homework_02

August 20, 2020

1 Homework 2 Solution

[1]:
```
# 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:
        local_filename = os.path.basename(url)
    with open(local_filename, 'wb') as fd:
        fd.write(data.content)
```

1.1 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:

[2]:
```
X = st.rv_discrete(values=([0, 1, 2, 3], [0.3, 0.1, 0.2, 0.4]))
print('E[X] = {0:1.2f}'.format(X.expect()))
```

```
E[X] = 1.70
```

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

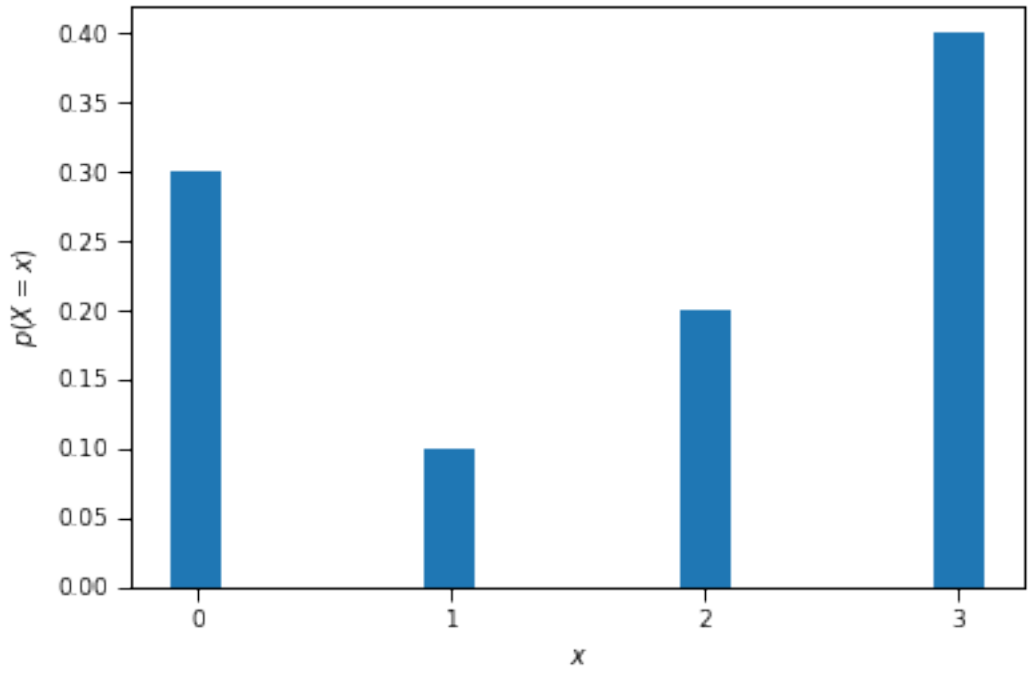
Answer:

[3]:
```
print('V[X] = {0:1.2f}'.format(X.var()))
```

```
V[X] = 1.61
```

C. Plot the probability mass function of X.

[4]:
```
fig, ax = plt.subplots()
ax.bar([0, 1, 2, 3], X.pmf([0, 1, 2, 3]), width=0.2)
ax.set_xticks([0, 1, 2, 3]);
ax.set_xlabel('$x$')
ax.set_ylabel('$p(X=x)$');
```



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

Answer:

[5]:
```
print('p(X in {{0, 2}}) = {0:1.2f}'.format(X.pmf(0) + X.pmf(2)))
```

```
p(X in {0, 2}) = 0.50
```

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

Answer:

[6]:
```
print('E[4 X + 3] = {0:1.2f}'.format(4 * X.expect() + 3))
```

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

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

Answer:

[7]:
```
print('V[4 X + 3] = {0:1.2f}'.format(4 ** 2 * X.var()))
```

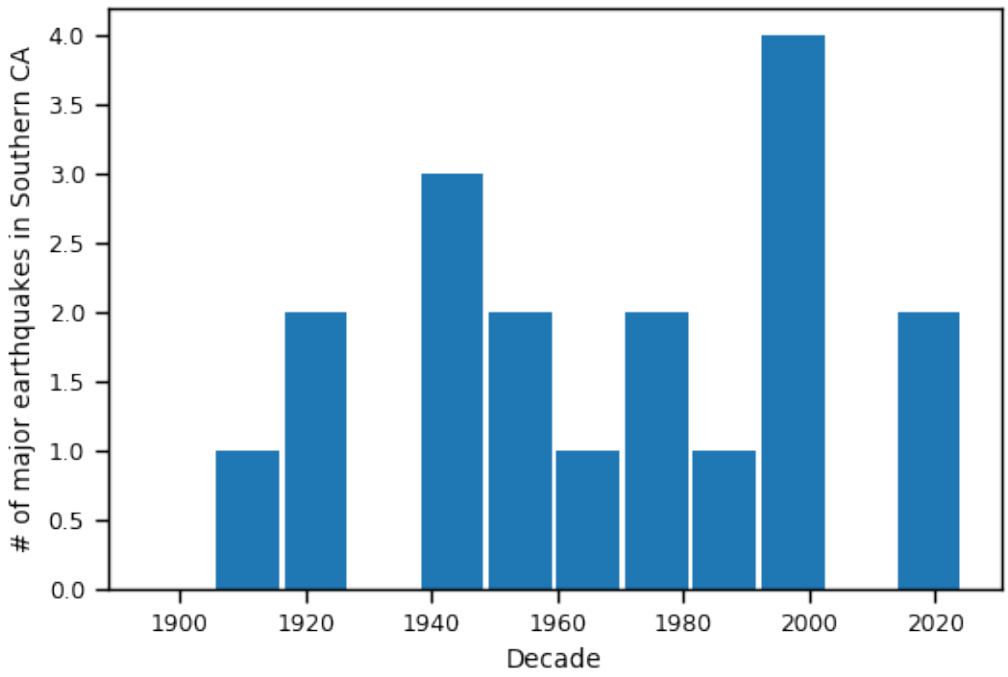
```
V[4 X + 3] = 25.76
```

1.2 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.

