DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 03:

HW No. 03: Maximum Likelihood vs. Bayesian Estimation

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

February 6th, 2022

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

A. TASK 1

As with any python project, one must first import any necessary libraries. For this task, I imported numpy, pandas, matplotlib, as well as random. To do so, the following snippet was included at the start of my script:

```
# %%import modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
```

At this point, we can start diving into the real work of this homework assignment. The first goal is to generate a set of 11 independent data points with a variance of 1 (from a Gaussian distribution). To do so, I listed all initial variables and created a dictionary that will allow me to format data for use. This dataset has 11 varying means, but all have the same size. The means to be used are 0.90, 0.92, 0.94, 0.96, 0.98, 1.00, 1.02, 1.04, 1.06, 1.08, and 1.10. These means are generated using numpy.arange and are saved in the variable deltrange.

```
# start with generation
mu = 1
var = 1**2
deltrange = np.arange(0.9, 1.1, 0.02)
output = [round(float(x), 2) for x in deltrange]

d = {}
for val in output:
    d["dataframe{0}".format(val)] = np.random.normal(
        loc=val, scale=var, size=1000000)
```

The code above functions by taking the specified means and supplying them to my dataframe generation command that utilizes np.random.normal. The outputted dictionary {d} has a size of 11 and is designed to have the value of the mean of the array that was specified.

Key	Type	Size	Value
dataframe0.9	Array of float64	(1000000,)	[0.12865282 0.69963437 0.45105246 ... 0.00811925 0.53251389 1.45324331 ...
dataframe0.92	Array of float64	(1000000,)	[0.30191303 1.89409288 0.37038235 ... -0.85619807 3.85497854 0.5 ...
dataframe0.94	Array of float64	(1000000,)	[-0.64594556 1.7985222 -0.42816019 ... 1.78108786 1.29069401 1.3 ...
dataframe0.96	Array of float64	(1000000,)	[0.66346672 1.49811455 -0.33543933 ... 2.16268506 -1.04527444 1.6 ...
dataframe0.98	Array of float64	(1000000,)	[1.5514458 2.07477704 0.87509391 ... 1.10293287 0.40060707 1.46120521 ...
dataframe1.0	Array of float64	(1000000,)	[-0.85372284 2.02396418 1.67700647 ... 3.12505353 -0.08371944 -0.3 ...
dataframe1.1	Array of float64	(1000000,)	[1.18946601 1.32574961 0.79302169 ... 2.34136013 2.08842609 1.26090956 ...
dataframe1.02	Array of float64	(1000000,)	[0.96255606 0.04631555 0.44738438 ... 0.76468552 0.5278064 1.49872568 ...
dataframe1.04	Array of float64	(1000000,)	[1.24216922 1.64124455 1.41177088 ... 0.5354751 0.42150015 1.96833663 ...
dataframe1.06	Array of float64	(1000000,)	[3.41859044 2.02318393 1.9571235 ... 1.37510427 0.91998787 0.85652602 ...
dataframe1.08	Array of float64	(1000000,)	[0.44715834 1.36041443 -0.04897185 ... 1.50519621 1.6052356 0.86 ...

This completes Task 1 of the homework assignment as step 1 was to just create 11 independent datasets consisting of 10^6 points with a variance of 1 and varying means.

