Homework 3: Applied Probability and Statistics

UA CSC 380: Principles of Data Science, Fall 2023

Homework due at 11:59pm on Sep 27

Deliverables Your must make two submissions: (1) your homework as a SINGLE PDF file by the stated deadline to the gradescope (Homework 3). Include your code and output of the code as texts in the PDF. and (2) your codes in HW03.ipynb file to a separate submission (Homework 3 code). Each subproblem is worth 10 points. More instructions:

- You can hand-write your answers and scan them to make it a PDF. If you use your phone camera, I recommend using TurboScan (smartphone app) or similar ones to avoid uploading a slanted image or showing the background. Make sure you rotate it correctly.
- Watch the video and follow the instruction for the submission: https://youtu.be/KMPoby5g_nE
- **Show all work along with answers to get the full credit**.
- **Paste all your codes and outputs in the PDF report to get full credit.**
- Place your final answer into an 'answer box' that can be easily identified.
- Map the questions with your solutions when submitting. Points will be deducted if not following this.
- There will be no late days. Late homeworks result in zero credit.

Failure to follow the submission instructions will result in a minor penalty in credit.

You can choose to work individually or in pairs.

- If you choose to work in pairs, you are free to discuss whatever you want with your partner; please make only one submission per group.
- Please do not discuss with people outside your group about the homework (refer to the academic integrity policy in Lecture 1).
- If you have clarification questions, please feel free to post on Piazza so that it can promote discussion.

Problem 1: Law of Large Numbers / Central Limit Theorem

This problem is about verifying theoretical results learned in the class by visualizing them. You will have to be familiar with googling and reading python library documents to solve these problems. You may also take a look at basic numpy operators like how addition works for vectors and matrices, and what 'mean(A,0)' does to a 2d numpy array A. I believe you must already be good at googling and reading API documents – one of the basic skills a computer science graduate must have. All the coding answers must be done with Jupyter lab.

a) Let us numerically verify the law of large numbers. We will simulate $m = 100$ sample mean trajectories of $X_1, \ldots, X_N \sim \text{Bernoulli}(\mu = 0.2)$ and plot them altogether in one plot. Here, a sample mean trajectory means a sequence of $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_N$ where $\bar{X}_i$ is the sample mean using samples $X_1, \ldots, X_i$. We will plot $\bar{X}_n$ as a function of n, but do this multiple times. Take n from 1 to $N = 1000$. An ideal plot would look like the following:

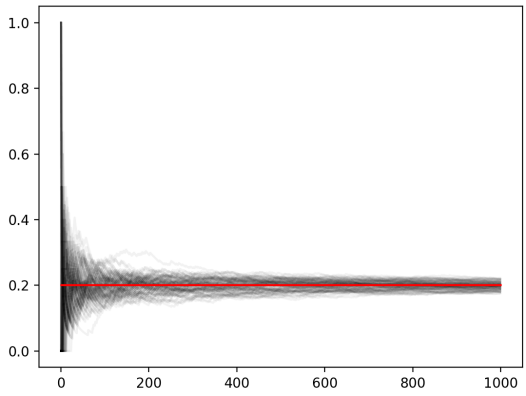


Figure 1:

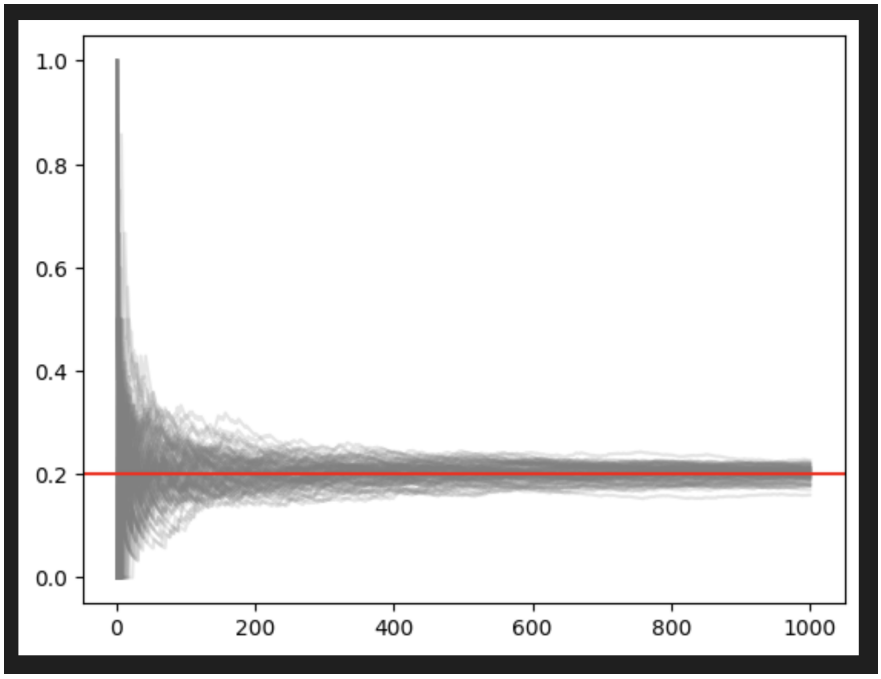
Hints: (1) To simulate numbers from Bernoulli, you can use numpy.random.rand() and compare each number with μ; (2) numpy.cumsum() may be helpful; (3) You must use the 'alpha' option to pyplot.plot() to give some transparency (you should obtain a similar look visualization as above). You may want to use the 'color' option to specify the color.

```
>>> import pandas as pd
>>> import numpy as np
>>> import numpy.random as ra
>>> import matplotlib.pyplot as plt
>>> ra.seed(31)
>>> # insert your code for a)
>>> # 100 mean trajectories
>>> for i in range(100):
...     # x values from 1 to 1000
...     x = np.arange(0, 1000) + 1
```

```
...     # 1000 y values in range 0 to 1
...     y = [np.random.rand() for x_val in x]
...     # compare y values to 0.2 to center on 0.2 mean
...     y1 = [1 if y_val <= 0.2 else 0 for y_val in y]
...     # cumlative sum of y at every index
...     y2 = np.cumsum(y1)
...     # mean of y at every index
...     y3 = [y2[x_val - 1]/x_val for x_val in x]
...     # plot mean trajectory
...     plt.plot(x, y3, alpha=0.2, color="gray")
...
[<matplotlib.lines.Line2D object at 0x117b51510>]
[<matplotlib.lines.Line2D object at 0x117b5df10>]
[<matplotlib.lines.Line2D object at 0x117b5e850>]
[<matplotlib.lines.Line2D object at 0x117b5f3d0>]
[<matplotlib.lines.Line2D object at 0x117b5ff50>]
[<matplotlib.lines.Line2D object at 0x117b78b90>]
[<matplotlib.lines.Line2D object at 0x117b79750>]
[<matplotlib.lines.Line2D object at 0x117b7a2d0>]
[<matplotlib.lines.Line2D object at 0x117b7ac50>]
[<matplotlib.lines.Line2D object at 0x117b7b8d0>]
[<matplotlib.lines.Line2D object at 0x117b84450>]
[<matplotlib.lines.Line2D object at 0x117b84f90>]
[<matplotlib.lines.Line2D object at 0x117b85b10>]
[<matplotlib.lines.Line2D object at 0x117b86710>]
[<matplotlib.lines.Line2D object at 0x117b87290>]
[<matplotlib.lines.Line2D object at 0x117b87dd0>]
[<matplotlib.lines.Line2D object at 0x117b8c9d0>]
[<matplotlib.lines.Line2D object at 0x117b8d3d0>]
[<matplotlib.lines.Line2D object at 0x117b8df90>]
[<matplotlib.lines.Line2D object at 0x117b8ea90>]
[<matplotlib.lines.Line2D object at 0x117b8f590>]
[<matplotlib.lines.Line2D object at 0x117b9c0d0>]
[<matplotlib.lines.Line2D object at 0x117b9cc90>]
[<matplotlib.lines.Line2D object at 0x117b9d750>]
[<matplotlib.lines.Line2D object at 0x117b9e310>]
[<matplotlib.lines.Line2D object at 0x117b9ee10>]
[<matplotlib.lines.Line2D object at 0x117b9f810>]
[<matplotlib.lines.Line2D object at 0x117bac410>]
[<matplotlib.lines.Line2D object at 0x117bace50>]
[<matplotlib.lines.Line2D object at 0x117bad950>]
[<matplotlib.lines.Line2D object at 0x117bae410>]
[<matplotlib.lines.Line2D object at 0x117baefd0>]
[<matplotlib.lines.Line2D object at 0x117bafb50>]
[<matplotlib.lines.Line2D object at 0x117bb46d0>]
```

```
[<matplotlib.lines.Line2D object at 0x117bb5210>]
[<matplotlib.lines.Line2D object at 0x117bb5c90>]
[<matplotlib.lines.Line2D object at 0x117bb66d0>]
[<matplotlib.lines.Line2D object at 0x117bb7090>]
[<matplotlib.lines.Line2D object at 0x117bb7c10>]
[<matplotlib.lines.Line2D object at 0x117bc07d0>]
