Homework 2

Due 9/15/2020.

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:** Kiara
- **Last Name:** Koeppen
- **Email:** kkoeppen@purdue.edu

```python
# Here are some modules that you may need - please run this block of code:
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_context('paper')
import numpy as np
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:
```

```
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 $\mathbb{E}[X]$.

Answer: $\mathbb{E}[X] = \sum_x xp(x)$ = (0)(0.3)+(1)(0.1)+(2)(0.2)+(3)(0.4)=1.7

```
# You can also answer with code here:
print('E[X]=',(0)*(0.3)+(1)*(0.1)+(2)*(0.2)+(3)*(0.4))
```

    E[X]= 1.7000000000000002

B. The variance $\mathbb{V}[X]$.

Answer: $\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$
$\mathbb{E}[X^2] = \sum_x x^2 p(x) = (0^2)(0.3) + (1^2)(0.1) + (2^2)(0.2) + (3^2)(0.4) = 4.5$
$\mathbb{V}[X] = (4.5) - (1.7)^2 = 1.61$

```
# You can also answer with code here:
print('E[X]^2=', (0**2)*(0.3)+(1**2)*(0.1)+(2**2)*(0.2)+(3**2)*(0.4))
print('V[X]=', 4.5-1.7**2)
```
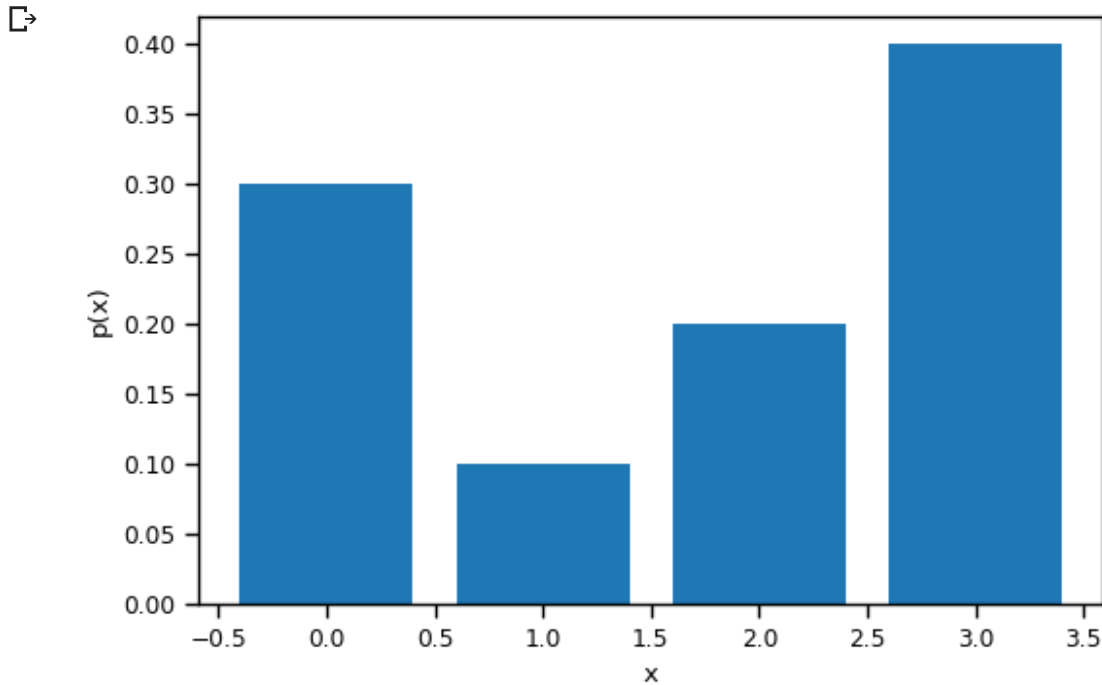
    E[X]^2= 4.5
    V[X]= 1.6100000000000003

C. Plot the probability mass function of X.

```python
# Your code here. Hint: use a bar plot
x = [0, 1, 2, 3]
p = [0.3, 0.1, 0.2, 0.4]
fig, ax = plt.subplots(dpi=100)
ax.bar(x,p)
ax.set_xlabel('x')
ax.set_ylabel('p(x)');
```


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

Answer:

$$p(X = 0 \, or \, X = 2) = p(X = 0) + p(X = 2) = 0.3 + 0.2 = 0.5$$

```python
# You can also answer with code here:
print('p(X=0 or X=2)=', p[0]+p[2])
```

```
p(X=0 or X=2)= 0.5
```

E. Find $\mathbb{E}[4X + 3]$.

Answer: $\mathbb{E}[4X + 3] = 4\mathbb{E}[X] + 3 = (4)(1.7) + 3 = 9.8$

```python
# You can also answer with code here:
(4)*(1.7)+3
```

```
9.8
```

F. Find $\mathbb{V}[4X + 3]$.

Answer: $\mathbb{V}[4X + 3] = 4^2 \mathbb{V}[X] = (4^2)(1.61) = $ <mark>25.76</mark>

```
# You can also answer with code here:
(4**2)*1.61
```

⊃ 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 occured in Southern California and collect the relevant data. We are going to start at 1900 because data before that time may are unreliable. Go over each decade and count the occurence of a major earthquake (i.e., count the number of organge and red colors in each decade). We have done this for you.

