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 should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

In [1]:

```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, "savefig.dpi":300})
sns.set_context("notebook")
sns.set_style("ticks")

import scipy
import scipy.stats as st
import urllib.request
import os

def download(
    url : str,
    local_filename : str = None
):
    """Download a file from a url.

    Arguments
    url            -- The url we want to download.
    local_filename -- The filemame to write on. If not
                      specified
    """
    if local_filename is None:
        local_filename = os.path.basename(url)
    urllib.request.urlretrieve(url, local_filename)
```

Student details

- **First Name: Ben**

- **Last Name: McAteer**
- **Email: bmcateer@purdue.edu**

Problem 1 - 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?

Here are some hypothetical data to work with. Suppose that we took ten gears and we worked them until failure. The failure times (say in years) are as follows:

In [2]:
```
time_to_fail_data = np.array(
    [
        10.5,
        7.5,
        8.1,
        8.4,
        11.2,
        9.3,
        8.9,
        12.4
    ]
)
```

Why does each gear fail at different times? There are several sources of uncertainty. The most important are:

- Manufacturing imperfections.
- Different loading conditions.

If this was a controlled fatigue experiment, then we could eliminate the second source of uncertainty by using exactly the same loading conditions.

Now, we are going to fit a probability density function to these data. Which one should we use? Well, new gears do not fail easily. So, the probability density function of T should be close to zero for small T. 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.

The Weibull has parameters and we are going to fit them to the available data. The method we are going to use is called the *maximum likelihood method*. We haven't really talked about this, and it is not important to know what it is to do this homework problem. We will learn about maximum likelihood in later lectures. Here is how we fit the parameters using `scipy.stats`:

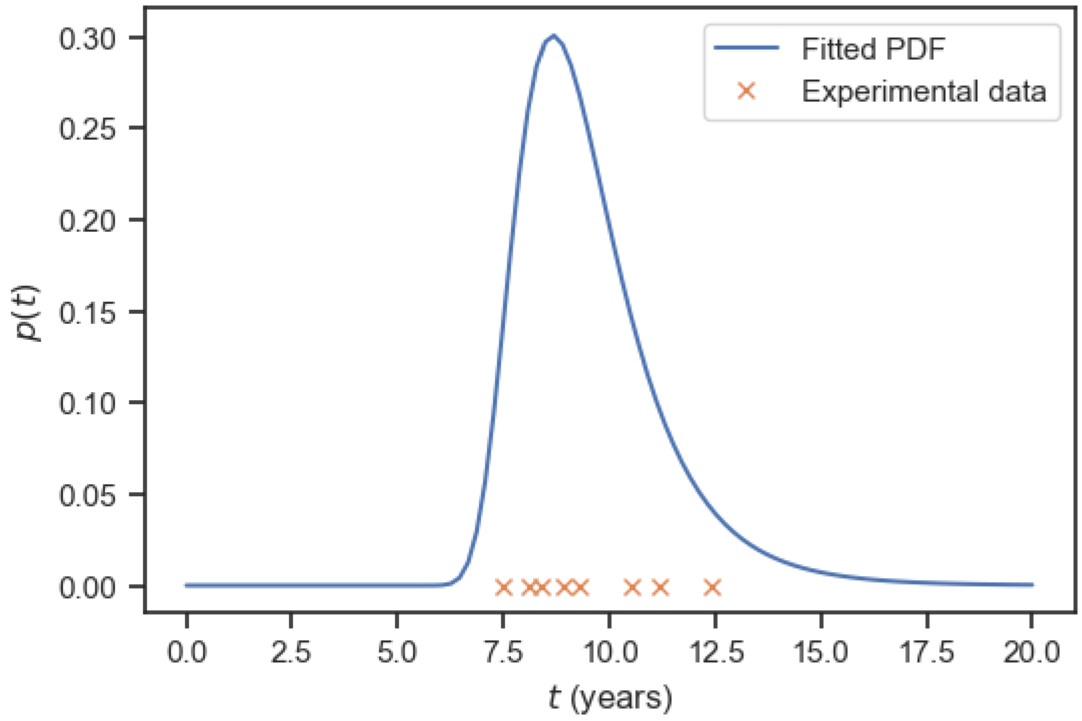
In [3]:
```
fitted_params = st.exponweib.fit(time_to_fail_data, loc=0)
```

```
T = st.exponweib(*fitted_params)
print(f"Fitted parameters: {fitted_params}")
```

```
Fitted parameters: (448.066965711728, 0.7099665338918923, 3.4218808260575804, 0.41627831
297126994)
```

Let's plot the fitted Weibul PDF and the data we used:

In [4]:
```
fig, ax = plt.subplots()
ts = np.linspace(0.0, 20.0, 100)
ax.plot(
    ts,
    T.pdf(ts),
    label="Fitted PDF"
)
ax.plot(
    time_to_fail_data,
    np.zeros_like(time_to_fail_data),
    "x",
    label="Experimental data"
)
ax.set_xlabel(r"$t$ (years)")
ax.set_ylabel(r"$p(t)$")
plt.legend(loc="best");
```



Now you have to answer a series of questions about the random variable T that we just fitted.

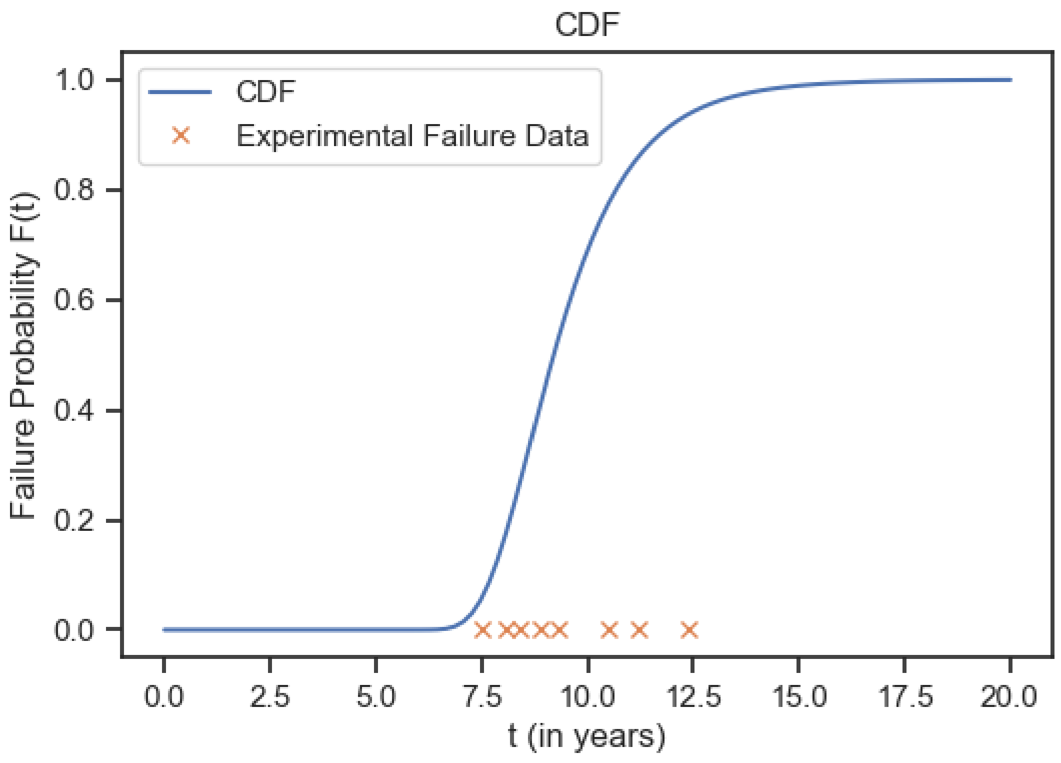
A. Find the mean fail time and its variance. Hint: Do not integrate anything by hand. Just use the functionality of `scipy.stats` .

