the following table.

Market Condition	μ	σ
Good	90	30
Bad	60	30

For this question, assume that demand can be fractional, and ignore the issue that the normal distribution might take negative values.

a) (4 points) Suppose that the warehouse stocks out (demand exceeds 100) in a given week, what is the probability that the market condition is Good?

Let G denote the event that the market condition is good, B the event that the condition is bad, and S the event of stocking out. Let X be a random variable denoting the demand in a week. We have

$$P(S|G) = P(X > 100|G) = 1 - P(X \leq 100|G) = 1 - F_G(100),$$
$$P(S|B) = P(X > 100|G) = 1 - P(X \leq 100|B) = 1 - F_B(100),$$

where F_G and F_B are the CDFs of the two normal distributions under Good and Bad conditions. Hence, by Bayes' rule, the desired probability is

$$P(G|S) = \frac{P(S|G)P(G)}{P(S|G)P(G) + P(S|B)P(B)} = \frac{0.3(1 - F_G(100))}{0.3(1 - F_G(100)) + 0.7(1 - F_B(100))}.$$

The above answer suffices for the exam. You can also express the answer using Python code, as below.

```
[1]: from scipy.stats import norm
     G=norm(90,30)
     B=norm(60,30)
     top=0.3*(1-G.cdf(100))
     bottom=0.3*(1-G.cdf(100))+0.7*(1-B.cdf(100))
     P=top/bottom
     P
```

```
0.6344872894031277
```

b) (3 points) In 52 weeks (approximately a year), what is the expected number of weeks in which the warehouse would stock out?

Let Y be a random variable denoting the number of weeks among 52 weeks of stocking out. We have that conditional on the market condition, Y is binomial distributed. Hence,

$$E[Y|G] = 52(1 - F_G(100)),$$
$$E[Y|B] = 52(1 - F_B(100)).$$

Hence,

$$E[Y] = P(G)E[Y|G] + P(B)E[Y|B] = (0.3)(52)(1 - F_G(100)) + (0.7)(52)(1 - F_B(100)).$$

The above answer suffices for the exam. However, you can also express the answer using Python code, as below.

```
[2]: from scipy.stats import norm
     G=norm(90,30)
     B=norm(60,30)
     M=0.3*52*(1-G.cdf(100))+0.7*52*(1-B.cdf(100))
     M
```

```
9.083373304857103
```

c) (3 points) Suppose that the answer to part a) is P and the answer to part b) is M. Suppose that in a given year, the warehouse only stocked out for 5 out of 52 weeks. What is the probability that the market condition was Good for at least 4 out of the 5 weeks?

Without loss of generality, we can label the 5 weeks in which the warehouse stocked out as $w_1, w_2, \cdots, w_5$. Because market conditions and demand are independent across weeks, knowing that in the other weeks we did not stock out does not give us any information about the market conditions in weeks $w_1, \cdots, w_5$.

Therefore, let random variable Z denote the number of weeks with Good condition out of these 5 weeks. Z is binomial distributed with $n = 5$ and $p = P$. Let F be the CDF of this distribution. The desired answer is

$$P(Z \geq 4) = 1 - P(Z \leq 3) = 1 - F(3).$$

```
[3]: from scipy.stats import binom
     1-binom(5,P).cdf(3)
```

```
0.3990145408342154
```

2. Generating Simulated Data using Python (10 points)

This question asks you to simulate the weekly price of a certain stock, given the initial price, the expected change in price per week, the standard deviation per week, and the number of weeks to simulate.

Write a function called "simulate" with the following input arguments:

- **initial**: a positive number representing the price in week -1.
- **mu**: the expected change in price from one week to the next.
- **sigma**: the standard deviation in the change in price.
- **N**: the number of weeks to simulate.

The function should return a Pandas Series containing the simulated price of the stock from weeks 0 through $N - 1$. (Note that the initial price in week -1 is not included in the returned Series.)

You should assume the following probabilistic model. Suppose the price in week $t-1$ is p_{t-1}, then the price in week t is given by

$$p_t = \begin{cases} 0 & \text{if } p_{t-1} = 0, \\ \max(0, p_{t-1} + \epsilon_t) & \text{if } p_{t-1} > 0, \end{cases}$$

where ϵ_t is independently and Normally distributed with mean **mu** and standard deviation **sigma**, as given by the input arguments.

```
[20]: from scipy.stats import norm
      import pandas as pd
      def simulate(initial,mu,sigma,N):
          price=initial
          data=[]
          dist=norm(mu,sigma)
          for i in range(N):
              if price>0:
                  price=max(0,price+dist.rvs())
              data.append(price)
          return pd.Series(data)
```

The following output shows the result of running the code with initial price of 1, mu=0.01, sigma=0.2, and N=5 weeks.