[<matplotlib.lines.Line2D object at 0x117bc1350>]
[<matplotlib.lines.Line2D object at 0x117bc1e50>]
[<matplotlib.lines.Line2D object at 0x117bc28d0>]
[<matplotlib.lines.Line2D object at 0x117bc3450>]
[<matplotlib.lines.Line2D object at 0x117bc3fd0>]
[<matplotlib.lines.Line2D object at 0x117bccad0>]
[<matplotlib.lines.Line2D object at 0x117bcd510>]
[<matplotlib.lines.Line2D object at 0x117bcdf90>]
[<matplotlib.lines.Line2D object at 0x117bce950>]
[<matplotlib.lines.Line2D object at 0x117bcf4d0>]
[<matplotlib.lines.Line2D object at 0x117bcff50>]
[<matplotlib.lines.Line2D object at 0x117bdca90>]
[<matplotlib.lines.Line2D object at 0x117bdd410>]
[<matplotlib.lines.Line2D object at 0x117bdde50>]
[<matplotlib.lines.Line2D object at 0x117bde9d0>]
[<matplotlib.lines.Line2D object at 0x117bdf450>]
[<matplotlib.lines.Line2D object at 0x117bdff10>]
[<matplotlib.lines.Line2D object at 0x117beca90>]
[<matplotlib.lines.Line2D object at 0x117bed350>]
[<matplotlib.lines.Line2D object at 0x117beddd0>]
[<matplotlib.lines.Line2D object at 0x117bee950>]
[<matplotlib.lines.Line2D object at 0x117bee890>]
[<matplotlib.lines.Line2D object at 0x117befe50>]
[<matplotlib.lines.Line2D object at 0x117bf8950>]
[<matplotlib.lines.Line2D object at 0x117bf9490>]
[<matplotlib.lines.Line2D object at 0x117bc2f90>]
[<matplotlib.lines.Line2D object at 0x117bfa990>]
[<matplotlib.lines.Line2D object at 0x117bfb390>]
[<matplotlib.lines.Line2D object at 0x117bfbed0>]
[<matplotlib.lines.Line2D object at 0x127d48a10>]
[<matplotlib.lines.Line2D object at 0x127d493d0>]
[<matplotlib.lines.Line2D object at 0x127d49f50>]
[<matplotlib.lines.Line2D object at 0x127d4aa90>]
[<matplotlib.lines.Line2D object at 0x127d4b590>]
[<matplotlib.lines.Line2D object at 0x127d54090>]
[<matplotlib.lines.Line2D object at 0x127d54b50>]
[<matplotlib.lines.Line2D object at 0x127d55610>]
[<matplotlib.lines.Line2D object at 0x127d55fd0>]
[<matplotlib.lines.Line2D object at 0x127d56b10>]
```

```
[<matplotlib.lines.Line2D object at 0x127d57610>]
[<matplotlib.lines.Line2D object at 0x127d60090>]
[<matplotlib.lines.Line2D object at 0x127d60b10>]
[<matplotlib.lines.Line2D object at 0x127d61650>]
[<matplotlib.lines.Line2D object at 0x127d620d0>]
[<matplotlib.lines.Line2D object at 0x127d62b50>]
[<matplotlib.lines.Line2D object at 0x127d63610>]
[<matplotlib.lines.Line2D object at 0x127d6c0d0>]
[<matplotlib.lines.Line2D object at 0x127d6cb90>]
[<matplotlib.lines.Line2D object at 0x127d6d550>]
[<matplotlib.lines.Line2D object at 0x127d6e050>]
[<matplotlib.lines.Line2D object at 0x127d6e990>]
[<matplotlib.lines.Line2D object at 0x127d6f410>]
[<matplotlib.lines.Line2D object at 0x127d6fd90>]
[<matplotlib.lines.Line2D object at 0x127d788d0>]
[<matplotlib.lines.Line2D object at 0x127d79310>]
[<matplotlib.lines.Line2D object at 0x127d79d90>]
[<matplotlib.lines.Line2D object at 0x127d7a790>]
[<matplotlib.lines.Line2D object at 0x127d7b190>]
[<matplotlib.lines.Line2D object at 0x127d7bb50>]
[<matplotlib.lines.Line2D object at 0x127d885d0>]
>>> # plot red horizontal line at 0.2
>>> plt.axhline(y=0.2, color="red")
<matplotlib.lines.Line2D object at 0x117b52650>
```