In [5]:
```
t_mean = T.mean()  #scipy.stats.tmean(time_to_fail_data, axis = 0)
t_var = T.var()  #scipy.stats.tvar(time_to_fail_data, limits = None)
print(f"E[T] = {t_mean:.2f}")
print(f"V[T] = {t_var:.2f}")
```

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

B. Plot the cumulative distribution function $F(t) = P(T \leq t)$ of T.

In [6]:
```
fig, ax = plt.subplots()
ts = np.linspace(0.0, 20.0, 100)
ax.plot(
    ts,
    T.cdf(ts),
    label="CDF"
)
ax.plot(
    time_to_fail_data,
    np.zeros_like(time_to_fail_data),
    "x",
    label="Experimental Failure Data"
)
ax.set_xlabel(r"t (in years)")
ax.set_title("CDF")
ax.set_ylabel(r"Failure Probability F(t)")
plt.legend(loc="best");
```

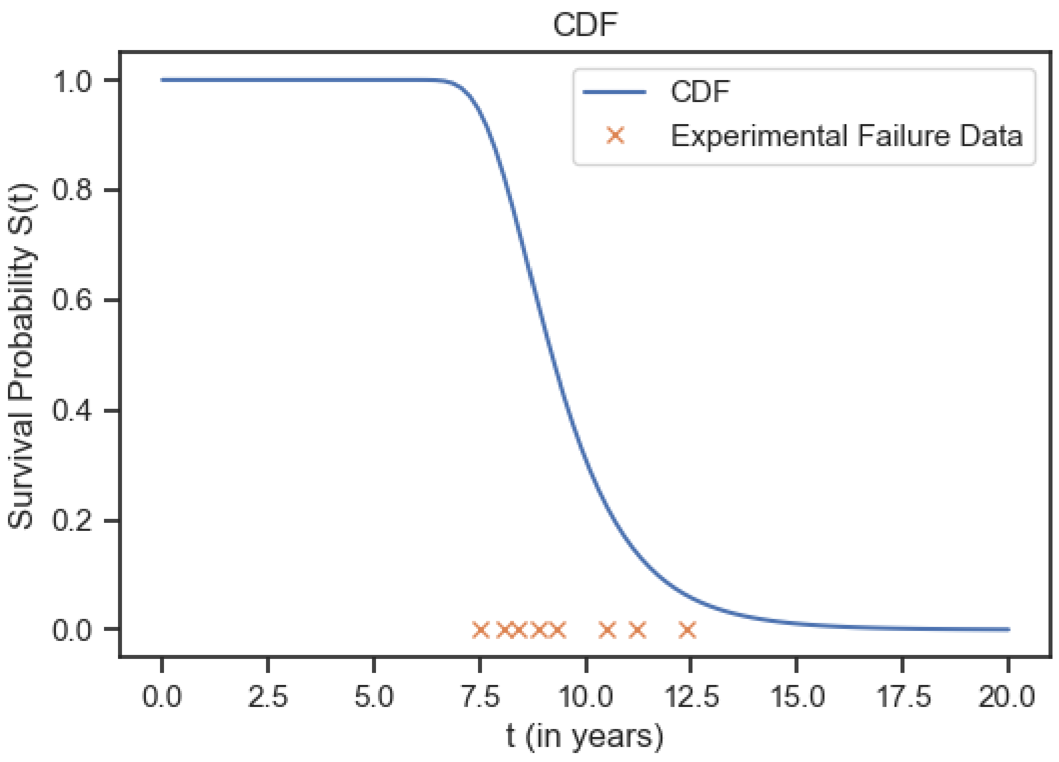


C. Plot the probability that gear survives for more than t as a function of t. That is, plot the function:

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

Hint: First connect $S(t)$ to the cumulative distribution function $F(t)$ of T.

In [7]:
```
#1C 1-prob of success
#1C:
fig, ax = plt.subplots()
ts = np.linspace(0.0, 20.0, 100)
ax.plot(
    ts,
    1 - T.cdf(ts),
    label="CDF"
)
ax.plot(
    time_to_fail_data,
    np.zeros_like(time_to_fail_data),
    "x",
    label="Experimental Failure Data"
)
ax.set_xlabel(r"t (in years)")
ax.set_title("CDF")
ax.set_ylabel(r"Survival Probability S(t)")
plt.legend(loc="best");
```



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

In [8]:
```
#1D:
prob8to10 =  T.cdf(10)-T.cdf(8)
print('The probability that the gear lasts anywhere between 8 and 10 years is {0:1.4f}'
```

```
The probability that the gear lasts anywhere between 8 and 10 years is 0.5343
```

E. If you were to sell the gear, how many years "warranty" would you offer?
Hint: This is subjective. There are many correct answers. But as a manufacturer of the gear, you really do not want to be replacing any... **Answer:** If I were to sell this gear, I would offer a 7 year warranty. This would provide a replacement to 1.11% of all gears. It would be reasonable to assume that replacing 1% of gears is in good faith to the customers without being too nice that the business cannot succeed. This would essentially only cover the early failures.

In [9]:
```
#1E:
prob7year =  T.cdf(7)
print('The probability that the gear fails at or before 7 years is {0:1.4f}'.format(pro
```

```
The probability that the gear fails at or before 7 years is 0.0111
```

Problem 2 - 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 [10]:
```
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:

In [11]:
```
print(f"p(X=2, Y=3) = {P[2, 3]:.3f}")
```

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

A. Verify that all the elements of P sum to one, i.e., that $\sum_{x,y}p(X=x,Y=y)=1$.

In [12]:
```
#2A
allP = np.sum(P)
print(f"sum of P = {allP:.2f}")
```

```
sum of P = 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.