```
eq_data = np.array([
    0, # 1900-1909
    1, # 1910-1919
    2, # 1920-1929
    0, # 1930-1939
    3, # 1940-1949
    2, # 1950-1959
    1, # 1960-1969
    2, # 1970-1979
    1, # 1980-1989
    4, # 1990-1999
    0, # 2000-2009
    2 # 2010-2019
])
fig, ax = plt.subplots(dpi=100)
ax.bar(np.linspace(1900, 2019, eq_data.shape[0]), eq_data, width=10)
ax.set_xlabel('Decade')
ax.set_ylabel('# of major earthquakes in Southern CA');
```

⊃
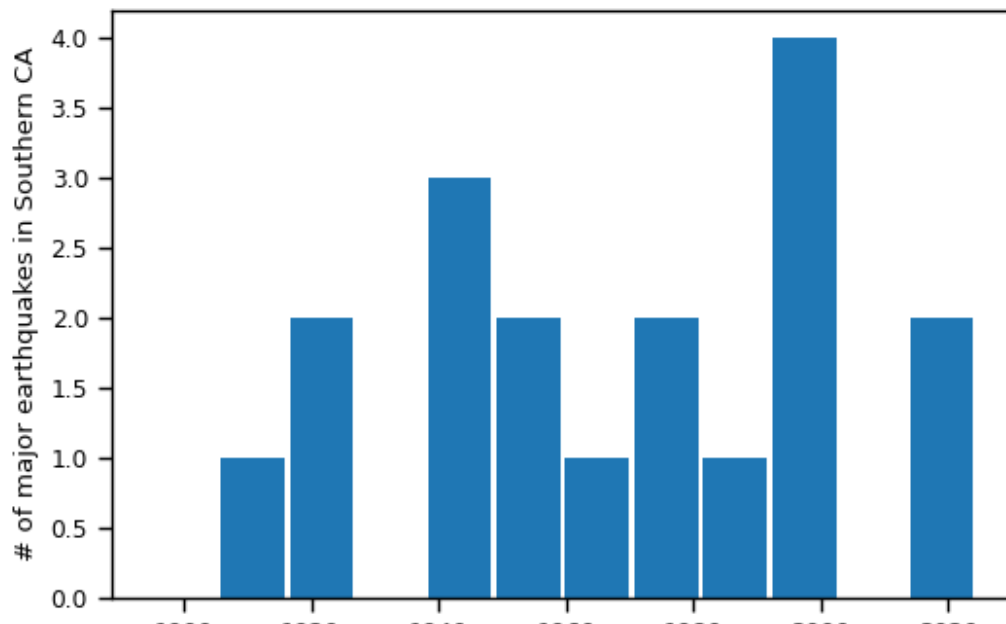
B. The [Poisson distribution](#) is a discrete distribution with values $\{0, 1, 2, \ldots\}$ which is commonly used to model the number of events occuring in a certain time period. It is the right choice when these events are happening independently and the probability of any event happening over a small period of time is constant. Let's use the Poisson to model the number of earthquakes X occuring 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:

```
r = np.mean(eq_data)
print('r = {0:1.2f} 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](#)):

```
X = st.poisson(r)
```

A. Plot the probability mass function of `x`.

```
# Your code here
fig, ax = plt.subplots(dpi=100)
ax.bar(eq_data, X.pmf(eq_data))
ax.set_xlabel('x (Number of Major Earthquakes in a Decade)')
ax.set_ylabel('p(x)');
```
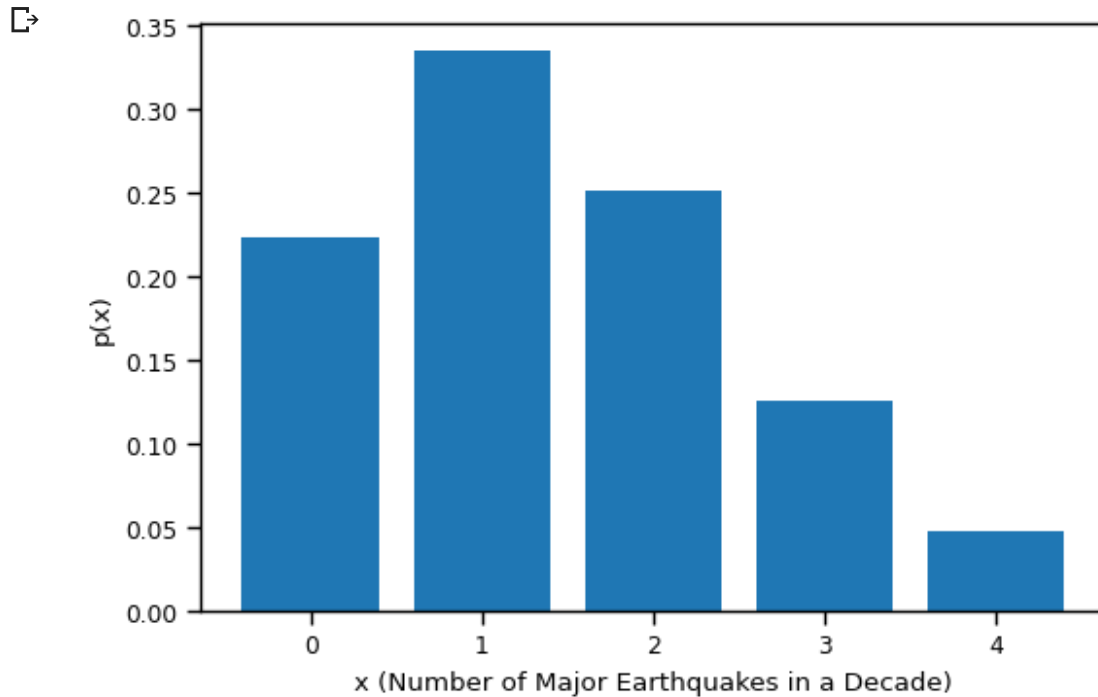

B. What is the probability that at least one major earthquake will occur during the next decade?

Answer:

The obvious rule:

1-X.pmf(0)=0.77687

```
# You can also answer with code here:
1-X.pmf(0)
```

0.7768698398515702

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.

Answer:

$$p(X_1 + X_2 \geq 1) = 1 - p(X_1 + X_2 = 0)$$
$$p(X_1 + X_2 \geq 1) = 1 - p(X_1 = 0, X_2 = 0)$$
$$p(X_1 + X_2 \geq 1) = 1 - p(X_1 = 0)p(X_2 = 0) = 0.9502$$

```
# You can also answer with code here:
1-X.pmf(0)**2
```

⌷→ 0.950212931632136

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

Answer:

0.999447

```
# You can also answer with code here:
1-X.pmf(0)**5
```

⌷→ 0.9994469156298522

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 opearating 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.

```
# Time to fail in years under normal operating conditions
# Each row is a different gear
time_to_fail_data = np.array([
    10.5,
    7.5,
    8.1,
    8.4,
    11.2,
    9.3,
    8.9,
    12.4
])

# Here is a Weibull distribution fitted to the data
fitted_params = st.exponweib.fit(time_to_fail_data, loc=0)
T = st.exponweib(*fitted_params)
# Fit picks the parameters of the data to match the distribution
# We will talk about what it does in subsequent lectures.

# Let's plot the probability density of this
fig, ax = plt.subplots(dpi=100)
ts = np.linspace(0.0, 20.0, 100)
ax.plot(ts, T.pdf(ts))
ax.set_xlabel('$t$ (years)')
ax.set_ylabel('$p(t)$');
```
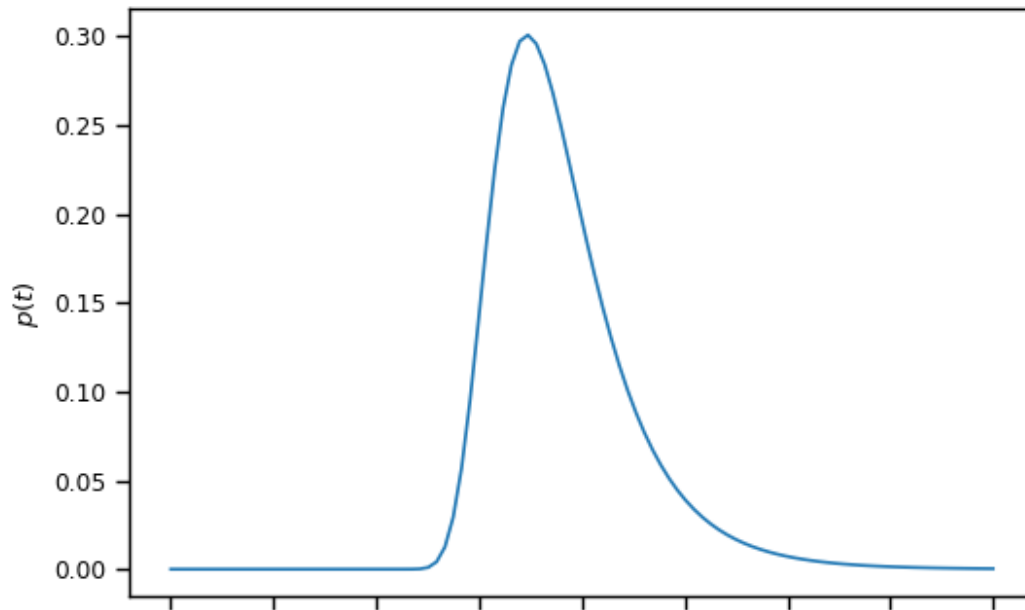
A. Find the mean fail time and its variance. Hint: Do not integrate anything by hand. Just use the functionality of `scipy.stats`.
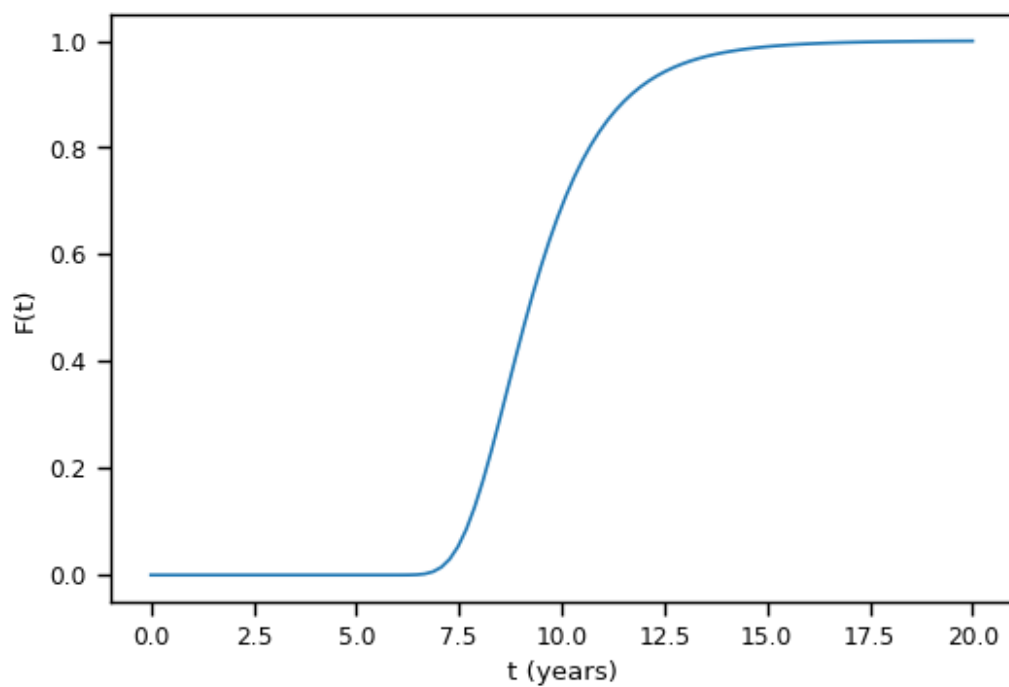
```
# Your code here
t_mean = np.average(time_to_fail_data)
t_var = np.var(time_to_fail_data)
print('E[T] = {0:1.2f}'.format(t_mean))
print('V[T] = {0:1.2f}'.format(t_var))
```

```
E[T] = 9.54
V[T] = 2.48
```

B. Plot the cumulative distribution function of T.

```
# Your code here
fig, ax = plt.subplots(dpi=100)
ax.plot(ts, T.cdf(ts))
ax.set_xlabel('t (years)')
ax.set_ylabel('F(t)')
```
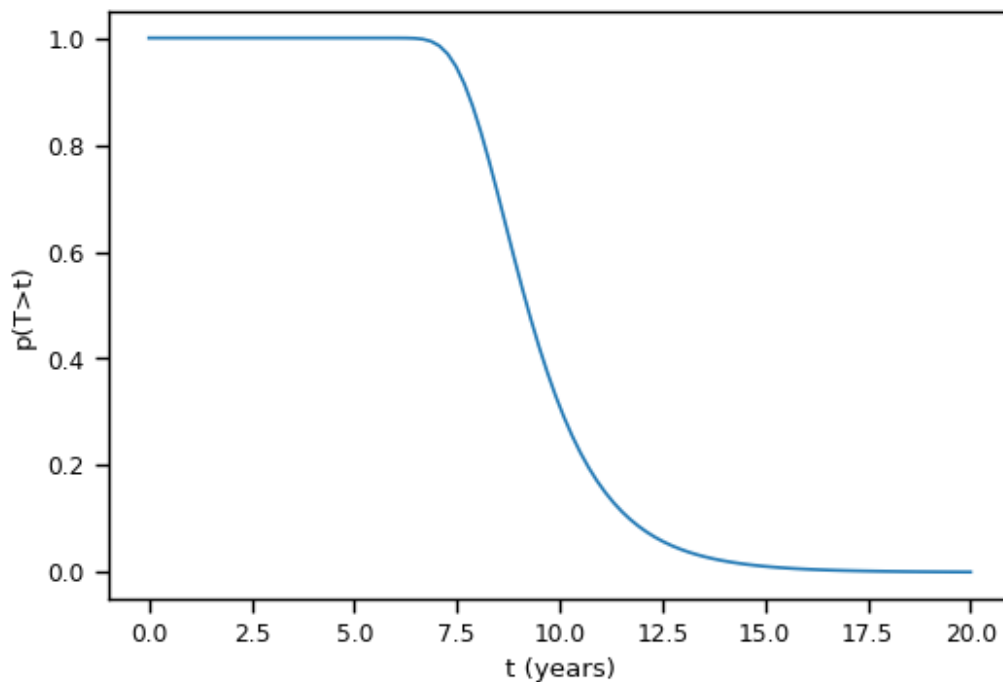
⊳ Text(0, 0.5, 'F(t)')

