Introducing Python

To enable us to remote work and learn the necessary tools for Data Analysis (and scientific research in general), we will be using two modern computing concepts: **Interactive coding** (with Jupyter notebooks) and **cloud computing (with Google Colab)**.

This notebook will help you get familiar with using Python in the Jupyter Notebook environment - this is important not only for practice required for the assessements, but also to go through some of the basic concepts of the course. Some core concepts of Data Analysis will only be taught via the Jupyter notebooks and live sessions (and of course the notes).

Jupyter notebooks allow us to write text in cells using the **Markdown** format.

It also allows us to write code in cells, we will use **Python** in this course.

Please ensure you have watched the Chapter 0 video and have set up Google Colab and read through the Chapter 0 jupyter notebook.

So how do we use Python to work with data? Useful things including reading in datafiles from experiments or telescopes, plotting data, plotting error bars, calculating some useful statistics, and perhaps probabilities, and fit models (lines/curves) to the data. Python is really useful because we can *import* packages that already have built in tools to make publication quality plots, or estimate the mean or median and so on. These packages include Numerical Python `numpy` and Scientific Python `SciPy` as well as plotting packages such as `Matplotlib` .

This introduction and the weekly guides are to support you to carry out scientific analysis using data. You do not need to be an experienced programmer, since there are some easy hints and tips that will allow you to do things in a simple way. Eg knowing that you can write an equation in python in the following way `y = x**2` and not `y=x^2` .

You will learn the following things this week

- How to use Jupyter (Colaboratory) notebooks
- How to set up your Google Colaboratory
- How to access and submit Jupyter Notebook assignments
- How to write a Python program and run it

How to get started - Python

If Python is new to you, we recommend getting started with Cardiff Python tutorials. This website also contains other resources (in left hand menu), such as a quick overview of python for coders. This url may require you to click Advanced to bypass the security exception.

For further help with Python, these sites are good:

- The Python Tutorial
- problem-solving with python
- short guide to python/data analysis
- and an online book from one of the best educators in python: Jake VanderPlas.

Also this notebook will help you get to grips with the basics. Any questions, please do raise them at the weekly live sessions or at office hours.

How to get started - Jupyter Notebooks

You are already using a Jupyter notebook if you are reading this! They are an excellent way to interactively write and execute codes in cells, whilst you can plot your data, discuss your results, fit a model and write the equations and interpretation all in one document. No need to cut and paste code from files into word documents.

Before we continue - please make sure you have read and followed the instructions in **Chapter0.ipynb**.

Jupyter notebook files end in .ipynb. The notebook contains cells. A cell is a place to either display text or to write some code to be executed by the notebook's kernel. A kernel is the engine that runs the code in the notebook.

Using notebooks to do some analysis

In a Jupyter notebook you can insert a new cell by clicking on the + cell box at the top of the notebook (Anaconda: Cell > insert cell). This time we'll keep it in the code format and do some basic maths in python.

If you entered the following text to a new code cell:

```
 x=5
y = x**2.
print(y)
```

Your cell will look like this:

In [1]:
```
x = 5
y = x**2.

print(y)
```

Chapter 1

```
25.0
```

And we can see that the cell has output the answer y. But we only get the output answer by compiling the code.

To do this for a single cell, select it with the mouse and click on the ▶ which appears in Colab. Or you can select the Runtime dropdown menu and "Run the focused cell" (Anaconda: ▶Run). To stop a cell from compiling/running we can go to manage sessions and stop some from running (Anaconda: click the ■ button). This is useful if we have complicated code or lots of data being plotted, as it can be very slow at times.

Each cell remembers what is above it, this is both good and bad as you only need to define things/import packages once, but you will need to be careful when writing code with similar looking parameters in one notebook!

Getting started with Markdown/Text

Text boxes in Jupyter Notebooks use a language called Markdown - this is a really simple way to write text. We can use it to describe our methods, state equations that we're using and to write our results and interpretation underneath our code. So in that sense we wish to use it as a word processor - ie with titles, subheadings, text, formatted equations etc.

We can use the following in a Markdown cell to bold text by writing __quantum__ or **quantum**

to make **quantum** or **quantum**

Titles and section headings use # in markdown cells.

You can write equations in markdown using $$.

Here are a few examples to get you started with Markdown.

Example

How would we write the table below in Markdown?

a	b	c
1	2	3

Solution:

Click below to see the solution.

a	b	c
1	2	3

Example

How would we write the following equations in markdown so they are formatted correctly?

- alpha = 10^2
- y = x^2+4
- y = asin(bx)+c
- y = x/x+3 * (z/(z^2-1))

Solution

Click below to see the solution.

`$\alpha = 10^{2}$` produces $\alpha = 10^{2}$.

`$y=x^2+4$` produces $y = x^2 + 4$.

`$y = a+sin(bx)+c$` produces $y = a + sin(bx) + c$

`$y = \dfrac{x}{x+3}+\left(\dfrac{z}{z^2-1}\right)$` produces $y = \dfrac{x}{x+3} + \left(\dfrac{z}{(z^2-1)}\right)$

Markdown Lists

We can write **lists** in Markdown in the following way.

Sometimes we want to include lists.

- Which can be indented.

Lists can also be numbered.

1. Item a
2. Item b

Or:

- Item 1
- Item 2

One can also use "-" to make a bullet point list.

Tips

- You can attach image files (jpg, png) directly to a notebook in Markdown cells by dragging and dropping it into the cell.
- Paragraphs must be separated by an empty line.
- For additional markdown tips, see this blog and this blog.

Getting started with Python

Importing packages

In general, it is good practice to start each notebook with a cell that explains what the notebook is for and a cell that imports useful python libraries e.g.

In [2]:
```
import numpy as np
import scipy as sp
import pylab as plt
%matplotlib inline
```

What does the above cell do?

`numpy` is a package consisting of lots of numerical tools. `scipy` is a package consisting of lots of useful scientific tools. `pylab` is a package for scientific plotting. `%matplotlib inline` is a 'magic command' used when running Python within a Jupyter notebook. It allows the display of data plots within the notebook.

Other useful packages to be used in this course include

`import scipy.stats as st`

`import pandas`

and `import math as m`

The `math` package includes tools like trig functions eg $sine$ etc. Once we've imported the packages, we can use tools inside them by writing `mean = np.mean(x)` to get the mean of a set of values x calling `numpy` to do the work for us. Similarly we can use `y = np.sqrt(x)` to get the square root of x, `y=np.log(x)` for log (base 10) of x and `np.pi` to call π.

Using Python as a calculator

You can use Python as a calculator. Below are some examples.

In [3]:
```
(1+2+3+4+5+6+7)/5
```

Out[3]: 5.6

In [4]:
```
9.**2
```

Out[4]: 81.0

In [5]:
```
(3./1.) + (4+5+3+2)**2.
```

Out[5]: 199.0

Sometimes Python spits out a zero when it is not expected, or a number with no decimal places (1). This is likely because you have typed 7/4 instead of 7./4. or 3/1 instead of 3./3 and Python has then rounded down to the nearest integer. To be safe always use the decimal point to let Python know it is a floating point value and not an integer.

Tips

- If something seems slightly "off" with your numbers, then go back and check the positioning of your brackets - often a misaligned bracket in an equation is responsible for all sorts of mishap.
- Note that you can also put *comments in the code cells* by writing what your line of code is doing using a #. This is really good practice so that when you return to your code later, you can remind yourself why that line of code is there.

Printing out results and text

You can get Python to do some calculations and then print results out (or even print out text if you'd like). We can do this by using `print("some text here")`.

To print out a result y and some explanation text, we can write `print("some text here",y)`.

Note that we can use words in Python code by defining them as strings. These are made clear by using the `""` or `''`. You can use strings and do manipulations on them - see examples later on in the notebook.

Example

Make a cell print out Hello World.

Solution

Click below for the Solution.

In [6]:
```
print('Hello World')
```

```
Hello World
```

Example

You've found a model $y(t) = \frac{1}{2}gt^2 + v_0t + y_0$ explains the experiment you've been working on in the lab where you've been searching for an equation that describes the position of a falling body y as a function of time t in free-fall. g is the acceleration due to gravity and v_0 and y_0 are the initial conditions.

What is the value of $y(t)$ for $t = 2.56$s if $v_0 = 1.26$m/s and $y_0 = 1.35$m?

Solution

Click below for the Solution.

In [7]:
```
t=2.56
v_0 = 1.26
y_0 = 1.35
g = 9.81

y = 0.5*g*t**2+v_0*t+y_0
print(y)
```

```
36.721008000000005
```

To explain our result to the reader of the notebook, we can provide a little more information in our answers:

In [8]:
```
print("the value of y is",y)
```

```
the value of y is 36.721008000000005
```

Formatting

Note that I didn't have to do all of the same numbers and equations again, because Jupyter notebooks *remembers all of the definitions and everything in the code above it* if it is compiled.

Now we want to be even more scientific and include units and formatting in our result, since the large amount of the significant figures above are not realistic (we very rarely know parameters to this level of precision).

This brings us to the point of **formatting numbers**. There are many ways to do this, we can format numbers using Python's `str.format()`. To do this, we can use

`print("FORMAT".format(NUMBER))`

where `FORMAT` is what we want our output to look like. Eg 2 significant figures is `"{:.2f}"` or `"{:.0f}"`. `NUMBER` is either a number which needs formatting or your result (in the case above we would replace this with y).

In [9]:
```
print('This is a bad example: the value of y is',y)
print()

print("This is a formatted example:","{:.2f}".format(y))
print()

print('Now this is a great example:')
print('The position of a falling body $y$ at time $t=2.56$s in free-fall is {:.2f} m.'.format(y)) # makes it 3 sig figs after decimal pt
```

```
This is a bad example: the value of y is 36.721008000000005

This is a formatted example: 36.72

Now this is a great example:
The position of a falling body $y$ at time $t=2.56$s in free-fall is 36.72 m.
```

Lists

You can create a list of objects in Python. This is handy because we can do all sorts of cool things with them, such as adding two separate lists together, pulling out values from the list. *The elements of a list are numbered from zero.* So if you have a list of 5 values, the elements in that list are numbered 0, 1, 2, 3 and 4.

If we take a list and add it it to another list, it doesn't add each item together, but simply adds the two lists to make a bigger list. Similarly, we can multiply a list, but this doesn't multiply each item, rather the list itself.

Example

a = [1,2,3,4,5] and b = [6,7,8,9,10] are lists.

- What is the list $c = a + b$ in python code?
- What is the list $c = 3a$ in python code?
- What is the 1st element of the list c?
- What is the 5th element?
- What is the number of data points in c?

Solution

Click below for the solution.

In [10]:
```
a = [1,2,3,4,5]
b = [6,7,8,9,10]

c = a+b
print('a+b where a and b are lists = ',c)

d = 3*a
print('3a where a is a list =',d)
print()

print('first element in c is',c[0])
print('fifth element in c is',c[4])
print('the number of data points in c is',len(c))
```

```
a+b where a and b are lists =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3a where a is a list = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

first element in c is 1
fifth element in c is 5
the number of data points in c is 10
```

Iterating over values - for loops

Python can iterate over the items in a list. For loops great for repeating something a fixed number of times. Python has a `range` function so that `range(10)` will give a set of 10 integers from 0 to 9. `range(1,5)` will give a set of integers from 1 to 5, and `range(6,10,2)` will give a range of 6-10 in steps of 2.

The for loop needs a colon (:) and everything after that (related to the for loop) must be indented.

We can use for loops to do lots of great things, but it is very important you know how they work - make sure you print out your values to check it is doing what you want it to.

Example

Write a for loop to print out the index of the loop and 2 times the index of the loop.

Solution

Click below to see the solution.

In [11]:
```
for k in range(1, 10):
    print('k in loop: ',k,'2k in loop:',(2*k))
```

```
k in loop:  1 2k in loop: 2
k in loop:  2 2k in loop: 4
k in loop:  3 2k in loop: 6
k in loop:  4 2k in loop: 8
k in loop:  5 2k in loop: 10
k in loop:  6 2k in loop: 12
k in loop:  7 2k in loop: 14
k in loop:  8 2k in loop: 16
k in loop:  9 2k in loop: 18
```

Example

Write a for loop to print out "I have a pet...." for all of the animals in the list animals = ["lemur","panda","shark","bat"].