In [13]:
```
# 2B: Hint, you can do this in one line if you read this:
#help(np.sum)
marginprobdenseX = np.sum(P, axis = 1)
print('The marginal probability density of $X$ is:',marginprobdenseX)
```

```
The marginal probability density of $X$ is: [0.204 0.258 0.198 0.178 0.162]
```

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

In [14]:
```
#2C
marginprobdenseY = np.sum(P, axis = 0)
print('The marginal probability density of $X$ is:',marginprobdenseY)
```

```
The marginal probability density of $X$ is: [0.149 0.15  0.091 0.096 0.092 0.151 0.059 0.
097 0.114]
```

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

In [15]:
```
#2D:
P_x = np.sum(P, axis=1)
P_y = np.sum(P, axis=0)

Xexpect = np.sum(np.arange(5) * P_x)
Xexpectsq = np.sum(np.arange(5) ** 2 * P_x)
Xvariance = Xexpectsq - Xexpect **2
Yexpect = np.sum(np.arange(9) * P_y)
Yexpectsq = np.sum(np.arange(9) ** 2 * P_y)
Yvariance = Yexpectsq - Yexpect **2
print('The expectation of X is {0:1.2f}'.format(Xexpect))
print('The variance of X is {0:1.2f}'.format(Xvariance))
print('The expectation of Y is {0:1.2f}'.format(Yexpect))
print('The variance of Y is {0:1.2f}'.format(Yvariance))
```

```
The expectation of X is 1.84
The variance of X is 1.87
The expectation of Y is 3.69
The variance of Y is 7.19
```

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

In [16]:
```
#2E
XandYexpect = Xexpect + Yexpect
print('The expectation of X and Y is {0:1.2f}'.format(XandYexpect))
```

```
The expectation of X and Y is 5.53
```

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

In [17]:
```
#2F (utilizing "Practicing with joint probability mass functions" document for help)
Cov_XY = 0.0
for x in range(5):
    for y in range(9):
        Cov_XY += (x - Xexpect) * (y - Yexpect) * P[x, y] # the += means add to the lef
print(f"Covariance of [X, Y] = {Cov_XY:.2f}")
```

```
Covariance of [X, Y] = 0.32
```

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

In [18]:
```
#2G
XandYvar = Xvariance + Yvariance + 2 * Cov_XY
print('The variance of X and Y is {0:1.2f}'.format(XandYvar))
```

```
The variance of X and Y is 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.

In [19]:
```
#2J
combvalue = 0  #sum for if X+Y > 5
for x in range(5):
    for y in range(9):
        if x + y <= 5:
            combvalue = combvalue + (P[x,y])

print('The probability that  X+Y  is less than or equal to 5 is {0:1.2f}'.format(combva
```

```
The probability that  X+Y  is less than or equal to 5 is 0.53
```

Problem 3 - 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. I have written a bit of code that allows you to put the data file in the right place without too much trouble. Simply run the following:

In [20]:
```
url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/lecturebook
download(url)
```

If everything worked well, then the following will work:

In [21]:
```
B_data = np.loadtxt('B_data.csv')
B_data
```

Out[21]:
```
array([[0.   , 0.005, 0.019, ..., 1.793, 1.793, 1.794],
       [0.   , 0.004, 0.014, ..., 1.837, 1.837, 1.837],
       [0.   , 0.004, 0.014, ..., 1.776, 1.776, 1.776],
       ...,
       [0.   , 0.003, 0.012, ..., 1.767, 1.767, 1.767],
       [0.   , 0.008, 0.031, ..., 1.777, 1.778, 1.778],
       [0.   , 0.003, 0.014, ..., 1.765, 1.765, 1.765]])
```

The shape of this dataset is:

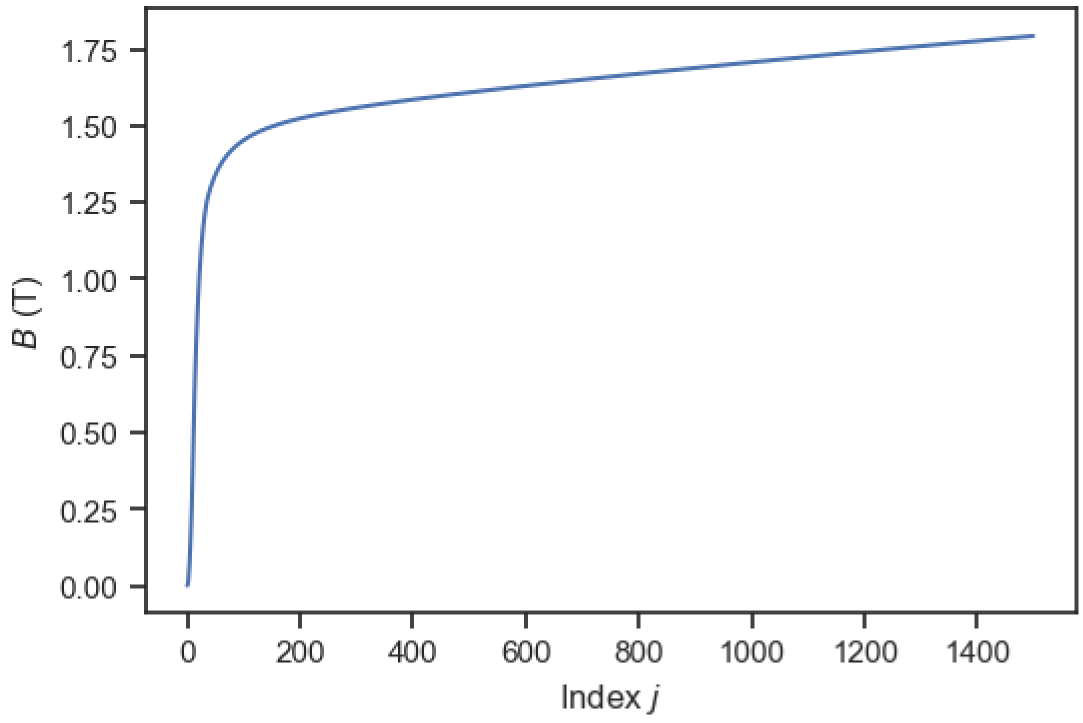
In [22]:
```
B_data.shape
```

Out[22]:
```
(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.

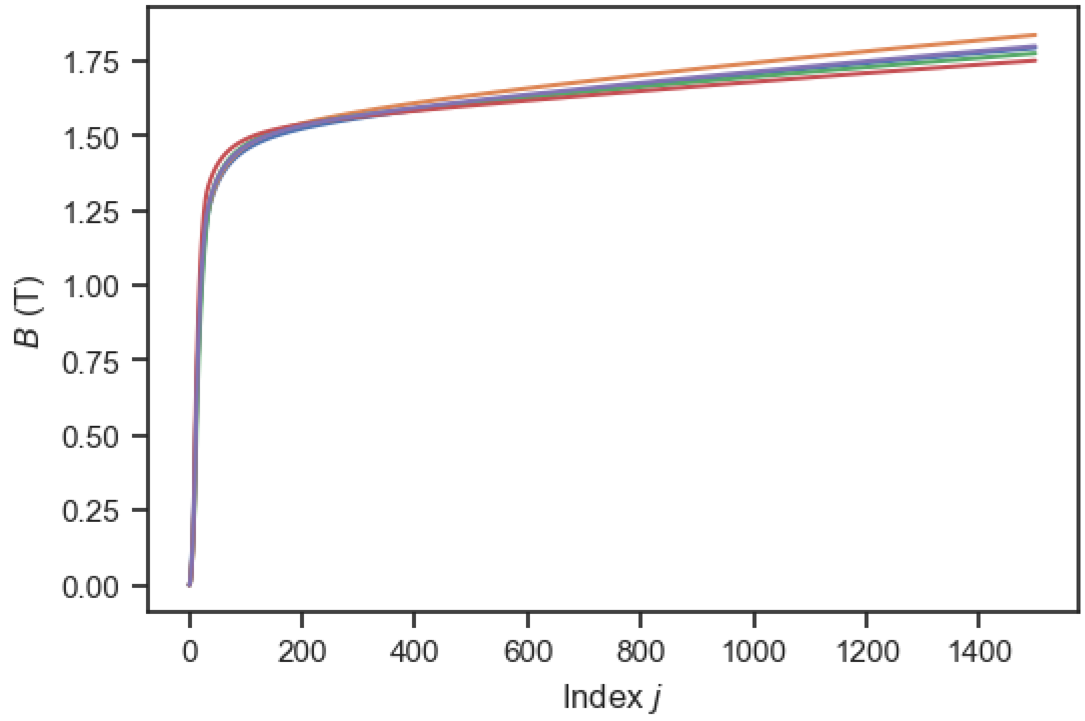
Here is one sample:

In [23]:
```
fig, ax = plt.subplots()
ax.plot(B_data[0, :])
ax.set_xlabel(r"Index $j$")
ax.set_ylabel(r"$B$ (T)");
```



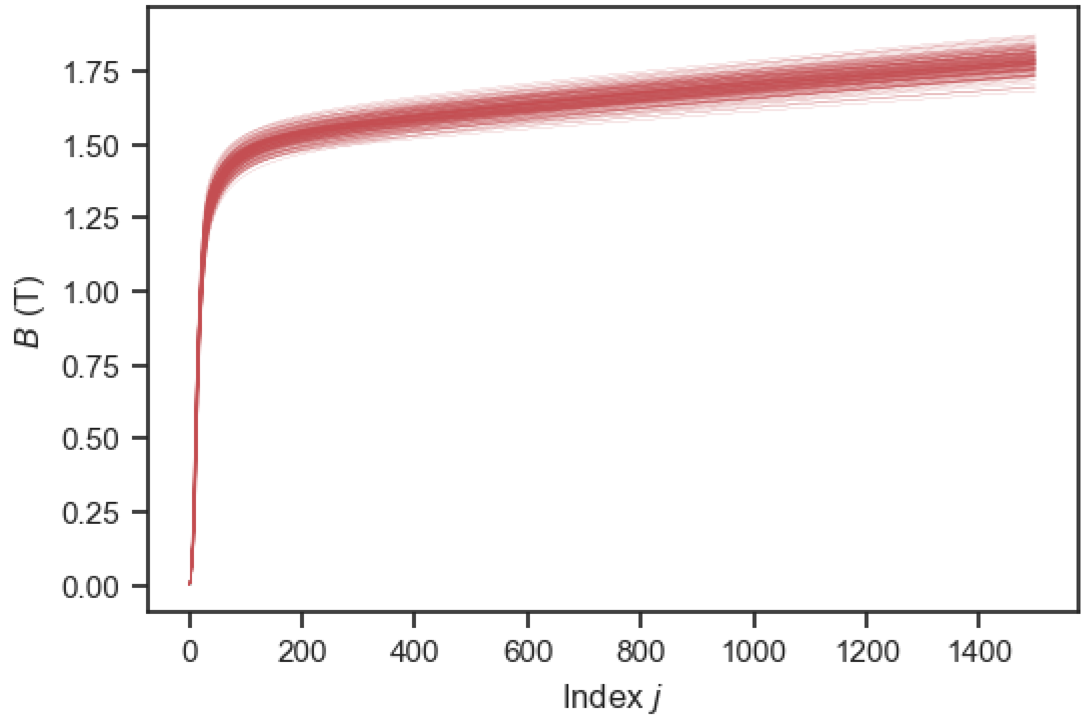
Here are five samples:

In [24]:
```
fig, ax = plt.subplots()
ax.plot(B_data[:5, :].T)
ax.set_xlabel(r"Index $j$")
ax.set_ylabel(r"$B$ (T)");
```



Here are all the samples:

```
In [25]:
fig, ax = plt.subplots()
ax.plot(B_data[:, :].T, 'r', lw=0.1)
ax.set_xlabel(r"Index $j$")
ax.set_ylabel(r"$B$ (T)");
```

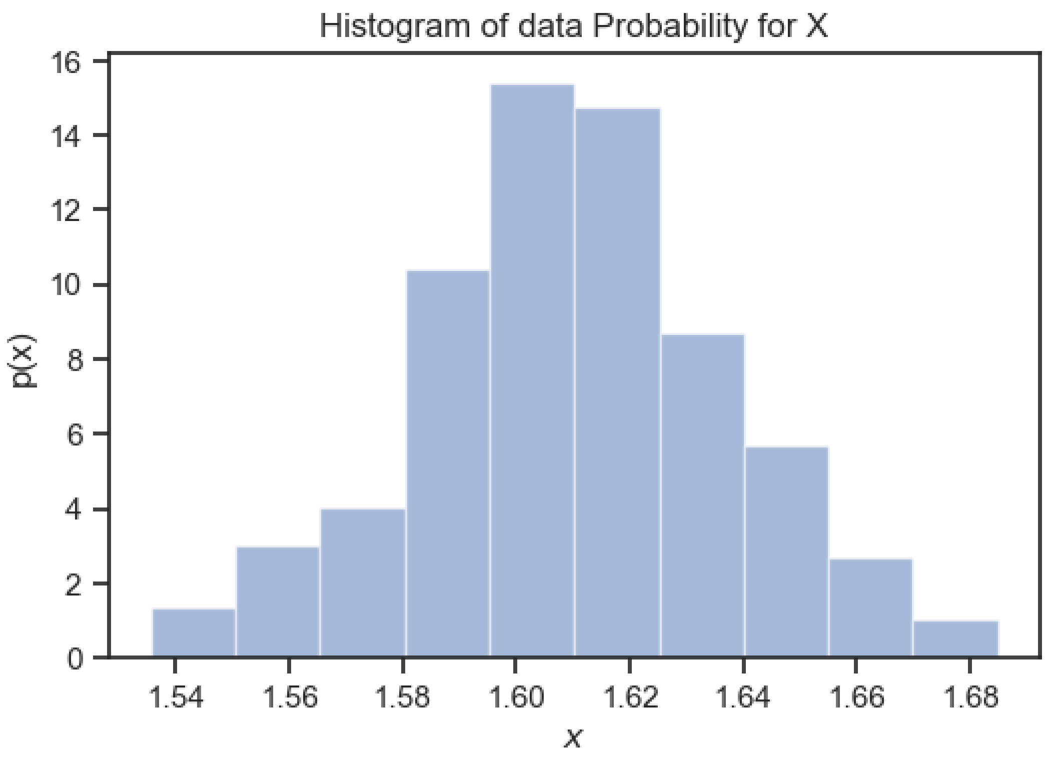


A. We are going to start by studying the data at only one index. Say index $j = 500$. Let's define a random variable

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

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

In [54]:
```
X_data = B_data[:, 500]
fig, ax = plt.subplots()
ax.hist(X_data, alpha=0.5, density=True)
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"p(x)");
ax.set_title('Histogram of data Probability for X');
```



This looks like a Gaussian $N(\mu_{500}, \sigma^2_{500})$. Let's try to find a mean and variance for that Gaussian. A good choice for the mean is the empirical average of the data:

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

Later we will learn that this is what the *maximum likelihood method* gives us.