[8]:
```
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 $\{0, 1, 2, \dots\}$ 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:

[9]:
```
r = np.mean(eq_data)
print('r = {0:1.2f} major earthquakes per decade'.format(r))
```

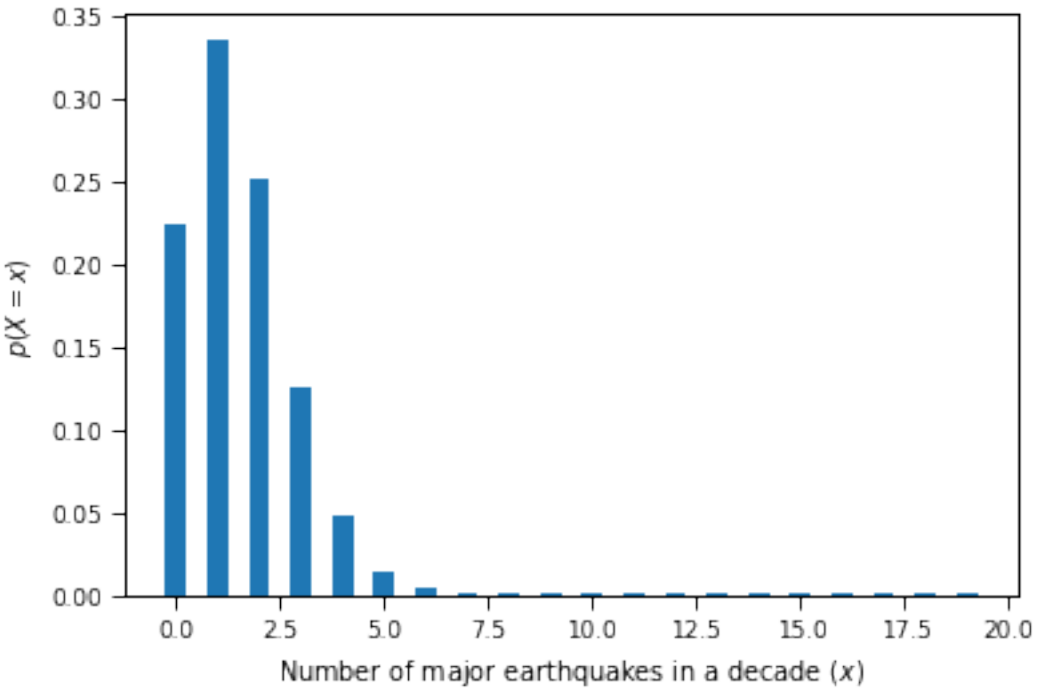
```
r = 1.50 major earthquakes per decade
```

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):

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

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

[11]:
```
fig, ax = plt.subplots()
xs = np.arange(20)
ax.bar(xs, X.pmf(xs), width=0.5)
ax.set_xlabel('Number of major earthquakes in a decade ($x$)')
ax.set_ylabel('$p(X=x)$');
```



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

The probability that at least one major earthquake will occure during the next decade is:

$$p(X \geq 1) = 1 - p(X = 0).$$

We can get this from the PMF of the Poisson:

[12]:
```
print('p(X >= 1) = {0:1.2f}'.format(1 - X.pmf(0)))
```

```
p(X >= 1) = 0.78
```

B. 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:

Let's solve this in two ways. First, following the hint. The sum of two independent Poisson is a Poisson with the rate being the sum of the two rates. In our example, this gives us:

$$X \sim \text{Poisson}(2r),$$

since $X_i \sim \text{Poisson}(r)$. Therefore the answer is:

[13]:
```
X2r = st.poisson(2 * r)
print('p(X1 + X2 >= 1) = {0:1.2f}'.format(1 - X2r.pmf(0)))
```

```
p(X1 + X2 >= 1) = 0.95
```

Let's also do it in another way that just uses simple probability rules. We are after:

$$p(X \geq 1) = p(X_1 + X_2 \geq 1) = p(X_1 \geq 1 \text{ or } X_2 \geq 1),$$

since the random variables are discrete and positive. Let's use the obvious rule on the last term:

$$p(X_1 \geq 1 \text{ or } X_2 \geq 1) = 1 - p(\text{not}(X_1 \geq 1 \text{ or } X_2 \geq 1)) = 1 - p(X_1 = 0 \text{ and } X_2 = 0).$$

Since X_1 and X_2 are independent random variables, we get:

$$p(X_1 = 0 \text{ and } X_2 = 0) = p(X_1 = 0)p(X_2 = 0),$$

and both these terms are given by the same Poisson PMF:

$$p(X_1 = 0) = p(X_2 = 0) = \text{Poisson}(0|r).$$

Putting everything together, the answer is:

$$p(X \geq 1) = 1 - (\text{Poisson}(0|r))^2 .$$

Let's calculate this to see if it matches the previous answer:

[14]:
```
print('p(X1 + X2 >= 1) = {0:1.2f}'.format(1 - X.pmf(0) ** 2))
```

```
p(X1 + X2 >= 1) = 0.95
```

C. What is the probability that at least one major earthquake will occur during the next five decades? **Answer:**

We may use any of the two methods above. Here is the easiest way to get it:

[15]:
```
print('p(X1 + X2 >= 1) = {0:1.2f}'.format(1 - X.pmf(0) ** 5))
```