Write another loop to print out all the letters in "sloth".

Solution

Click below to see the solution.

In [12]:
```
animals=["lemur","panda","shark","bat"]

for item in animals:
    print("I have a pet " + item)

print()
for item in "sloth":
    print(item)
```

```
I have a pet lemur
I have a pet panda
I have a pet shark
I have a pet bat

s
l
o
t
h
```

Example

Calculate the position of a falling body in free-fall from $t = 0$s to 1000s in steps of 50s.

Solution

Click below to see the solution.

In [13]:
```
t=range(0,1000,50)
# set up an array of times which starts at t=0, ends at t =
# 1000s in steps of 50s

# now call the equation for each time by using the for loop:
for i in range(0,len(t)):   # each value of t is called as t[i] where i is an index
                            # from t[start] to t[end]
    y = 0.5*g*t[i]**2+v_0*t[i]+y_0
    print(y)
```

```
1.35
12326.85
49177.35
110552.85
196453.35
306878.85
441829.35
601304.85
785305.35
993830.85
1226881.35
1484456.85
1766557.35
2073182.85
2404333.35
2760008.85
3140209.35
3544934.85
3974185.35
4427960.85
```

If instead we used the following code, we would only get the last value of y.

In [14]:
```
# now call the equation for each time by using a for loop:
for i in range(0,len(t)):
    y = 0.5*g*t[i]**2+v_0*t[i]+y_0

print(y)
print('so we see we only get one value of y if we print outside the for loop')

print()
# though we should use the following for our print statement:
print('Note: the correct way to write this would be something like:')
print('The position of body in free fall at t=1000s is {:.2e} m'.format(y))
```

```
4427960.85
so we see we only get one value of y if we print outside the for loop

Note: the correct way to write this would be something like:
The position of body in free fall at t=1000s is 4.43e+06 m
```

Append and factorising code

One way to get around this issue (where you can only get the values of y for all t inside the for loop and not outside) is to use Python `append` function.

A much simpler alternative way of calculating y for every value of t without having to do the for loop iteration over each t value is shown below (in fact we can do it in one line).

In [15]:
```
y = [] # tell the code that y will have more than one value
```

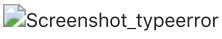
```
for i in range(0,len(t)):
    y.append(0.5*g*t[i]**2+v_0*t[i]+y_0)

print('using a loop to calculate y for each value of t and then append all values to a list gives')
print(y)

print()
print(r'or we can write the following code: y = [0.5*g*x**2+v_0*x+y_0 for x in t] to give:')
# the little r in the line above tells python you want to print out exactly what you have written!
print()

y2 = [0.5*g*x**2+v_0*x+y_0 for x in t]
print(y2)
```

```
using a loop to calculate y for each value of t and then append all values to a list gives
[1.35, 12326.85, 49177.35, 110552.85, 196453.35, 306878.85, 441829.35, 601304.85, 785305.35, 993830.85, 1226881.35, 1484456.85, 176655
7.35, 2073182.85, 2404333.35, 2760008.85, 3140209.35, 3544934.85, 3974185.35, 4427960.85]

or we can write the following code: y = [0.5*g*x**2+v_0*x+y_0 for x in t] to give:

[1.35, 12326.85, 49177.35, 110552.85, 196453.35, 306878.85, 441829.35, 601304.85, 785305.35, 993830.85, 1226881.35, 1484456.85, 176655
7.35, 2073182.85, 2404333.35, 2760008.85, 3140209.35, 3544934.85, 3974185.35, 4427960.85]
```

Yay - we have basically saved all the values of y into one list.

Data types

Now there may be times when you wish to take your list of y values and do some mathematical operations on them, for example what is y^2? Naively we might think we can simply just multiply all the values in our y list by itself.



Obviously this has not worked. This is because it's a list and we can't multiply a list this way (we'd have to do another for loop). To find out if you have a list or an array you can use the `type` command. We will return to `numpy` arrays later.

We can convert our Python list into a Python array using `numpy` and then do mathematical calculations with our data - let's say y^2 for example by using the following code

```
y_arr = np.array(y)
y_sq = y_arr*y_arr
print(y_sq)
```

Example

Find out the types of the following `a=1` , `b=1.0` , and `c="1.0"` .

Solution

Click below to see the solution.

In [16]:
```
a=1; b=1.0; c='1.0'

print(type(a))
print(type(b))
print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

So what is the type of our data $y(t)$?

In [17]:
```
print(type(y))
```

```
<class 'list'>
```

It's a list!

If Statements

If statements allow us to have conditions. For example, if the speed of the falling body goes above a critical speed v_{crit}, then perhaps a different equation applies.

Example

You are asked whether you have any symptoms of a fever or a continuous cough. Use Python `if` statements to print an appropriate message if the answer to this question is yes, no, or not sure.

Solution

Click below to see the solution.

In [18]:
```
your_symptoms = "ill"
```

```
if your_symptoms == 'ill':
    print('You are '+ your_symptoms + ' - You need to go home and rest in isolation.')
elif your_symptoms == 'not ill':
    print('You are '+ your_symptoms +' - OK, take care of yourself.')
elif your_symptoms == 'not sure':
    print('You are '+ your_symptoms + '- Best to be safe and go home and rest.')
else:
    print('I am sorry, I do not know what to say.')
```

```
You are ill - You need to go home and rest in isolation.
```

Note that here we are making use of strings `" "` instead of numbers.

Example

Let's suppose we want to estimate the position of a body falling in free fall y only when $t \leq 100$s. At any other times, we set it to equal to zero (ie they hit the ground at 100s). We can do this using Python `if` statements.

Solution

Click below to see the solution.

In [19]:
```
y = []

for i in range(0,len(t)):
    if (t[i] <= 100):
        y.append(0.5*g*t[i]**2+v_0*t[i]+y_0)
    else:
        y.append(0.)

print(y)
```

```
[1.35, 12326.85, 49177.35, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

In these examples you will see that Python tests for equality uses the `==` to separate it from the `a = [1,2,3,4,5]` assignments of variables. We can also use `<`, `<=`, `>=` and `>` in Python code.

Arrays

In science we often have data $x_0, x_1, x_2, \ldots, x_{n-1}$ and related $y_0, y_1, y_2, \ldots, y_{n-1}$. We can use a Python list a for the x values with elements a[0], a[1], a[3],..., a[n-1] and so on. Python lists can contain any type of Python object, but sometimes we want one to contain numbers only. We can use arrays, which can be mutidimensional (eg 3x3 array instead of the 1d list). Python arrays are lists of objects of the same type and can make your code faster if you use them.

Example

Convert the $y(t)$ values from above (which are currently in a Python list) into an array and check it has worked.

Solution

Click below to see the solution.

In [20]:
```
y = np.array(y)

print(type(y))
```

```
<class 'numpy.ndarray'>
```

arange

It works! We have used `numpy` to convert our list of values into an array that can be manipulated mathematically.

As we used `range` earlier to make a list of time array of values, we can use `np.arange()` to make an array of time values. Examples are below:

In [21]:
```
a = np.arange(10) # make an array with 10 values (produces integers)

print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [22]:
```
a = np.arange(0,10,1) # now go from 0-10 in steps of 1 (produces integer)
print('a=',a)


b = np.arange(0,10,0.5)  #now go from 0-10 in steps of 0.5 (produces float)
print('b=',b)
```

```
a= [0 1 2 3 4 5 6 7 8 9]
b= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
```

How can we check the types of the numbers in a and b? Are they integers or floats? If we use `type(a)` this gives us the type of a (an array) so we need to call

one of the numbers in the array and check it's type. We will do this using the 1st value in the array `a[0]` .

In [23]:
```
# what is the type of the 1st number in a
# what is the type of the 1st number in b
print('The type of 1st number in a is', type(a[0]), '-- is it an integer? Yes.')
print('The type of 1st number in b is', type(b[0]), '-- is it a float? Yes.')
```

```
The type of 1st number in a is <class 'numpy.int64'> -- is it an integer? Yes.
The type of 1st number in b is <class 'numpy.float64'> -- is it a float? Yes.
```

Slicing data, getting subsets of data

Some examples of slicing data samples where `a` is an array of data with length `len(a)` and has a starting data point `a[0]` and an end data point `a[len[a]-1]` . We call the starting and ending data points we wish to pull out of the array `start` and `stop` in the examples below.

`a[start:stop]` ---- items start through stop-1

`a[start:]` ---- items start through the rest of the array

`a[:stop]` ---- items from the beginning through stop-1

`a[:]` ---- a copy of the whole array

`a[start:stop:step]` ---- can have a step size to pull out say every 100 points from the array

Tip

You can use `numpy` to get the min, max and mean, standard deviation for your arrays using `np.mean(a)` etc.

Equations and functions

If you have to use the same function a few times, or you like tidy code, it's easiest to define a function. Here is an example function for $y = x^2 + 2$.

In [24]:
```
def my_equation(x):
    return x**2+2
```

In [25]:
```
a = 3.4

print(my_equation(a))
```

```
13.559999999999999
```

In [26]:
```
print('for a =  3.4s, the equation gives {:.2f}'.format(my_equation(a))) # no units given in question.
```

```
for a =  3.4s, the equation gives 13.56
```

Example

Write a function for the position of a body falling in free fall for time $t = 30$s. The initial speed and position are 3m/s and 1m.

Solution

Click below to see the solution.

In [27]:
```
def position(time,accel,v_init,y_init):
    value = 0.5*accel*time**2+v_init*time+y_init
    return value

t = 30.
g = 9.81
v_0 = 3.
y_0 = 1.

y = position(t,g,v_0,y_0)

print('at t =  30s, the value of position is {:.2f} m'.format(y))
```

```
at t =  30s, the value of position is 4505.50 m
```

Example

Now get values of the position of a body falling in free fall for times from $t = 0$s to $t = 100s$ in steps of 10s. The initial speed and position are 3m/s and 1m.

Solution

Click below to see the solution.

In [28]:
```
t = np.arange(0,100,10) # set up time values

y = position(t,g,v_0,y_0)
```

```
print('time in secs',t)
print('position in m',y)
```

```
time in secs [ 0 10 20 30 40 50 60 70 80 90]
position in m [1.00000e+00 5.21500e+02 2.02300e+03 4.50550e+03 7.96900e+03 1.24135e+04
 1.78390e+04 2.42455e+04 3.16330e+04 4.00015e+04]
```

Now we can use a for loop to print this out nicely so that we have t and y values in columns:

In [29]:
```
for i in range(0,len(t)):
    y = position(t[i],g,v_0,y_0)
    print(t[i],y)
```

```
0 1.0
10 521.5
20 2023.0
30 4505.5
40 7969.0
50 12413.5
60 17839.0
70 24245.5
80 31633.0
90 40001.5
```

Importing data files

The data file you want to export must be in the same directory as the notebook or you will need to tell the notebook where to look for the file. You can upload the file to Google Colab. In this example we will use the `numpy` package (shortened to `np`) and call the `numpy genfromtxt` tool to read the data file:

```
data = np.genfromtxt('file.dat')
```

Sometimes an error occurs with the `np.genfromtxt`. If this happens, you might want to skip the first row (often in data files there are words in the first row, known as the header information. To get around this you can use:

```
data = np.genfromtxt('file.dat',skip_header=1)
```

Sometimes you need to specify the delimiter in the data file (tab/space/comma), in this case it looks like:

```
data = np.genfromtxt('file.dat',delimiter=' ')
```

(Note you can also use `np.loadtxt`).

Example

Read in the data file DataAnalysis_testfile.dat and print it out. You can find this file here.

Solution

Click below to see the solution.

In [30]:
```
import numpy as np

data = np.genfromtxt('DataAnalysis_testfile.dat')

print(data)
```

```
[[ 0.223279 26.857211  1.6     ]
 [ 0.547843 20.331691  1.6     ]
 [ 0.56679  21.507016  1.6     ]
 [ 0.50147  15.098177  1.6     ]
 [ 0.944974 24.821554  1.6     ]
 [ 0.853123 21.838086  1.6     ]
 [ 0.759773 18.597575  1.6     ]
 [ 0.319978 22.060243  1.6     ]
 [ 0.907024 20.913449  1.6     ]
 [ 0.482762 20.570473  1.6     ]
 [ 0.599497 32.412242  1.6     ]
 [ 0.540597 21.428759  1.6     ]
 [ 0.892695 26.270349  1.6     ]]
```

So we can see we have 3 columns of data. Let's check if there are column headings:

In [31]:
```
data = np.genfromtxt('DataAnalysis_testfile.dat',names=True)
print(data.dtype.names) # this gives us our column names from the data file
```