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.

```python
# Your code here
fig, ax = plt.subplots(dpi=100)
ax.plot(ts, 1-T.cdf(ts))
ax.set_xlabel('t (years)')
ax.set_ylabel('p(T>t)')
```

Text(0, 0.5, 'p(T>t)')

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

```
# Your code here
T.cdf(10)-T.cdf(8)
```

> 0.5342898192604528

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

Answer:

There are many factors that would ideally play in deciding how many years of warranty a company would offer. Ideally knowing the profit margins, industry these gears are being applied in, and the cost to produce these gears would be beneficial, but since we don't know this information I will assume that there is a low cost to manufacture these gears with a large profit margin. Since it doesn't cost much to manufacture these I would say offering a warranty that covers 90% of failures would be good so our customers are satisfied. Plugging the numbers in we find that a 12 year warranty will cover about 90% of failures, so I would say that would be a good place to start.

If we say this gear will be used in a powertrain, it says the typical warranty coverage is 5 years (https://www.carchex.com/content/what-does-a-powertrain-warranty-cover). So perhape if these gears are being applied to a system that has it's own standard warranty it'd only be 5 years (the buyer won't be getting recovery after 5 years so they wouldn't request for a suppliers part warranty to be greater than the warranty they're offering their customers). However for this gear if we offered only a 5 year warranty it'd cover basically none of the failures so this would not be good for our customer. Because of this, I'll assume that this is not being used in a powertrain and the 5 year warranty is not applicable.

Again, there's no 'right' answer because there are so many unknown variables but I believe starting at 90% would ensure customer satisfaction.

```
# Your code here
T.cdf(12)
```

> 0.918379553519117

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

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

```
P = np.array([[0.03607908, 0.03760034, 0.00503184, 0.0205082 , 0.01051408,
        0.03776221, 0.00131325, 0.03760817, 0.01770659],
       [0.03750162, 0.04317351, 0.03869997, 0.03069872, 0.02176718,
        0.04778769, 0.01021053, 0.00324185, 0.02475319],
       [0.03770951, 0.01053285, 0.01227089, 0.0339596 , 0.02296711,
        0.02187814, 0.01925662, 0.0196836 , 0.01996279],
       [0.02845139, 0.01209429, 0.02450163, 0.00874645, 0.03612603,
        0.02352593, 0.00300314, 0.00103487, 0.04071951],
       [0.00940187, 0.04633153, 0.01094094, 0.00172007, 0.00092633,
        0.02032679, 0.02536328, 0.03552956, 0.01107725]])
X=np.array([0,1,2,3,4])
Y=np.array([0,1,2,3,4,5,6,7,8])
```

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:

```
print('p(X=2, Y=3) = {0:1.3f}'.format(P[2, 3]))
```

⌐→ p(X=2, Y=3) = 0.034

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

```
# Your code here
np.sum([P])
```

⌐→ 0.9999999900000001

B. Find the marginal probability density of X:
$$p(x) = \sum_y p(x, y).$$
You can represent this as a 5-dimensional vector.

```
mpdX=np.sum(P, axis =1)
print('pmf of just X:', mpdX)
```

⌐→ pmf of just X: [0.20412376 0.25783426 0.19822111 0.17820324 0.16161762]

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

```
# Your code here
mpdY=np.sum(P, axis=0)
print('pmf of just Y:', mpdY)
```

> pmf of just Y: [0.14914347 0.14973252 0.09144527 0.09563304 0.09230073 0.15128076
> 0.05914682 0.09709805 0.11421933]

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

```
# Your code here
EX= np.sum(np.multiply(mpdX,X))
EX2= np.sum(np.multiply(mpdX, np.square(X)))
VX= EX2-np.square(EX)
EY= np.sum(np.multiply(mpdY,Y))
EY2= np.sum(np.multiply(mpdY, np.square(Y)))
VY= EY2-np.square(EY)
print('E[X]=', EX)
print('V[X]=', VX)
print('E[Y]=', EY)
print('V[Y]=', VY)
```

> E[X]= 1.83535668
> V[X]= 1.8718956371793776
> E[Y]= 3.69345081
> V[Y]= 7.1905898441103435

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

```
# Your code here
print('E[X + Y]=', EX + EY)
```

> E[X + Y]= 5.52880749

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

```
# Your code here
CXY=0.0
for x in range(5):
  for y in range(9):
    CXY += (x-EX)*(y-EY)*P[x,y]
print('C[X,Y]=', CXY)
print('Yes, the two are positively correlated because C[X,Y] is greater than 0')
```

```
C[X,Y]= 0.31867736582709294
Yes, the two are positively correlated because C[X,Y] is greater than 0
```

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

```
# Your code here
print('V[X+Y]=', VX + VY + 2*CXY)
```

```
V[X+Y]= 9.699840212943908
```

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

```
# Your code here
PA=0.0
for x in range(5):
  for y in range(9):
    if x+y<=5:
      PA += P[x,y]
print('The probability that X + Y is less than or equal to 5 is', PA)
```

```
The probability that X + Y is less than or equal to 5 is 0.5345903099999999
```

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 alos 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:

```
url = 'https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/homework/B_data.csv'
download(url)
```