```
p(X1 + X2 >= 1) = 1.00
```

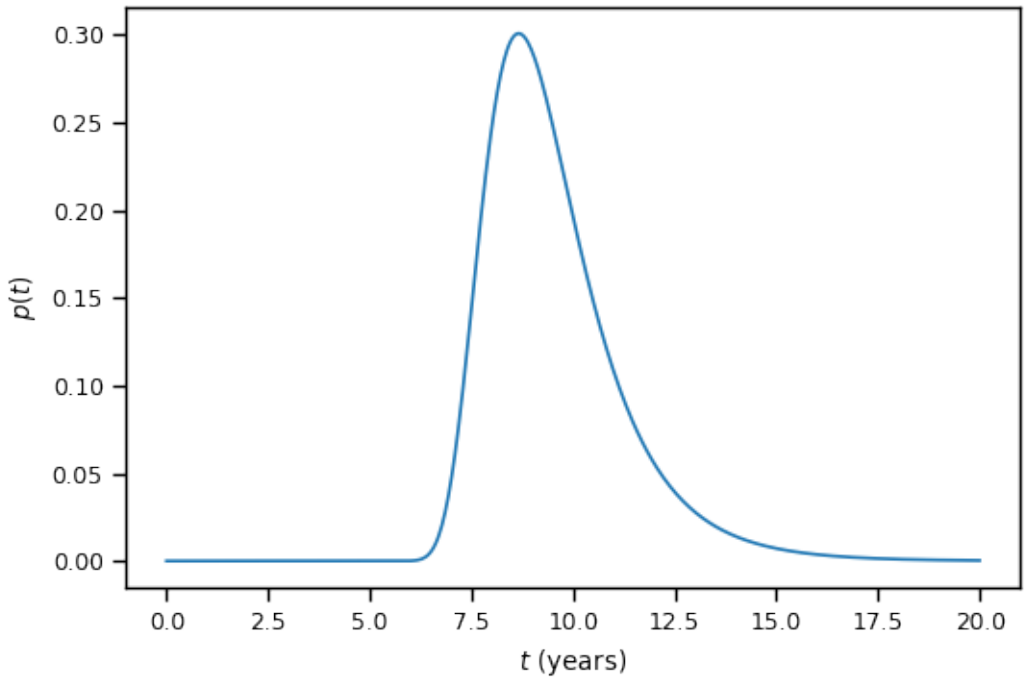
1.3 Problem 3 - Failure of a mechanical component

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.

[16]:
```
# 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, 500)
ax.plot(ts, T.pdf(ts))
ax.set_xlabel('$t$ (years)')
ax.set_ylabel('$p(t)$');
```



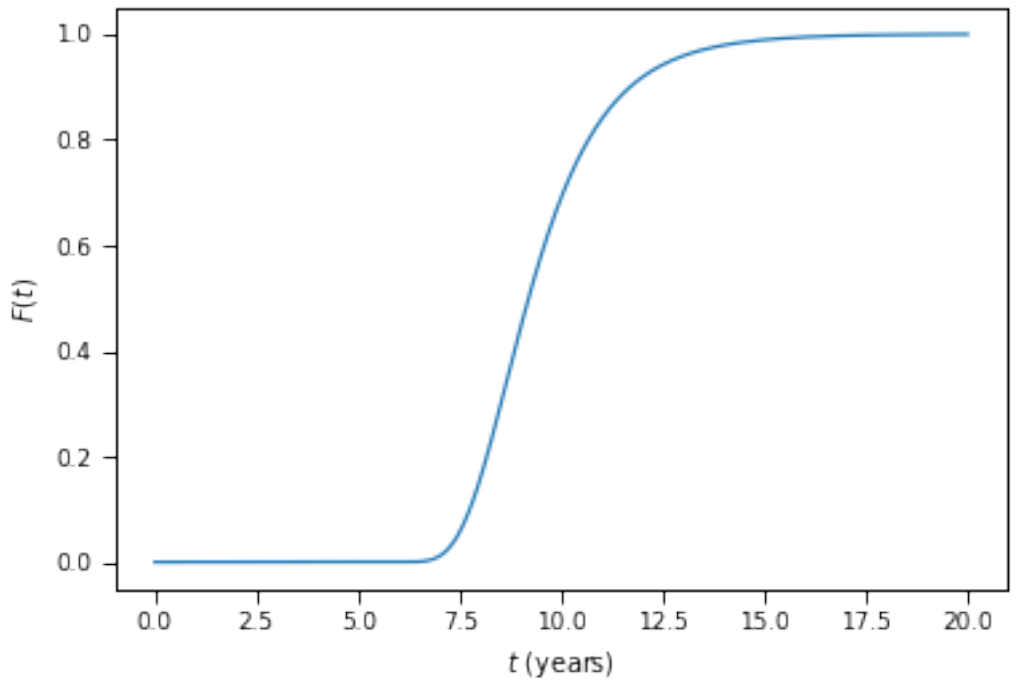
A. Find the mean fail time and its variance. Hint: Do not integrate anything by hand. Just use the functionality of `scipy.stats`.

[17]:
```
t_mean = T.expect()
t_var = T.var()
print('E[T] = {0:1.2f}'.format(t_mean))
print('V[T] = {0:1.2f}'.format(t_var))
```

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

B. Plot the cumulative distribution function of T.

[18]:
```
fig, ax = plt.subplots()
ax.plot(ts, T.cdf(ts))
ax.set_xlabel('$t$ (years)')
ax.set_ylabel('$F(t)$');
```

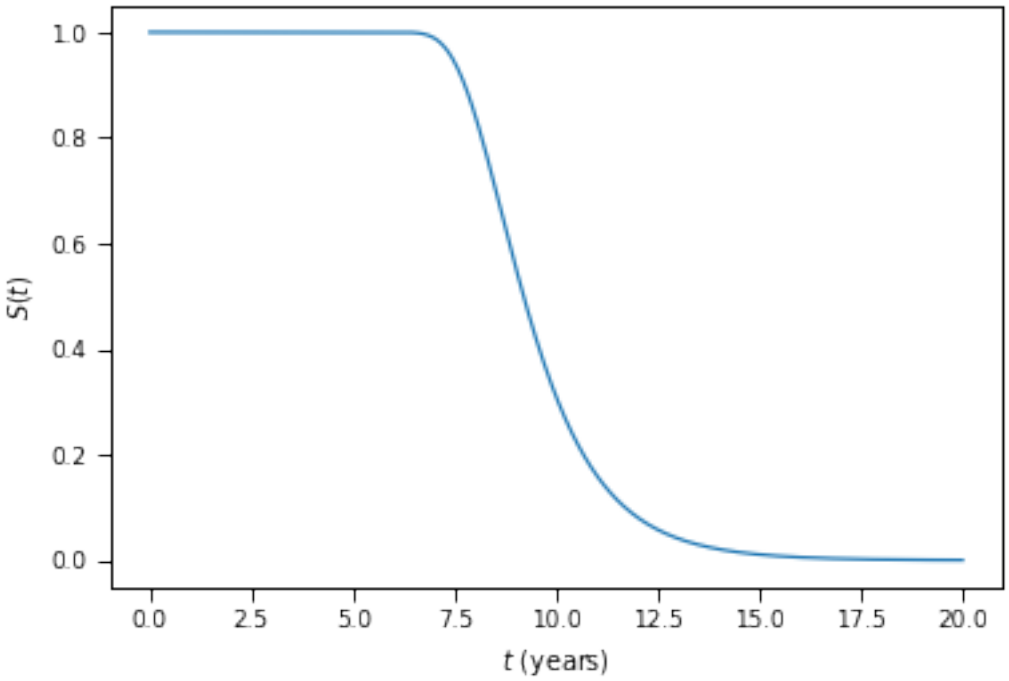


C. Plot the probability that gear survives for more than t as a function of t. That is, plot the function:

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

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

[19]:
```
fig, ax = plt.subplots()
ax.plot(ts, 1 - T.cdf(ts))
ax.set_xlabel('$t$ (years)')
ax.set_ylabel('$S(t)$');
```



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

[20]:
```
print('p(8 <= T <= 10)) = {0:1.2f}'.format(T.cdf(10) - T.cdf(8)))
```