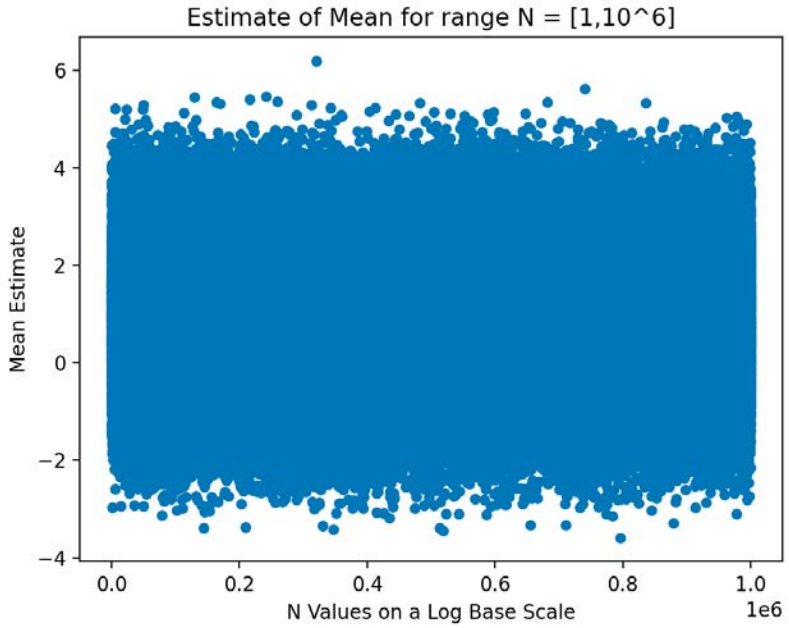
B. TASK 2

Task 2's goal was first to utilize a dataset with a mean of 1.00 and estimate the mean value through the maximum likelihood estimate method. In order to do this, I utilized the same dictionary {d} that was generated in the previous task. Out of pure anxiety, I double checked all of the values in my dictionary to ensure they all had the proper means. I wrote a for-loop that iterates through the name and values of the data frame and calculated the mean by diving the sum of the values by the total length of the array.

```
#%% q1 we need to start with mean 1
#and then perform the max likelihood
#use the log scale
for i,j in d.items(): #name, value
    mean = sum(j)/len(j) #calculate mean
    print("Mean of the dataset with delta value {val} is: {kval}".format(
        val=i, kval=mean)) #print statement for each value
```

The outputted value of the mean using the maximum likelihood estimation for the dataset with an intended mean of 1.00 is 1.00135. This is good as it was exactly as intended. The other datasets follow in that all of their means are incredibly close to the intended value.

The next goal of this task is to plot the estimated mean for the range of N = [1,10^6] using a log base 10 scale. Therefore, I plotted all estimated means for all 1,000,000 data points and utilized a log base scale as required. The outputted graph is as follows:



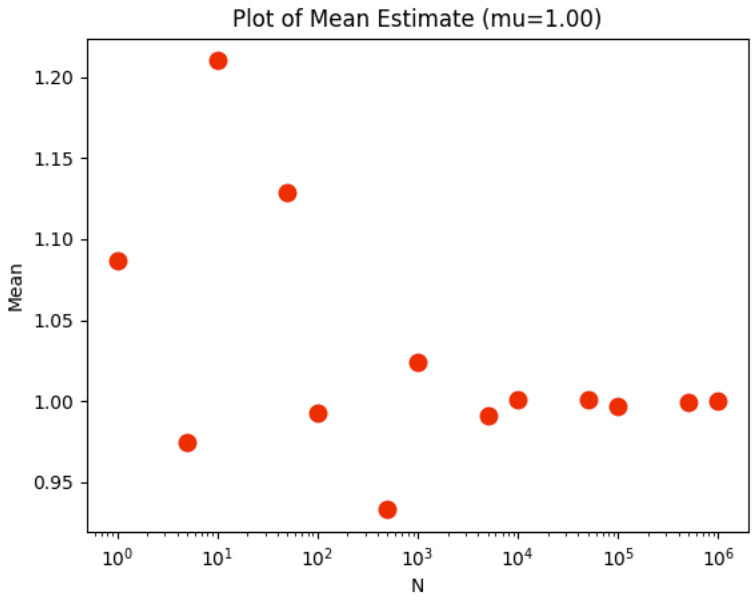
The Task then requires students to compute the mean estimated for varying values of N (1, 5, 10^1, 50, 10^2, …, 10^6). In order to complete this task, I first defined the samples as specified in the homework assignment:

```
# we need to define the samples
samples = [1, 5, 10**1, 50, 10**2, 500, 10**3,
           5000, 10**4, 50000, 10**5, 500000, 10**6]
```