```
('x', 'y', 'y_error')
```

It looks like our data file contains x, y data and error bars on the y data.

Example

Print out the first column, the second row, the first value in the first column and the number of datapoints.

Solution

Click below for the solution.

In [32]:

```
# use the header names
print('all of the first column:')
print(data['x'])
print()

#Print second row
print('all of the second row:')
print(data[1]) # python starts at zero!
print()


# print first value in first column
print('first value in first column:')
print(data[0][0])
print()


# print length of data
print('number of datapoints:')
print(len(data))
print()
```

```
all of the first column:
[0.223279 0.547843 0.56679  0.50147  0.944974 0.853123 0.759773 0.319978
 0.907024 0.482762 0.599497 0.540597 0.892695]

all of the second row:
(0.547843, 20.331691, 1.6)

first value in first column:
0.223279

number of datapoints:
13
```

Plotting data

The most commonly used library for plotting sciencitific data in Python is matplotlib which can be used by importing the `pylab` module. There are a wealth of online examples for making plots, some publication level quality, and some more simple. In this course, we don't need to spend time on making publication level quality plots, but rather to try and make scientific data appear clear - ie labelled axes, large enough fontsize, legends.

Plotting in Python is very intuitive, we basically use plot(x,y) where x and y are lists of numbers or arrays. You can add more data via using plot(x2,y2) to add more lines/datapoints on the same curve. We can logscale our plots or our data points if we wish using `plt.loglog(x,y)` or `plt.semilogy(x,y)` . We can show scatter plots (data points) using scatter(x,y). Histograms can be drawn using hist(x,bins) and so on.

Example

Plot $y = sin(x)$ for x ranging from 0 to 6π in steps of 0.1π.

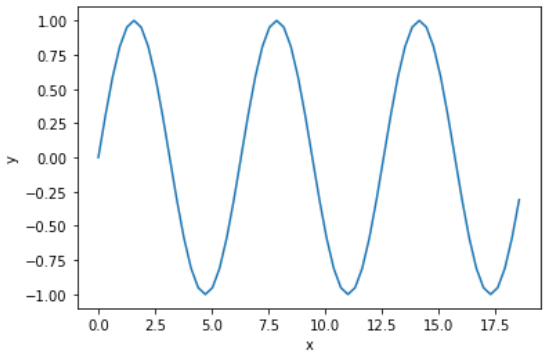
Solution

Click below to see the solution.

In [33]:

```
x = np.arange(0,6*np.pi,0.1*np.pi)
plt.plot(x,np.sin(x))
plt.xlabel('x')
plt.ylabel('y')
```

Out[33]: `Text(0, 0.5, 'y')`



Example

- Plot $y = sin(x)$ for x ranging from 0 to 6π in steps of 0.1π.
  - Add a curve to show $y = cos(x)$.
  - Make the linewidths width 2
  - Make the $cos(x)$ curve have an opacity of 0.6.

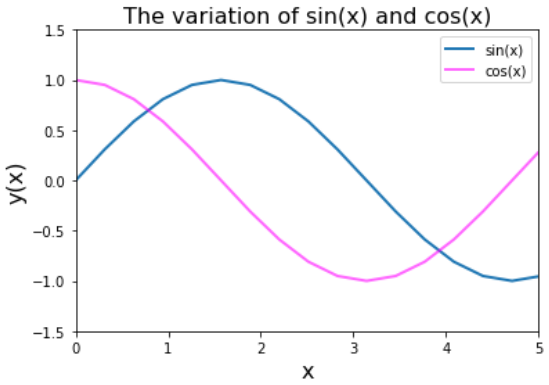
- Set the y limits to be (-1.5,1.5) and x limits of (0,5).
- Add a title.
- Set the fontsize of the title and labels to equal size 16.
- Add a legend with labels for the curves.

Solution

Click below to see the solution.

In [34]:
```
plt.plot(x,np.sin(x),label='sin(x)',lw=2) #lw = thickness of line, label = the name of curve fn or datafile
plt.plot(x,np.cos(x),label='cos(x)',lw=2,c='magenta',alpha=0.6) # c=colour,alpha=transparency
plt.xlabel('x',fontsize=16)
plt.ylabel('y(x)',fontsize=16)
plt.ylim(-1.5,1.5)
plt.xlim(0,5)
plt.legend(loc='upper right')  # makes the legend
plt.title('The variation of sin(x) and cos(x)',fontsize=16) # add a title
```

Out[34]: Text(0.5, 1.0, 'The variation of sin(x) and cos(x)')



Example

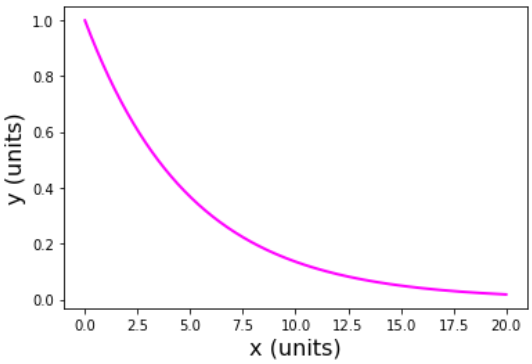
Plot $y = \exp\left(\frac{-x}{5}\right)$

Solution

Click below to see the solution.

In [35]:
```
import numpy as np
import pylab as plt # for plotting
# the line below makes the plot appear in the jupyter notebook
%matplotlib inline

# define function to calculate y
def func(x):
    return np.exp(-x / 5.0)

t = np.arange(0.01, 20.0, 0.01)
plt.plot(t, func(t),c='magenta',lw=2)
plt.xlabel('x (units)',fontsize=16)
plt.ylabel('y (units)',fontsize=16)
```

Out[35]: Text(0, 0.5, 'y (units)')



Example

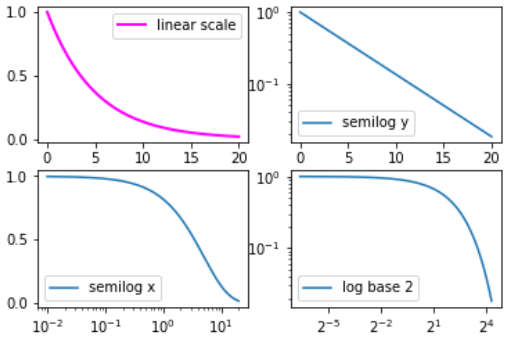
Plot $y = \exp\left(\frac{-x}{5}\right)$ on different logscales.

Solution

Click below to see the solution.

In [36]:
```
plt.subplot(221) # makes a row=2 x column=2 grid of plots, 3rd number is index of each subplot 1-4
plt.plot(t, func(t),c='magenta',lw=2,label='linear scale')
plt.legend(loc='upper right')

plt.subplot(222)
plt.semilogy(t, func(t),label='semilog y')
plt.legend(loc='lower left')

plt.subplot(223)
plt.semilogx(t, func(t),label = 'semilog x')
plt.legend(loc='lower left')