```
p(8 <= T <= 10)) = 0.53
```

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

For the warranty, you would probably want to make sure that you don't have to send a replacement. Let's say that the warranty lasts for y years. You offer a replacement when $T \leq y$. Let's try to make the probability of offering a replacement very small, say one in a million. So, we would like to solve:

$$p(T <= y) = 1e - 3.$$

You can pose this as a root finding problem and solve for y if you wish. However, let's just use the scipy.stats functionality:

[21]:
```
# The following gives you the 99% quantile:
y = T.ppf(1e-3)
print('Offer a warranty for {0:1.1f} years'.format(y))
print('Sanity check: ')
print('F({0:1.1f}) = p(T <= {0:1.1f}) = {1:1.0e}'.format(y, T.cdf(y)))
```

```
Offer a warranty for 6.5 years
Sanity check:
F(6.5) = p(T <= 6.5) = 1e-03
```

1.4 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:

[22]:
```
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]])
```

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:

[23]:
```
print('p(X=2, Y=3) = {0:1.3f}'.format(P[2, 3]))
```

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

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

[24]:
```
print('Sum of everything = {0:1.2f}'.format(P.sum()))
```

```
Sum of everything = 1.00
```

B. Find the marginal probability density of X:

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

You can represent this as a 5-dimensional vector.

[25]:
```
p_x = P.sum(axis=1)
print('p_x = ', p_x)
```

```
p_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.

[26]:
```
p_y = P.sum(axis=0)
print('p_y = ', p_y)
```

```
p_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.

[27]:
```
# I could do it using scipy.stats.rv_discrete, but I will do it by hand
# The values X takes
xs = np.arange(5)
# The expectation of X
E_X = np.sum(xs * p_x)
# The variance of X
E_X2 = np.sum(xs ** 2 * p_x)
V_X = E_X2 - E_X ** 2
# Repeat for Y
ys = np.arange(9)
E_Y = np.sum(ys * p_y)
E_Y2 = np.sum(ys ** 2 * p_y)
V_Y = E_Y2 - E_Y ** 2
# Print everything
print('E[X] = {0:1.2f}'.format(E_X))
print('V[X] = {0:1.2f}'.format(V_X))
print('E[Y] = {0:1.2f}'.format(E_Y))
print('V[Y] = {0:1.2f}'.format(V_Y))
```

```
E[X] = 1.84
V[X] = 1.87
E[Y] = 3.69
V[Y] = 7.19
```

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

[28]:
```
print('E[X + Y] = {0:1.2f}'.format(E_X + E_Y))
```

```
E[X + Y] = 5.53
```

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

[29]:
```
# There are faster ways, but I am just going to implement
# the covariance formula using for loops
C_XY = 0.0
for i in range(xs.shape[0]):
    for j in range(ys.shape[0]):
        C_XY += P[i, j] * (xs[i] - E_X) * (ys[j] - E_Y)
print('C[X, Y] = {0:1.2f}'.format(C_XY))
```

```
C[X, Y] = 0.32
```

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

[30]:
```
# For this we are going to use the formula
print('V[X + Y] = V[X] + V[Y] + 2C[X, Y] = {0:1.2f}'.format(V_X + V_Y + 2 *␣
 ↪C_XY))
```

```
V[X + Y] = V[X] + V[Y] + 2C[X, Y] = 9.70
```

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.

[31]:
```
prob = 0.0
for i in range(xs.shape[0]):
    for j in range(ys.shape[0]):
        if xs[i] + ys[j] <= 5:
            prob += P[i, j]
print('p(X + Y <= 5) = {0:1.2f}'.format(prob))
```

```
p(X + Y <= 5) = 0.53
```

1.5 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:

[32]:
```
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:

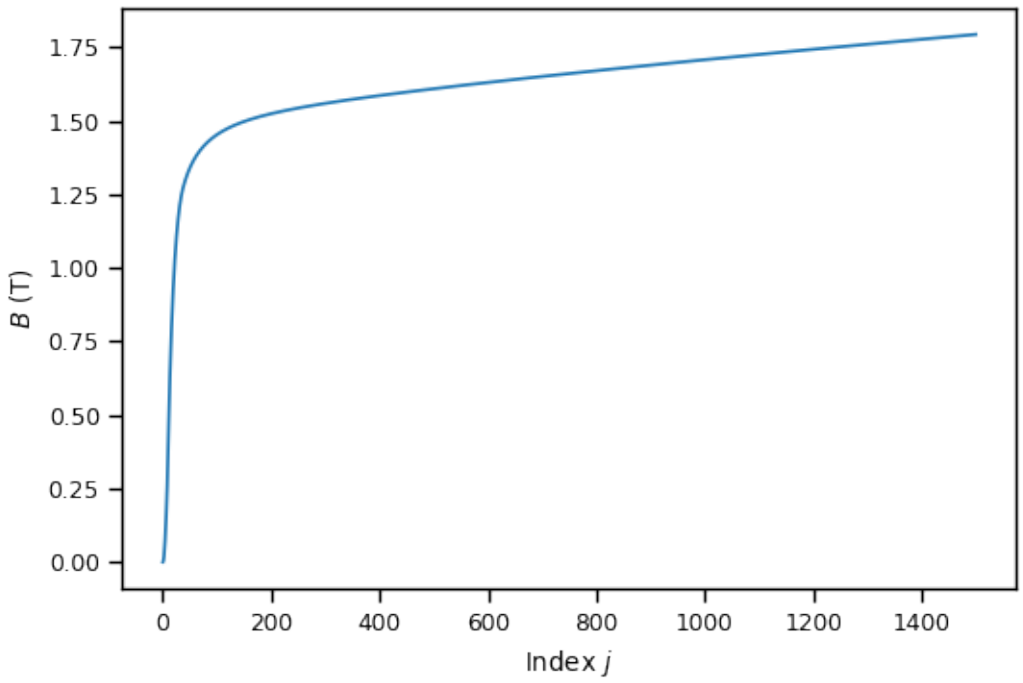
[33]:
```
# Load the data - You need the B_data.csv
B_data = np.loadtxt('B_data.csv')
# This file contains a single matrix the shape of which is:
print(B_data.shape)
```

```
(200, 1500)
```

