

Homework 2

References

- Lectures 3-6 (inclusive).

Instructions

- Type your name and email in the "Student details" section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you can either:
 - Type the answer using the built-in latex capabilities. In this case, simply export the notebook as a pdf and upload it on gradescope; or
 - You can print the notebook (after you are done with all the code), write your answers by hand, scan, turn your response to a single pdf, and upload on gradescope.

Note: Please match all the pages corresponding to each of the questions when you submit on gradescope.

Student details

- First Name:** Alex
- Last Name:** Shank
- Email: shank14@purdue.edu

```
In [1]: # Here are some modules that you may need - please run this block of code:
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
sns.set_context('paper')
import scipy
import scipy.stats as st
# A helper function for downloading files
import requests
import os
def download(url, local_filename=None):
    """
    Downloads the file in the 'url' and saves it in the current working directory.
    """
    data = requests.get(url)
    if local_filename is None:
        local_filename = os.path.basename(url)
    with open(local_filename, 'wb') as fd:
        fd.write(data.content)
```

Problem 1 - Practice with discrete random variables

Consider the Categorical random variable:

$$X \sim \text{Categorical}(0.3, 0.1, 0.2, 0.4).$$

taking values in $\{0, 1, 2, 3\}$. Find the following (you may use `scipy.stats.rv_discrete` or do it by hand):

A. The expectation $E[X]$.

Answer:

$$E[X] = 1.7$$

```
In [2]: ps = [0.3, 0.1, 0.2, 0.4]
xs = np.array([0, 1, 2, 3])
X = st.rv_discrete(name='Practice categorical', values=(xs, ps))
print('E[X] = {:.2f}'.format(Ex := X.expect()))

E[X] = 1.70
```

B. The variance $V[X]$.

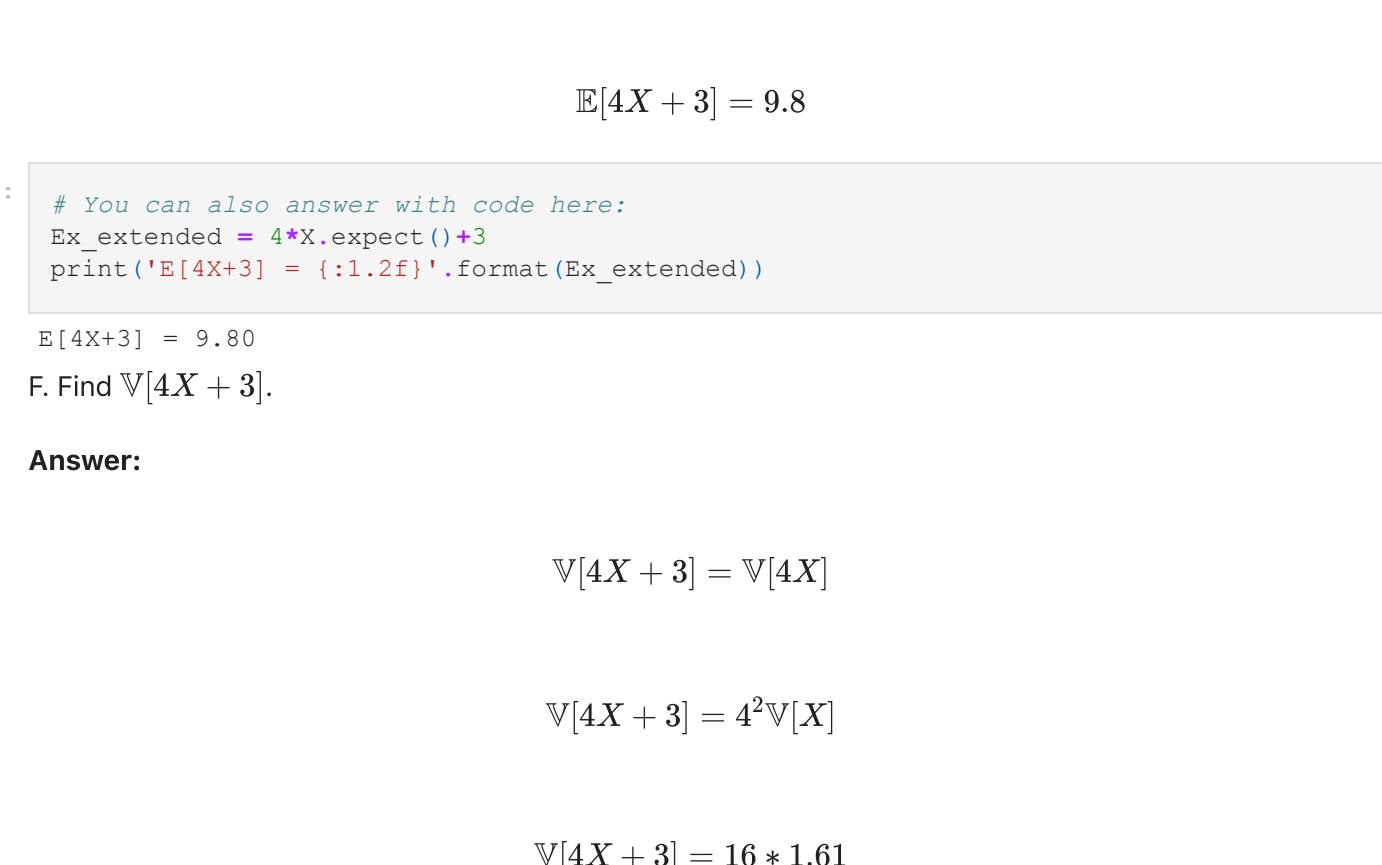
Answer:

$$V[X] = 1.61$$

```
In [3]: # You can also answer with code here:
print('V[X] = {:.2f}'.format(Vx := X.var()))

V[X] = 1.61
```

C. Plot the probability mass function of X .



D. Find the probability that X is in $\{0, 2\}$.

Answer:

$$P(X \in \{0, 2\}) = P(X = 0) + P(X = 2)$$

$$P(X \in \{0, 2\}) = P(X = 0) + P(X = 2)$$

$$P(X \in \{0, 2\}) = 0.3 + 0.2$$

$$P(X \in \{0, 2\}) = 0.5$$

```
In [5]: # You can also answer with code here:
print('P[X in {0, 2}] = {:.2f}'.format(sum([X.pmf(0), X.pmf(2)])))

P[X in {0, 2}] = 0.50
```

E. Find $E[4X + 3]$.

Answer:

$$E[4X + 3] = E[4X] + E[3]$$

$$E[4X + 3] = 4E[X] + 3$$

$$E[4X + 3] = 4 * 1.7 + 3$$

$$E[4X + 3] = 9.8$$

```
In [6]: # You can also answer with code here:
Ex_extended = 4*X.expect()+3
print('E[4X+3] = {:.2f}'.format(Ex_extended))

E[4X+3] = 9.80
```

F. Find $V[4X + 3]$.

Answer:

$$V[4X + 3] = V[4X]$$

$$V[4X + 3] = 4^2 V[X]$$

$$V[4X + 3] = 16 * 1.61$$

$$V[4X + 3] = 25.76$$

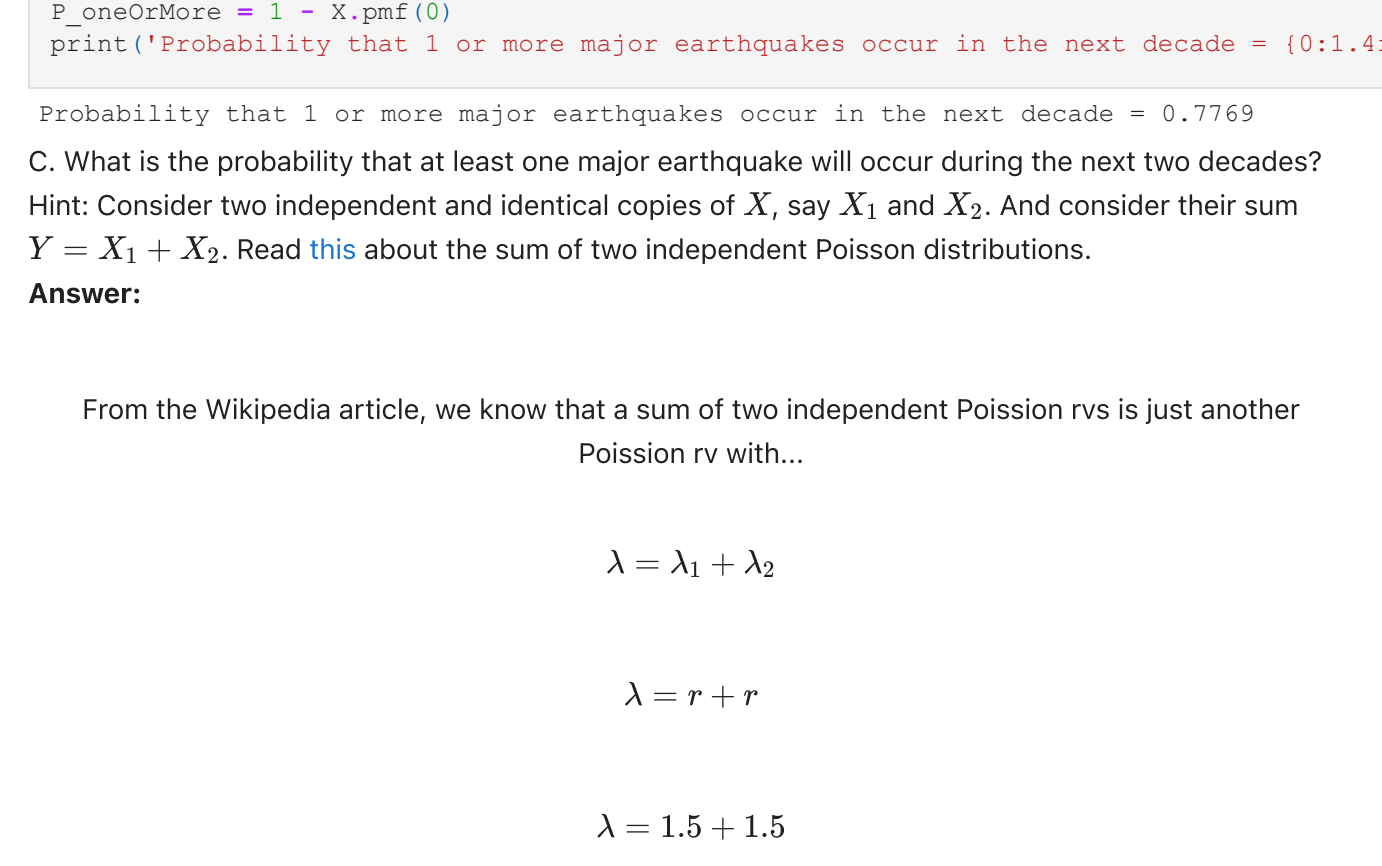
```
In [7]: # You can also answer with code here:
Vx_extended = pow(4,2)*X.var()
print('V[4X+3] = {:.2f}'.format(Vx_extended))

V[4X+3] = 25.76
```