plt.subplot(224)
plt.loglog(t,func(t),basex=2,label='log base 2')
plt.legend(loc='lower left')
```

Out[36]:
```
<matplotlib.legend.Legend at 0x1127c7c10>
```



Example

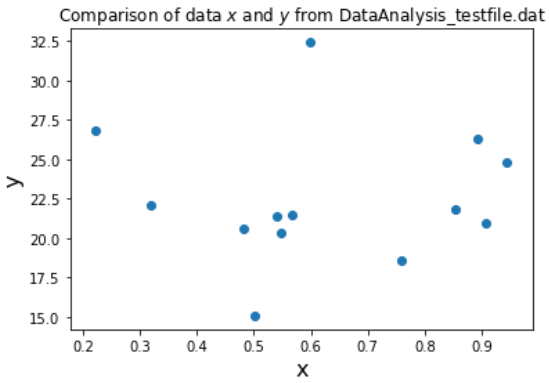
Plot the data in the data file DataAnalysis_testfile.dat.

Solution

Click below to see the solution.

In [37]:
```
data = np.genfromtxt('DataAnalysis_testfile.dat',names=True)
plt.scatter(data['x'],data['y'])
plt.xlabel('x',fontsize=16)
plt.ylabel('y',fontsize=16)
plt.title('Comparison of data $x$ and $y$ from DataAnalysis_testfile.dat')
```

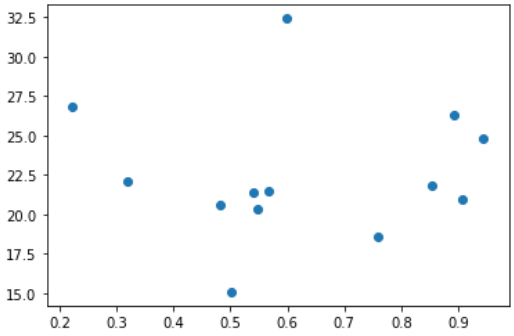
Out[37]:
```
Text(0.5, 1.0, 'Comparison of data $x$ and $y$ from DataAnalysis_testfile.dat')
```



We could also plot the above if we don't have any names for columns, see below:

In [38]:
```
data = np.genfromtxt('DataAnalysis_testfile.dat')
plt.scatter(data[:,0],data[:,1])
```

Out[38]:
```
<matplotlib.collections.PathCollection at 0x112b9a9d0>
```

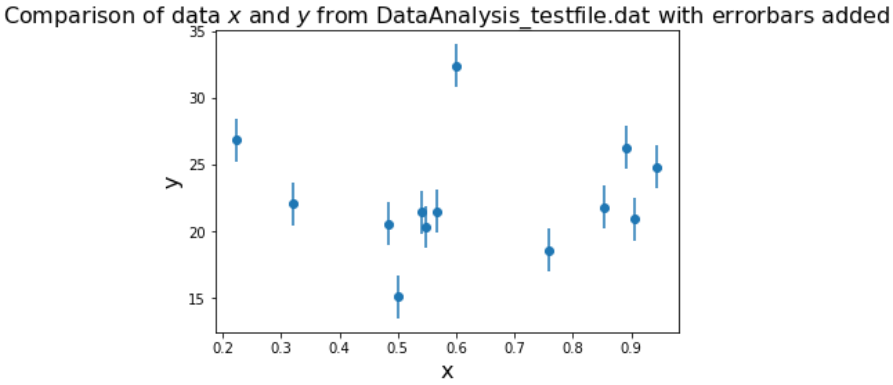


Errorbars

Often in physics, we have errors in our measurements. These can be easily added via the `errorbar` option in `matplotlib` .

In [39]:
```
data = np.genfromtxt('DataAnalysis_testfile.dat',names=True)
plt.errorbar(data['x'],data['y'],yerr=data['y_error'],fmt='o')
plt.xlabel('x',fontsize=16)
plt.ylabel('y',fontsize=16)
plt.title('Comparison of data $x$ and $y$ from DataAnalysis_testfile.dat with errorbars added',fontsize=16)
```

Out[39]:
```
Text(0.5, 1.0, 'Comparison of data $x$ and $y$ from DataAnalysis_testfile.dat with errorbars added')
```



Tip

All of the plotting options that can be used in matplotlib can be found here.

General Tips

- Make sure your code works in a new clean environment. In Colab: Runtime->restart Runtime, in Anaconda's Jupyter: Kernel->Restart.
- Dont forget to save your notebook regularly.
- Use spaces and empty lines to make program clearer.
- Add comments to your code so you know why you're doing something.
- Use sensible names for variables and functions, choose names you would be able to understand if you came back to your code a year later.
- With equations and calculations, use same variables as your equations in the notes/lectures so you will be able to directly compare. Do all of the calculation in the python cells, don't do part of it in your head/in your notes.
- Always have a markdown box before and after code cells explaining what you're going to do, and discussing the results.
- Don't forget to compile your cells, and run all before submitting assessed work.
- Don't be afraid to google for bits of code - particularly make use of the SciPy and Python documents.

Now you are ready to tackle the **Chapter 1 quiz** on Learning Central and the Chapter1_yourturn notebook.

Chapter 2

Introducing Probability

Please ensure you have watched the Chapter 2 video(s).

You will learn the following things in this Chapter

- Basic probability rules
- Bayes Theorem
- How to use Python programming to estimate probabilities
- After completing this notebook you will be able to attempt CA 1 questions 1 and 4.

Probably useful: probability revision

We are going to be using the concept of probability a lot in this course, so it is best to first review the basic ideas behind probability theory, and get used to the notation. Unfortunately, there are many notations; this course will adopt one, but we will also mention the others, so that you can recognise them when reading further.

- An experiment results in a set of outcomes, which we will call Ω. This can be a discrete set of outcomes, such is in the classic coin toss, $\Omega = \{H, T\}$, or the roll of a die, $\Omega = \{1, 2, 3, 4, 5, 6\}$. However in many real life experiments, Ω, which is referred to as the *outcome space*, or *event space*, can have an infinite continuum of values. We will return to this idea later, but for the moment we will consider just discrete events to help outline the basic properties of probability.

- Returning to the coin toss, we can say that a fair coin will have a probability of heads of $P(H) = 0.5$, and a probability of tails of $P(T) = 0.5$. Each outcome of the experiment of tossing the coin, $\Omega = \{H, T\}$, are thus equally likely. Similarly, for a die roll, $\Omega = \{1, 2, 3, 4, 5, 6\}$, with $p(i) = 1/6$. When an experiment has m equally likely outcomes, the probability of any outcome x is then:

  $$P(x) = \frac{N(x)}{m}$$

  where $N(x)$ is the number of times that x occurs. For example, consider the more complicated case where we toss three different coins together, a 50p, a 20p and a pound coin. The outcome space is then

  $$\Omega = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}.$$

  Assuming all outcomes were equally likely, then the probability of getting any one is 1/8.

- Not all outcomes are equally likely. For example, we could ask what the probability of the event that our 3-coin toss comes up with n heads, so the outcome space is then $\Omega = \{0, 1, 2, 3\}$. The outcome $\omega \in \Omega$ in this new experiment is just given by considering the outcomes in our previous 3-coin experiment, but ignoring which exact coin lands on Heads or Tails ie

  - $\omega = 1$: n = 0, corresponds to TTT
  - $\omega = 2$: n = 1 corresponds to HTT, THT, or TTH
  - $\omega = 3$: n = 2 corresponds to HHT, HTH, or THH
  - $\omega = 4$: n = 3 corresponds to HHH

  Thus $P(n = 0) = P(n = 3) = 1/8$ while $P(n = 1) = P(n = 2) = 3/8$.

This example problem originates from this site.

The above uses of P hide an important aspect of probabilities: $P(x_i)$ is *normalised*, such that the sum of the probabilities of all possible events adds up to 1,

$P(X) = \sum_i^N P(x_i) = 1,$

where $X = (x_i, \ldots, x_N)$.

To put this another way:

- the probability of something which is certain to happen is 1,
- the probability of something which is impossible to happen is 0,
- the probability of something not happening is 1 minus the probability that it will happen.

Technically this in only true for either finite or countably infinite outcome spaces. If the outcome space is truly uncountably infinite, then the definition of probability has to be relaxed.

Axioms

Some definitions before we continue:

- **Disjoint**: Two events that cannot occur at the same time are called disjoint or mutually exclusive.
- **Discrete** variables: this is a variable whose value can be obtained by counting, and has gaps within each value that the variable can take on.
- **Continuous** variables: this is a variable whose value cannot be obtained by counting, and can take on all values within the range.

For a discrete variable, if I pick any two consecutive outcomes. I cannot get an outcome that lies in between. For example, if we consider 1 and 2 as outcomes of rolling a six-sided die, then I cannot have an outcome of 1.5).

The axioms of probability are:

Axiom 1 : $0 \leq P(A) \leq 1$

Axiom 2 : $P(\Omega) = 1$

Axiom 3 : $P(A_1 \cup A_2 \cup A_3 \dots) = P(A_1) + P(A_2) + P(A_3) + \dots$

Here the symbol $\cup$ denotes **or** ie P(event A_1 occurs or event A_2 occurs or both occur etc). This is true for disjoint events and can be used for any number of disjoint events.

The axioms permit us to work out the probability of event A **not** occurring,

$$P(A^c) = 1 - P(A),$$

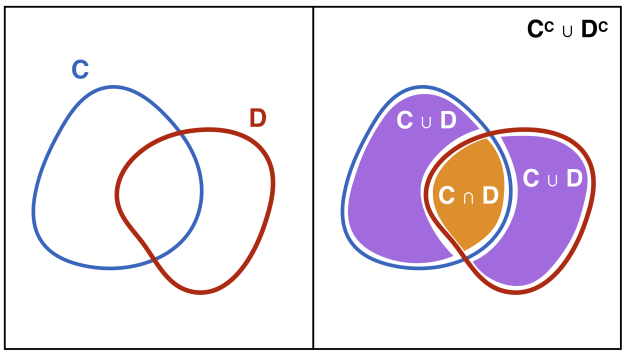
where ^{c} is called the **complement** ie P(not A).

This result can be generalised. For example, for any two events C, D, the probability of getting C **or** D is:

$$P(C \cup D) = P(C) + P(D) - P(C \cap D),$$

where now the symbol $\cap$ denotes **and** (also written as $P(CD)$ or $P(C, D)$ in the literature). A simple way to think of this is that the probability of getting either C or D is just **the sum** of the chances of getting either ($P(C) + P(D)$) minus that chances of getting both at the same time, $P(C \cap D)$.

The last part is important since we're asking for the probability of *either* C **or** D, not both! The best way to see this is by considering the diagram below.



The orange bit in the middle is the bit we need to subtract.

Note that the last expression is valid *whether or not the events are mutually exclusive (disjoint)*. That is the purpose of the **and** term, since it subtracts the probability that both the events occur at the same time.

How to calculate the probability of C **and** D ie $P(C \cap D)$? Well, if the events are independent (i.e. do not depend on the other event occurring), then

$$P(C \cap D) = P(C)P(D),$$

i.e. just **the product** of the probability of the two events.

Example

Someone has a bag of M&Ms in which there are red, blue, green and orange colours. M&Ms are picked out and replaced. Someone did this 1000 times and obtained the following results:

number of blue M&Ms: 300, number of red M&Ms: 200, number of green M&Ms: 450, Number of orange M&Ms: 50.

1. What is the probability of picking a green M&M?
2. If there are 100 sweets in the bag, how many of them are likely to be green?

Solution

Click below to see the solution.

1. For every 1000 M&Ms picked out, 450 are green. Therefore P(green) = 450/1000 = 0.45.
2. The experiment suggests that 450 out of 1000 M&Ms are green. Therefore, out of 100, we would expect that 45 will be green (using ratios).

Example

All human blood can be typed as O, A, B or AB. The frequency of occurance varies dependent on group. The probabilities for the different human blood types O, B and AB in the US are

Blood Type	Probability
O	0.44
A	?
B	0.1
AB	0.04

A person in the United States is chosen at random. What is the probability of the person having blood type A?

Solution

Click below to see the solution.

Since the four blood types O, A, B, and AB should sum to 1 we simply need to sum O, B, and AB together and the probability of type A must be what is remaining.

This example problem originates from this site.

In [6]:
```
p_O = 0.44
p_B = 0.1
p_AB =0.04
p_A = 1. - (p_O+p_B+p_AB)

print('The probability of a person at random having blood type A is {:.2f}'.format(p_A))
```

```
The probability of a person at random having blood type A is 0.42
```

Example

What is the probability that a randomly chosen person does not have blood type O?

Solution

Click below to see the solution.

- So we need to find out what is the the probability that a randomly chosen person does not have blood type O? Ie we need to calculate: $P(\text{not}O)$ or $P(O^c)$.

In [7]:
```
p_not_O = 1-p_O
print('The probability of a person at random not being able to donate blood to anyone is {:.2f}'.format(p_not_O))
```

```
The probability of a person at random not being able to donate blood to anyone is 0.56
```

- From the information given, we know that being a potential donor for a person with blood type A means having blood type A **or** O.

We therefore need to find P(A or O). Since the events A and O are disjoint, we can use the addition rule for disjoint events to get:

$$P(A \cup O) = P(A) + P(O)$$

In [8]:
```
p_A_or_O = p_A + p_O
print('The probability of a person at random being able to donate to someone with type A is {:.2f}'.format(p_A_or_O))
```

```
The probability of a person at random being able to donate to someone with type A is 0.86
```

Conditional Probabilities

We now have enough knowledge of the probability basics to consider *conditional probabilities*. Technically speaking, all probabilities are conditional. For example, the probability that my coin will land either heads or tails is conditioned by the probability that the coin will land. Or indeed have a heads and a tails! In a less contrived example, the probability that an astronomer is observing a specific type of star, say an A2 star, is first conditioned on the probability that what they are observing is indeed a star, and not another astronomical phenomenon.

So conditional probabilities are important. They are denoted in the following way: **the probability of event A given condition (or event) B is $P(A|B)$.**

Take a classic example: what is the probability of randomly drawing the Queen of Spades (QoS) from a well-shuffled, true pack of cards?

There are 52 cards in total, so the probability of drawing any card (C) is simply $P(C) = 1/52$. The probability of drawing our desired QoS is then $P(QoS) = 1/52$. Now consider that the dealer is truthful, and tells you that the card you have just drawn is a face (F) card. Now what is the probability $P(QoS)$?

Well, first, the probability of drawing a face card that's also the QoS is given by $P(QoS \cap F)$. But since we know that the card is a face card, our probabilities for $P(QoS)$ and $P(F)$ are wrong in the sense that they were derived by dividing by all cards - 1/52, or 12/52, since there are 12 face cards in the pack. Instead we want to renormalise our probabilities to the region of outcome space where F is true, so we need to divide by $P(F)$.

Bayes Theorem

Bayes' theorem, named after 18th-century British mathematician Thomas Bayes, is a mathematical formula for determining conditional probabilities. This theorem has huge importance in the field of data science. It is used in finance to rate the risk of lending money to potential borrowers. It can be used to determine the accuracy of medical test results by accounting for both the probability of a person having the disease and the accuracy of the test itself. The famous Bayes Theorem is derived from considering the idea of conditional probabilities introduced above.

Imagine for instance that someone has a cough and we want to know if this means they have lung disease. Let's say you know:

- the probability of somebody having a cough *given* that they have lung disease X ie P**(cough|lung disease)**,
- the probability of somebody in general having lung disease X ie P**(lung disease)**,
- the probability of somebody in general having a cough ie P**(cough)**.

With these 3 pieces of information you should be able to calculate the probability of somebody having lung disease X *given* that they have a cough ie P**(lung disease|cough)** - but how?

Standard form of Bayes Theorem

Starting from the equations above, given that $P(A \cap B) = P(B \cap A)$ and two independent events, then we can write

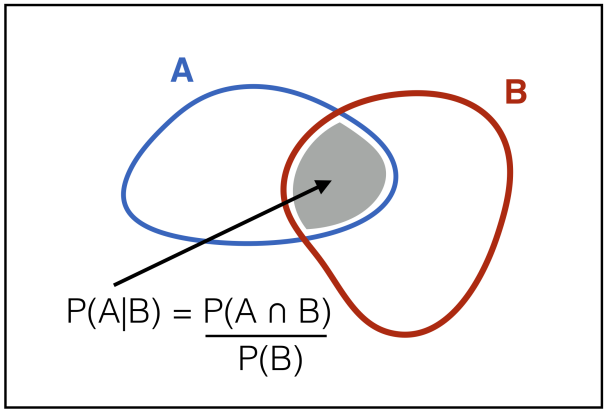
$$P(A|B)P(B) = P(B|A)P(A)$$

such that,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

which is the standard form of Bayes Theorem.

A simple way to visualise this result is to consider the areas in the figure below.



We can think of the denominator as re-normalising the probability into a section of probability space in which B has occured.

One can generalise the denominator by considering that,

$$P(B) = P(B \cap A) + P(B \cap A^c) = P(B|A)P(A) + P(B|A^c)P(A^c)$$

to give the result

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)}$$

Bayes with Models and Datasets

Bayes Rule is normally used to determine the probability of a specific model, θ, given some data D, such that

$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$ where

where $P(D|\theta)$ is the *likelihood*, $P(\theta)$ is the *prior*, and $P(D)$ is the *evidence*. $P(\theta|D)$ is the *posterior*.

The standard way to write Bayes Rule is then,

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Let's define the terms:

- $P(\theta|D)$ the posterior - the probability of model parameter θ being true, given the data
- $P(D|\theta)$ the likelihood - given model parameter θ what is likelihood of obtaining the data
- $P(\theta)$ the prior - the probability of the model parameter θ being 'true'
- $P(D)$ the evidence - the probability of getting the data, give all possible model parameter values (θ and others!)

Why is Bayes so powerful? Here's an example: if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer.

Priors

People feel uncomfortable about 'priors', since often they are a 'best guess'. Indeed, different analysts may have differing opinions about what the prior for a given experiment should be. Although 'frequentists' disagree with the use of priors, note that technically they do assume one: they assume that all is equally likely (i.e. a 'flat' prior).

Clearly this is also wrong. So priors are useful — but they must be clearly stated. They provide a formal means for the analyst to include previous information that is relevant to the experiment. They also allow you test whether the model is good.

Where we have new data in an experiment, we can reapply Bayes Rule to get a new posterior. But what to use for the prior? Well, the posterior from the previous analysis! Hence the statement:

Yesterdays posterior is tomorrow's prior

This feedback loop makes Bayes Theorem particularly useful in machine learning, in which decision making needs to adapt to new information as it comes in.

Example

Imagine that a box contains five coins, one of which is a joke (J) coin, with heads on both sides. A coin is selected at random from the box, and flipped 3 times. The result is 3 heads (3H). What is probability that the coin is the trick coin?

Solution

Click below to see the Solution.

First, we should define what we are trying to work out.

We are interested in $P(J|3H)$. We will let the normal coin be denoted by C. So using Bayes Theorem we can write,

$$P(J|3H) = \frac{P(3H|J)P(J)}{P(3H)}$$

To get the probability of $P(3H)$ we need to add up all possibilities of getting it.

$$P(J|3H) = \frac{P(3H|J)P(J)}{P(3H|J)P(J) + P(3H|C)P(C)}$$

The probability of randomly selecting the joke coin is $P(J) = 1/5$.

The probability of not selecting it, is $P(J^c) = 1 - 1/5 = 4/5 = P(C)$.

The probability of getting 3 heads with the joke coin is 1, so

$$P(3H|J) = 1$$

The probability of getting 3 heads with a standard coin is $(1/2) \times (1/2) \times (1/2)$ (remember these are independent events), so

$$P(3H|C) = 1/8$$

$$P(J|3H) = \frac{1 \times 1/5}{1 \times 1/5 \ + \ 1/8 \times 4/5} = 2/3$$

So there's a 66% chance the coin that we are seeing flipped is the joke coin!

Example

A couple has 2 children and the older child is a boy. If the probabilities of having a boy or a girl are both 50%, what's the probability that the couple has two boys?

We already know that the older child is a boy. The probability of two boys then is equivalent to the probability that the younger child is a boy, which is 50%. Show this is indeed true using Bayes Theorem.

Solution

Click below to see the Solution.

Using Bayes Theory let's evaluate this formally:

Define the events, A and B as follows:

- $A = \text{both children are boys}$
- $B = \text{the older child is a boy}$

So we need to find out

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ where

$P(B) = \frac{1}{2}$ (the probability of the older child being a boy)

$P(A) = \frac{1}{4}$ (probability of both children being boys - this is an **and** probability - $= 0.5 \times 0.5$).

$P(B|A) = 1$ (the probability of the older child being a boy given both children are boys).

In [9]:
```
p_A = 1./4
p_B = 1./2
p_B_given_A = 1.