b) Let us verify the central limit theorem (CLT) by simulation. For $N \in \{10, 100, 1000, 10000\}$, perform:

- Take N samples from Bernoulli($\mu = 0.05$) and compute the sample mean. Repeat this 1000 times.
- Plot those 1000 numbers as a histogram (pyplot.hist) with a proper number of bins. Use density=True.
- With a red line, overlay the pdf of a Gaussian distribution with the parameters suggested by the CLT (figure this out!).

An ideal answer would look like Figure 2. To receive full credit, you must use py-

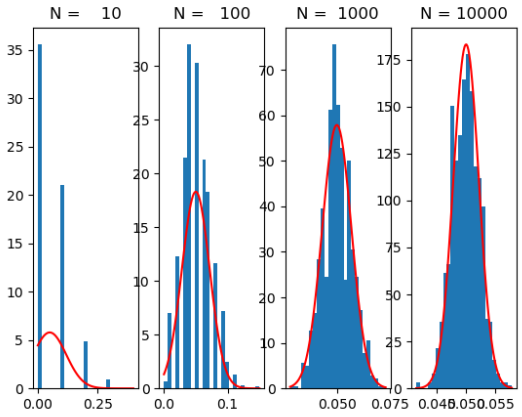


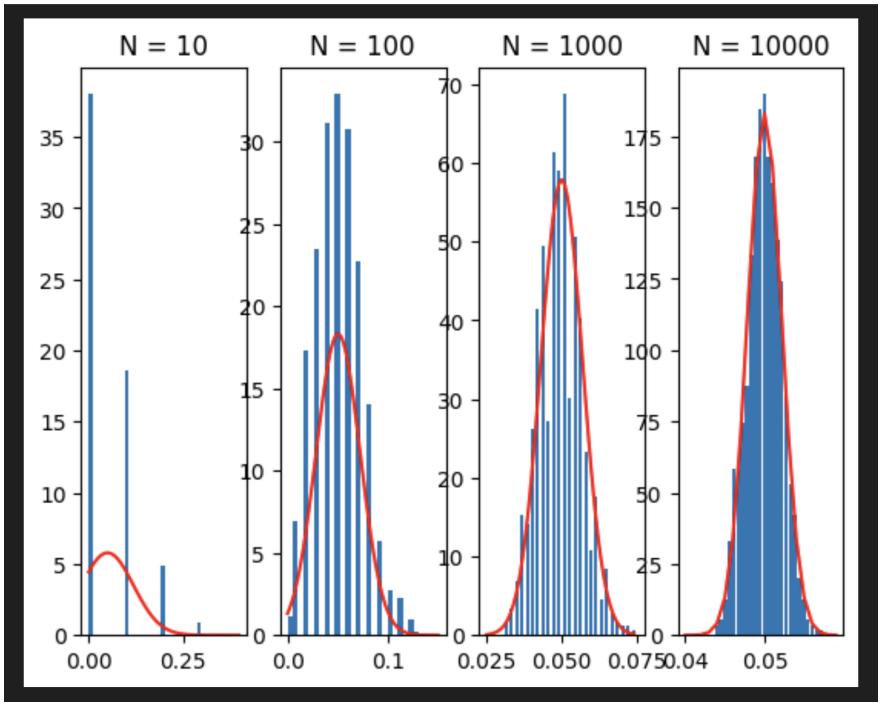
Figure 2:

plot.subplot to have four plots in one figure. Hint: numpy.random.rand() can be used to generate any given shape array with random numbers.

```
>>> # insert your code for b)

>>> # for plotting normal pdf
>>> import scipy.stats as stats

>>> # init sample sizes
>>> n = [10, 100, 1000, 10000]
>>> # init width values for each histogram plot
>>> width = [0.01, 0.005, 0.001, 0.0005]
>>> # init histogram plots
>>> fig, axs = plt.subplots(1, 4)
>>> # init x values for normal pdf
>>> x1 = [np.arange(0, 0.4, 0.001), np.arange(0, 0.15, 0.001),
...         np.arange(0.025, 0.075, 0.001), np.arange(0.04, 0.06, 0.001)]

>>> # plot 4 histogram subplots
>>> for i in range(4):
...     # get N samples
```

```
...     n_val = n[i]
...     # set subplot
...     subplot = axs[i]
...     # compute size N sample mean of bernoulli 1000 times
...     binom = [np.mean(np.random.binomial(1,0.05,n[i])) for x_val in x]
...     # draw histogram subplot
...     axs[i].hist(binom, density=True, bins=25, width=width[i])
...     # set subplot title
...     axs[i].set_title('N = ' + str(n[i]))
...     # calculate std of bernoulli
...     std = np.sqrt(0.05*(1-0.05)/n[i])
...     # generate normal pdf
...     p1 = [stats.norm.pdf(x1_val, 0.05, std) for x1_val in x1[i]]
...     # plot normal pdf on histogram subplot
...     axs[i].plot(x1[i], p1, color="red")
...
(array([38.5625,  0.    ,  0.    ,  0.    ,  0.    ,  0.    , 18.3125,
        0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  4.6875,  0.    ,
        0.    ,  0.    ,  0.    ,  0.    ,  0.875 ,  0.    ,  0.    ,
        0.    ,  0.    ,  0.    ,  0.0625]), array([0.   , 0.016, 0.032, 0.048, 0.064
       0.144, 0.16 , 0.176, 0.192, 0.208, 0.224, 0.24 , 0.256, 0.272,
       0.288, 0.304, 0.32 , 0.336, 0.352, 0.368, 0.384, 0.4  ]), <BarContainer object
Text(0.5, 1.0, 'N = 10')
[<matplotlib.lines.Line2D object at 0x29f92e610>]
(array([ 1.15384615,  5.19230769,  0.        , 18.26923077,  0.        ,
       26.92307692,  0.        , 33.65384615,  0.        , 37.88461538,
        0.        , 25.96153846,  0.        , 17.5       ,  0.        ,
       11.34615385,  0.        ,  7.11538462,  0.        ,  3.46153846,
        0.        ,  2.30769231,  0.        ,  0.96153846,  0.57692308]), array([0.
       0.0416, 0.0468, 0.052 , 0.0572, 0.0624, 0.0676, 0.0728, 0.078 ,
       0.0832, 0.0884, 0.0936, 0.0988, 0.104 , 0.1092, 0.1144, 0.1196,
       0.1248, 0.13  ]), <BarContainer object of 25 artists>)
Text(0.5, 1.0, 'N = 100')
[<matplotlib.lines.Line2D object at 0x29f964d10>]
(array([ 1.13636364,  3.97727273,  5.68181818, 13.06818182,  7.95454545,
       25.        , 25.56818182, 43.18181818, 25.56818182, 56.25      ,
       61.93181818, 63.63636364, 27.27272727, 60.22727273, 40.90909091,
       33.52272727, 15.90909091, 20.45454545, 13.06818182, 11.36363636,
        2.84090909,  3.40909091,  3.40909091,  1.70454545,  1.13636364]), array([0.03
       0.04232, 0.04408, 0.04584, 0.0476 , 0.04936, 0.05112, 0.05288,
       0.05464, 0.0564 , 0.05816, 0.05992, 0.06168, 0.06344, 0.0652 ,
       0.06696, 0.06872, 0.07048, 0.07224, 0.074  ]), <BarContainer object of 25 artis
Text(0.5, 1.0, 'N = 1000')
[<matplotlib.lines.Line2D object at 0x29f966390>]
(array([  1.57232704,   0.        ,   0.        ,   1.57232704,
```

```
          1.57232704,  17.29559748,  23.58490566,  28.30188679,
         66.03773585,  81.76100629, 136.79245283, 182.38993711,
        144.65408805, 180.81761006, 161.94968553, 163.52201258,
        125.78616352,  80.18867925,  50.31446541,  62.89308176,
         23.58490566,  17.29559748,   7.86163522,   7.86163522,
          4.71698113]), array([0.0413  , 0.041936, 0.042572, 0.043208, 0.043844, 0.044
        0.045116, 0.045752, 0.046388, 0.047024, 0.04766 , 0.048296,
        0.048932, 0.049568, 0.050204, 0.05084 , 0.051476, 0.052112,
        0.052748, 0.053384, 0.05402 , 0.054656, 0.055292, 0.055928,
        0.056564, 0.0572  ]), <BarContainer object of 25 artists>)
Text(0.5, 1.0, 'N = 10000')
[<matplotlib.lines.Line2D object at 0x29f9dc690>]
```