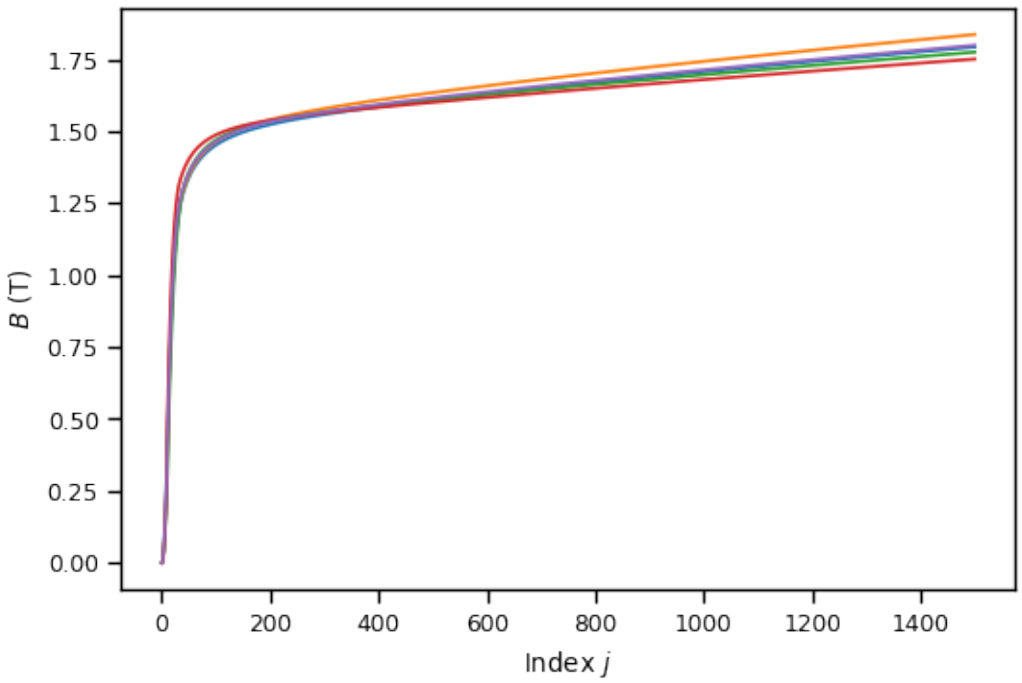
The rows (200) corresponds to different samples of the $B-H$ curves (different suppliers and different times). The columns (1500) corresponds to different values of H. That is, the i, j element is the value of B at the a specific value of H, say H_j. The values of H are the equidistant and identical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

[34]:
```
# Here is one sample
fig, ax = plt.subplots(dpi=100)
ax.plot(B_data[0, :])
ax.set_xlabel('Index $j$')
```