Problem 2 - Predicting the probability of major earthquakes in Southern California

The **San Andreas fault** extends through California forming the boundary between the Pacific and the North American tectonic plates. It has caused some of the major earthquakes on Earth. We are going to focus on Southern California and we would like to assess the probability of a major earthquake, defined as an earthquake of magnitude 6.5 or greater, during the next ten years.

A. The first thing we are going to do is go over a [database of past earthquakes](#) that have occurred in Southern California and collect the relevant data. We are going to start at 1900 because data before that time may be unreliable. Go over each decade and count the occurrence of a major earthquake (i.e., count the number of orange and red colors in each decade). We have done this for you.



B. The **Poisson distribution** is a discrete distribution with values $\{0, 1, 2, \dots\}$ which is commonly used to model the number of events occurring in a certain time period. It is the right choice when these events are happening independently and the probability of any event happening over a small period of time is constant. Let's use the Poisson to model the number of earthquakes X occurring in a decade. We write:

$$X \sim \text{Poisson}(r),$$

where r is the **rate parameter** of Poisson. The rate is the number of events per time period. Here, r is the number of earthquakes per decade. Using the data above, we can set the rate as the empirical average of the observed number of earthquakes per decade:

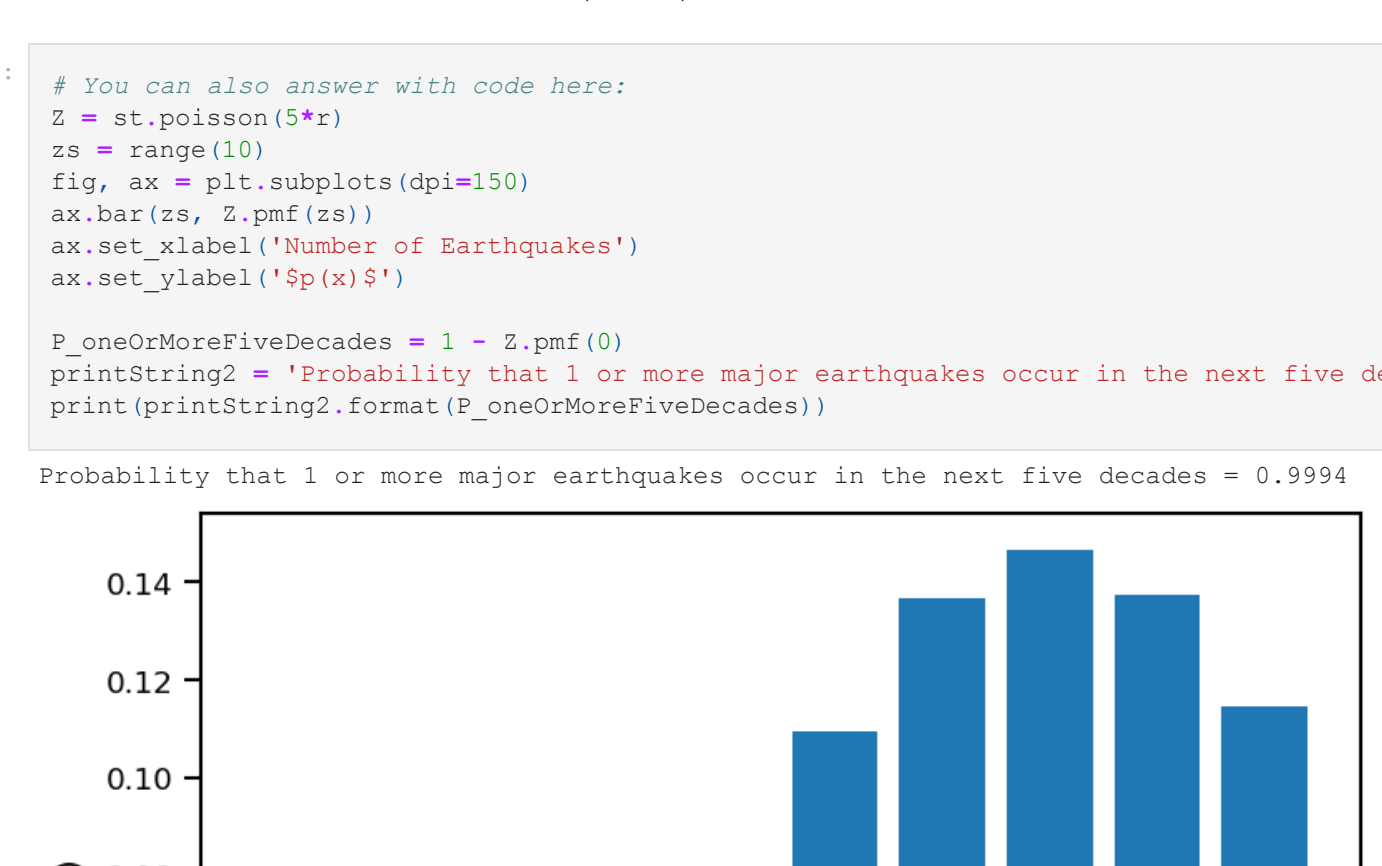
```
In [9]: r = np.mean(eq_data)
print('r = {:.2f} major earthquakes per decade'.format(r))

r = 1.50 major earthquakes per decade
```

Strictly speaking, **this is not how you should be calibrating models!!!** We will learn about the **right** way (which uses Bayes' rule) in the subsequent lectures. But it will do for now as the answer you would get using the **right** way is, for this problem, almost the same. Let's define a Poisson distribution using `scipy.stats.poisson` (see documentation [here](#)):

```
In [10]: X = st.poisson(r)
```

A. Plot the probability mass function of X .



B. What is the probability that at least one major earthquake will occur during the next decade?

Answer:

$$P(X \geq 1) = 1 - P(X = 0)$$

$$P(X \geq 1) = 1 - 0.2231$$

$$P(X \geq 1) = 0.7769$$

```
In [12]: # You can also answer with code here:
P_oneOrMore = 1 - X.pmf(0)
print('Probability that 1 or more major earthquakes occur in the next decade = {:.4f}'.format(P_oneOrMore))

Probability that 1 or more major earthquakes occur in the next decade = 0.7769
```

C. What is the probability that at least one major earthquake will occur during the next two decades? Hint: Consider two independent and identical copies of X , say X_1 and X_2 . And consider their sum $Y = X_1 + X_2$. Read this about the sum of two independent Poisson distributions.