I then created a new dictionary that will allow me to for-loop through both the desired means and the desired sample amount. This will facilitate the plotting of the estimated mean of the data frame with a mean of 1.00 while also accounting for the desired *N* values. For brevity's sake, I have elected not to explain my code line by line but instead include a commented-out version of my code to offer an explanation.

```
distributions = {}#here we have distribution dict
for i in samples:#we need all samp values but only with mean 1
    data = [random.gauss(1,var) for j in range(i)]
    distributions[i] = data

#calculate the mean
mean1 = []#define a list for the mean
for i,data in distributions.items():#name,val
    mean=sum(data)/len(data)#calc mean
    mean1.append(mean)#append

# question 2
plot = list(zip(samples, mean1))#tuples!!
df2 = pd.DataFrame(plot, columns=['n', 'mean'])#make a dataframe

fig, ax = plt.subplots()#plot
ax.scatter(samples,mean1, s=80, color='r')#plot
ax.set_xscale("Log")
plt.title('Plot of Mean Estimate (mu=1.00)')
plt.ylabel('Mean')
plt.xlabel('N')
```

The outputted plot looks as follows and shows our mean converging to 1.00 with an increase in *N* values.



We are then tasked by taking the average of the first *N* of points of the first 6 sets. These plots all vary above and below the desired mean. For example, the first point of the dataset with mean 0.9 is 1.07801 and the first point of the dataset with mean 0.94 is 0.73374. These values are biased because they do not show the true mean of the data. We will only see the true mean of our data as we increase the sample size and allow our data to converge to the true mean.

C. TASK 3

For Task 3, we were to perform the same as Task 2, but with all the previously generated means given by the variable 'sample'. I had to scour a few Stack Overflow forums to figure out how to plot multiple graphs in an easier manner than just brute forcing it. My endeavors were rewarded, and I was able to implement a for-loop that calls each of the unique means of a data frame and plot them sequentially. First, however, I had to create a dictionary that had *all* of the distributions and their corresponding values. A simple nested for-loop can be implemented to do this.

```
#%% god bless stack overflow
all_distributions = {}
for i in output: #these are the delta values
    datasets = {} #empty
    for j in samples: #n values
        key = "{}".format(j) #naming system
        #value = distribution of varying mean and var of 1 for
        #all values in the range of the sample number
        value = [random.gauss(i,var) for k in range(j)]
        dataset[key] = value #key value pair
    all_distributions["mean_{}".format(i)] = dataset
```

Key	Type	Size	Value
mean_0.9	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_0.92	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_0.94	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_0.96	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_0.98	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_1.0	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_1.1	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_1.02	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650 ...
mean_1.04	dict	13	{'1':[0.7730213816021818], '5':[1.7421902935196805, 0.1076149513136650

Here we see that our all_distributions dictionary has all the means requested (13 total values) and that each value has an array of length that matches the *N* value. From here, we want to now calculate each value of the mean for each *N* value. This can be done by following the same syntax that was implemented in Task 2. We iterate through the bigger dictionary, reference the keys and values of the smaller dictionary, and then calculate the mean. The only difference is that at the end, we will create a new pandas dataframe that will have specified column names and will allow us to plot graphs notably easier than otherwise.

```
#%% define means for them
mu_array = []
for name,datasets in all_distributions.items(): #iterate name & values of main dict
    for key, val in datasets.items(): #iterate values and key in the smaller dict
        mean = np.mean(val)
        mu_array.append((name,key,mean))

all_means = pd.DataFrame(mu_array,columns=['mean','n','estimate'])
```

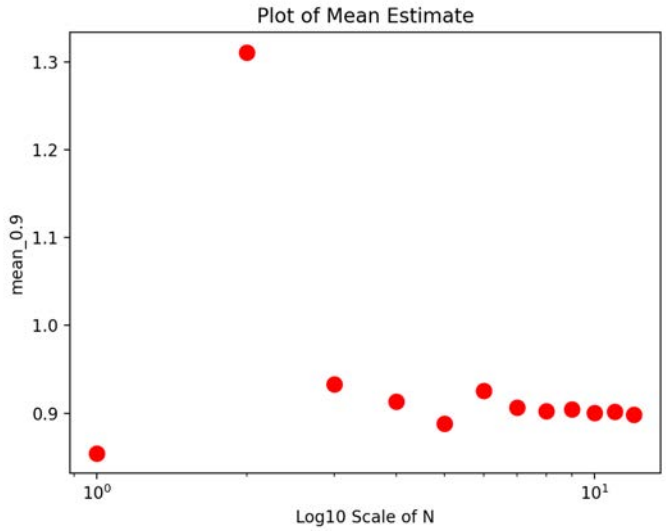
The outputted dataframe will look as such and will be of dimensions 143x3. This is because we are supplying 11 different datasets with 13 varying sample sizes (11 * 13 = 143).