So, the mean is:

In [27]:
```
mu_500 = X_data.mean()
print(f"mu_500 = {mu_500:.2f}")
```

```
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 [28]:
```
sigma2_500 = np.var(X_data)
print(f"sigma_500 = {sigma2_500:.2e}")
```

```
sigma_500 = 7.42e-04
```

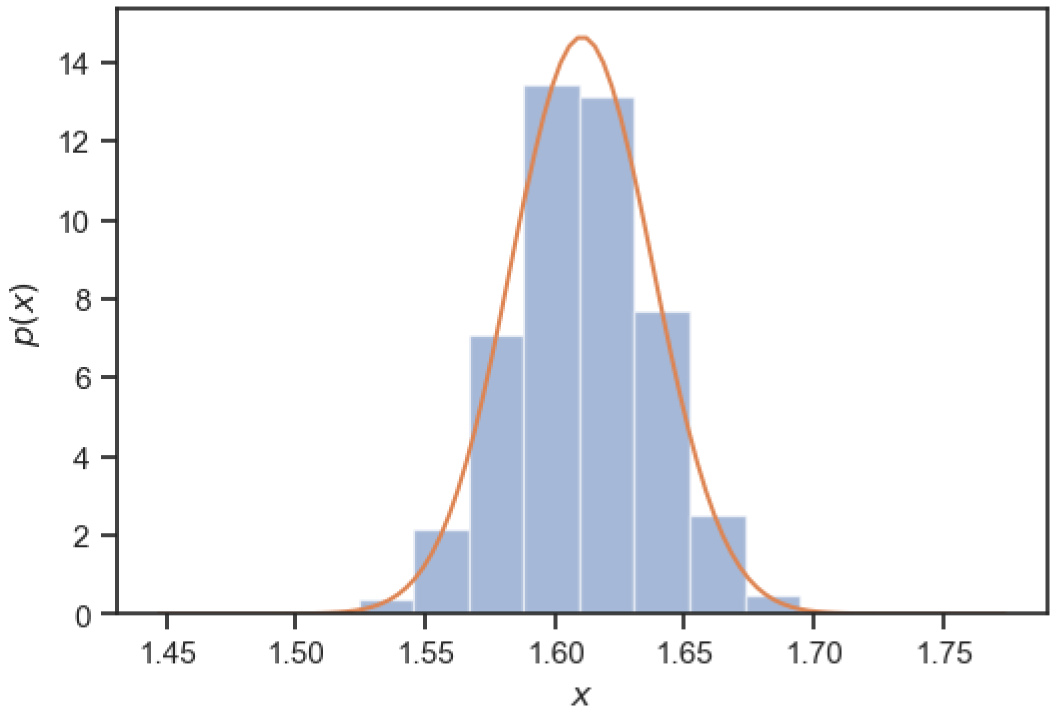
Repeat the plot of the histogram of X along with the PDF of the normal variable we have just identified using the functionality of `scipy.stats`.

In [29]:
```
#3A Using help from "The Guassian Distribution" notes

Z = st.norm()
sigma_500 = np.sqrt(sigma2_500)

X = st.norm(mu_500, sigma_500)
xs = np.linspace(mu_500 - 6.0 * sigma_500, mu_500 + 6.0 * sigma_500, 100)
x_samples = mu_500 + sigma_500 * Z.rvs(size=10000)

fig, ax = plt.subplots()
ax.hist(x_samples, density=True, alpha=0.5)
ax.plot(xs, X.pdf(xs))
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.

In [30]:
```
#3B
prob166 = X.sf(1.66)
print('The probability that X = B(H500) is greater than 1.66 is {0:1.3f}'.format(prob16
```

```
The probability that X = B(H500) is greater than 1.66 is 0.034
```

C. Let us now consider another random variable

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

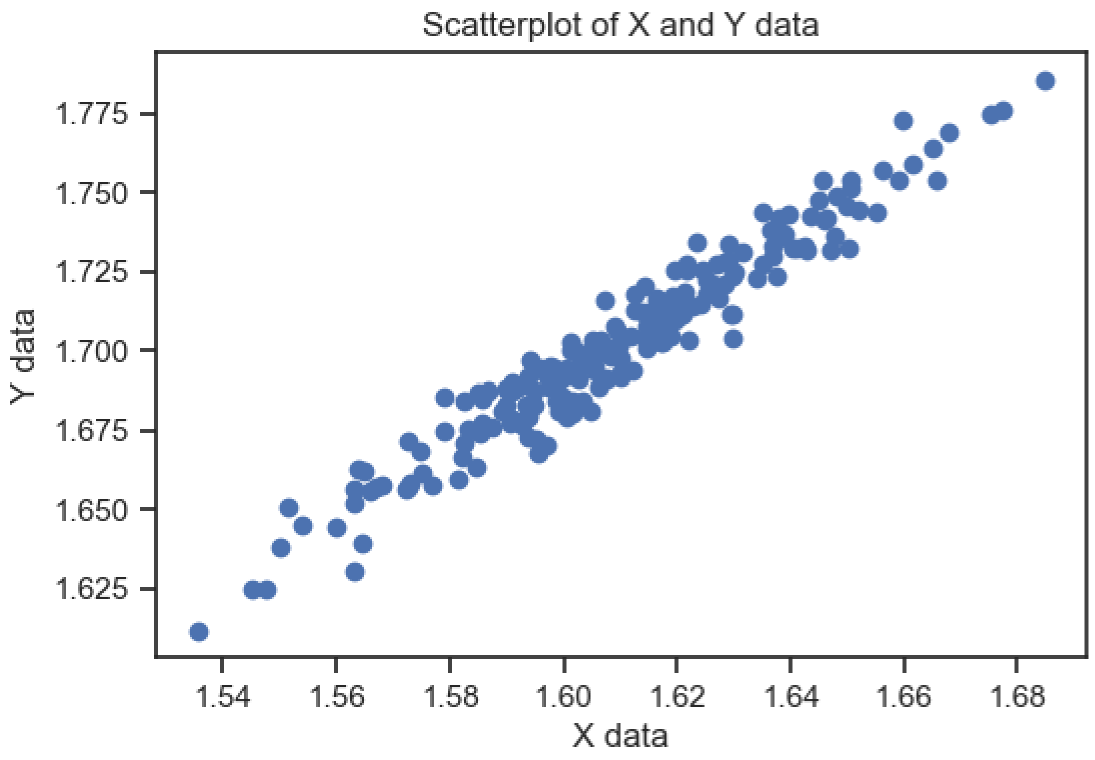
Isolate the data for this as well:

In [31]:
```
Y_data = B_data[:, 1000]
```

Do the `scatter` plot of X and Y:

In [32]:
```
plt.figure()
plt.scatter(X_data,Y_data)
plt.xlabel('X data')
plt.ylabel('Y data')
plt.title('Scatterplot of X and Y data')
```

Out[32]:
```
Text(0.5, 1.0, 'Scatterplot of X and Y data')
```



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 [33]:
```
XY_data = np.hstack([X_data[:, None], Y_data[:, None]]) # I NEVER KNEW THIS WAS THIS SI
```

The mean vector is:

In [55]:
```
mu_XY = np.mean(XY_data, axis=0)
print(f"The mean vector is: {mu_XY}")
```

```
The mean vector is: [1.61  1.703]
```

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 [35]:
```
# Careful with np.cov because it requires you to transpose the matrix
C_XY = np.cov(XY_data.T)
print(f"C_XY =")
print(C_XY)
```

```
C_XY =
[[0.001 0.001]
 [0.001 0.001]]
```

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

Answer: Because the covariance of X & Y is greater than 1, they are positively correlated

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

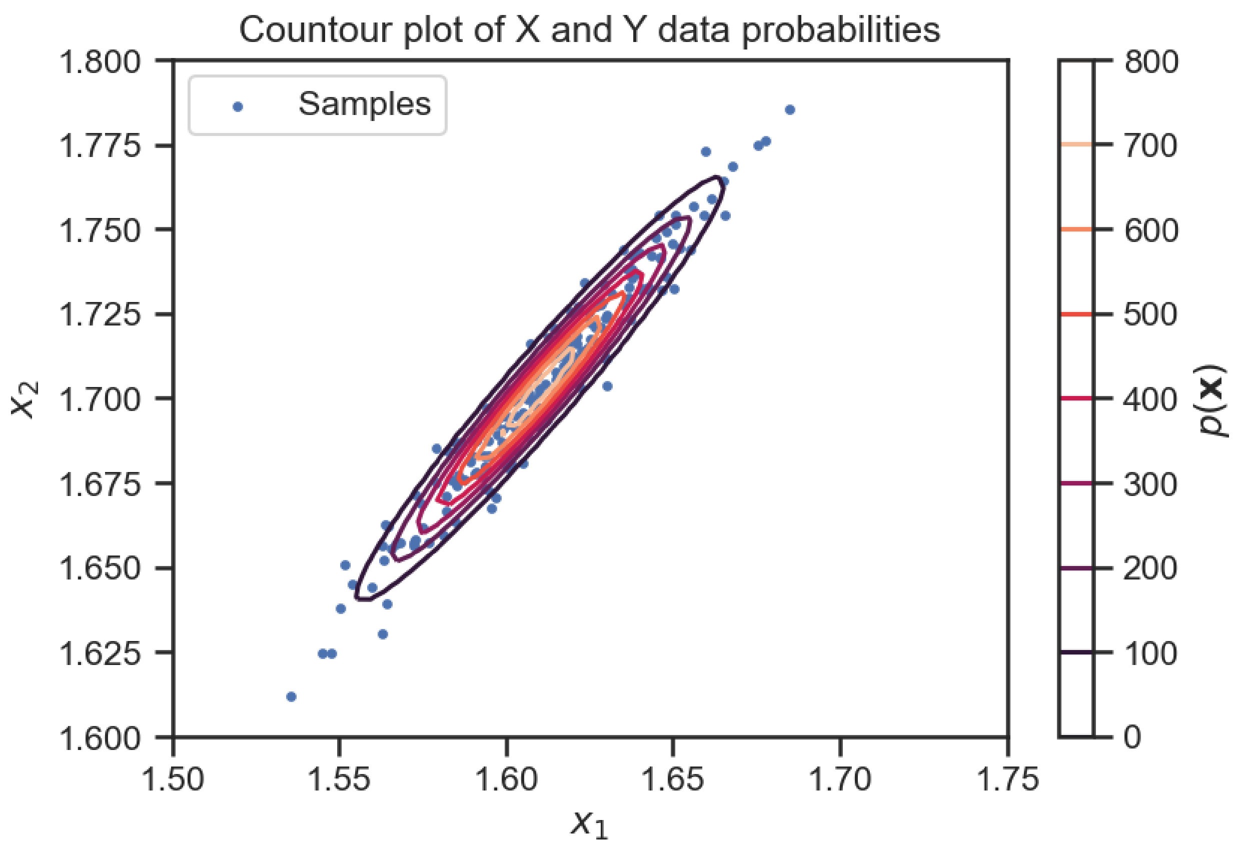
In [36]:
```
Eigens = np.linalg.eigh(C_XY)
print(Eigens[0])
print('Since the Eigen values are positive, C_XY is indeed positive definite')
```

```
[2.474e-05 1.688e-03]
Since the Eigen values are positive, 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 [57]:
```
#3F
XY = scipy.stats.multivariate_normal(mean=mu_XY, cov=C_XY)