From the Wikipedia article, we know that a sum of two independent Poisson rvs is just another Poisson rv with...

$$\lambda = \lambda_1 + \lambda_2$$

$$\lambda = r + r$$

$$\lambda = 1.5 + 1.5$$

$$\lambda = 3$$

We can then use the same process as before, where we look at the pmf and sum all probabilities that are not...

$$P(X = 0)$$

So,

$$P(X \geq 1) = 1 - P(X = 0)$$

$$P(X \geq 1) = 1 - P(X = 0)$$

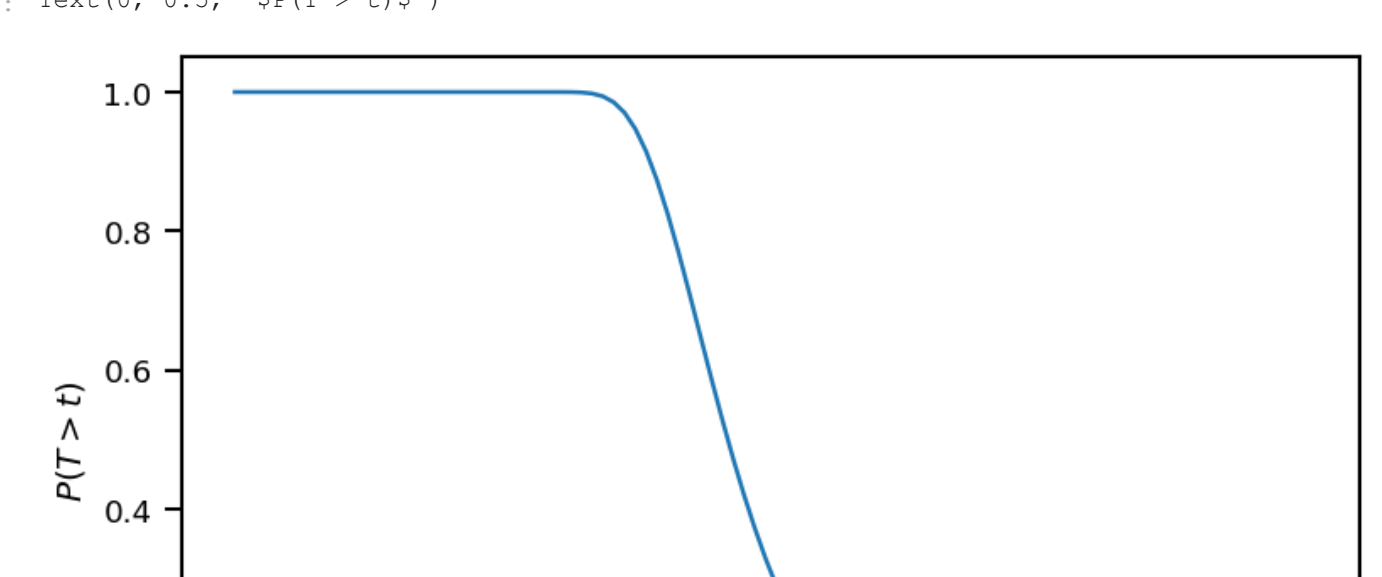
$$P(X \geq 1) = 1 - 0.4988$$

$$P(X \geq 1) = 0.9502$$

```
In [13]: # You can also answer with code here:
Y = st.poisson(r * r)
ys = range(10)
fig, ax = plt.subplots(dpi=150)
ax.bar(ys, Y.pmf(ys))
ax.set_xlabel('Number of Earthquakes')
ax.set_ylabel('$p(x)$')

P_oneOrMoreTwoDecades = 1 - Y.pmf(0)
print('Probability that 1 or more major earthquakes occur in the next two decades = {:.4f}'.format(P_oneOrMoreTwoDecades))

Probability that 1 or more major earthquakes occur in the next two decades = 0.9502
```



D. What is the probability that at least one major earthquake will occur during the next five decades?

Answer:

Similar to the previous problem, we can use a sum of 5 identical poisson rvs that are independent

$$P(X \geq 1) = 1 - P(X = 0)$$

$$P(X \geq 1) = 1 - P(X = 0)$$

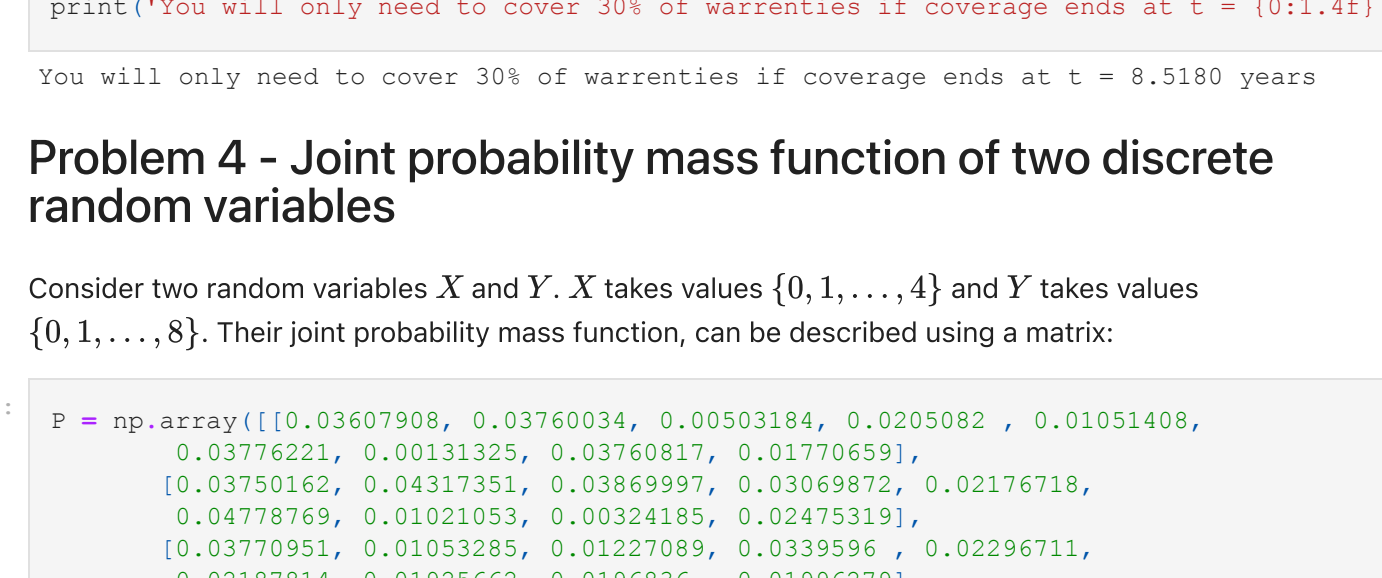
$$P(X \geq 1) = 1 - 0.0006$$

$$P(X \geq 1) = 0.9994$$

```
In [14]: # You can also answer with code here:
Z = st.poisson(5*r)
zs = range(10)
fig, ax = plt.subplots(dpi=150)
ax.bar(zs, Z.pmf(zs))
ax.set_xlabel('Number of Earthquakes')
ax.set_ylabel('$p(x)$')

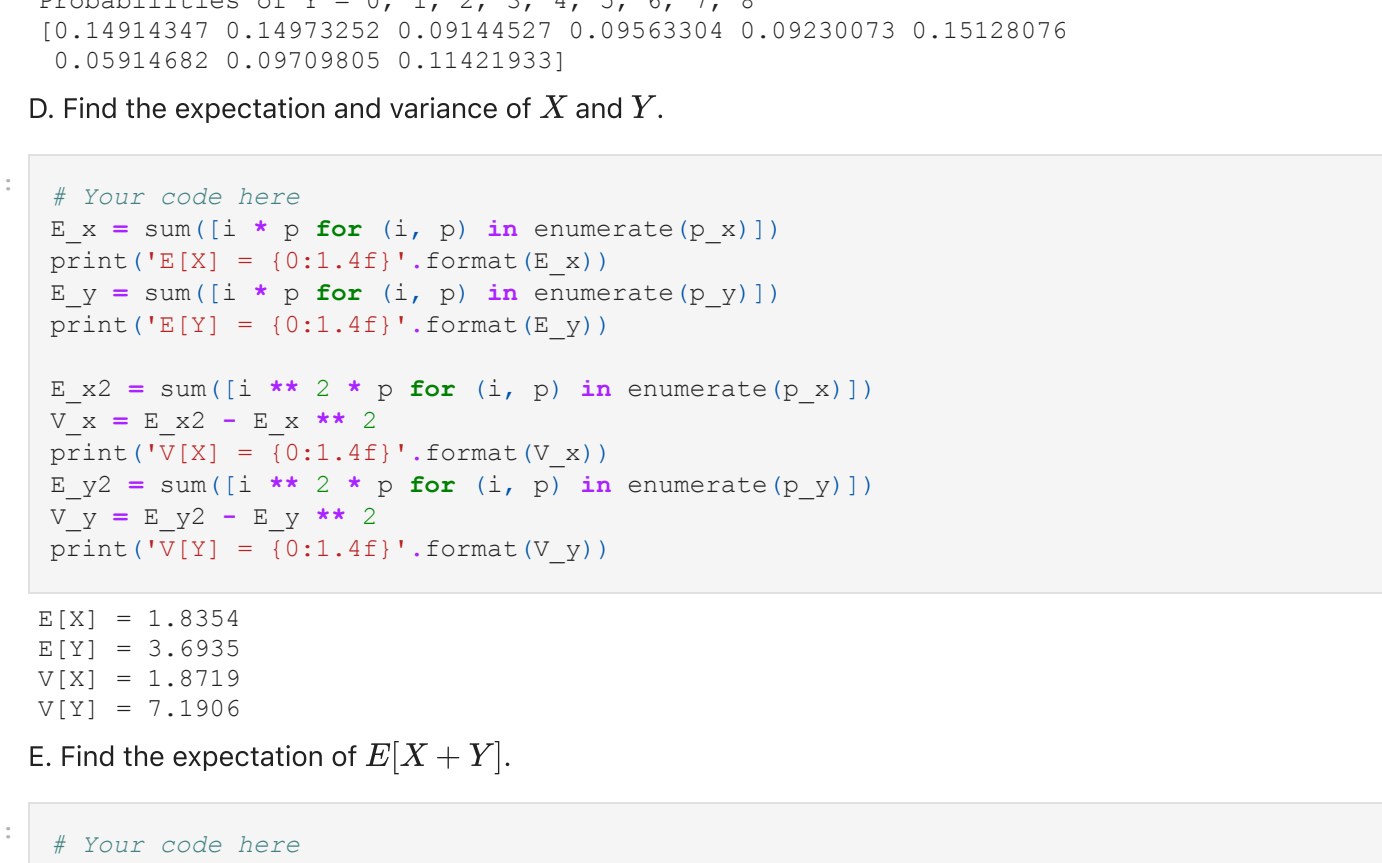
P_oneOrMoreFiveDecades = 1 - Z.pmf(0)
print('Probability that 1 or more major earthquakes occur in the next five decades = {:.4f}'.format(P_oneOrMoreFiveDecades))

Probability that 1 or more major earthquakes occur in the next five decades = 0.9994
```