p_A_given_B = (p_A*p_B_given_A)/p_B

print('Probability that both children are boys given older child is a boy is {:.2f}'.format(p_A_given_B))
```

```
Probability that both children are boys given older child is a boy is 0.50
```

Example

A woman's DNA matches that of a sample found at a crime scene. The chances of a DNA match are just one in two million, the court interprets this as the chance that it came from someone else is 1 in 2 million.

Given the court concludes that the probability that she is guilty $= 1 - \frac{1}{2,000,000}$ which is basically 100%, she gets sent to jail for a very long time.

The women comes from a city where approximately 400,000 women with ages greater than 18 live, approximately 300,000 of which are from a similar ethnic group. Use a simple Bayes theorem approach to show why the justification above for sending her to jail is completely wrong (this is known as the prosecutor's fallacy).

Solution

Click below to see the Solution.

The answer above mistakes the one in two million for the probability of the woman's innocence. In order to assess the woman's guilt properly, we need to need to take the fact that she matched the sample as a given, and see *how much more likely this makes her to be guilty than she was before the DNA evidence came to light*.

But how likely is it that the DNA match profile also exists in the population at large? Who else could there be out there with that profile? The match probability of one in two million tells how likely it is that a random person's DNA profile will match the crime sample not how guilty that person is.

To take the match probability into account we need to calculate the likelihood ie the

$$\text{likelihood } = 2,000,000$$
$$= \frac{\text{Probability of observing the DNA if the defendant is GUILTY}}{\text{Probability of observing the DNA if the defendant is INNOCENT}}$$

This tells us that the 2 million times number simply tells us how more likely we are to observe the evidence if the woman is guilty, than if she is innocent.

Bayes theorem allows us to write

$$\text{Posterior odds of guilt} = \text{likelihood} \times \text{prior}$$

Assuming that this woman is no more likely to be guilty than any other woman in the local area we can estimate that there is a 1 in 300,000 chance that she is guilty - this is then our prior odds.

In [10]:
```
prior = 1./3e5
likelihood = 2e6

answer_odds = likelihood*prior # let's do it in odds to compare with the 1 in 2 million number

print('Odds woman is guilty after seeing the DNA evidence is 1 in {:.0f}'.format(answer_odds))

answer = answer_odds/(answer_odds+1)*100
print('Probability woman is guilty after seeing the DNA evidence is {:.0f}%'.format(answer))
print('So there is a high probability that she is guilty of the crime given DNA evidence but not 100% sure.')
```

```
Odds woman is guilty after seeing the DNA evidence is 1 in 7
Probability woman is guilty after seeing the DNA evidence is 87%
So there is a high probability that she is guilty of the crime given DNA evidence but not 100% sure.
```

Bayesian vs Frequentist:

Bayesian analysis has a somewhat formidable reputation for being extremely difficult... why is that?

- In general, the denominator can be difficult to evaluate
- Tricky integrals
- Often require numerical solutions
- Large (multivariate) parameter space
- In the 20th century, the development of Monte Carlo Markov Chains have made the evaluation of the integrals and the probabilities much easier.

You will learn more about this later in the course!

Bayesian statistician

- Philosophy of science: we do not "rule out" models, just determine their probabilities
- Argument: *"the prior probability is a logical necessity when assessing the probability of a model. It should be stated, and if it is unknown you can just use an uninformative (wide) prior"*

Frequentist statistician

- Philosophy of science: we attempt to "rule out" or falsify models if P(data) given a model is too small.
- Argument: *"setting the prior is subjective - two experimenters could use the same data to come to two different conclusions just by taking different priors"*

To recap: how do we estimate probability?

Our examples of coin flipping, die rolling and card selecting, we introduced the notion that the probability of a particular outcome or event is simply the number of times that event occurs, divided by the number of all possible outcomes.

But say you have a coin, and you want to know $P(H)$ -- how do you proceed? You could guess that the coin is fair and assign 0.5 to outcome heads/tails. But is the coin fair? One way you could test this is to perform lots of experiments (coin flips) and keep track of the outcome. If you do enough of these, eventually you will get an empirical measure,

$$P(H) = \frac{n_H}{n_{\text{flips}}}$$

where n_H is the number of heads that appeared in the experiment and n_{flips} is the number of times you flipped the coin (and counted the result). But when do you stop? Well, that depends on how accurately you want to know $P(H)$. But for now, we will simply note that this type of determination of P is *frequentist*, in that the probability is defined by counting the instances of occurrence.

However, what about the probability that it will rain tomorrow? You can see straight away that such a probability is more difficult to define. In fact, the use of Bayes Theorem, and in particular the prior, introduces the idea that P represents the belief that something will occur.

Now you are ready to tackle the **Chapter 2 quiz** on Learning Central and the Chapter 2 yourturn notebook.

Chapter 3

Experimental Data: Best Estimates and Probability Density Functions

Please ensure you have watched the Chapter 3 video(s).

You will learn the following things in this Chapter

- How to get the best estimate of a variable from data
- Probability Distribution Functions: where experimental data sample from the limiting distribution of a variable and histograms of our data are an approximation of the underlying limiting distribution.
- How to use common distributions to estimate expected values of a parameter as well as probabilities.
- Marginalisation.
- How to use Python programming to do the above.
- After completing this notebook you will be able to attempt CA 1 questions 2 and 3.

Note that this is a long notebook as some of the concepts are revision from GCSE, A Level or year 1 UG level.

The best estimate from data

Imagine we are trying to measure the length of a snake, where length is defined as x. The snake keeps moving around so we know we will have some errors in our measurements. Not only that but our measurements of the snake length will 'jiggle' around the true length of the snake (of which we're trying to make a measurement). We decide to take *10 measurements* and we measure the length x to be:

26, 24, 26, 28, 23, 24, 25, 24, 26, 25

The length of the snake is the **parameter** we want to know, and now we need a **statistic** to estimate the parameter using our data.

The best way to estimate the length would be to simply take the *mean*.

$$\hat{x} = \frac{\sum_i^N x_i}{N}$$

where N is the number of measurements.

What about an error on our mean? We can do this by asking what is the difference between each value we measure and our mean value eg $d = x_i - \hat{x}$? But we have 10 of these estimates and we only want one number for our error, so we need to sum these values and divide by N. *However* this does not account for the fact that d can be positive or negative, so what we really want is a value of the absolute difference between the mean and our individual measurements. To do this we need to square our differences. The error then is the well known standard deviation σ_x,

$$\sigma_x = \sqrt{\frac{\sum_i^N (x_i - \hat{x})^2}{N}},$$

except that we need to take into account the number of degrees of freedom. This is because we've had to use the data to estimate our mean $\hat{x}$ in order to calculate the *sample* standard deviation so it is in fact:

$$\sigma_x = \sqrt{\frac{\sum_i^N (x_i - \hat{x})^2}{N-1}}.$$

Another measurement of error in the mean is the standard error,

$$\text{se} = \frac{\sigma_x}{\sqrt{N}}.$$

Think of this the following way - if we took 10 sets of samples of the snake lengths and calculate a mean in each, then the standard deviation of these means would eventually converge to the standard error.

We can cheat a little and use inbuilt stat functions from `numpy` and `scipy` .

Note that the square of the standard deviation σ_x^2 is the *variance*.

In [14]:
```
import scipy.stats as stats
import numpy as np

x = [26,24,26,28,23,24,25,24,26,25]
n=len(x)

mean = np.mean(x)

standard_dev = np.std(x)  # standard deviation function in numpy

standard_error = stats.sem(x)  # standard error on the mean function in scipy.stats

standard_dev_samp = np.sqrt( np.sum((x-mean)**2.0)/(n-1)) # standard deviation sample N-1

print('The mean length of the snake is is {:.2f} '.format(mean))
print('The standard deviation in the length is {:.2f}'.format(standard_dev) )
print('The error on our estimate of the mean length is {:.2f}'.format(standard_error))
print('The sample standard deviation is {:.2f}'.format(standard_dev_samp) )
```

```
The mean length of the snake is is 25.10
The standard deviation in the length is 1.37
The error on our estimate of the mean length is 0.46
The sample standard deviation is 1.45
```

Bias

In statistics, the bias (or bias function) of an estimator is the difference between this estimator's expected value $E(x)$ and the true value of the parameter being estimated. An estimator or decision rule with zero bias is called unbiased. Or we can write this another way: an estimator is said to be unbiased, if the estimator tends towards the expected value as the sample size (i.e. the number of values / measurements) tends towards infinity.

The mean is an unbiased estimator of $E(x)$, since as the number of points increases, the mean tends towards $E(x)$. This is not true for the standard deviation equation $\sigma_x = \sqrt{\frac{\sum_i^N (x_i - \hat{x})^2}{N}}$ but is true for $\sigma_x = \sqrt{\frac{\sum_i^N (x_i - \hat{x})^2}{N-1}}$.

This is why the degrees of freedom concept is so important.

Suppose we collect a random sample of observations. Now, imagine that we know the mean equals 6.9, but we don't know the value of one of our observations (the `X`) in the list of values below:

```
data = [6,8,5,9,6,8,4,11,7,X]
```

We know that the mean is given by the following equation:

$$\text{mean} = \frac{\text{sum (data)} + \text{X}}{N},$$

and as $N = 10$ this means that the sum(data)+X =69 ie X **has to be equal to 5**.

The last number has no freedom to vary. It is not an independent piece of information because it cannot be any other value. Therefore when we next use the mean to derive a standard deviation, it means we already have lost 1 degree of freedom and we need to account for this $N - 1$ to ensure that the standard deviation of our sample is unbiased. If we do not do this it will be biased away from the true value of the parameter.

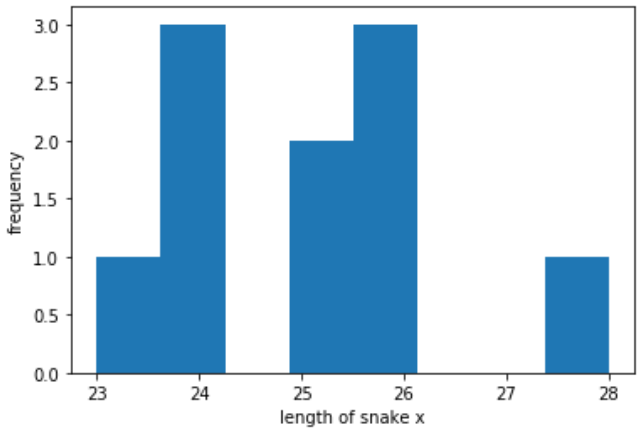
Distributions of data

So let's now take a look at our distribution of data for the length of the snake. Let's take each *different* measured length to be x_k and look at how many times that value was measured.

Measured length of snake	23	24	25	26	27	28
Frequency of how many times that length is measured	1	3	2	3	0	1

If we were to plot this as a bar graph, this would be the histogram.

In [15]:
```
# to make a plot we need to import plotting package
from pylab import plt
# this makes the plot appear in the notebook
%matplotlib inline

# length of snake measurements
x = [26,24,26,28,23,24,25,24,26,25]

# plot the histogram
plt.hist(x,bins=8,histtype='bar')
plt.ylabel('frequency')
plt.xlabel('length of snake x')
plt.show()
```



The mean in this case would be

$$\hat{x} = \frac{\sum_k x_k n_k}{N}$$

where n_k is the number of instances that that the measurement x_k was made. Note that $\sum n_k = N$.

We can look at this another way, each result x_k occurs a certain fraction of times F_k where, out of 10 measurements, we measured a value of 24 three times ie $F_k = n_k/N$. The mean would then be

$$\hat{x} = \sum_k x_k F_k$$

and $\sum_k F_k = 1$.

What if our measurements were not exactly 23 and 24, but were instead 23.6 and 24.3? Our new, more precise, measurements of the length of the snake are

26.4, 23.9, 25.1, 24.6, 22.7, 23.8, 25.1, 23.9, 25.3, 25.4.

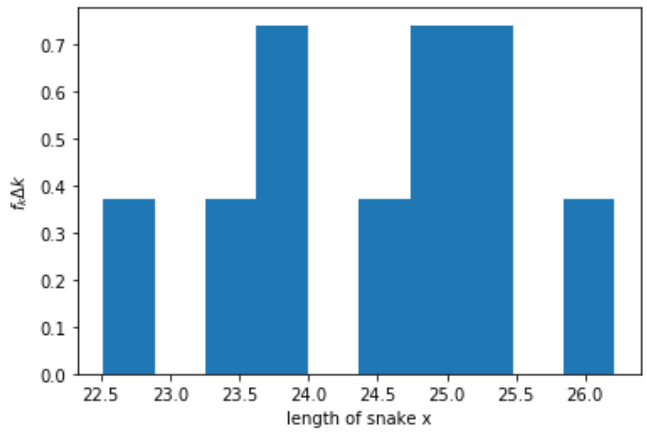
So we now need to think of distributing them in the following way

Bin	22 to 23	23 to 24	24 to 25	25 to 26	26 to 27	27 to 28
Frequency of how many times that length is measured	1	3	1	4	1	0

where the bin width is denoted as Δ_k, and the area of the bin represents the fraction of measurements that fall within the kth bin ie $f_k \Delta_k$.

In [16]:
```
# to make a plot we need to import plotting package
from pylab import plt
# this makes the plot appear in the notebook
%matplotlib inline

# length of snake measurements
x = [26.4,23.9,25.1,24.6,22.7,23.8,25.1,23.9,25.3,25.4]

# get the histogram counts (frequency) and the bin edges from python
```

```
hist, bin_edges = np.histogram(x)
# need to work out size of bins as we want to plot freq x bin width on y axis
binWidth = bin_edges[1] - bin_edges[0]

# plot
plt.bar(bin_edges[:-1], hist*binWidth, binWidth)
plt.ylabel('$f_k \Delta k$')
plt.xlabel('length of snake x')
plt.show()
```



Probability Mass Functions and Probability Density Functions

Up until now, we have only considered the probability of discrete events, such as a coin-flip resulting in heads, or a die turning up a 6. However, probabilities can also be determined for continuous variables, for example, the probability that a child will be a certain height at a given age, or that the intensity in a spectrum will be a given value, or that a molecule will have a given velocity. In this case, the height $h(\mathrm{age})$ or intensity $I(\lambda)$, or velocity v, are all **continuous** variables. These are variables that are uncountable.

Not only are there discrete and continuous variables, but even with discrete data, if we have a lot of trials in our experiments, looking at the data can get tedious. In rolling a six-sided die, there were only six possible outcomes so we could write down the entire probability distribution in a table. Similarly it was rather easy to do it for our snake length measurements above. In many scenarios, the number of outcomes can be much larger and hence a table would be tedious to write down. Worse still, the number of possible outcomes could be infinite, in which case, good luck to anyone writing a table for that!

Now we can start talking about what happens as our number of measurements approaches infinity $N \to \infty$. Two things will happen:

- the distribution of data (our histogram) is said to approach the *limiting distribution* of the variable
- $f_k \to f(x)$ and $\Delta_k \to dx$ so that, whereas before we had $\sum f_k \, \Delta_k = 1$ for all k, now we have

  $\int_{-\infty}^{\infty} f(x) \, dx = 1$

  and so the histrogram of the data approaches the **probability mass function** or the **probability density function** (PDF) of the limiting distribution depending on whether the variable is discrete or continuous.

The **PDF** is used to specify the probability of the random variable falling within a particular range of values, as opposed to taking on any one value. It therefore gives the density of probability rather than the probability mass. The concept is very similar to mass density in physics: its unit is probability per unit length.

Sometimes we don't know the underlying limiting distribution, but we can make a rough one out of our data.

We can now introduce the formal definition of the mean as $N \to \infty$:

$$\hat{x} = \sum_k x_k \, F_k = \sum_k x_k \, f_k \, \Delta_k \to \int_{-\infty}^{\infty} x \, f(x) \, dx$$

and similarly for the standard deviation:

$$\sigma_x = \sqrt{\int_{-\infty}^{\infty} (x - \hat{x})^2 \, f(x) \, dx}$$

where we don't need to worry about N or $N-1$ in the denominators as we're in the regime where $N \to \infty$. We therefore formally define the mean as the expected value of a discrete random variable - the probability-weighted average of all its possible values. In other words, each possible value the random variable can assume is multiplied by its probability of occurring, and the resulting products are summed to produce the expected value.

Because we now have this terminology, we can also define the **cumulant** distribution (CDF) where

$F(x') = \int_{-\infty}^{x'} f(x) \, dx$

where $F(x')$ tells us the percentile that x' represents, eg if $F(x') = 0.6$ then 0.6 or 60% of the area under the function $f(x)$ would lie in the range $\leq x'$. Note that the lower limit could be any bound over which the distribution is valid. The CDF returns the expected probability for observing a value less than or equal to a given value.

The **median** is the special case where $F(x) = 0.5$ - the 50th percentile. Sometimes (depending on the data) the median is a better estimate than the mean.

Finally the Probability Percent Function (PPF) is the inverse of the CDF. and gives the value of the variate for which the cumulative probability has the given value.

- **The distribution of a discrete random variable is characterised by its probability mass function.**
- **The distribution of a continuous random variable is characterised by its probability density function.**

The probability density function is not a probability

The probability mass function (PMF) of a discrete variable θ is a function that gives you, for any real number x, the probability that θ will be equal to x.

However if θ is a continuous variable, its probability density function (PDF) evaluated at a given point x is **not** the probability that θ will be equal to x.

The PMF does not work for continuous random variables because for a continuous random variable $P(\theta = x) = 0$ for all x. Instead, we use the PDF.

Example

Let X be a continuous random variable with the following PDF:

$f_X(x) = ce^{-x}$ if $x \geq 0$ or $f_X(x) = 0$ otherwise.

where c is a positive constant.

1. Find c.
2. Find the CDF, $F_X(x)$ and plot.
3. Find $P(1 < X < 3)$.

Solution

Click below to see the solution.

1. To find c we use the fact that $\int_{-\infty}^{\infty} f(u) \, du = 1$. As it is zero for anything other than $x \geq 0$, the limits become:

   $\int_0^{\infty} ce^{-u} \, du = 1 \to c[-e^{-u}]_0^{\infty} = 1$

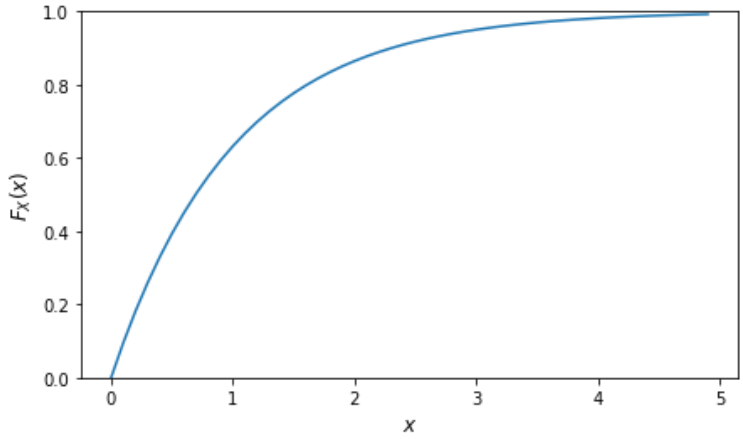
   Therefore $c = 1$.

2. The CDF $F_X(x)$ is given by $F_X(x') = \int_{-\infty}^{x} f(u) \, du$. This is zero unless $x \geq 0$. So this becomes $F_X(x) = \int_0^x e^{-u} du \to 1 - e^{-x}$.

The CDF is then $1 - e^{-x}$ if $x \geq 0$ or $F_X(x) = 0$ otherwise.

In [17]:

```
# define the CDF function from part 2 of question
def cdf_F(k):
    if k < 0:
        val = 0.
    else:
        val = 1-np.exp(-k)
    return val