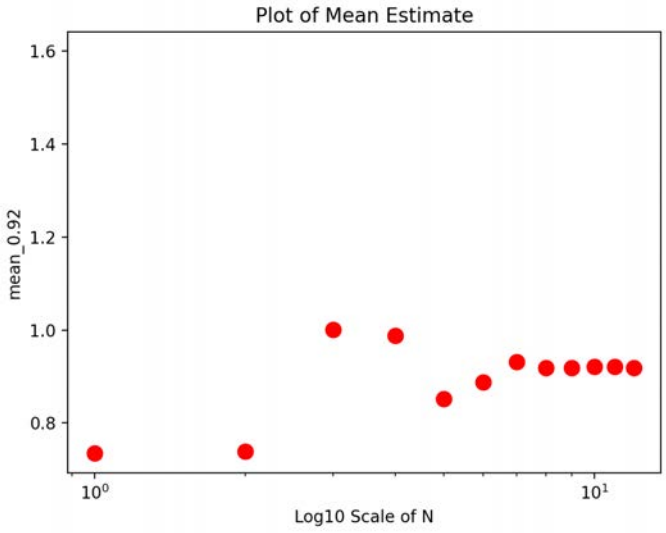
Index	mean	n	estimate
0	mean_0.9	1	1.05905
1	mean_0.9	5	0.854294
2	mean_0.9	10	1.3108
3	mean_0.9	50	0.932814
4	mean_0.9	100	0.913252
5	mean_0.9	500	0.888249
6	mean_0.9	1000	0.925616
7	mean_0.9	5000	0.906362
8	mean_0.9	10000	0.902556
9	mean_0.9	50000	0.904525
10	mean_0.9	100000	0.900061

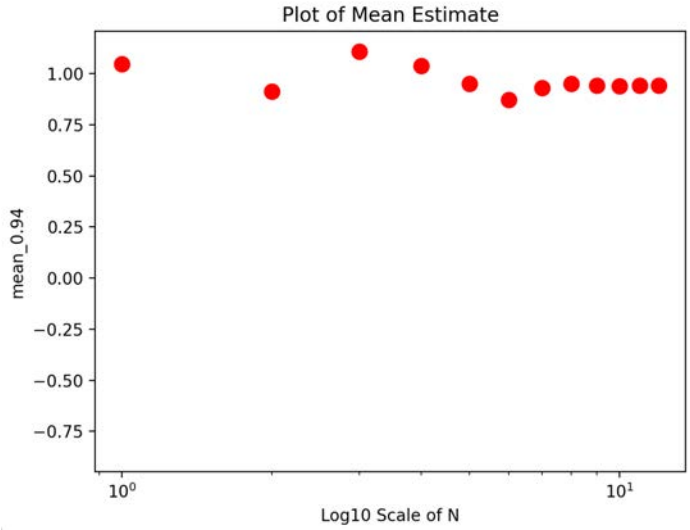
We can then implement the previously mentioned for-loop to plot all of our graphs. Note that the column names are ‘mean’, ‘n’, and ‘estimate.’ We will use these to call on our data by only calling on the unique means. This is done by looked at the ‘mean’ column and selecting all of the unique variables (mean_0.9, mean_0.92, mean_0.94, etc…). Then, the for-loop will iterate through these columns and plot every possible combination of the mean values, the mean estimate, and the sample size.

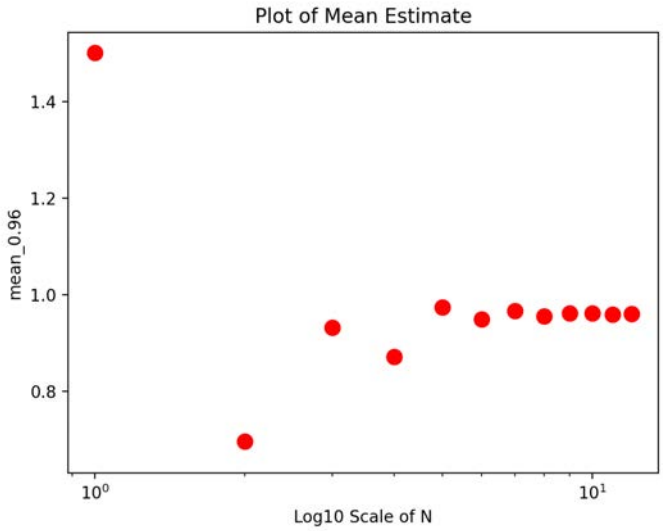
```
#%% we should plot them all:
mean_values = df_mean.iloc[:, 0].unique()

for i in mean_values:
    fig, ax = plt.subplots()
    plot_data = df_mean[df_mean["mean"] == i]
    ax.scatter(plot_data['n'],plot_data['estimate'], s=80, color='r')
    plt.title('Plot of Mean Estimate')
    plt.ylabel('{}'.format(i))
    plt.xlabel('Log10 Scale of N')
    ax.set_xscale('log')
```