Problem 3 - Failure of a mechanical component

Assume that you designing a gear for a mechanical system. Under normal operating conditions the gear is expected to fail at a random time. Let T be a random variable capturing the time the gear fails. What should the probability density of T look like? Well, when the gear is brand new, the probability density should be close to zero because a new gear does not fail under normal operating conditions. As time goes by, the probability density should increase because various things start happening to the material, e.g., crack formation, fatigue, etc. Finally, the probability density must again start going to zero as time further increases because nothing lasts forever... A probability distribution that is commonly used to model this situation is the **Weibull**. We are going to fit some fail time data to a Weibull distribution and then you will have to answer a few questions about failing times.

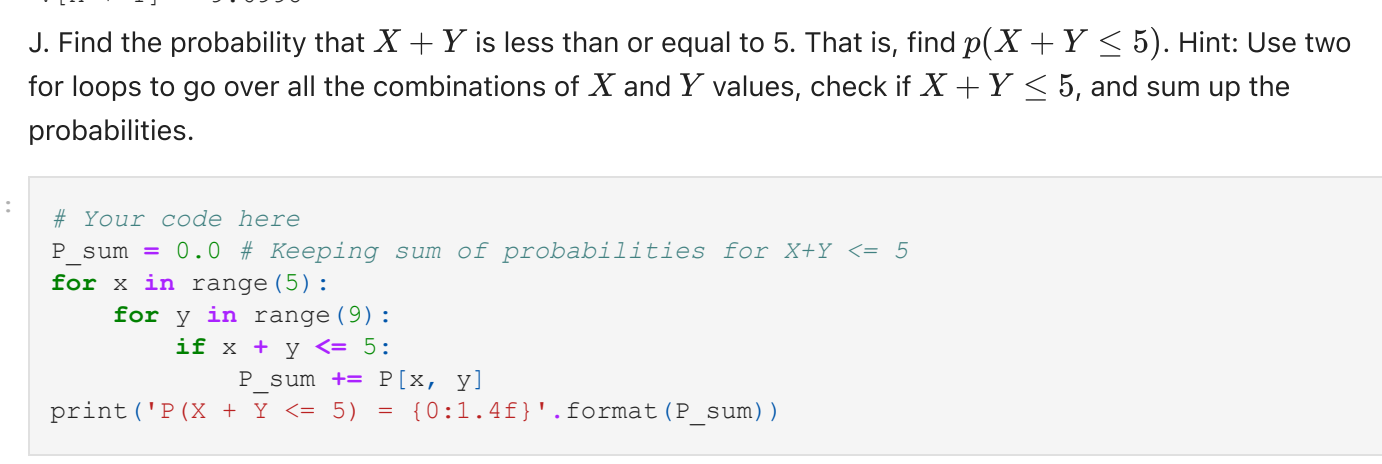


A. Find the mean fail time and its variance. Hint: Do not integrate anything by hand. Just use the functionality of `scipy.stats`.

```
In [16]: # Your code here
t_mean = T.expect() # Change me
errors = T.var() # Change me
print('E[T] = {:.2f}'.format(t_mean))
print('V[T] = {:.2f}'.format(T.var()))

E[T] = 9.53
V[T] = 2.88
```

B. Plot the cumulative distribution function of T .



C. Plot the probability that gear survives for more than t as a function of t . That is, plot the function:

$$S(t) = p(T > t).$$

Hint: First connect this function to the cumulative distribution function of T .

We know that...

$$P(T > t) = 1 - P(T \leq t)$$

and, by definition, the cdf of T is...

$$P(T \leq t)$$

So, lets flip the graph...

D. Find the probability that the gear lasts anywhere between 8 and 10 years.

We know that...

$$P(t_1 \leq T \leq t_2) = F_T(t_2) - F_T(t_1)$$

if $t_2 > t_1$, so...

$$P(8 \leq T \leq 10) = 0.6900 - 0.1557$$

$$P(8 \leq T \leq 10) = 0.5343$$

```
In [19]: # Your code here
prob8to10 = T.cdf(10) - T.cdf(8)
print('Probability that gear lasts between 8 and 10 years = {:.4f}'.format(prob8to10))

Probability that gear lasts between 8 and 10 years = 0.5343
```

E. If you were to sell the gear, how many years "warranty" would you offer?

If I were a scummy product manager, I would want the product warranty to end just before the majority of customs have gear failures. If I consider 70% of customers the "majority", I would check for the value of t that makes $P(T \leq t) = 0.30$.

This number is roughly equal to $t = 8.5180$ years

```
In [20]: # Your code here
cdf_values = [(t, T.cdf(t)) for t in np.linspace(7.5, 9, 1000)]
errors = [(t, abs(0.30 - cdf)) for (t, cdf) in cdf_values]
min_error = min(errors, key=lambda error:error[1])
print('You will only need to cover 30% of warranties if coverage ends at t = {:.4f}'.format(min_error[0]))

You will only need to cover 30% of warranties if coverage ends at t = 8.5180 years
```

Problem 4 - Joint probability mass function of two discrete random variables

Consider two random variables X and Y . X takes values $\{0, 1, \dots, 4\}$ and Y takes values $\{0, 1, \dots, 8\}$. Their joint probability mass function, can be described using a matrix:

```
In [21]: P = np.array([[0.03607908, 0.03760034, 0.00503184, 0.02050882, 0.01051408,
    [0.03776221, 0.00131325, 0.03760817, 0.01770659],
    [0.03750162, 0.04317351, 0.03869997, 0.03049872, 0.02176718],
    [0.0479769, 0.01021053, 0.00324185, 0.02475319],
    [0.03770951, 0.01053285, 0.01227089, 0.0339596, 0.02296711],
    [0.02187814, 0.01925622, 0.0196836, 0.01996279],
    [0.02845139, 0.01209429, 0.02450163, 0.00874645, 0.03612603],
    [0.02352593, 0.00303314, 0.00134687, 0.04071351],
    [0.00940187, 0.04633153, 0.01094094, 0.00172007, 0.00092633],
    [0.0202679, 0.02536328, 0.0352956, 0.01107725]])

The rows of the matrix correspond to the values of  $X$  and the columns to the values of  $Y$ . So, if you wanted to find the probability of  $p(X = 2, Y = 3)$  you would do:
```

```
In [22]: print('P(X=2, Y=3) = {:.14f}'.format(P[2, 3]))

P(X=2, Y=3) = 0.034
```

A. Verify that $\sum_{x,y} P(X = x, Y = y) = 1$.

```
In [23]: # Your code here
P_sum = P.sum()
print('Sum of all probabilities = {:.14f}'.format(verification))

Sum of all probabilities = 1.0000
```

B. Find the marginal probability density of X :

$$p(x) = \sum_y p(x, y).$$

You can represent this as a 5-dimensional vector.

```
In [24]: # Hint, you can do this in one line if you read this:
p_x = np.sum(P, axis=1)
print('Probabilities of X = 0, 1, 2, 3, 4')
print(p_x)
```

Probabilities of X = 0, 1, 2, 3, 4
[0.20412376 0.25783426 0.19822111 0.17820324 0.16161762]

C. Find the marginal probability density of Y . This is a 9-dimensional vector.

```
In [25]: # Your code here
p_y = np.sum(P, axis=0)
print('Probabilities of Y = 0, 1, 2, 3, 4, 5, 6, 7, 8')
print(p_y)
```

Probabilities of Y = 0, 1, 2, 3, 4, 5, 6, 7, 8
[0.14914347 0.14973292 0.0944527 0.09563304 0.09230073 0.15128076 0.05914682 0.09709805 0.11421933]

D. Find the expectation and variance of X and Y .

```
In [26]: # Your code here
E_x = sum((i * p) for (i, p) in enumerate(p_x))
print('E[X] = {:.4f}'.format(E_x))
E_y = sum((i * p) for (i, p) in enumerate(p_y))
print('E[Y] = {:.4f}'.format(E_y))

E_x2 = sum((i ** 2 * p) for (i, p) in enumerate(p_x))
V_x = E_x2 - E_x ** 2
print('V[X] = {:.4f}'.format(V_x))

E_y2 = sum((i ** 2 * p) for (i, p) in enumerate(p_y))
V_y = E_y2 - E_y ** 2
print('V[Y] = {:.4f}'.format(V_y))

E[X] = 1.8354
E[Y] = 3.6935
V[X] = 1.8719
V[Y] = 7.1906
```

E. Find the expectation of $E[X + Y]$.

```
In [27]: # Your code here
E_x_y = E_x + E_y
print('E[X+Y] = {:.14f}'.format(E_x_y))

E[X+Y] = 5.5288
```

F. Find the covariance of X and Y . Are the two variable correlated? If yes, are they positively or negatively correlated?

```
In [28]: # Your code here.
C_xy = 0.0 # Keeping track of the sum
for x in range(5):
    C_xy += sum((i - E_x)*(j - E_y) * P(x, j) for j in range(9))
print('C(X,Y) = {:.14f}'.format(C_xy))
print('Yes, X and Y are positively correlated.')
```

C(X,Y) = 0.3187
Yes, X and Y are positively correlated.