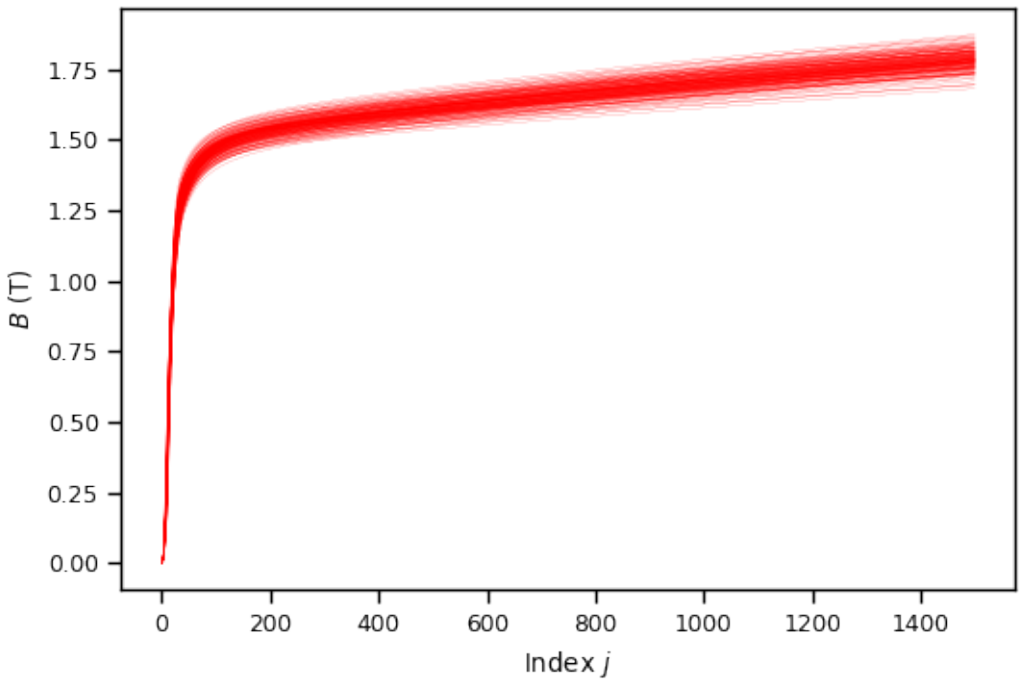
```
ax.set_ylabel('$B$ (T)');
```



[35]:
```
# 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)');
```



[36]:
```
# 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)');
```

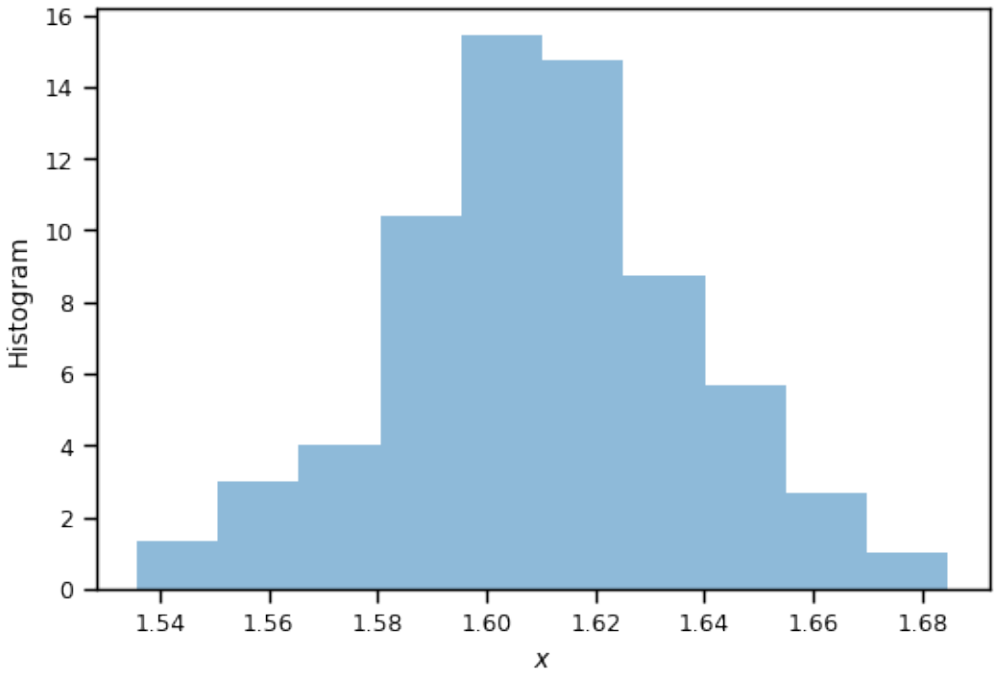


A. We are going to start by studying the data at only one index. Say index $j = 500$. Let's define a random variable

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

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

[37]:
```
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:

[38]:
```
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:

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

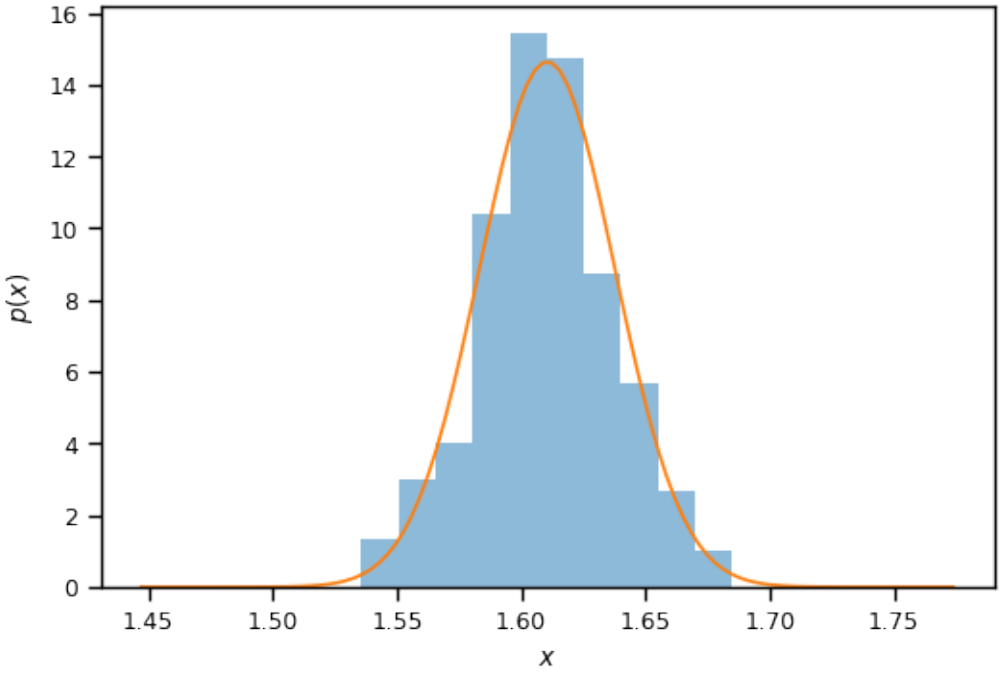
That is:

[39]:
```
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`.

[40]:
```
sigma_500 = np.sqrt(sigma2_500)
X = st.norm(loc=mu_500, scale=sigma_500)
xs = np.linspace(mu_500 - 6 * sigma_500, mu_500 + 6 * sigma_500, 100)
fig, ax = plt.subplots(dpi=100)
ax.hist(X_data, alpha=0.5, density=True, label='Histogram')
ax.plot(xs, X.pdf(xs), label='Identified Normal PDF')
ax.set_xlabel('$x$')
ax.set_ylabel('$p(x)$');
```



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

[41]:
```
print('p(X >= 1.66T) = {0:1.2f}'.format(1 - X.cdf(1.66)))
```