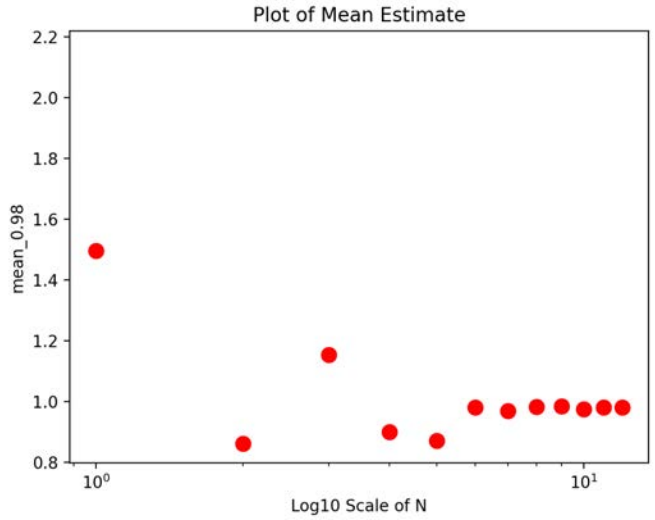
Each of the graphs has been screenshotted and uploaded on the following pages. The mean value is listed on the y-axis and the x-axis are all log10 scale.