fig, ax = plt.subplots(dpi=150)
x1 = np.linspace(1.5, 1.75, 64)
y2 = np.linspace(1.6, 1.8, 64)
X1, Y2 = np.meshgrid(x1, y2)
X_flat = np.hstack(
    [
        X1.flatten()[:, None],
        Y2.flatten()[:, None]
    ]
)
pdf_XY = XY.pdf(X_flat).reshape(X1.shape)
c = ax.contour(X1, X2, pdf_XY)
plt.colorbar(c, label='$p(\mathbf{x})$')
plt.scatter(X_data,Y_data, label = 'Samples', s = 5)
plt.title('Countour plot of X and Y data probabilities')
plt.legend(loc='best')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$');
```



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.

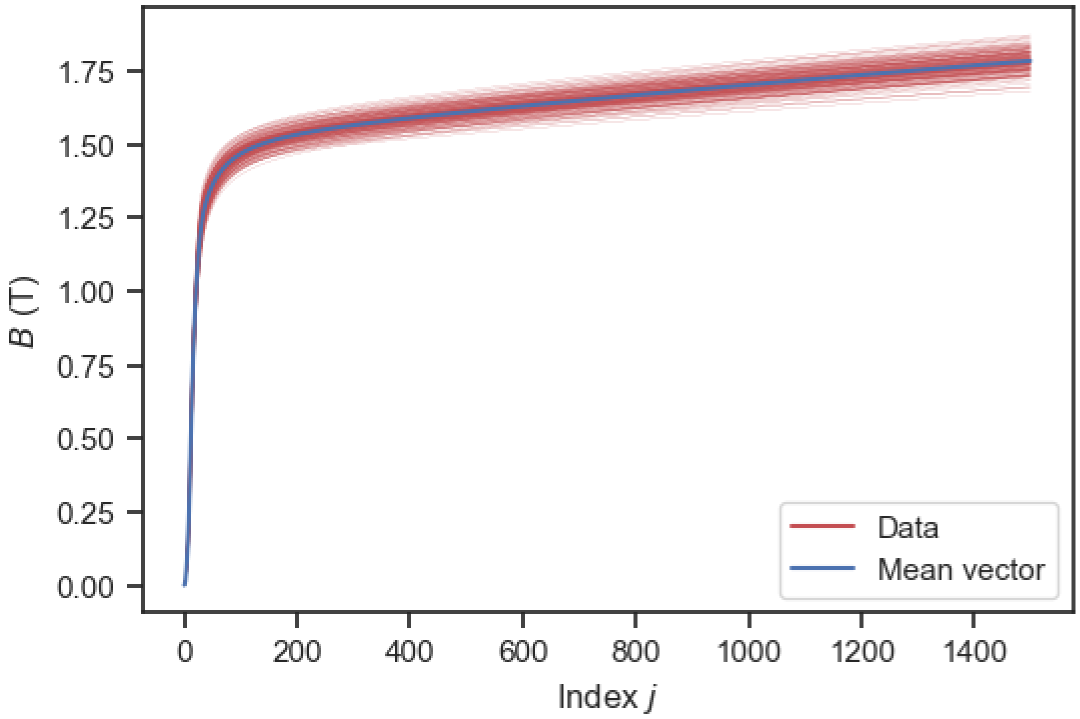
First, let's find the mean of this random vector:

In [38]:
```
B_mu = np.mean(B_data, axis=0)
B_mu
```

Out[38]:
```
array([0.   , 0.004, 0.015, ..., 1.784, 1.784, 1.784])
```

Let's plot the mean on top of all the data we have:

In [39]:
```
fig, ax = plt.subplots()
ax.plot(B_data[:, :].T, 'r', lw=0.1)
plt.plot([],[], 'r', label='Data')
ax.plot(B_mu, label="Mean vector")
ax.set_xlabel(r"Index $j$")
ax.set_ylabel(r"$B$ (T)")
plt.legend(loc="best");
```



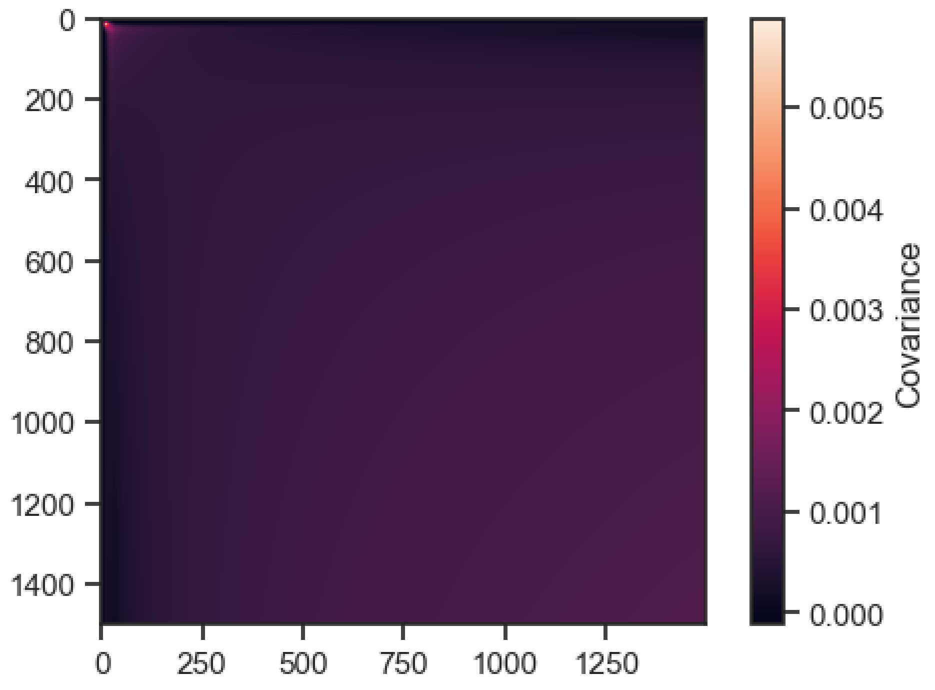
It looks good. Now, find the covariance matrix of $\mathbf{B}$. This is going to be a 1500x1500 matrix.

In [40]:
```
B_cov = np.cov(B_data.T)
B_cov
```

Out[40]:
```
array([[0.000e+00, 0.000e+00, 0.000e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [0.000e+00, 1.163e-06, 4.420e-06, ..., 3.182e-06, 3.184e-06,
        3.185e-06],
       [0.000e+00, 4.420e-06, 1.680e-05, ..., 1.228e-05, 1.229e-05,
        1.229e-05],
       ...,
       [0.000e+00, 3.182e-06, 1.228e-05, ..., 1.203e-03, 1.203e-03,
        1.203e-03],
       [0.000e+00, 3.184e-06, 1.229e-05, ..., 1.203e-03, 1.203e-03,
        1.203e-03],
       [0.000e+00, 3.185e-06, 1.229e-05, ..., 1.203e-03, 1.203e-03,
        1.204e-03]])
```

Let's plot this matrix:

In [41]:
```
fig, ax = plt.subplots()
c = ax.imshow(B_cov, interpolation='nearest')
plt.colorbar(c, label="Covariance");
```



The numbers are very small. This is because the covariance depends on the scale of X and Y. If we change the units it will change. To get a more objective measure, we should be using the correlation. The correlation is essentially the covariance of X and Y scaled by their standard deviation, i.e.:

$$\rho(X,Y) = \frac{\mathbb{C}[X,Y]}{\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}}.$$

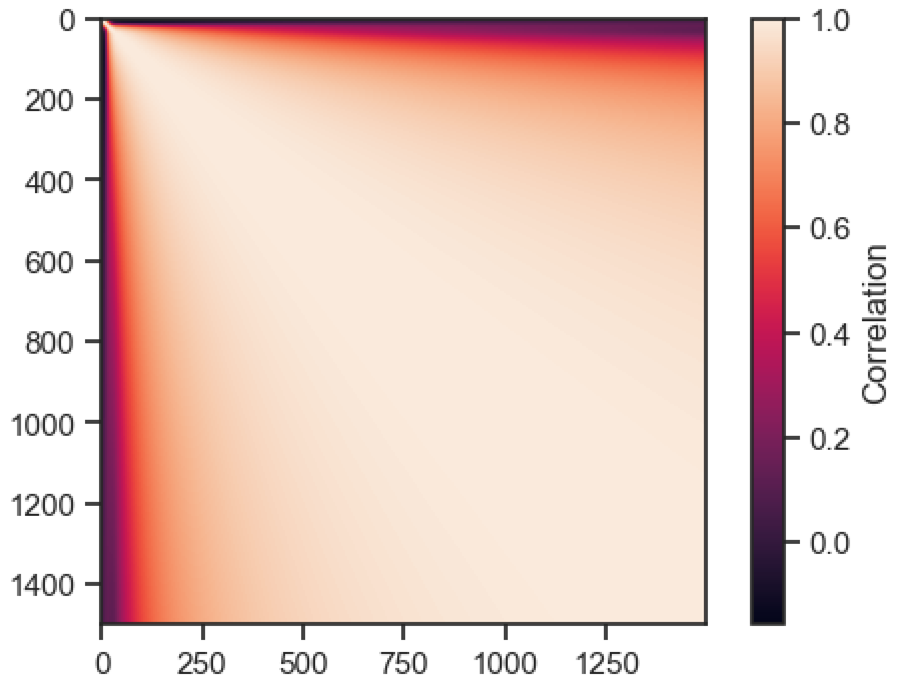
Obviously the correlation has the same sign as the covariance. The addendum is that it is unitless and unambiguously scaled between -1 and +1. Here is how you can find it:

In [42]:
```
# Note that I have to remove the first point because it is always zero
# and it has zero variance.
B_corr = np.corrcoef(B_data[:,1:].T)
B_corr
```

Out[42]:
```
array([[1.   , 1.   , 0.999, ..., 0.085, 0.085, 0.085],
       [1.   , 1.   , 1.   , ..., 0.086, 0.086, 0.086],
       [0.999, 1.   , 1.   , ..., 0.088, 0.088, 0.088],
       ...,
       [0.085, 0.086, 0.088, ..., 1.   , 1.   , 1.   ],
       [0.085, 0.086, 0.088, ..., 1.   , 1.   , 1.   ],
       [0.085, 0.086, 0.088, ..., 1.   , 1.   , 1.   ]])
```

Here is the correlation visualized:

In [43]:
```
fig, ax = plt.subplots()
c = ax.imshow(B_corr, interpolation='nearest')
plt.colorbar(c, label="Correlation");
```



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.

Let's check if the covariance is indeed positive definite:

In [44]:
```
print("Eigenvalues of B_cov:")
print(np.linalg.eigh(B_cov)[0])
```