G. Find the variance of $X + Y$.

```
In [29]: # Your code here
V_x_y = V_x + V_y + 2 * C_xy
print('V[X + Y] = {:.4f}'.format(V_x_y))

V[X + Y] = 9.6998
```

J. Find the probability that $X + Y$ is less than or equal to 5. That is, find $p(X + Y \leq 5)$. Hint: Use two for loops to go over all the combinations of X and Y values, check if $X + Y \leq 5$, and sum up the probabilities.

```
In [30]: # Your code here
P_sum = 0.0 # Keeping sum of probabilities for X+Y <= 5
for x in range(5):
    for y in range(9):
        if x + y <= 5:
            P_sum += P(x, y)
print('P(X + Y <= 5) = {:.14f}'.format(P_sum))

P(X + Y <= 5) = 0.3187
```


$$P(X \rightarrow Y \leftarrow S) = 0.5346$$

Problem 5 - Creating a stochastic model for the magnetic properties of steel

The magnetic properties of steel are captured in the so called $B-H$ curve) which connects the magnetic field H to the magnetic flux density B . The shape of this curve depends on the manufacturing process of the steel. As a result the $B-H$ differs across different suppliers but also across time for the same supplier.

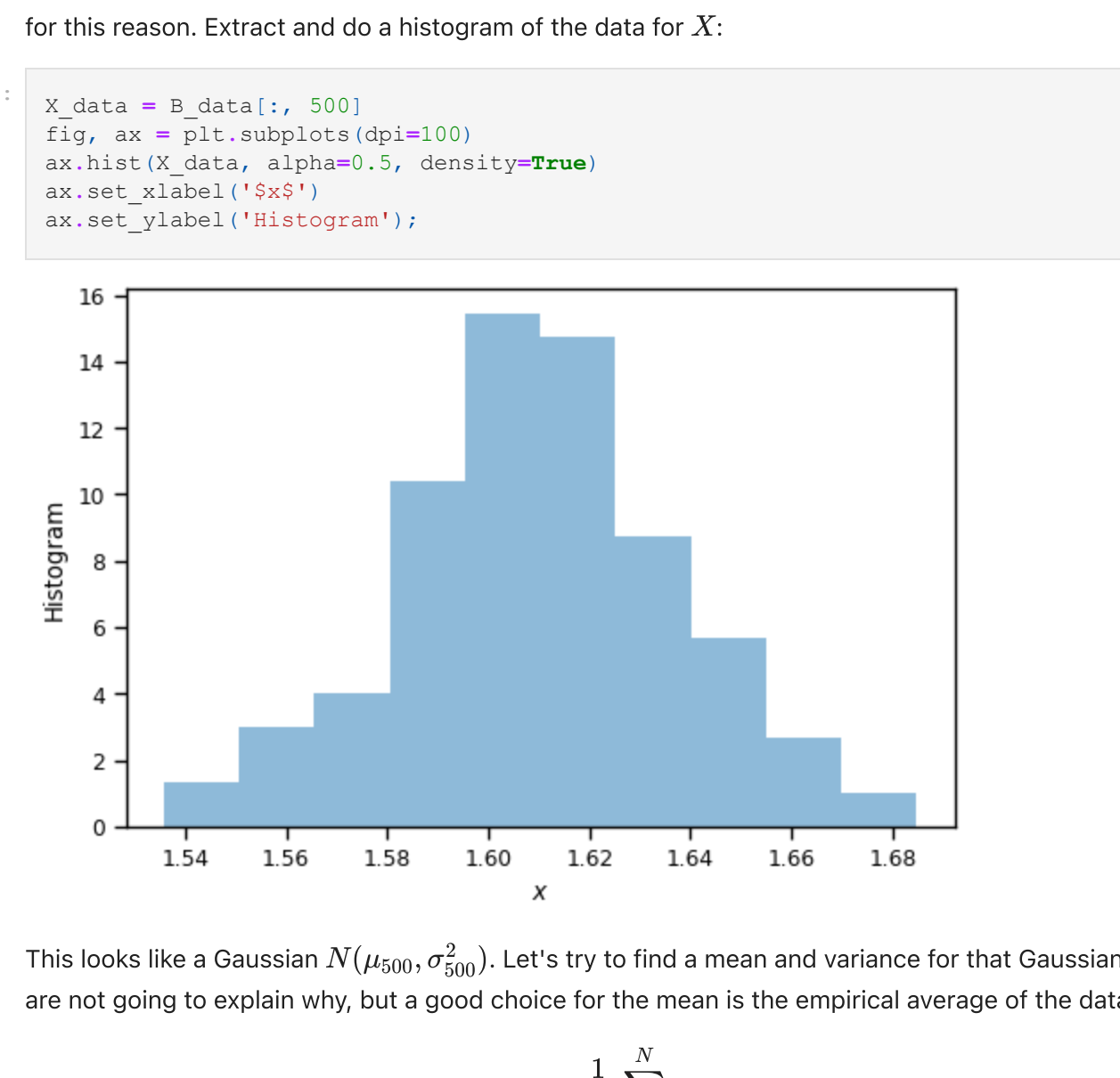
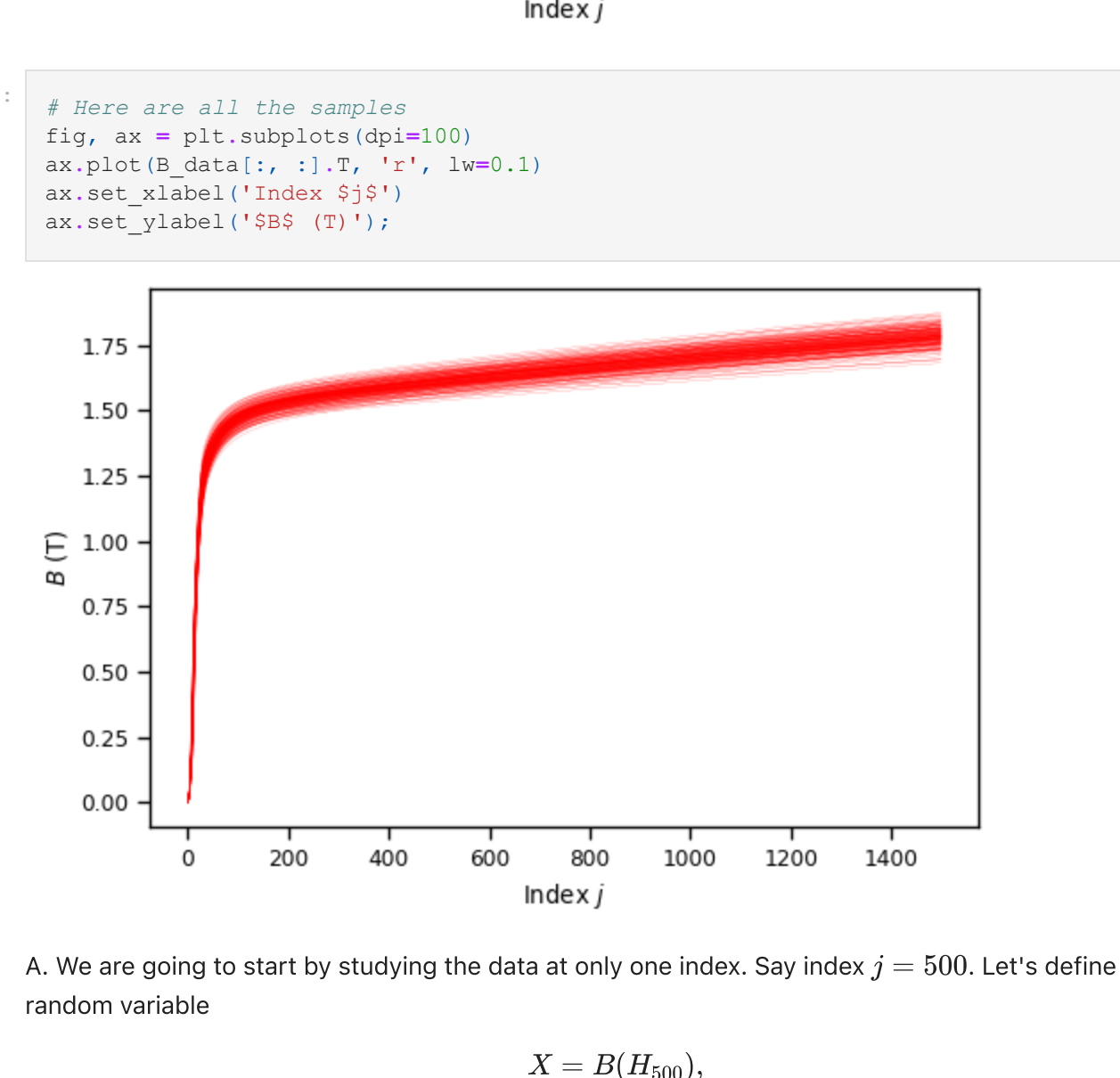
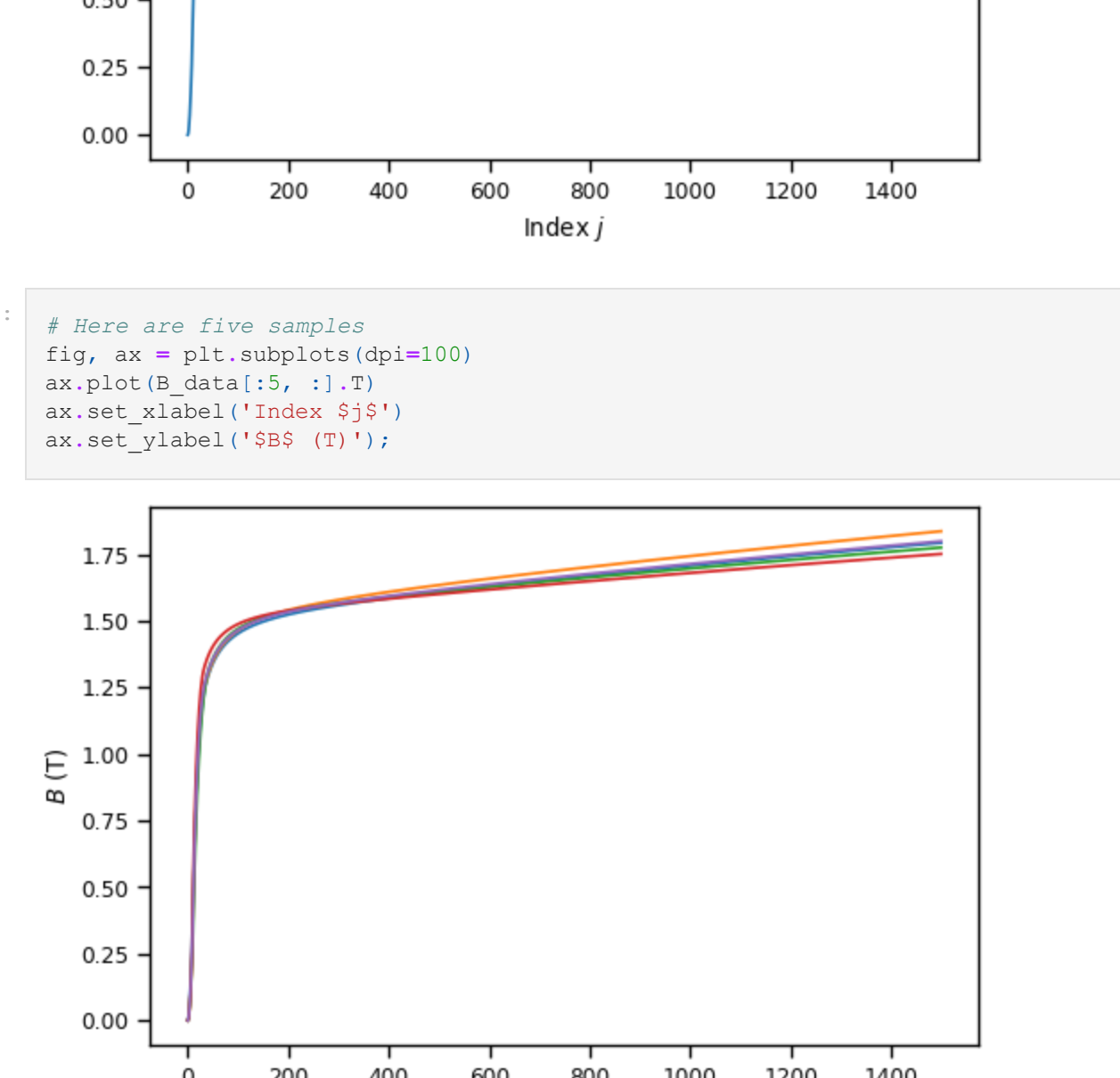
Let's use some real manufacturer data to visualize these differences. The data are [here](#). It will take a while to explain how to upload data on Google Colab. We will do it in the next homework set. For now, you should just know that the data file `B_data.csv` needs to be in the same working directory as this Jupyter notebook. We have written a piece of code that allows you to put the data file in the right place without too much trouble. Simply run the following:

```
In [31]: url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/homework1/download(url)
```