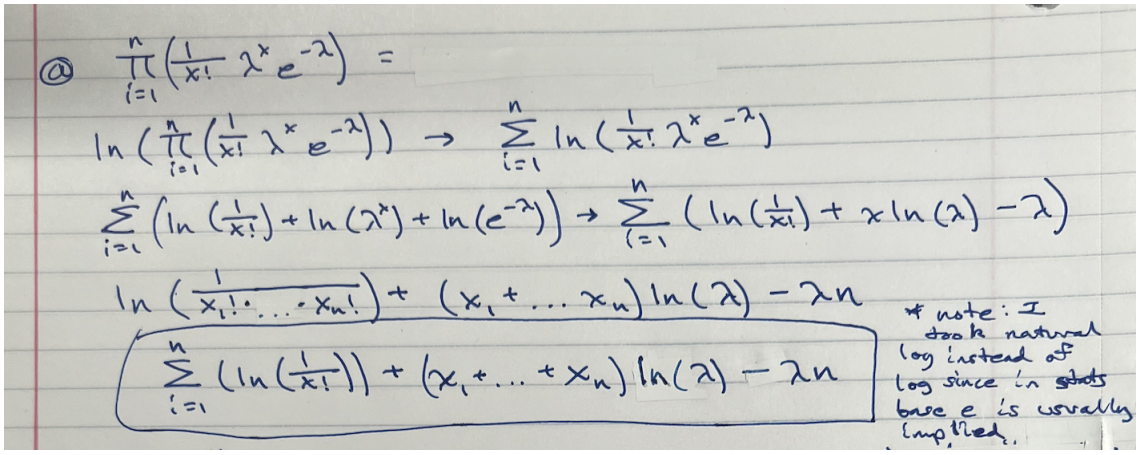
Problem 2: Maximum Likelihood Estimation

I would like to build a simple model to predict how many students are likely to come to my office hours this semester. Because this is an arrival process, I will model the number of arrivals during office hours as Poisson distributed. Recall that the Poisson is a discrete distribution over the number of arrivals (or events) in a fixed time-frame. The Poisson distribution has a probability mass function (PMF) of the form,

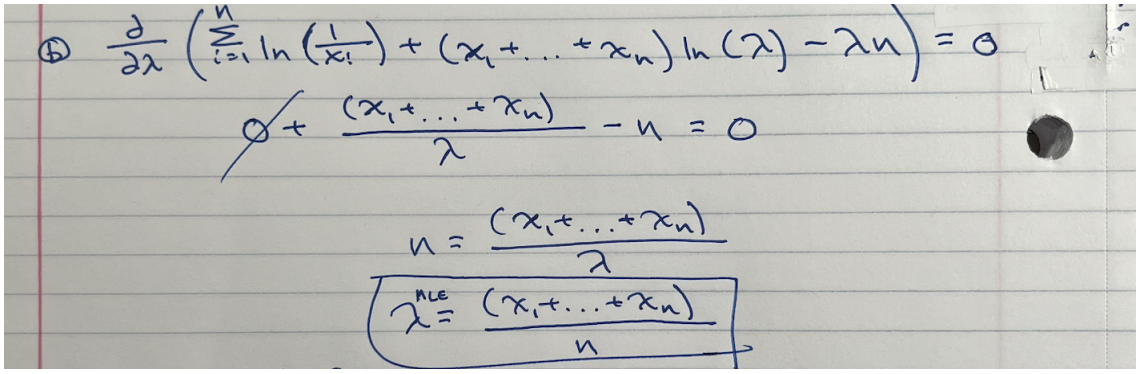
$$\text{Poisson}(x;\lambda) = \frac{1}{x!}\lambda^x e^{-\lambda}.$$

The parameter λ is the *rate* parameter, and represents the expected number of arrivals $\mathbb{E}[x] = \lambda$. To fit the model I will need to estimate the rate parameter using some data.