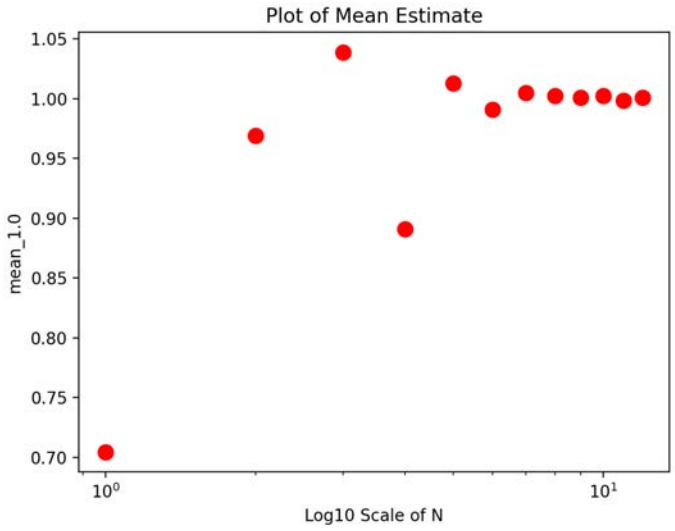

Plot of Mean Estimate
mean_0.9
1.3
1.2
1.1
1.0
0.9
10^0
10^1
Log10 Scale of N

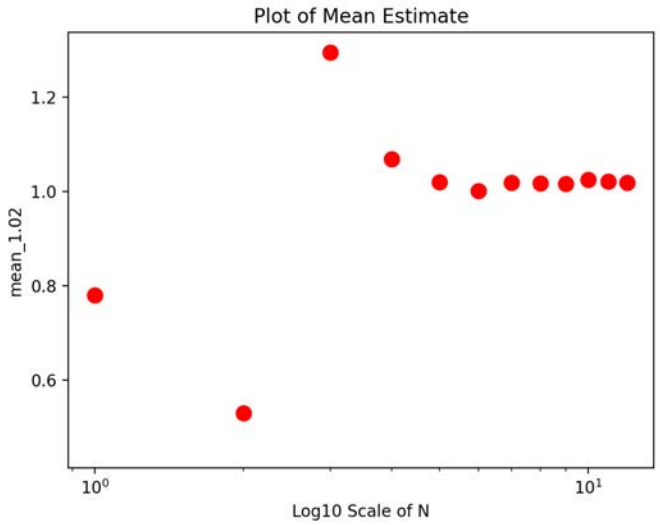

Plot of Mean Estimate
mean_0.92
1.6
1.4
1.2
1.0
0.8
10^0
10^1
Log10 Scale of N

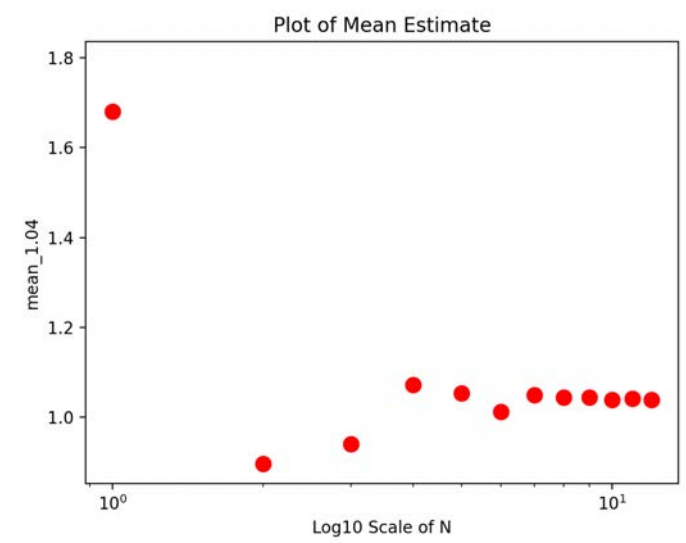

Plot of Mean Estimate
mean_0.94
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
10^0
10^1
Log10 Scale of N

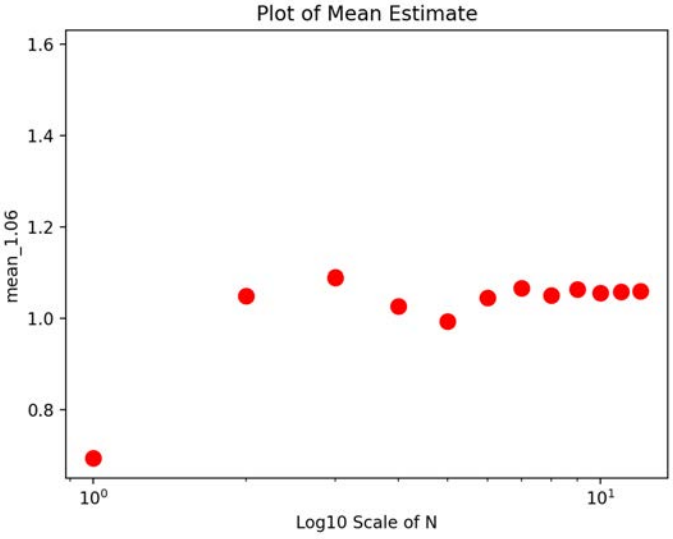

Plot of Mean Estimate
mean_0.96
1.4
1.2
1.0
0.8
10^0
10^1
Log10 Scale of N

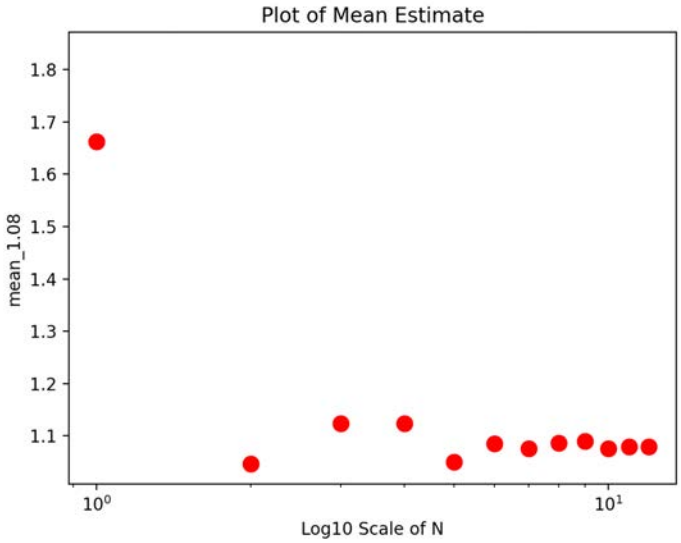

Plot of Mean Estimate
mean_0.98
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
10^0
10^1
Log10 Scale of N