If everything worked well, then the following will work:

```
In [32]: # Load the data - You need the B_data.csv
B_data = np.loadtxt('B_data.csv')
# This file contains a single matrix the shape of which is:
print(B_data.shape)
```

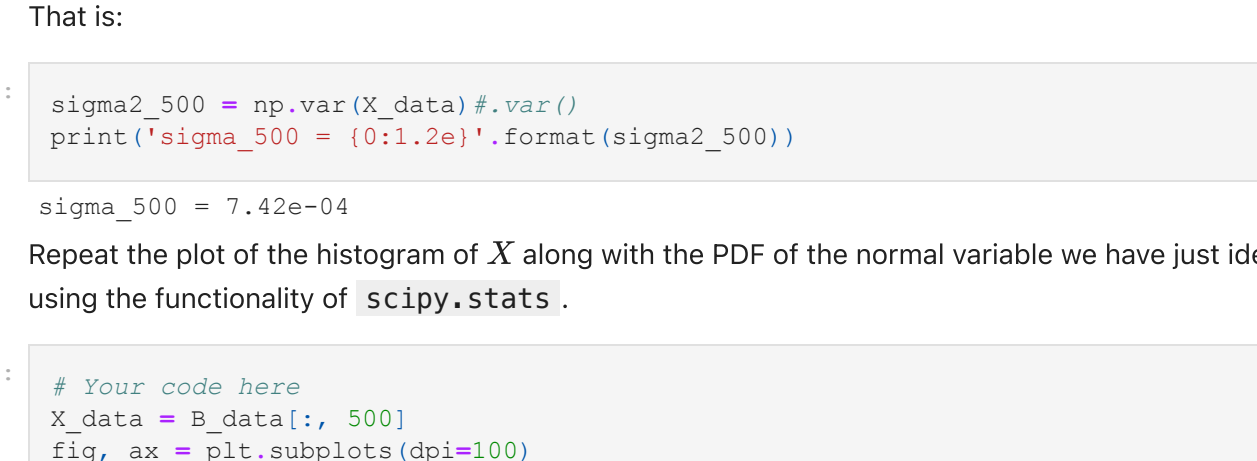
The rows (200) corresponds to different samples of the $B-H$ curves (different suppliers and different times). The columns (1500) corresponds to different values of H . That is, the i, j element is the value of B at the a specific value of H , say H_j . The values of H are the equidistant and identical and we are going to ignore them in this analysis. Let's visualize some of the samples.



A. We are going to start by studying the data at only one index. Say index $j = 500$. Let's define a random variable

$$X = B(H_{500}),$$

for this reason. Extract and do a histogram of the data for X :



This looks like a Gaussian $\mathcal{N}(\mu_{500}, \sigma_{500}^2)$. Let's try to find a mean and variance for that Gaussian. We are not going to explain why, but a good choice for the mean is the empirical average of the data:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N B_{ij}.$$

That is:

```
In [37]: mu_500 = X_data.mean()
print('mu_500 = {0:1.2f}'.format(mu_500))
```

mu_500 = 1.61

Similarly, for the variance a good choice is the empirical variance defined by:

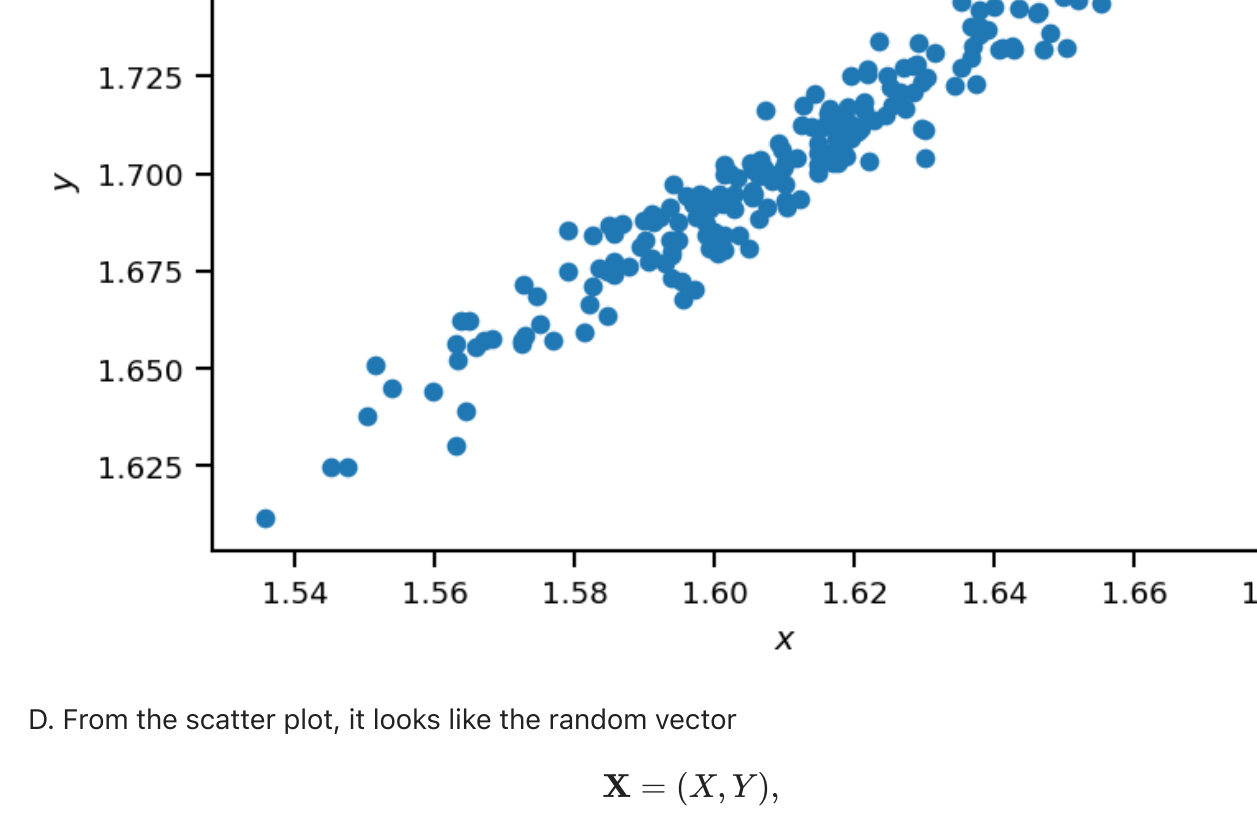
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (B_{ij} - \mu_j)^2.$$

That is:

```
In [38]: sigma2_500 = np.var(X_data) #.var()
print('sigma_500 = {0:1.2e}'.format(sigma2_500))
```

sigma_500 = 7.42e-04

Repeat the histogram of X along with the PDF of the normal variable we have just identified using the functionality of `scipy.stats`.



B. Using your normal approximation to the PDF of X , find the probability that $X = B(H_{500})$ is greater than 1.66 T.

```
In [40]: # Your code here
p_166 = 1-X.cdf(1.66)
print('P(X > 1.66) = {0:1.4}'.format(p_166))
```

P(X > 1.66) = 0.03436

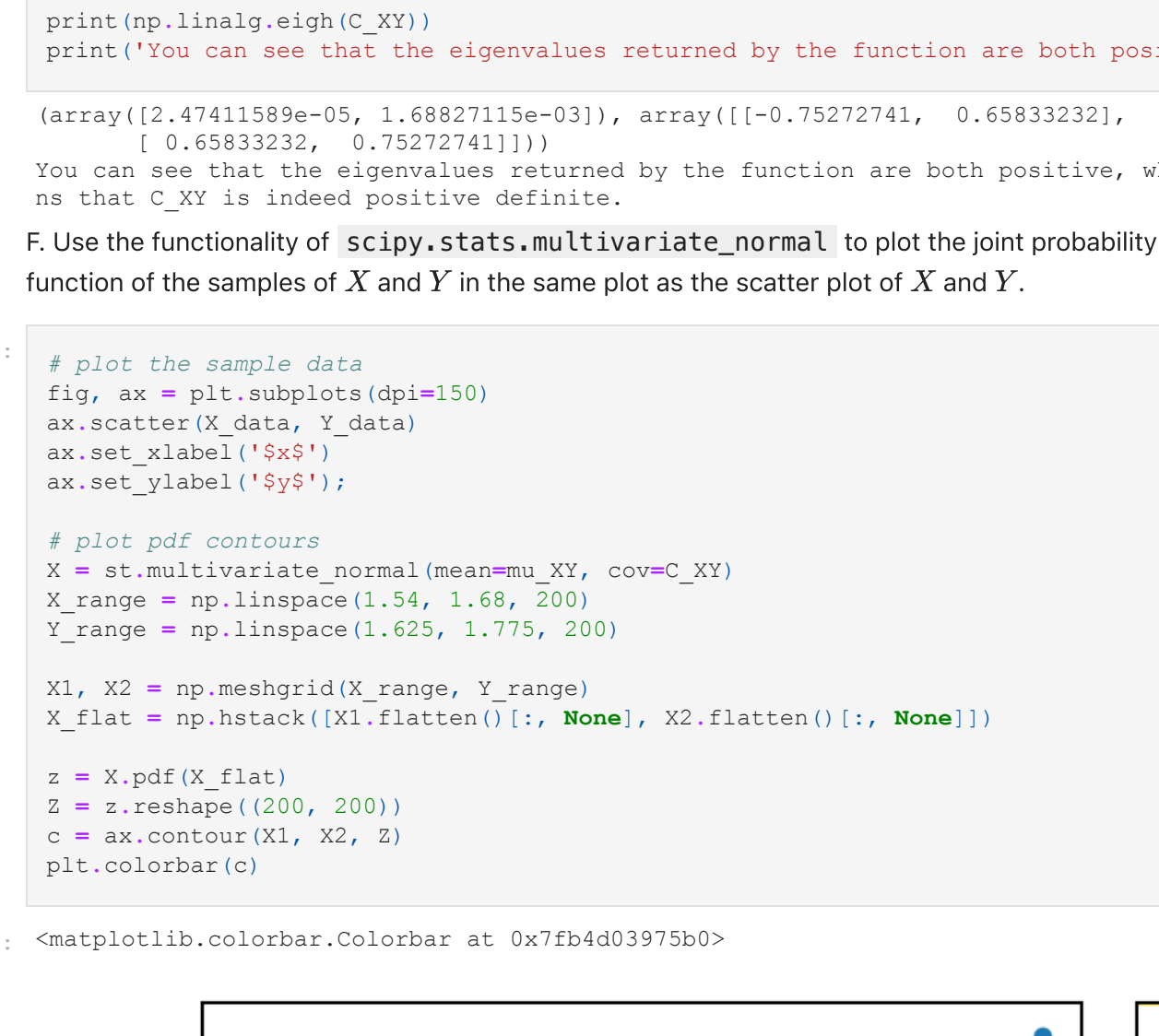
C. Let us now consider another random variable

$$Y = B(H_{1000}).$$

Isolate the data for this as well:

```
In [41]: Y_data = B_data[:, 1000]
```

Do the `scatter` plot of X and Y :



D. From the scatter plot, it looks like the random vector

$$\mathbf{X} = (X, Y),$$

follows a multivariate normal distribution. What would be the mean and covariance of the distribution. Well, first organize the samples of X and Y in a matrix with the number of rows being the number of samples and two columns (one corresponding to X and one to Y).

```
In [43]: XY_data = np.hstack([X_data[:, None], Y_data[:, None]])
```

The mean vector is:

```
In [44]: mu_XY = np.mean(XY_data, axis=0)
print(mu_XY)
```

[1.61041566 1.70263681]

The covariance matrix is a little bit trickier. We have already discussed how to find the diagonals of the covariance matrix (it is simply the variance). For the off-diagonal terms, this is the formula that is being used:

$$C_{jk} = \frac{1}{N} \sum_{i=1}^N (B_{ij} - \mu_j)(B_{ik} - \mu_k).$$

This is how you can find it:

```
In [45]: # Careful with np.cov because it requires you to transpose the matrix
C_XY = np.cov(XY_data.T)
print(C_XY)
```

```
[[0.00074572 0.00082435]
 [0.00082435 0.00096729]]
```

Are the two variables X and Y positively or negatively correlated?

Answer:

From the (1,2) and (2,1) entries in the covariance matrix, you can see that X and Y are positively correlated (value > 0).

E. Use `np.linalg.eigh` to check that the matrix `C_XY` is indeed positive definite.

```
In [46]: # Your code here
print(np.linalg.eigh(C_XY))
print('You can see that the eigenvalues returned by the function are both positive, w
```

```
(array([2.47411589e-05, 1.68827115e-03]), array([[[-0.75272741, 0.65833232],
[0.65833232, 0.75272741]]])
```

You can see that the eigenvalues returned by the function are both positive, which means that C_XY is indeed positive definite.