```
p(X >= 1.66T) = 0.03
```

C. Let us now consider another random variable

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

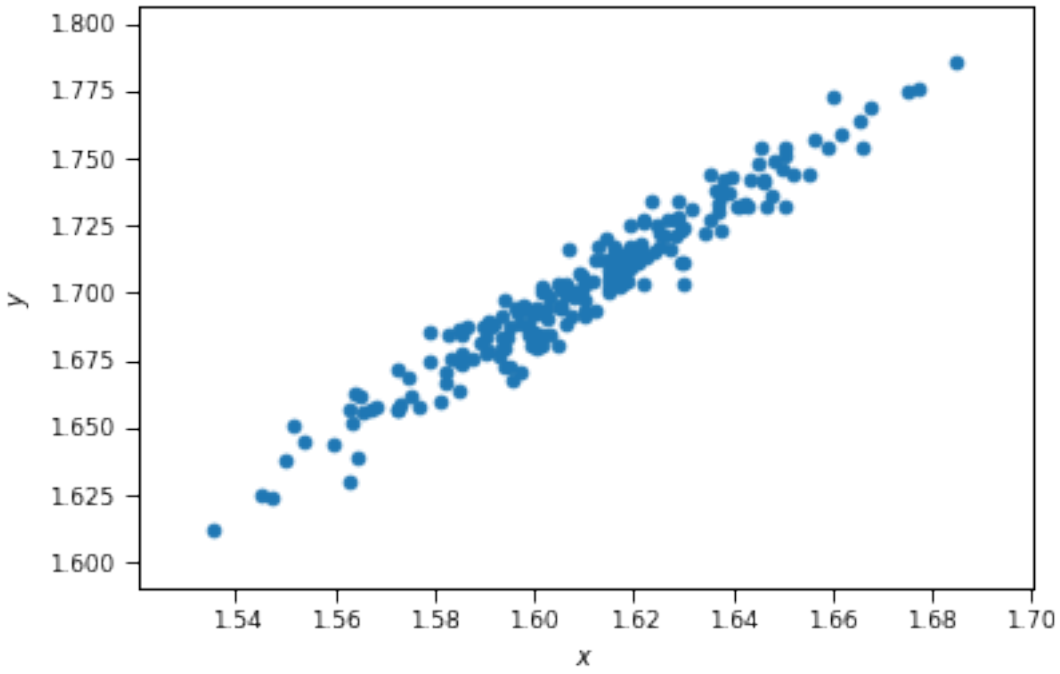
Isolate the data for this as well:

[42]:
```
Y_data = B_data[:, 1000]
```

Do the `scatter` plot of X and Y:

[43]:
```
fig, ax = plt.subplots()
ax.scatter(X_data, Y_data)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[43]:
```
Text(0, 0.5, '$y$')
```



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).

[44]:
```
XY_data = np.hstack([X_data[:, None], Y_data[:, None]])
```

The mean vector is:

[45]:
```
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:

[46]:
```
# 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:**

They are positively correlated. This is because the the off diagonal element of the covariance matrix, which gives the correlation between X and Y, is positive.

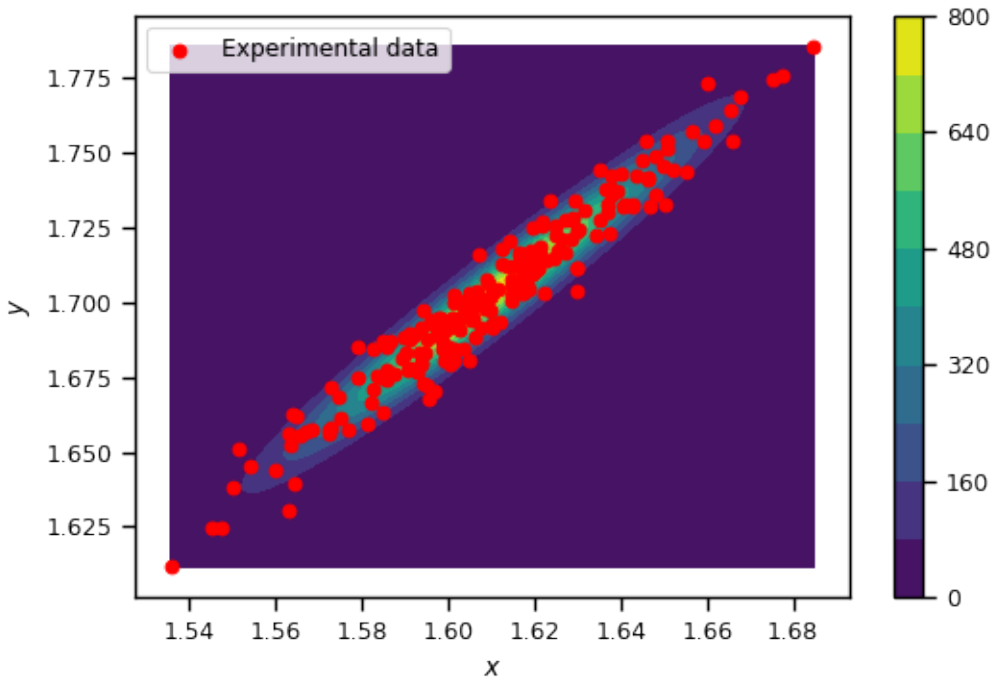
E. Use `np.linalg.eigh` to check that the matrix `C_XY` is indeed positive definite.

[47]:
```
lam, v = np.linalg.eigh(C_XY)
print('Are all the eigenvalues positive?', np.all(lam > 0))
```

```
Are all the eigenvalues positive? True
```

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.

[48]:
```
XY = st.multivariate_normal(mu_XY, cov=C_XY)
xs = np.linspace(X_data.min(), X_data.max(), 100)
ys = np.linspace(Y_data.min(), Y_data.max(), 100)
Xs, Ys = np.meshgrid(xs, ys)
XY_flat = np.hstack([Xs.flatten()[:, None], Ys.flatten()[:, None]])
Zs_flat = XY.pdf(XY_flat)
Zs = Zs_flat.reshape(Xs.shape)
fig, ax = plt.subplots(dpi=100)
c = ax.contourf(Xs, Ys, Zs, levels=10)
ax.scatter(X_data, Y_data, color='r', label='Experimental data')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.legend(loc='best')
plt.colorbar(c);
```