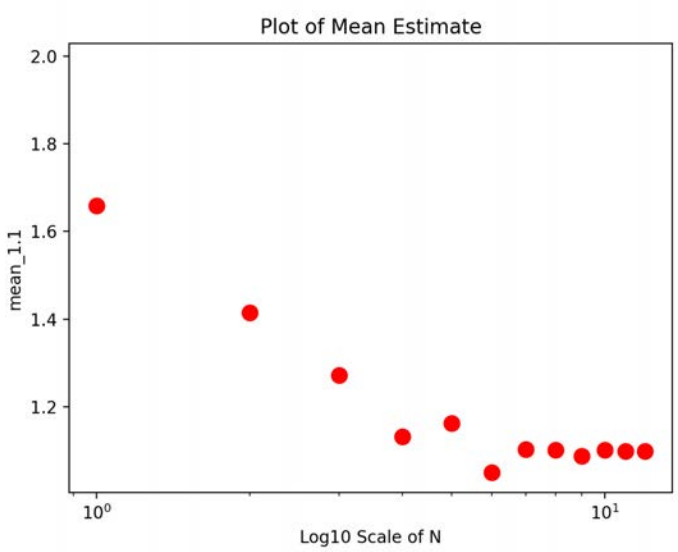

Plot of Mean Estimate
mean_1.0
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
10^0
10^1
Log10 Scale of N


Plot of Mean Estimate
mean_1.02
1.2
1.0
0.8
0.6
10^0
10^1
Log10 Scale of N

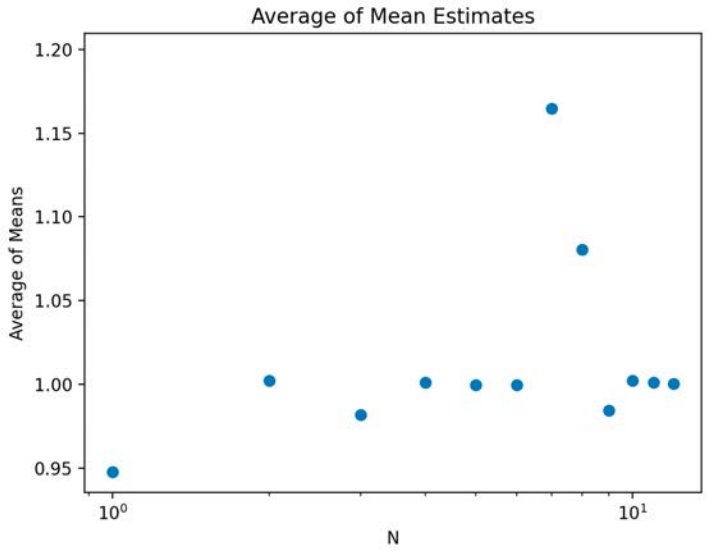

Plot of Mean Estimate
mean_1.04
1.8
1.6
1.4
1.2
1.0
10^0
10^1
Log10 Scale of N


Plot of Mean Estimate
mean_1.06
1.6
1.4
1.2
1.0
0.8
10^0
10^1
Log10 Scale of N





I am confident in the means of these datasets as each graph shows a convergence to the true mean. Now, the goal is to compute the average of the averages of these means. We should ideally see a convergence at 1.00 because we have taken an equal number of steps above and below 1.00 (1.00 ± 0.10). In order to do this, we must take the average of each value of 'estimate' once we have sorted through our dataframe by 'n'. An easy explanation to these steps is to sort the values by 'n' and then compute the mean as previously done. The only difference is that we must first group the values, then call on the 'estimate' column, take the sum, and then divide by 11 (the total count of varying means). This code was omitted due to simplicity and to avoid further repetition as this is the same mean calculation as previously done in this task and in task 2. Once completed, we see that our graph also converges to 1.00 which I had assumed that it would.