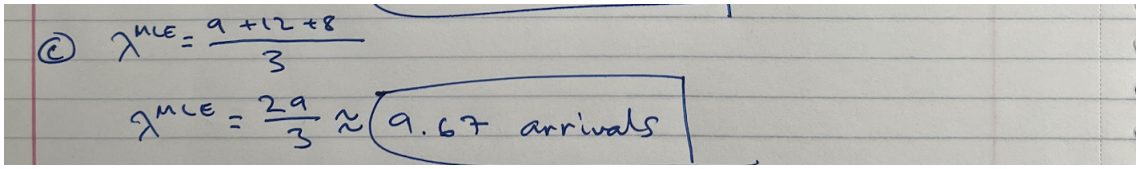
a) Write the logarithm of the joint probability distribution $\log p(X_1, ..., X_n; \lambda)$.



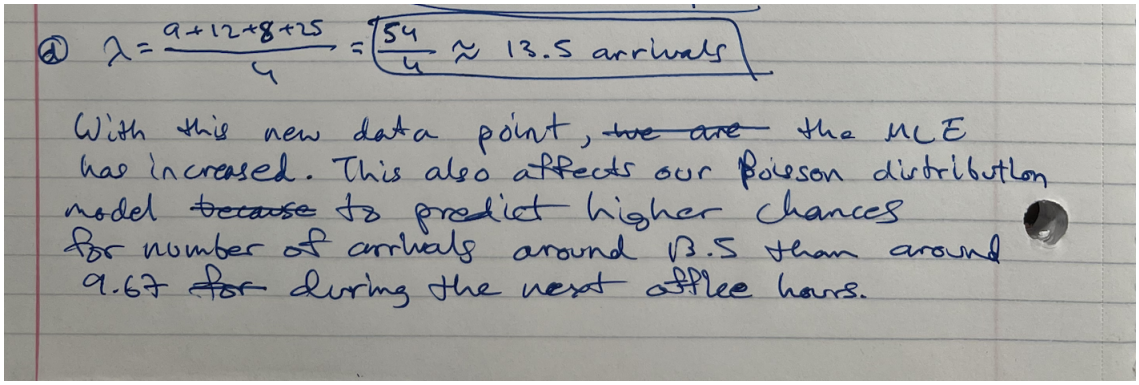
b) Compute the maximum likelihood estimate (MLE) of the rate parameter λ^{MLE} which maximizes the joint probability. The log-likelihood function is concave and so the MLE can be computed by finding the zero-derivative solution. Make sure to show all of your calculations.



c) During my last three office hours I received $X_1 = 9, X_2 = 12, X_3 = 8$ students. How many arrivals should I expect at my next office hours under this model? Hint: use λ calculated at b).



d) I have assigned a particularly challenging homework which has led to a lot of students $X_4 = 25$ arriving at my office hours. Compute the MLE again, but include this new training point. How has the model changed with this new data point?



Problem 3: Estimating Pearson Correlation

This question will walk you through the process of estimating the Pearson correlation of two random variables, along with confidence intervals using the bootstrap method. Lecture slides and Chapter 8 of the textbook (Wasserman) will be helpful if you need a refresher. For our chosen model, we will use a bivariate Gaussian distribution: $P(X, Y; \mu, \Sigma) = \mathcal{N}(\mu, \Sigma)$. For simplicity, let us assume that the distribution is zero-mean $\mu = (0, 0)^T$. Let us assume that the true (unknown) covariance matrix is,

$$\Sigma = \begin{pmatrix} \text{Var}(X) & \text{Cov}(X,Y) \\ \text{Cov}(Y,X) & \text{Var}(Y) \end{pmatrix} = \begin{pmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}.$$