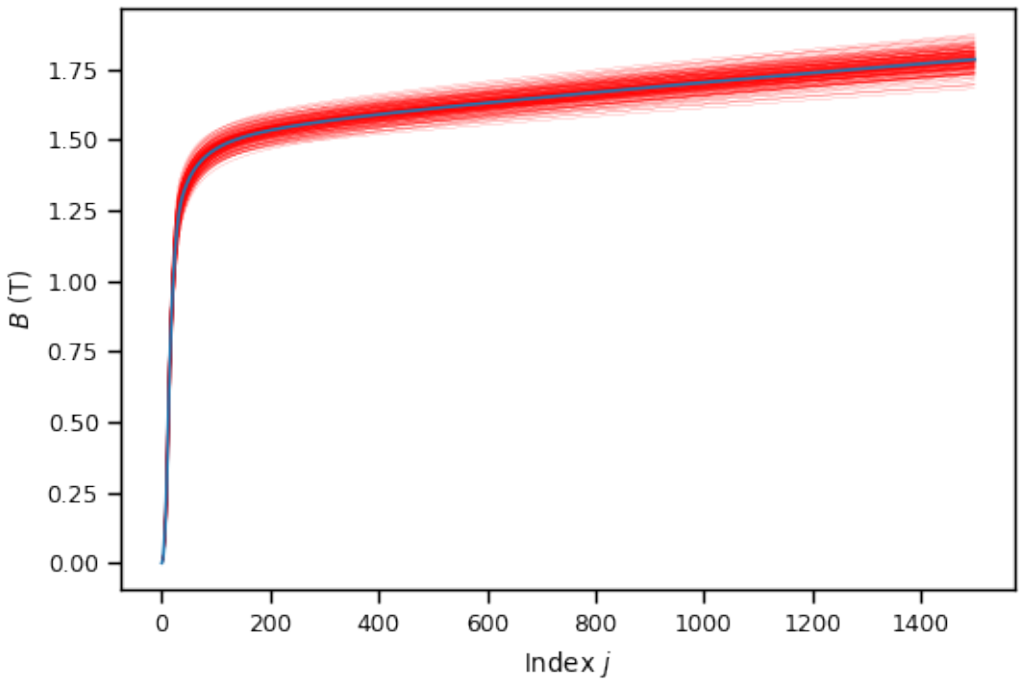


G. Now, let's think each $B-H$ curve as a random vector. That is, the random vector $\mathbf{B}$ corresponds to the values of the magnetic flux density at a fixed number of H-values. It is:

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

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:

[49]:
```
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)');
```



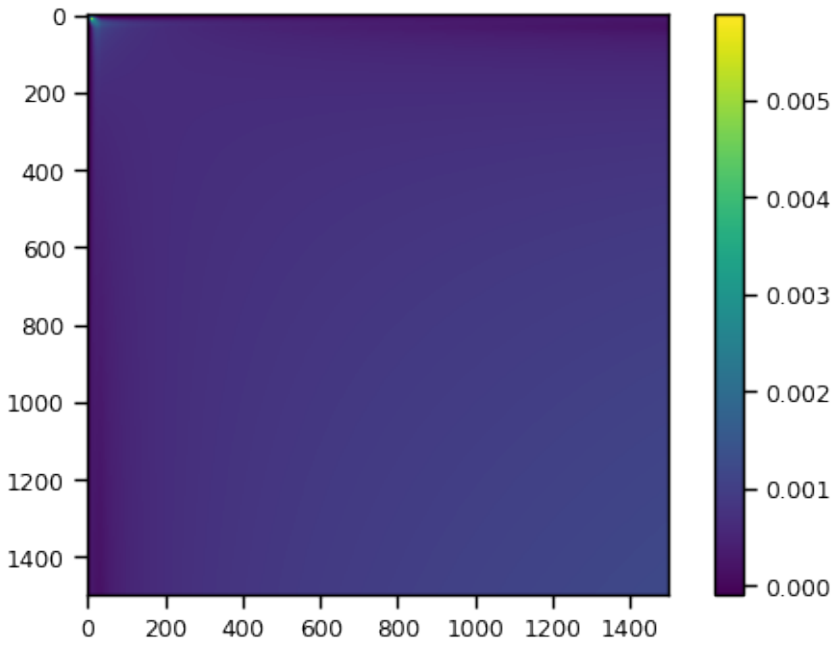
It looks good. Now, find the covariance matrix of **B**. This is going to be a 1500x1500 matrix.

[50]:
```
B_cov = np.cov(B_data.T)
```

Let's plot this matrix:

[51]:
```
fig, ax = plt.subplots(dpi=100)
c = ax.imshow(B_cov, interpolation='nearest')
plt.colorbar(c)
```

[51]:
```
<matplotlib.colorbar.Colorbar at 0x7ffa5c95aa10>
```



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:

[52]:
```
print('Eigenvalues of B_cov:')
print(np.linalg.eigh(B_cov)[0])
```

```
Eigenvalues of B_cov:
[-2.36393511e-16 -2.16484010e-16 -1.66104682e-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:

[53]:
```
B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

```
␣
↪---------------------------------------------------------------------------

LinAlgError                               Traceback (most recent call␣
↪last)

<ipython-input-53-42ddfd48bf34> in <module>
----> 1 B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

```
       ~/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_multivariate.py␣
↪in __call__(self, mean, cov, allow_singular, seed)
       361         return multivariate_normal_frozen(mean, cov,
       362                                           ␣
↪allow_singular=allow_singular,
   --> 363                                           seed=seed)
       364
       365     def _process_parameters(self, dim, mean, cov):


       ~/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_multivariate.py␣
↪in __init__(self, mean, cov, allow_singular, seed, maxpts, abseps, releps)
       734         self.dim, self.mean, self.cov = self._dist.
↪_process_parameters(
       735                                                         None,␣
↪mean, cov)
   --> 736         self.cov_info = _PSD(self.cov, allow_singular=allow_singular)
       737         if not maxpts:
       738             maxpts = 1000000 * self.dim


       ~/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_multivariate.py␣
↪in __init__(self, M, cond, rcond, lower, check_finite, allow_singular)
       161         d = s[s > eps]
       162         if len(d) < len(s) and not allow_singular:
   --> 163             raise np.linalg.LinAlgError('singular matrix')
       164         s_pinv = _pinv_1d(s, eps)
       165         U = np.multiply(u, np.sqrt(s_pinv))


       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.

[54]:
```
# 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.')
```

```
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.

[55]:
```
B_samples = B.rvs(size=10)
fig, ax = plt.subplots(dpi=100)
ax.plot(B_samples.T, 'r', lw=0.1)
ax.set_xlabel('Index $j$')
ax.set_ylabel('$B$ (T)');
```