If everything worked well, then the following will work:

```
# 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)
```

    (200, 1500)

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.

```
# Here is one sample
fig, ax = plt.subplots(dpi=100)
ax.plot(B_data[0, :])
ax.set_xlabel('Index $j$')
ax.set_ylabel('$B$ (T)');
```
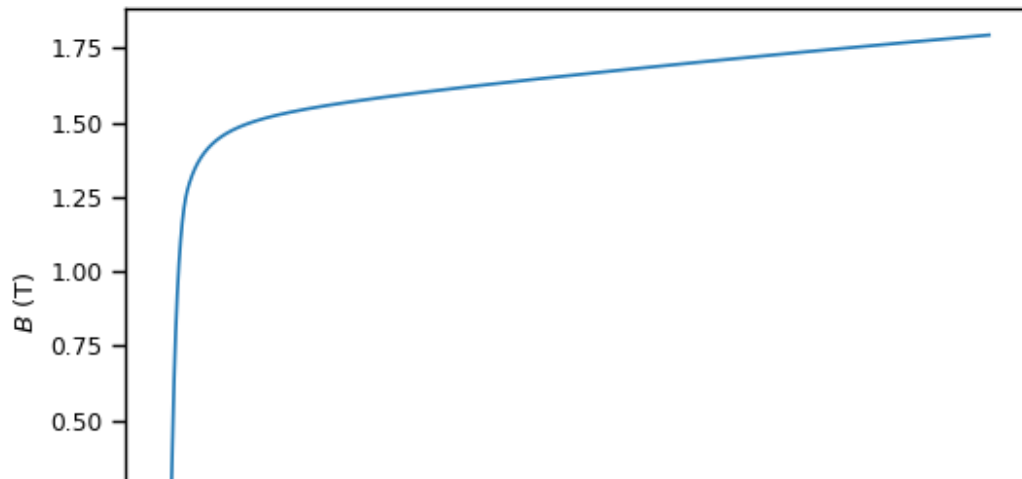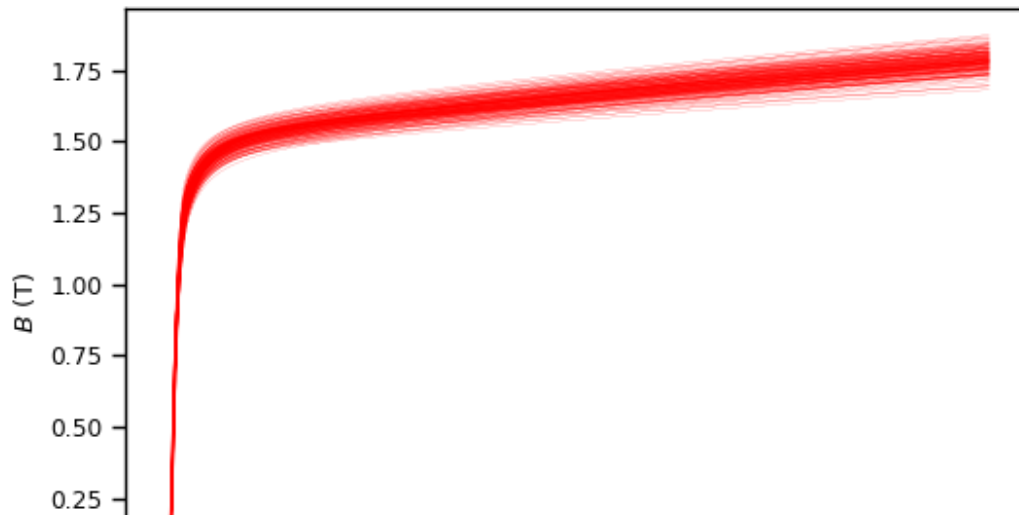
```
# Here are five samples
fig, ax = plt.subplots(dpi=100)
ax.plot(B_data[:5, :].T)
ax.set_xlabel('Index $j$')
ax.set_ylabel('$B$ (T)');
```