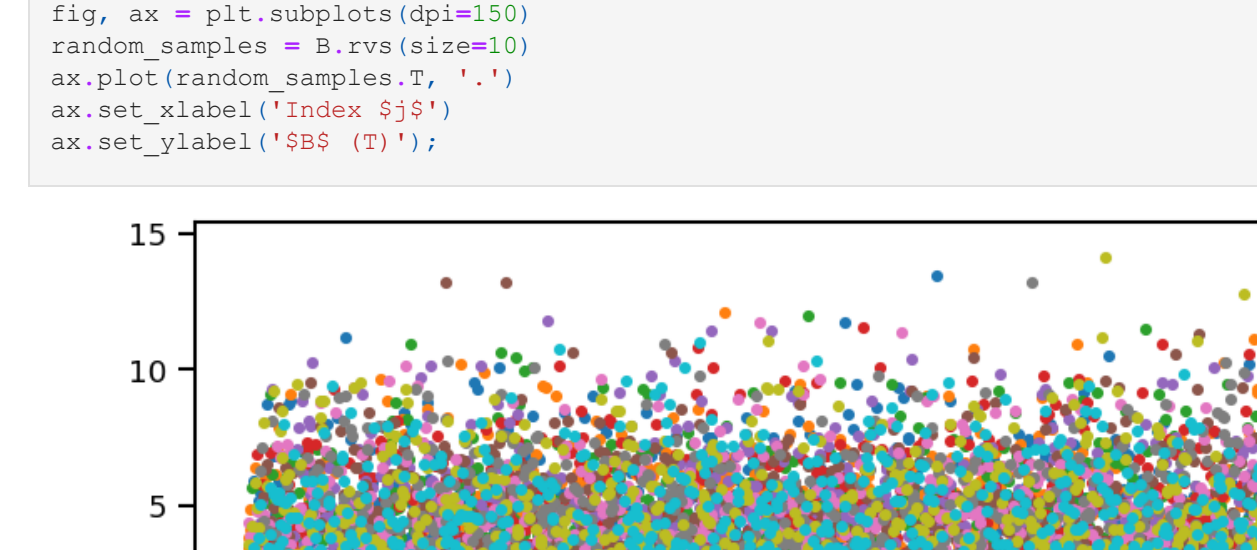
F. Use the functionality of `scipy.stats.multivariate_normal` to plot the joint probability function of the samples of X and Y in the same plot as the scatter plot of X and Y .

```
In [47]: # plot the sample data
fig, ax = plt.subplots(dpi=150)
ax.scatter(X_data, Y_data)
ax.set_xlabel('X ($T$)')
ax.set_ylabel('Y ($T$)');
```

```
# plot pdf contours
X = st.multivariate_normal(mean=mu_XY, cov=C_XY)
X_range = np.linspace(1.54, 1.68, 200)
Y_range = np.linspace(1.625, 1.775, 200)
```

```
X1, X2 = np.meshgrid(X_range, Y_range)
X_flat = np.hstack([X1.flatten()[1:], None], X2.flatten()[1:], None])
z = X.pdf(X_flat)
Z = z.reshape((200, 200))
c = ax.contour(X1, X2, Z)
plt.colorbar(c)
```

```
Out[47]: <matplotlib.colorbar.Colorbar at 0x7fb4d03975b0>
```



G. Now, let's think each $B-H$ curve as a random vector. That is, the random vector \mathbf{B} corresponds to the values of the magnetic flux density at a fixed number of H -values. It is:

$$\mathbf{B} = (B(H_1), \dots, B(H_{1500})).$$

It is like $\mathbf{X} = (X, Y)$ only now we have 1500 dimensions instead of 2. First, let's find the mean of this random vector:

```
In [48]: B_mu = np.mean(B_data, axis=0)
# Let's plot the mean on top of the data
# Here are all the samples
fig, ax = plt.subplots(dpi=100)
ax.plot(B_data[:, :100], label='B', lw=0.1)
ax.plot(B_mu)
ax.set_xlabel('Index $j$')
ax.set_ylabel('B ($T$)');
```



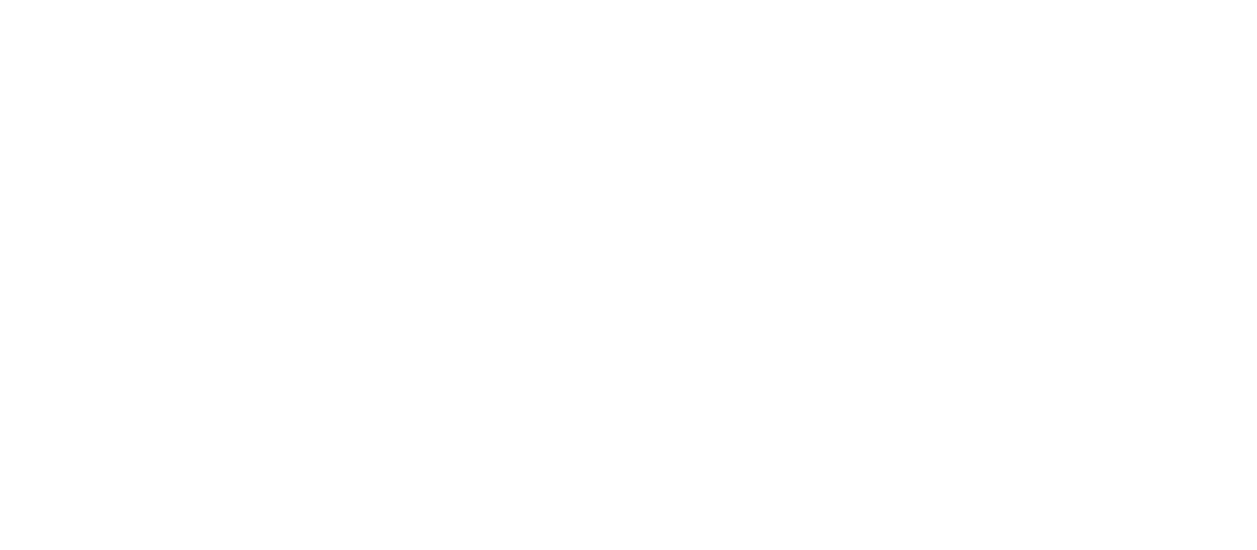
It looks good. Now, find the covariance matrix of \mathbf{B} . This is going to be a 1500x1500 matrix.

```
In [49]: B_cov = np.cov(B_data.T)
```

Let's plot this matrix:

```
In [50]: fig, ax = plt.subplots(dpi=100)
c = ax.imshow(B_cov, interpolation='nearest')
plt.colorbar(c)
```

```
Out[50]: <matplotlib.colorbar.Colorbar at 0x7fb4d03975b0>
```



You see that the values are quite a bit correlated. This makes sense because the curves are all very smooth and they look very much alike. But let's check if the covariance is indeed positive definite:

```
In [51]: print('Eigenvalues of B_cov:')
print(np.linalg.eigh(B_cov)[0])
```

```
Eigenvalues of B_cov:
[-3.34668263e-16 -3.34612879e-16 -7.43049575e-17 ... 4.66244763e-02
1.16644070e-01 1.20726782e+00]
```

Hmm, notice that there are several eigenvalues that are negative, but they are too small. Very close to zero. This happens very often in practice when you are finding the covariance of a very large random vectors. It arises from the fact that we are using floating point arithmetic instead of a real numbers. It is a numerical artifact. If you tried to use this covariance to make a multivariate normal random vector using `scipy.stats` it would fail. Try this:

```
In [52]: B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

```
-----
LinAlgError: Traceback (most recent call last)
--> 1 B = st.multivariate_normal(mean=B_mu, cov=B_cov)
~/opt/anaconda3/lib/python3.8/site-packages/scipy/stats/_multivariate.py in __call__(self, mean, cov, allow_singular, seed)
365         return multivariate_normal_frozen(mean, cov,
366                                           allow_singular=allow_singular,
367                                           seed=seed)
~/opt/anaconda3/lib/python3.8/site-packages/scipy/stats/_multivariate.py in __init__(self, mean, cov, allow_singular, seed, maxpts, abseps, releps)
743         self.dim, self.mean, self.cov = self._dist._process_parameters(
744             None, mean, cov)
--> 745         self.cov_info = _PSD(self.cov, allow_singular=allow_singular)
746         if not maxpts:
747             maxpts = 1000000 * self.dim
LinAlgError: singular matrix
```

The way to overcome this problem is to add a small positive number to the diagonal. This needs to be very small so that the distribution does not change very much. It must be the smallest possible number that makes the covariance matrix behave well. This is known as the *jitter* or the *nugget* . Find the nugget playing with the code below. Every time you try, multiply the nugget by ten.

```
In [53]: # Pick the nugget here
nugget = 1e-12 * 10
# This is the modified covariance matrix
B_cov_w_nugget = B_cov + nugget * np.eye(B_cov.shape[0])
# Try building the distribution:
try:
    B = st.multivariate_normal(mean=B_mu, cov=B_cov_w_nugget)
    print('It worked! Move on.')
except:
    print('It did not work. Increase nugget by 10.')
```

It worked! Move on.

H. Now you have created your first stochastic model of a complicated physical quantity. By sampling from your newly constructed random vector \mathbf{B} you have essentially quantified your uncertainty about the $B-H$ curve as induced by the inability to perfectly control the production of steel. Take 10 samples of this random vector and plot them.

```
In [54]: # Your code here
fig, ax = plt.subplots(dpi=150)
random_samples = B.rvs(size=10)
ax.plot(random_samples.T, ',')
ax.set_xlabel('Index $j$')
ax.set_ylabel('B ($T$)');
```



```
In [ ]:
```