```
Eigenvalues of B_cov:
[-3.347e-16 -1.346e-16 -7.430e-17 ...  4.662e-02  1.166e-01  1.207e+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 [45]:
```
B = st.multivariate_normal(mean=B_mu, cov=B_cov)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_26156/3953023254.py in <module>
----> 1 B = st.multivariate_normal(mean=B_mu, cov=B_cov)

D:\School\Programing\Anaconda\envs\BME301\lib\site-packages\scipy\stats\_multivariate.py
in __call__(self, mean, cov, allow_singular, seed)
    358         See `multivariate_normal_frozen` for more information.
    359         """
--> 360         return multivariate_normal_frozen(mean, cov,
    361                                           allow_singular=allow_singular,
    362                                           seed=seed)

D:\School\Programing\Anaconda\envs\BME301\lib\site-packages\scipy\stats\_multivariate.py
```

```
in __init__(self, mean, cov, allow_singular, seed, maxpts, abseps, releps)
    728         self.dim, self.mean, self.cov = self._dist._process_parameters(
    729                                                             None, mean, cov)
--> 730         self.cov_info = _PSD(self.cov, allow_singular=allow_singular)
    731         if not maxpts:
    732             maxpts = 1000000 * self.dim

D:\School\Programing\Anaconda\envs\BME301\lib\site-packages\scipy\stats\_multivariate.py
in __init__(self, M, cond, rcond, lower, check_finite, allow_singular)
    163         d = s[s > eps]
    164         if len(d) < len(s) and not allow_singular:
--> 165             raise np.linalg.LinAlgError('singular matrix')
    166         s_pinv = _pinv_1d(s, eps)
    167         U = np.multiply(u, np.sqrt(s_pinv))

LinAlgError: singular matrix
```

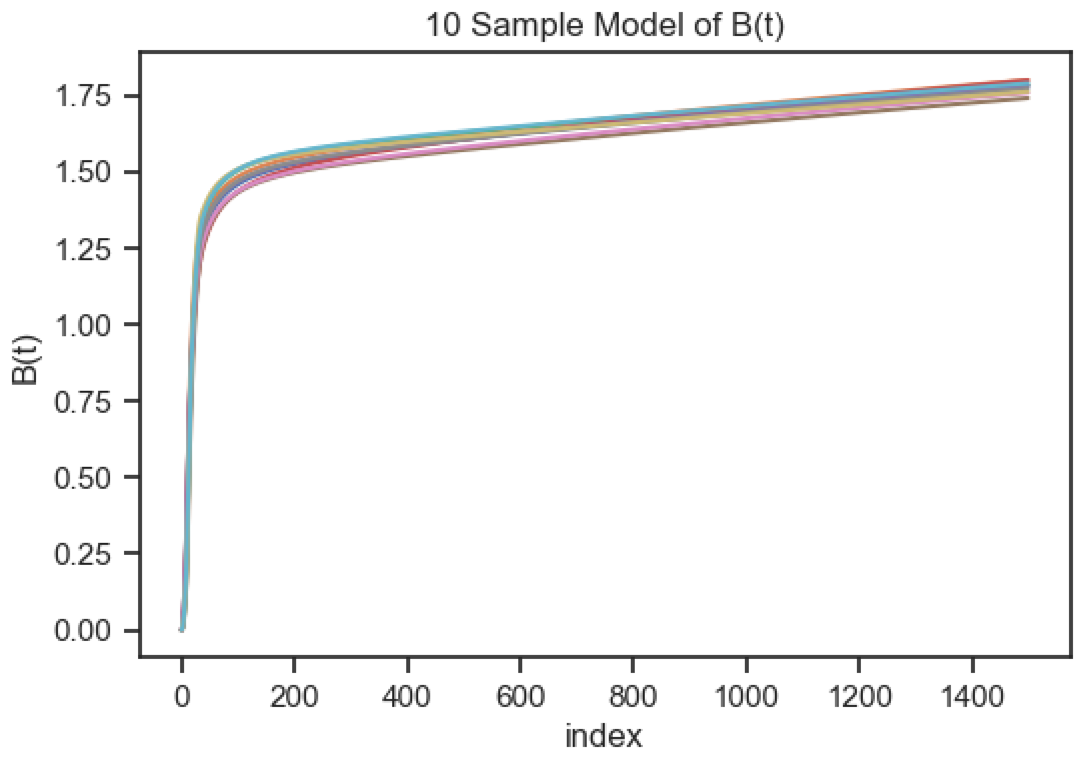
The way to overcome this problem is to add a small positive number to the diagonal. This needs to be very small so that the distribution does not change very much. It must be the smallest possible number that makes the covariance matrix behave well. This is known as the *jitter* or the *nugget*. Find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

In [60]:
```
# Pick the nugget here
nugget = 1e-9 #smallest nugget that worked
# 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 [61]:
```
#3H
plt.figure()
plt.plot(B.rvs(10).T)
plt.ylabel('B(t)')
plt.xlabel('index')
plt.title('10 Sample Model of B(t)')
plt.show()
```



Congratulations! You have made your first stochastic model of a physical field quantity. You have now the ability to sample $B-H$ curves in a way that honors the manufacturing uncertainties. This is the first step to uncertainty quantification studies. The next step would be to propagate these samples through Maxwell's equations to characterize the effect on performance of an electric machine. If you are interested to see how that looks take a look at {cite} `sahu2020` and {cite} `beltran2020` .

```
In [ ]:
```