⊳
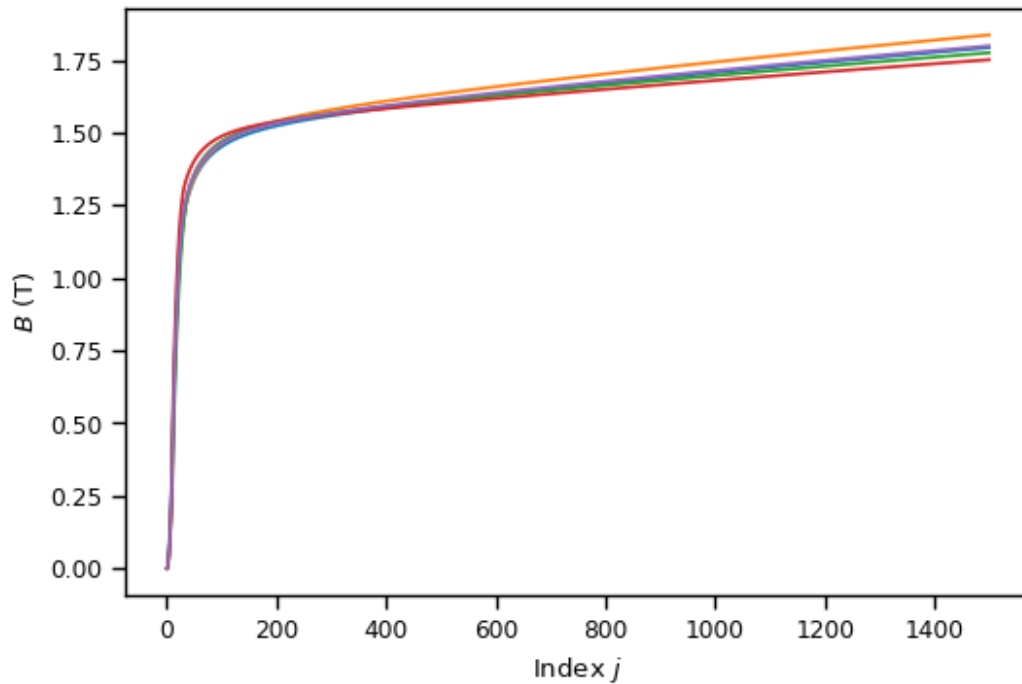

```
# Here are all the samples
fig, ax = plt.subplots(dpi=100)
ax.plot(B_data[:, :].T, 'r', lw=0.1)
ax.set_xlabel('Index $j$')
ax.set_ylabel('$B$ (T)');
```
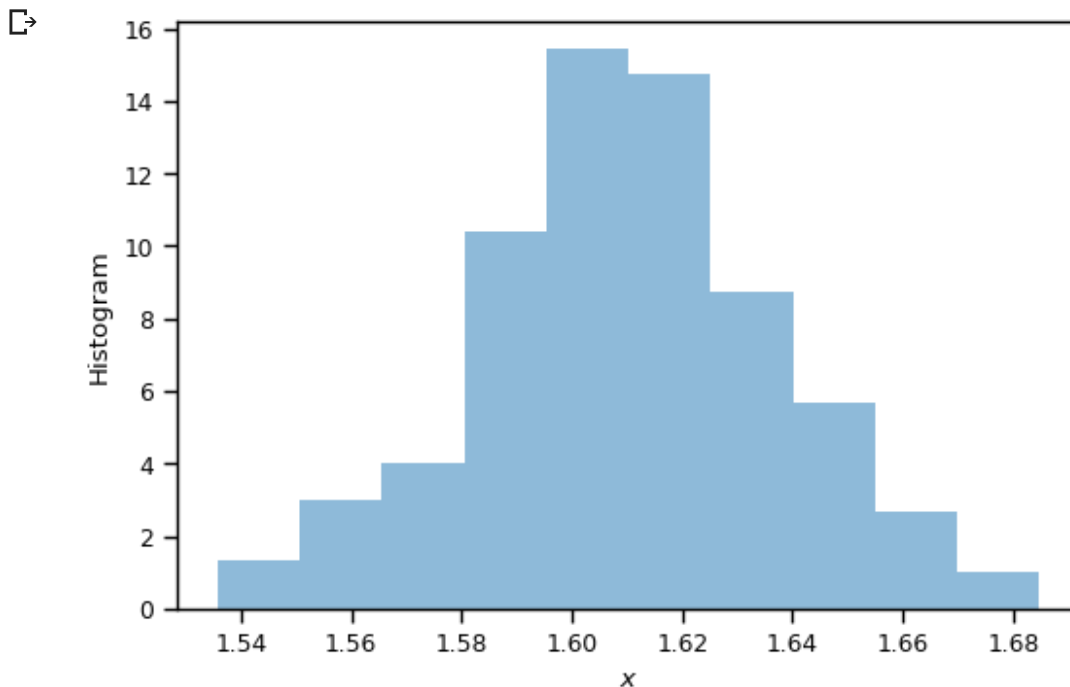
⊳

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:

```
X_data = B_data[:, 500]
fig, ax = plt.subplots(dpi=100)
ax.hist(X_data, alpha=0.5, density=True)
ax.set_xlabel('$x$')
ax.set_ylabel('Histogram');
```


This looks like a Gaussian $N(\mu_{500}, \sigma^2_{500})$. 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:

```
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:

```
sigma2_500 = np.var(X_data)#.var()
print('sigma2_500 = {0:1.2e}'.format(sigma2_500))
```

⬚→   sigma2_500 = 7.42e-04

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

```
# Your code here

X_Norm=np.linspace(1.5,1.7,1000)
px=st.norm.pdf(X_Norm, mu_500, np.sqrt(sigma2_500))
fig, ax = plt.subplots(dpi=100)
ax.hist(X_data, alpha=0.5, density=True)
ax.plot(X_Norm, px)
ax.set_xlabel('$x$')
ax.set_ylabel('Histogram');
```
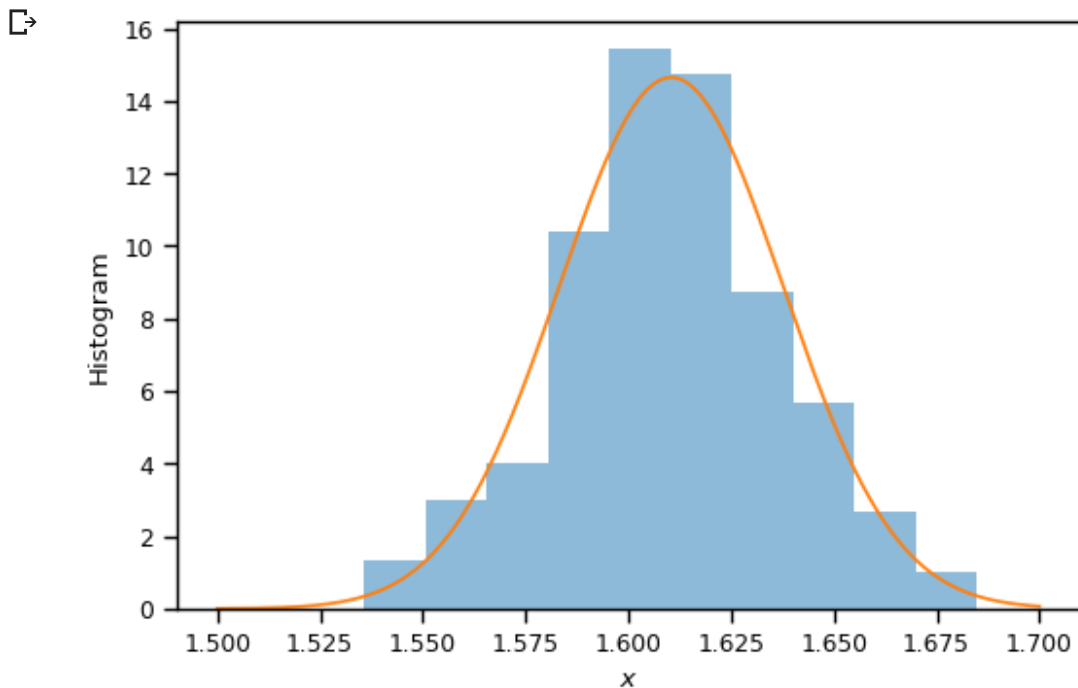

B. Using your normal approximation to the PDF of X, find the probability that $X = B(H_{500})$ is geater than 1.66 T.