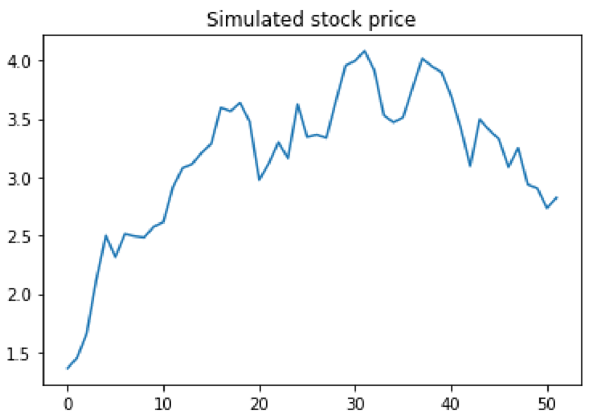
```
[21]: import numpy as np
      np.random.seed(0)
      simulate(1,0.01,0.2,5)
```

```
0    1.362810
1    1.452842
2    1.658590
3    2.116768
4    2.500280
dtype: float64
```

The following plots one random sequence of 52 weeks.

```
[17]: np.random.seed(0)
      simulate(1,0.01,0.2,52).plot(title='Simulated stock price')
```

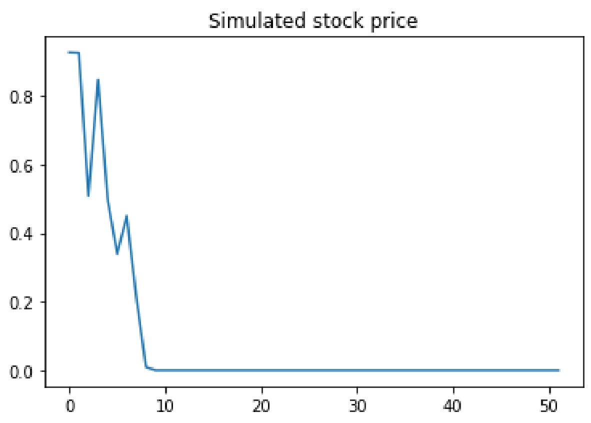
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7b17bf898>
```



The following plots another random sequence and illustrates that once price hits zero, it stays there forever.

```
[18]: np.random.seed(2)
      simulate(1,0.01,0.2,52).plot(title='Simulated stock price')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7b179bda0>
```



3. Analyzing Data using Python (10 points)

This question asks you to evaluate an algorithmic trading strategy using a time series of stock prices.

Write a function called "analyze" with the following input arguments:

- **prices**: a non-empty list of stock prices, with each entry representing the price of the stock in a given period. You can assume that all given prices are strictly positive.
- **cash**: a positive decimal number designating the total amount of money you have at the beginning of all periods.
- **p_L**: a threshold on the price to purchase. As soon as the stock price is less than or equal to this price, you would purchase as many shares as you can using all the cash you have. (**Note: you can only purchase an integer number of shares.** You can round down a number n using the code `int(n)`. i.e., `int(3.6)` would yield 3.)
- **p_H**: a threshold on the price to sell. As soon as the stock price is greater than or equal to this price, you would sell all the shares you have and convert it to cash.

Assume that at the beginning, you have zero shares of the stock. Moreover, in the last week, you have to sell all the shares you have regardless of the price. **The function should return the total amount of cash you have at the end of all periods.**

Examples:

Suppose that `prices=[1.1,1.3,0.8,1.3,1.6,0.4]`. Then

- `analyze(prices,8,0.8,1.6)` yields 16.0 because you would buy 10 shares when the price hits 0.8 and sell then for 1.6 each when the price reaches 1.6.
- `analyze(prices,8,0.8,1.7)` yields 4.0 because while the sell price of 1.7 is never reached, you are stuck with 10 shares, which you are forced to sell in the last period for 0.4 each.
- `analyze(prices,10,0.8,1.6)` yields 19.6 because you buy 12 shares when the price reaches 0.8, with 0.4 cash left, and you sell all 12 shares at a price of 1.6. Your ending cash is $0.4 + 1.6 \times 1.2 = 1.96$.

- `analyze(prices,8,0.7,1.6)` yields 8.0 because the price is never low enough for you to buy, so you end up with the same amount of cash you had at the beginning.
- `analyze(prices,8,0.8,1.2)` yields 13.0 because you buy the 10 shares when the price hits 0.8 and sell everything when the price hits 1.3.

Write your code in the spaces below:

```
[64]: def analyze(prices,cash,p_L,p_H):
          shares=0
          for p in prices:
              if p<=p_L:
                  bought=int(cash/p)
                  shares+=bought
                  cash-=p*bought
              elif p>=p_H:
                  cash+=p*shares
                  shares=0
          cash+=p*shares
          shares=0
          return cash
```

```
[65]: prices=[1.1,1.3,0.8,1.3,1.6,0.4]
      analyze(prices,8,0.8,1.6)
```

```
16.0
```

```
[66]: analyze(prices,8,0.8,1.7)
```

```
4.0
```

```
[30]: analyze(prices,10,0.8,1.6)
```

```
19.6
```

```
[38]: analyze(prices,8,0.7,1.6)
```

```
8.0
```

```
[37]: analyze(prices,8,0.8,1.2)
```

```
13.0
```