The homework asks: "Note that in this case, for $N = 10$, you are using 11x10 = 110 points, to compute the average. How does that compare to $N = 110$ for the plot of problem no. 2?" We know that we can calculate a Bayesian model average by averaging the estimates or the predictions of different models. Therefore, we know that by averaging the averages of the varying models, we have ultimately created a Bayesian estimate of the aggregated models we have generated. The main differences we see in the plots are varying data points caused by fluctuations during the averages. However, as expected, we see that our final graph has converged to a value of 1.00. This is exactly as we would expect.

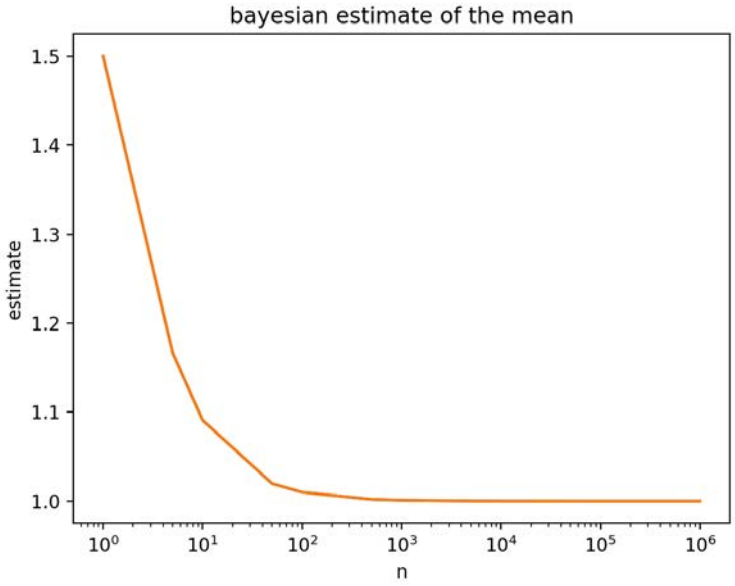
D. TASK 4

The final task requires us to construct a Bayesian estimate of the mean. I will preface this task by saying that I was not entirely sure how to go about this. We are given an initial mean guess of 2 but we are also told that the mean is unknown. However, I assumed that we would want our plot to converge to 1 as we assumed that it would in the previous task.

As previously discussed at the end of Task 3, we have essentially created a Bayesian estimate by averaging the averages of our model. We see that our average of averages converges to 1 as expected and we hope to see a similar occurrence when we create a Bayesian estimate of the mean. To do so, we will iterate through our samples again using a simple for-loop and pass the sample values to a hard coded Bayesian estimation. This was done as follows and I was able to read complete this by reading through an article on StatLect.com. The article applies basic principles of Bayesian statistics to estimate parameters.

Thus, we are able to take the initial mean (2.00) and divide by our variance which results in 2. Our denominator is (1/var+i) where I elected to just enter 1 for ease of reading later. This equation then provides the following graph which shows the mean estimate converging to our true value of 1. For this problem, I did assume that we were trying to approach a true mean of 1 as to be able to compare it to our previous graphs.

```
for i in samples:
    post_mean = ((2)+i*true_mean)/(1/1+i)
    bayes.append(post_mean)
```



We see that it takes a notably larger number of guesses to converge to the true mean using the Bayesian Estimation. It is important to recall that the horizontal axis of this graph and therefore we do not start to see convergence to the true mean until approximately 10^2 *N* size. However, we see convergence on all Maximum Likelihood Estimations *before* 10^1 *N* size. From an initial glance, I would say that the estimates are only equal to each other when the sample size is very small. At this point, both estimators are very far off but will eventually converge to the true mean at high *N* values.

E. SUMMARY

The goal of this assignment was to gain a better understanding and application of maximum likelihood as well as a Bayesian Estimation. The assignment was fairly straight forward for the first two tasks but I did stumble a lot on Tasks 3 and 4. The ML application seemed to work with little to no problems but I am unsure if my application of the Bayesian estimator was correct or not, albeit it converging to 1.00.

After completing this assignment, I realize that I am going to have to spend more time reading through notes and reading outside material to gain the understanding of Bayesian Estimation to apply this in a more facilitated manner.

Overall, the assignment helped me a lot with application, but I do hope that we can spare a few minutes at the start of next class to further discuss how one would apply these techniques. Primarily out of curiosity but also to ensure I don't make any mistakes with these techniques in the future.