```
# Your code here
X_Norm_2=st.norm(mu_500,np.sqrt(sigma2_500))
#Obvious Rule
1-X_Norm_2.cdf(1.66)
```

0.034355748631309635

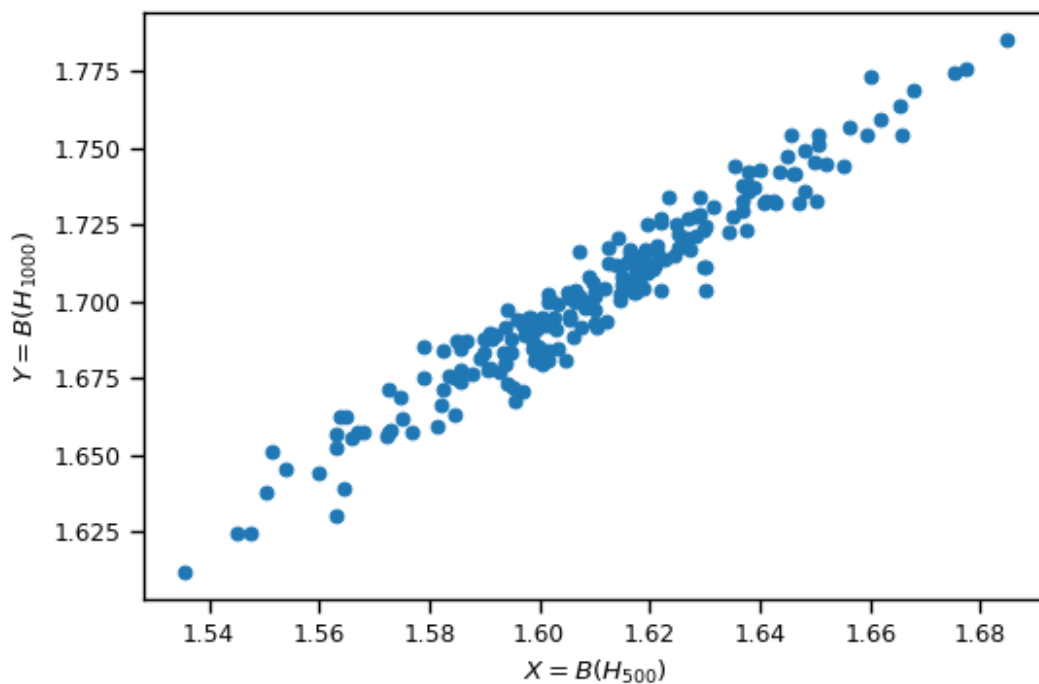C. Let us now consider another random variable

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

Isolate the data for this as well:

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

Do the `scatter` plot of X and Y:

```
# Your code here
fig, ax=plt.subplots(dpi=100)
ax.scatter(X_data, Y_data)
ax.set_xlabel('$X=B(H_{500})$')
ax.set_ylabel('$Y=B(H_{1000})$')
```

> Text(0, 0.5, '$Y=B(H_{1000})$')

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).

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

The mean vector is:

```
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:

```
# 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:

The two vectors are positively correlated because the covariance matrix has all positive values

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

```
# Your code here
print('Eigenvalues=', np.linalg.eigh(C_XY)[0])
#Because the eigenvalues are positive it means that C_XY is positive definite
```

➟ Eigenvalues= [2.47411589e-05 1.68827115e-03]

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.

```
# Your code here
#Referencing hands on activity
fig, ax= plt.subplots(dpi=100)
ax.scatter(X_data, Y_data)
XY=st.multivariate_normal(mu_XY, C_XY)
XY_x=np.linspace(1.5, 1.7, 64)
XY_y=np.linspace(1.6,1.8,64)
XY_X,XY_Y=np.meshgrid(XY_x, XY_y)
X_flat=np.hstack([XY_X.flatten()[:, None], XY_Y.flatten()[:, None]])
pdf_X=XY.pdf(X_flat).reshape(XY_X.shape)
k=ax.contour(XY_X, XY_Y, pdf_X)
plt.colorbar(k, label='$p(\mathbf{x})$')
ax.set_xlabel('$X=B(H_{500})$')
ax.set_ylabel('$Y=B(H_{1000})$')
ax.plot(XY.mean[0], XY.mean[1], 'ro', label='$\mu$')
plt.legend(loc='best')
```

<matplotlib.legend.Legend at 0x7eff2dce2908>
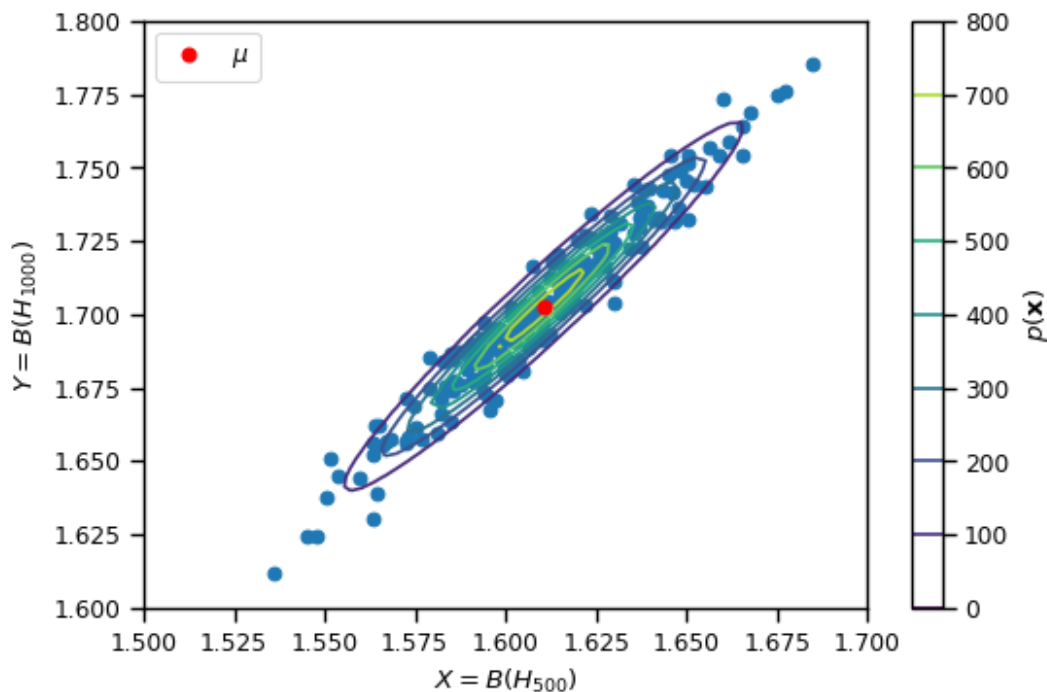
G. Now, let's think each $B - H$ curve as a random vector. That is, the random vector $\mathbf{B}$ corresponds to the values of the magnetic flux density at a fixed number of H-values. It is:
$$\mathbf{B} = (B(H_1), \ldots, B(H_{1500})).$$
It is like $\mathbf{X} = (X, Y)$ only now we have 1500 dimensions instead of 2. First, let's find the mean of this random vector:

```
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[:, :].T, 'r', lw=0.1)
ax.plot(B_mu)
ax.set_xlabel('Index j')
ax.set_ylabel('B(T)');
```
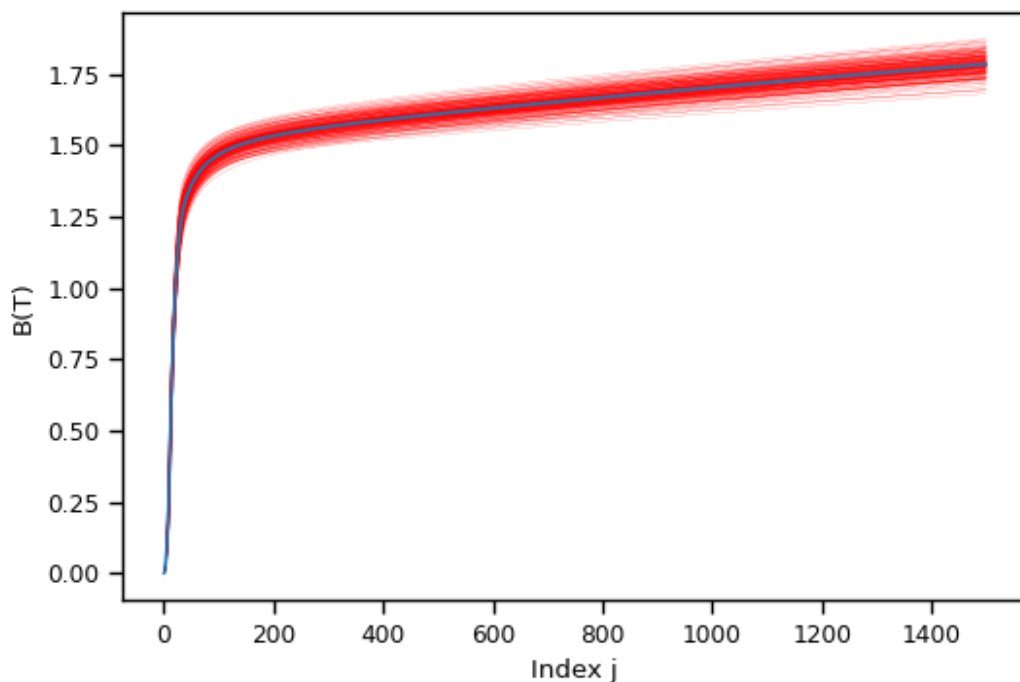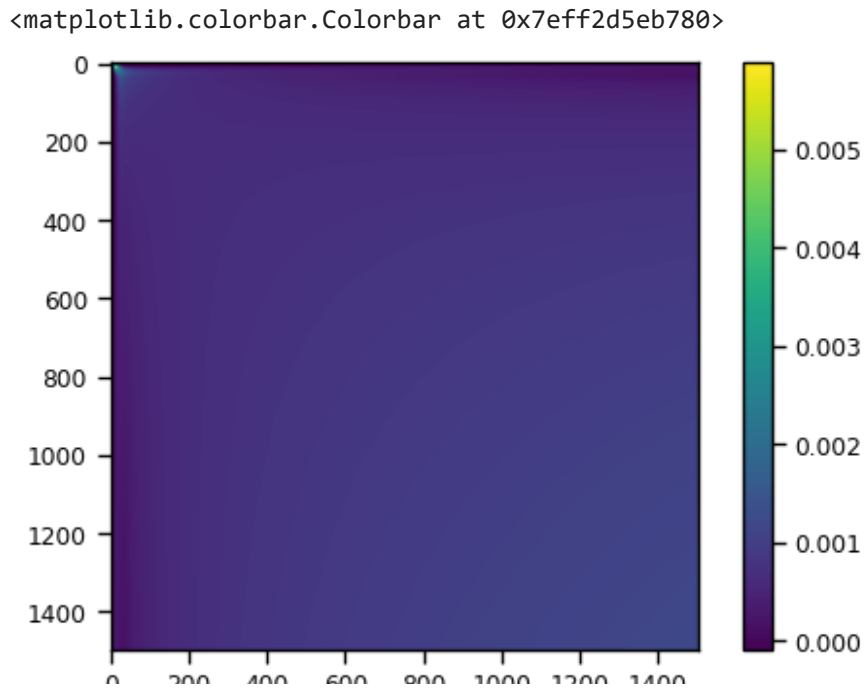

It looks good. Now, find the covariance matrix of $\mathbf{B}$. This is going to be a 1500x1500 matrix.

```
B_cov = np.cov(B_data.T)
```

Let's plot this matrix:

```
fig, ax = plt.subplots(dpi=100)
c = ax.imshow(B_cov, interpolation='nearest')
plt.colorbar(c)
```

<matplotlib.colorbar.Colorbar at 0x7eff2d5eb780>

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:

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

```
Eigenvalues of B_cov:
[-3.85082041e-16 -3.39494803e-16 -2.71195160e-16 ...  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:

```
B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

```
------------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-230-42ddfd48bf34> in <module>()
----> 1 B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

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.

```
# Pick the nugget here
nugget = 1e-9
# 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.')
print('Nugget=1e-9')
```

⊡→ It worked! Move on.
     Nugget=1e-9

H. Now you have created your first stochastic model of a complicated physical quantity. By sampling from your newly constructed random vector 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.

```
# Your code here
n=10
B_n=B.rvs(size=n)
fig, ax=plt.subplots(dpi=100)
ax.plot(B_n.T, 'r', lw=0.1)
ax.set_xlabel('Index j')
ax.set_ylabel('B(T)')
```

⊡→

Text(0, 0.5, 'B(T)')