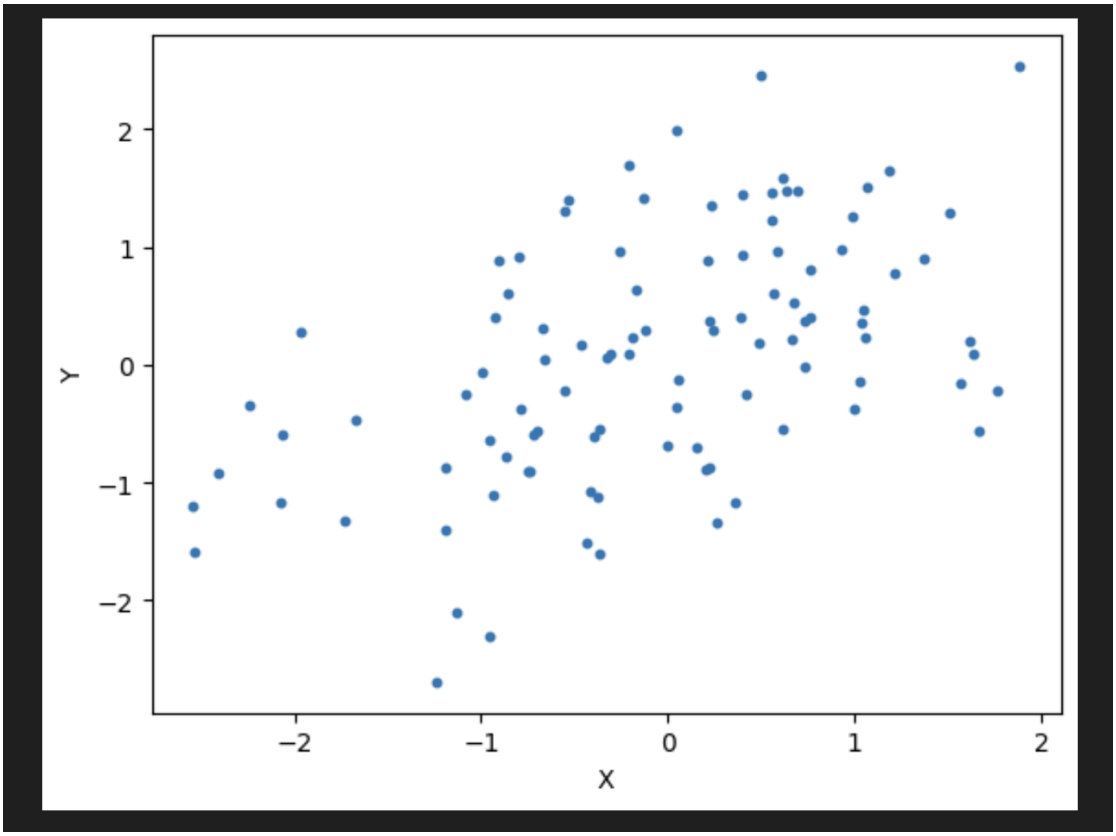
We have written the covariance matrix in terms of the Pearson correlation between X and Y,

$$\rho = \frac{\text{Cov}(X,Y)}{\sigma_X\sigma_Y} = \frac{\mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right]}{\sqrt{\mathbb{E}\left[(X - \mathbb{E}[X])^2\right]\mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right]}} = \frac{0.5}{1 \cdot 1} = 0.5.$$

Using **numpy.random.seed** set your random number generator seed to 0 and answer the following:

*a) Create a dataset by drawing $N = 100$ samples from our model using **numpy.random** function **multivariate_normal**. Create a scatterplot of your data using **matplotlib.pyplot** functions **scatter**. Label your axes X and Y.*

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # insert your code for a)

>>> # setting random seed to 0
>>> np.random.seed(0)
>>> # init distribution mean
>>> mean = [0, 0]
>>> # init covariance matrix
>>> cov = [[1, 0.5], [0.5, 1]]
>>> # generate dataset for scatter plot
>>> x, y = np.random.multivariate_normal(mean, cov, size=100).T
>>> # draw scatter plot
>>> plt.scatter(x, y, 10)
<matplotlib.collections.PathCollection object at 0x29f982890>
>>> # label x axis
>>> plt.xlabel("X")
Text(0.5, 0, 'X')
>>> # label y axis
>>> plt.ylabel("Y")
Text(0, 0.5, 'Y')
```



b) *Compute and report the plug-in estimator of correlation, given by:*

$$\hat{\rho}_N = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X_i - \bar{X})^2 \sum_j (Y_j - \bar{Y})^2}}$$

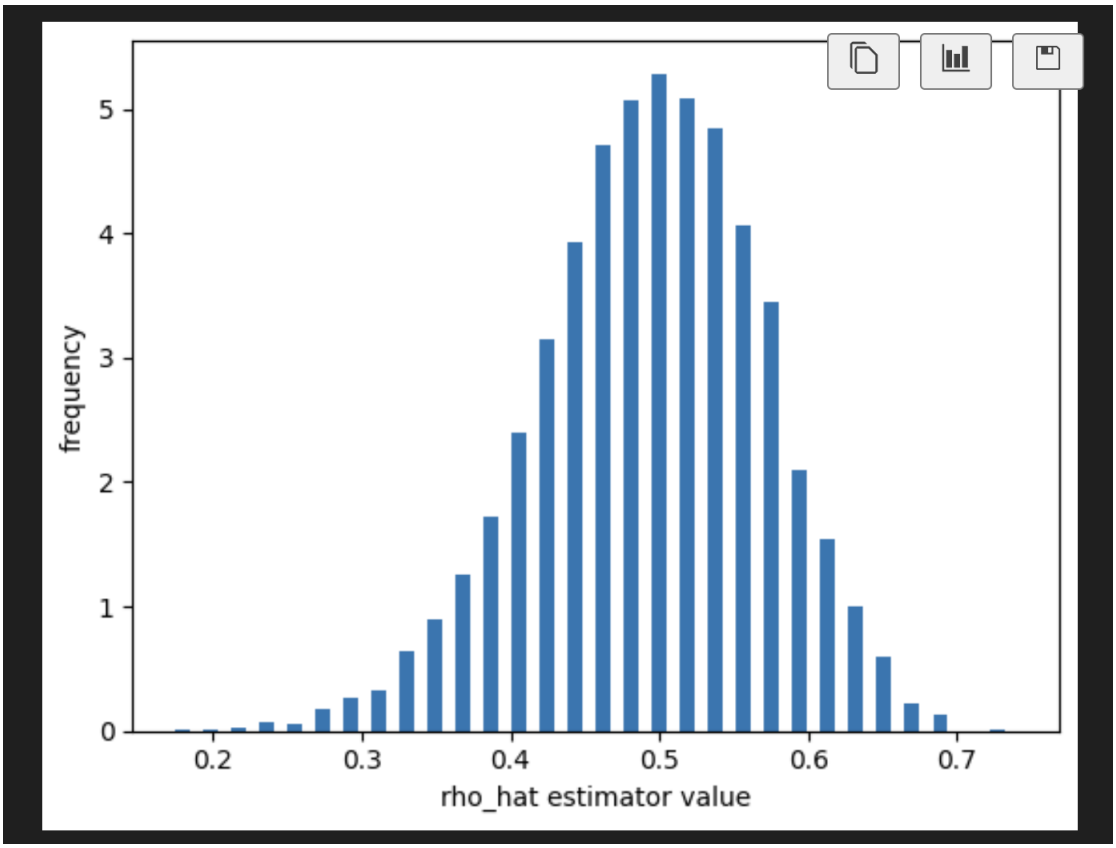
Where $\bar{X} = \frac{1}{N}\sum_i X_i$ *is the sample mean (and similarly for* $\bar{Y}$*).*

```
>>> # insert your code for b)
>>> np.random.seed(0)
>>> # define rho_hat calculation function
>>> def rho_hat():
...         # compute x_bar
...         x_bar = sum(x) / len(x)
...         # compute y_bar
...         y_bar = sum(y) / len(y)
...         # compute summation terms in numerator
...         num_sum = [(x_val - x_bar)*(y_val - y_bar)
...                 for x_val, y_val in zip(x, y)]
...         # compute numerator
...         num = sum(num_sum)
...         # compute summation terms in denominator
...         d0 = [np.square(x_val - x_bar) for x_val in x]
```

```
...         d1 = [np.square(y_val - y_bar) for y_val in y]
...         # compute denominator
...         denom = np.sqrt(sum(d0) * sum(d1))
...         # compute and return rho_hat
...         return num / denom
...
>>> # call rho_hat function and write result
>>> print('Plug-in estimator = ' + str(rho_hat()))
Plug-in estimator = 0.5046162424282595
```

c) *Repeat the above process* $B = 5,000$ *times to generate* $\hat{\rho}_{N,1}, \ldots, \hat{\rho}_{N,B}$*, each one based on a fresh set of* $N = 100$ *samples. Display a histogram of your* B *estimates using* ***matplotlib.pyplot.hist*** *with 30 bins. Label your axes.*

```
>>> # insert your code for c)
>>> np.random.seed(0)
>>> # init array of p_hats
>>> rho_hats = []

>>> # generate rho_hat for 5000 samples and append to rho_hats
>>> for i in range(5000):
...     x, y = np.random.multivariate_normal(mean, cov, size=100).T
...     rho_hats.append(rho_hat())
...
>>> plt.hist(rho_hats, density=True, bins=30, width = 0.01)
(array([0.01059535, 0.01059535, 0.02119069, 0.06357208, 0.05297673,
       0.16952554, 0.26488366, 0.31786039, 0.63572078, 0.89000909,
       1.25025087, 1.71644611, 2.39454827, 3.14681786, 3.93087349,
       4.71492912, 5.06457555, 5.28707783, 5.08576625, 4.85266863,
       4.068613  , 3.44348756, 2.09787858, 1.53632522, 0.99596256,
       0.5933394 , 0.22250227, 0.12714416, 0.        , 0.01059535]), array([0.17415
       0.26853486, 0.28741107, 0.30628728, 0.32516349, 0.3440397 ,
       0.36291591, 0.38179213, 0.40066834, 0.41954455, 0.43842076,
       0.45729697, 0.47617318, 0.4950494 , 0.51392561, 0.53280182,
       0.55167803, 0.57055424, 0.58943045, 0.60830667, 0.62718288,
       0.64605909, 0.6649353 , 0.68381151, 0.70268772, 0.72156394,
       0.74044015]), <BarContainer object of 30 artists>)
>>> plt.xlabel('rho_hat estimator value')
Text(0.5, 0, 'rho_hat estimator value')
>>> plt.ylabel('frequency')
Text(0, 0.5, 'frequency')
```



d) Use the B estimates obtained in the above question to estimate $\mathbb{E}\left[(\hat{\rho}_N - \rho)^2\right]$, the mean square error (MSE) of plug-in estimator $\hat{\rho}_N$. What is the value of your MSE estimate?

```
>>> # insert your code for d)

>>> # compute mse estimate
>>> mse = np.mean([np.square(rho_hat - 0.5) for rho_hat in rho_hats])
>>> # write MSE estimate
>>> print('MSE estimate = ' +str(mse))
MSE estimate = 0.005749295118261227
```