# Plot CDF by deriving the function for the CDF first and plotting that
x = np.arange(0,5,0.1)
cdf = [cdf_F(item) for item in x]

plt.figure(figsize=(7,4))
plt.plot(x,cdf)
plt.ylim(0,1)
plt.xlabel('$x$',fontsize=12)
plt.ylabel('$F_X(x)$',fontsize=12)
plt.show()
```



Note that if X had been a discrete variable we could have simply used the `numpy cumsum()` function on the outputs of $f_X(x)$ (the PDF) to get the CDF function.

1. We can calculate $P(1 < X < 3)$ using the PDF or the CDF. Using the PDF we would do the following:

   $$P(1 < X < 3) = \int_1^3 f(u)du \rightarrow \int_1^3 e^{-u}du \rightarrow e^{-1} - e^{-3}.$$

   Using the CDF, we would instead do:

   $P(1 < X < 3) = F_X(3) - F_X(1) = 1 - e^{-3} - (1 - e^{-1}) \rightarrow e^{-1} - e^{-3}$ - they give the same answer.

Question from this site.

Introduction to the idea of Maximum likelihood

Maximum likelihood is the idea is that the best guess for the values of x_0 and σ are those that maximise the probability. Given N observed measurements $x_1, \ldots, x_N$, the best estimates for the mean and standard deviation are those values for which the observed $x_1, \ldots, x_N$ are most likely, or rather where $p_{\hat{x},\sigma}(x_1, \ldots, x_N)$ is the maximum.

Why do statitiscians use maximum likelihood and not maximum probability?

The likelihood is formally written as

$$\mathcal{L}(\mu, \sigma|\text{data})$$

where this is the likelihood that the parameters μ and σ (mean and std) take on certain values given the data we've observed.

However probability is usually written as

$P(\text{data}|\mu,\sigma)$

ie the probability (or probability density) of observing data with model parameters μ and σ.

One is asking about the data and the other (maximum likelihood) is asking about the parameter values.

The Normal Distribution

The central limit theorem states that if a quantity is subject to many **small, but independent, random processes**, the spread of the quantity can be described by a bell-like curve, known as a Gauss function or normal distribution. This is a continuous probability distribution. This distribution is possibly the most important distribution in probability theory as it describes natural phenomena such as people's weights, heights and so on.

The form of the normal distribution with width σ, for data value x centred on a value of x given by x_0

$$N_{x_0,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-x_0)^2/2\sigma^2}.$$

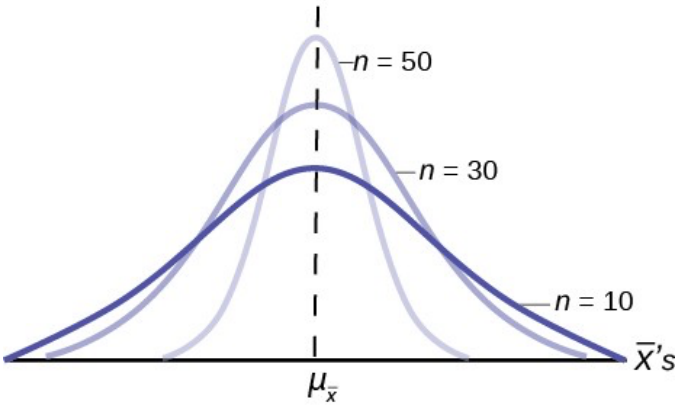
We can show that the mean of the normal distribution is simply

$$\hat{x} = x_0$$

and the standard deviation of the data that follow a normal distribution is

$$\sigma_x = \sigma.$$

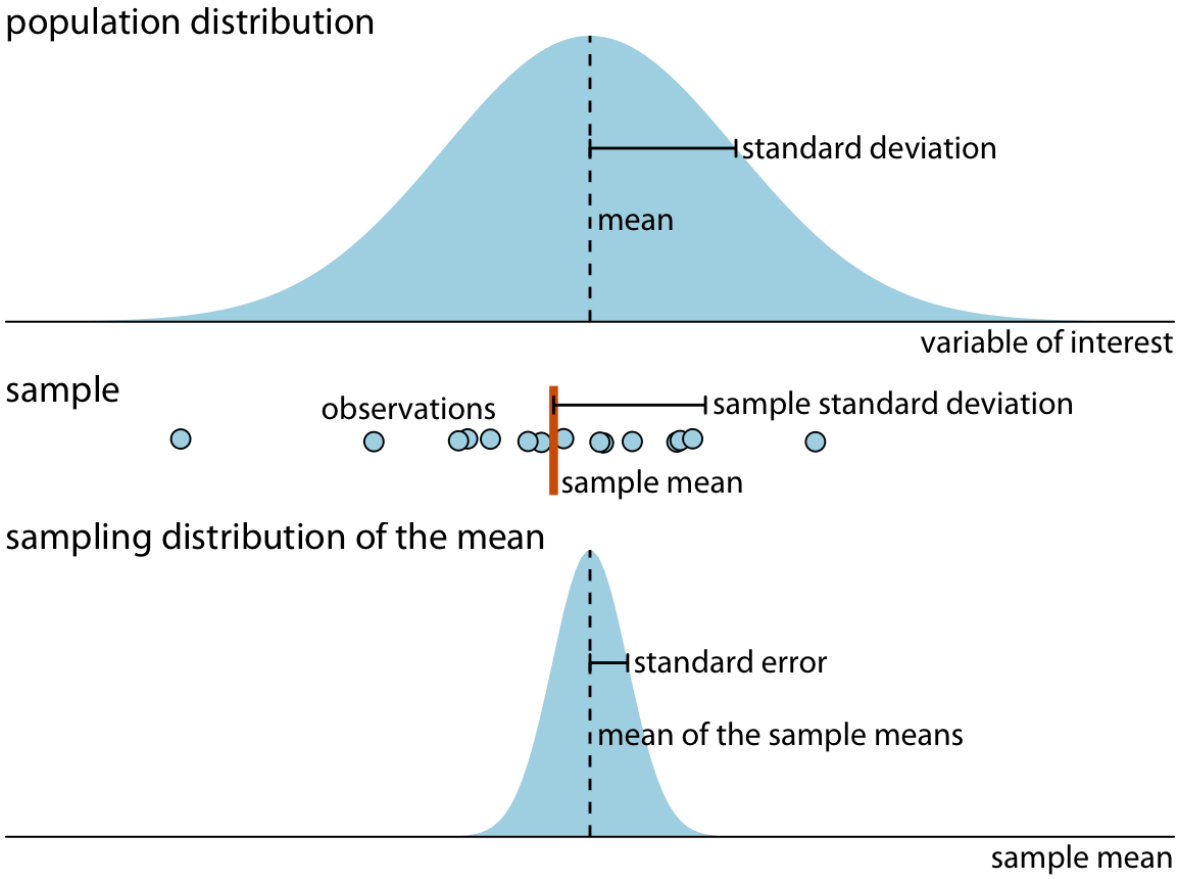
The central limit theorem means that as we get more and more data of say, our snake's length, then the averages of random variables behave like normally-distributed random variables regardless of their limiting distribution. Practically, a sample size of 30 or more is considered large.



An important comment: the mean itself is a random variable. This is because our averages are based on a sample and it is impossible to sample every snake in the world for example. Every time we take a new sample of measurements, our average will be different from our previous ones.

This is where something like the normal distribution becomes very important — if we take a lot of samples, then the mean of our sample means would itself look like a normally distributed random variable.

The figure below (taken from here) shows what we mean by sampling statistics and population. The snake length has a true distribution (population) with population means and standard deviations. Our sample of experimental data has a sample mean and a sample standard deviation which differ from the population parameters. If we keep taking data and recalculate the mean, they would be distributed also as a normal. The standard error of this distribution informs us how precisely we are measuring the true snake length (the population mean).



What does this mean? Well, even for relatively small samples we can make some inferences about the populations (eg the length of snakes) using the normal distribution and our sample statistics.

Derivation

How did we get the above result?

We first have to normalise the function to use it as a probability, such that it satisfies,

$\int_{-\infty}^{\infty} N(x)dx = 1$

We introduce a constant C, such that,

$N(x) = C\, e^{-(x-x_0)^2/2\sigma^2}.$

Note that C only serves to the move the curve up an down in y, but leaves the shape and centring undisturbed; it obviously changes the area under the curve though, which is the whole point in the normalisation. To evaluate the integral, we make a change of variable, by setting $(x - x_0)/\sigma = z$, such that $dx = \sigma dz$ to get,

$\int_{-\infty}^{\infty} N(z)dz = C\sigma \int_{-\infty}^{\infty} e^{-z^2/2}dz$

This is a standard integral in physics, and has the result,

$\int_{-\infty}^{\infty} e^{-z^2/2}dz = \sqrt{2\pi},$

which yields the value for the normalisation $C = 1/\sigma\sqrt{2\pi}$. We can then write the final form for the normal distribution as,

$$N_{x_0,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-x_0)^2/2\sigma^2}$$

The know that the mean of a PDF is

$$\hat{x} = \int_{-\infty}^{\infty} x\, N_{x_0,\sigma}(x)dx = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} x\, e^{-(x-x_0)^2/2\sigma^2}dx.$$

Again, this can be evaluated with a change of variables, replacing $x - x_0 = y$, such that $dx = dy$ and $x = y + x_0$. This results in,

$$\hat{x} = \frac{1}{\sigma\sqrt{2\pi}}\left(\int_{-\infty}^{\infty} y\, e^{-y^2/2\sigma^2} dy \; + \; x_0 \int_{-\infty}^{\infty} e^{-y^2/2\sigma^2} dy\right).$$

The first integral is zero, since although the exponential term is symmetric about $y = 0$, the y is not, and so the points from $-y$ are exactly cancelled by those from $+y$. The second integral is the same as that we seen above, and is just $\sigma\sqrt{2\pi}$, which cancels with the term at the front, leaving us with,

$$\hat{x} = x_0.$$

Similarly for the standard deviation:

$$\sigma_x^2 = \int_{-\infty}^{\infty} (x - \hat{x})^2 N_{x_0,\sigma}(x) dx.$$

By noting that $\hat{x} = x_0$, and then by making the substitutions $x - x_0 = y$, and $y/\sigma = z$ gives,

$$\sigma_x^2 = \sigma^2.$$

Probabilities from Normal distributions

Since $N_{x_0,\sigma}(x)$ is a PDF, the probability of x lying in the range a to b is then given by,

$$\int_a^b N_{x_0,\sigma}(x) dx.$$

So what about the probability of lying within $\pm t\sigma$ eg 1, 2 or 3σ, where t is some real (positive) number? This is given by,

$$P(\text{within t}\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{\text{x}_0-\text{t}\sigma}^{\text{x}_0+\text{t}\sigma} \text{e}^{-(\text{x}-\text{x}_0)^2/2\sigma^2} \text{dx}.$$

Once again, substitution of $(x - x_0)/\sigma = z$, with $dx = \sigma dz$ and now limits of $-t$ to t, we have,

$$P(\text{within t}\sigma) = \frac{1}{\sqrt{2\pi}} \int_{-\text{t}}^{+\text{t}} \text{e}^{-z^2/2} \text{dz}.$$

The equation above is known as the *error function*. Unfortunately, it can not be evaluated analytically, however using a computer, it is possible to obtain values for the integral as a function of t, and we can use reference tables to look this up.

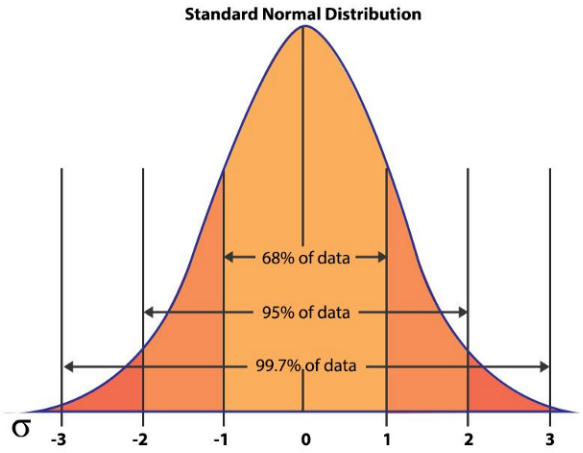
t	0.25	0.5	0.75	1.0	1.5	2.0	2.5	3.0	3.5	4.0
P within $t\sigma$	0.2	0.38	0.55	0.68	0.87	0.954	0.988	0.997	0.9995	0.9999

We can use this information for data that is independent and random, to state our best esimtate of the value as the mean and an error on that value as $X\sigma$ with an associated probability. For example, the probability that a measurement lies within 1σ of the mean is 68%.

$$P(\mu - 1\sigma < X < \mu + 1\sigma) \sim 0.6827$$

$$P(\mu - 2\sigma < X < \mu + 2\sigma) \sim 0.9545$$

$$P(\mu - 3\sigma < X < \mu + 3\sigma) \sim 0.9973$$



Normal distribution figures are taken from here.

A good reference for other distributions that are commonly used in data analysis can be found here.

Example

You order pizza from your favourite place and want to know how long it will take. The restaurant states it has an average (mean) delivery time of 30 minutes with a standard deviation of 5 minutes. Estimate the range of delivery times for 68% of the deliveries. Do the same for 95 and 99.7%.

Solution

Click below for the solution.

So delivery times are on average 30 ± 5 minutes. Now how to match that to a probability?

Using the table above, we can immediately determine that:

- 68% of the delivery times will be between 25-35 minutes (30 ± 5)
- 95% are between 20-40 minutes $(30 \pm (2 \times 5))$
- 99.7% are between 15-45 minutes $(30 \pm (3 \times 5))$.

Example

At a facility that manufactures electrical resistors, a statistical sample of 1 kΩ resistors is pulled from the production line. The resistor's resistances are measured and recorded. A mean resistance of 979.8 kΩ and a standard deviation of 73.10 kΩ represents the sample of resistors. The desired resistance tolerance for the 1-kΩ resistors is $\pm$ 10%. This tolerance range means the acceptable range of resistance is 900 to 1100 Ω.

Assuming a normal distribution, show the probability that a resistor picked off the production line is within the desired tolerance on a plot.

Solution

Click below for the solution.

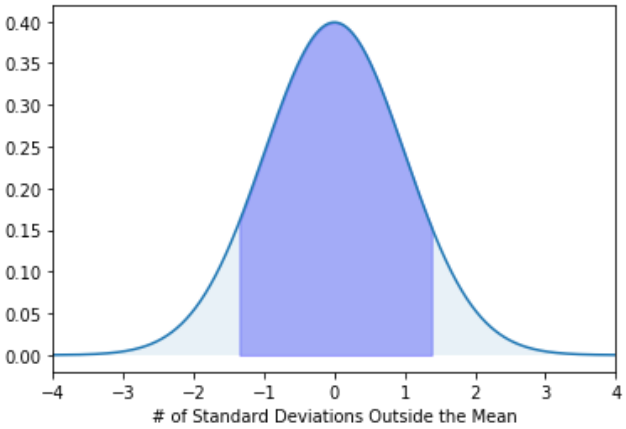
As we are interested in the probability, we use the equation above

$$P(\text{within t}\sigma) = \frac{1}{\sqrt{2\pi}} \int_{-\text{t}}^{+\text{t}} \text{e}^{-\text{z}^2/2}\text{dz}.$$

In [18]:

```
import numpy as np
import pylab as plt
# install packaged to do a normal function
from scipy.stats import norm

%matplotlib inline

# define constants
mu = 998.8
sigma = 73.10

# acceptable values of resistance
x1 = 900
x2 = 1100

# calculate the z-transform x - x_o /sigma
# this will move x axis to +/- sigmas rather than resistance
z1 = ( x1 - mu ) / sigma
z2 = ( x2 - mu ) / sigma

# range of x in spec
x = np.arange(z1, z2, 0.001)

# plot distribution for +/1 10 sigma as comparison
x_all = np.arange(-10, 10, 0.001)

# mean = 0, stddev = 1, since Z-transform was calculated
y = norm.pdf(x,0,1)
y2 = norm.pdf(x_all,0,1)
```

```
# build the plot
plt.plot(x_all,y2)

# this shows dark shaded region for acceptable values of resistor
plt.fill_between(x,y,0, alpha=0.3, color='b')
# this gills in the gap between the PDF and the accepted values
plt.fill_between(x_all,y2,0, alpha=0.1)
plt.xlim([-4,4])
plt.xlabel('# of Standard Deviations Outside the Mean')
plt.show()
```



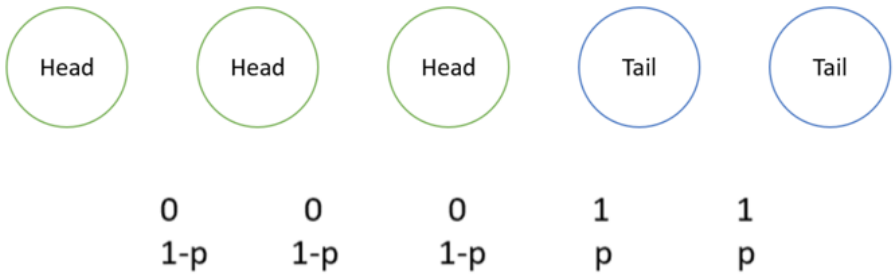
Notice how the area corresponding to resistors in the given specification (between the upper and lower bounds) is shaded.

Bernoulli Distribution

A Bernoulli event is one in which the outcomes are of the yes/no variety, such as, did the coin land heads? Did the patient survive 3 years after treatment? Due to the widely applicable nature of this type of event, Bernoulli distributions are a common feature of statistics and data analysis. This is a discrete probability distribution.

Let's flip a coin 5 times and see how many tails we get. Let's denote and a success (tail) as 1, and a fail (head) as 0. Each trial has probability 0.5 of success and 0.5 of failure. We might be interested in knowing what is the probability of obtaining a given number of successes. How shall we proceed?

The figure below shows us an example of the outcomes from 1 trial of flipping the coin 5 times and got $\Omega = H, H, H, T, T$. If we assign 0s and 1s then we have the following:



So in this case the probability of getting the above outcomes would be (if independent) $p^2(1-p)^3$ or rather $p^\nu(1-p)^{N-\nu}$ where N is number of coins flipped and ν is the number of successes.

If our probability of success in one trial is represented by θ, we can then write the probability of success in p given θ as,

$p(x|\theta) = \theta^x(1-\theta)^{(1-x)}$.

where $x = 0$ for failure, or 1 for success. Eg for rolling a six with a fair dice, $x = 1$ for rolling a 6, and 0 for anything else. In this case the probability for rolling a six in one trial θ would be 1/6.

We can write the probability of obtaining any particular sequence of successes for $\nu =$ number of successes

$p(\nu|N,\theta) = \theta^{\nu}(1-\theta)^{(N-\nu)}$

Example

M&M's created in the US have 50% red sweets, whereas those created in the UK only have 30% red M&Ms. You draw 5 M&Ms from an unlabeled bag and find that 2 are red. Use maximum likelihoods to find out which country your M&Ms came from.

Solution

Click below to see the solution.

Probability distribution is a Bernoulli with successes 2, trials 5. The probability of getting a red M&M in one trial is $\theta = 0.3$ for the UK and 0.5 for the US. Likelihood can be written as:

$\mathcal{L}(p|x) = \theta^{\nu}(1-\theta)^{N-\nu}$

In [19]:

```
import math
theta = [0.3,0.5] #prob red M&Ms in UK, prob red M&Ms in US
N=5  # No of trials
nu=2 # No of successes

lik = [x**nu * (1-x)**(N-nu) for x in theta]

print('likelihood for theta=0.3 and theta=0.5 is {:.5f} and {:.5f}'.format(lik[0],lik[1]))
```

```
likelihood for theta=0.3 and theta=0.5 is 0.03087 and 0.03125
```

$\mathcal{L}(0.5|x) > \mathcal{L}(0.3|x)$ therefore more likely bag came from the US.

The Binomial Distribution

Binomial distribution is a discrete probability distribution like Bernoulli. It can be used to obtain the number of successes from N Bernoulli trials.

Let's consider rolling 3 dice at the same time. What is the probability of getting ν sixes, where now $\nu = \{0,1,2,3\}$? First, consider $\nu = 0$,

$p(\text{not 6, not 6, not 6}) = \left(\frac{5}{6}\right)^3$ and then consider $\nu = 3$,

$p(6,\ 6,\ 6) = \left(\frac{1}{6}\right)^3$.

These were the most straightforward as there is only 1 way in which they can occur. But now let's consider ($\nu = 1$). This can occur in 3 ways:

$$p(\text{one 6 in 3}) = p(6, \text{not 6, not 6}) + p(\text{not 6}, 6, \text{not 6}) + p(\text{not 6, not 6},\ 6) \tag{1}$$

$$= 3\left(\frac{1}{6}\right)\left(\frac{5}{6}\right)^2. \tag{2}$$

Similarly for $\nu = 2$:

$$p(\text{two 6 in 3}) = p(6, 6, \text{not 6}) + p(6, \text{not 6},\ 6) + p(\text{not 6}, 6, 6) \tag{3}$$

$$= 3\left(\frac{1}{6}\right)^2\left(\frac{5}{6}\right). \tag{4}$$

The coefficients that sit in front of the θ and $\theta - 1$ terms are given by the Binomial Coefficient,

$$\binom{N}{\nu} = \frac{N(N-1)\cdots(N-\nu+1)}{1\times 2\times\cdots\times\nu} \tag{5}$$

$$= \frac{N!}{\nu!(N-\nu)!} \tag{6}$$

The *Binomial Distribution* is therefore,

$B_{N,\theta}(\nu) = \binom{N}{\nu}\theta^{\nu}(1-\theta)^{(N-\nu)}$.

The mean of the Binomial Distribution are given by,

$$\hat{\nu} = \sum \nu B_{N,\theta}(\nu) \tag{7}$$
$$= N\theta, \tag{8}$$

that is, if you repeat the experiment N times, the average number of successes is simply the probability of success in any one trail times the number of trials. The standard deviation is little trickier to evaluate, but is given by,

$\sigma_{\nu} = \sqrt{N\theta(1-\theta)}$.

You can get access to python's inbuilt binomial calculator by using `scipy.stats` . As it is a discrete probability distribution, the python function is called `binom.pmf()` .

The binomial distribution approaches the normal distribution as $N \to \infty$.

Example

Suppose a dice is tossed 5 times. What is the probability of getting exactly 2 fours?

Solution

Click below for the solution.

This is a binomial experiment in which the number of trials is equal to 5, the number of successes is equal to 2, and the probability of success on a single trial is 1/6. The binomial distribution is given by

$B_{N,\theta}(\nu) = \binom{N}{\nu}\theta^{\nu}(1-\theta)^{(N-\nu)}$.

In [20]:

```
from scipy.stats import binom

# set up values
N=5
nu = 2

# prob of single trial
theta = 1./6

prob = binom.pmf(nu,N, theta)

print('probability of getting exactly 2 fours when tossing coin 5 times is {:.2f}'.format(prob))
```

```
probability of getting exactly 2 fours when tossing coin 5 times is 0.16
```

Beta Distributions

The functional family that has the same form as Bernoulli and Binomial distributions are called *beta distributions*. These are suitable for describing the random behavior of percentages and proportions. One example is where a scientists believes that the (unknown) probability of having flu is not fixed and not the same for the entire population, but is another random variable with its own distribution. The probability density function is (usually) denoted by,

$$p(\theta|\ a,b) = \mathrm{beta}(\theta|\mathrm{a},\mathrm{b}) = \frac{\theta^{(\mathrm{a}-1)}(1-\theta)^{(\mathrm{b}-1)}}{\mathrm{B}(\mathrm{a},\mathrm{b})}$$

where a and b are shape paramaters and $B(a,b)$ is the normalisation factor that ensures that the area under the curve integrates to unity,

$B(a,b) = \int_0^1 \theta^{(a-1)}(1-\theta)^{(b-1)}d\theta$.

The mean and variance of the beta distribution are given by,

$$\hat{\theta}_B = \frac{a}{a+b}$$

$$\sigma_\sigma^2 = \frac{\hat{\theta}(1-\hat{\theta})}{a+b+1}.$$

Example

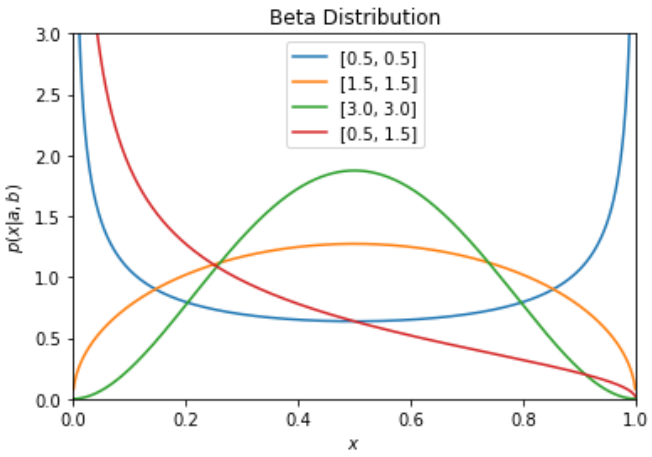
Let's plot some beta distributions for different values of a and b, we can import beta function from `scipy.stats` .

Solution

Click below to see the solution.

In [21]:

```
from scipy.stats import beta

# let's do it for 4 different parameters of a and b

a = [0.5, 1.5, 3.0, 0.5]
b = [0.5, 1.5, 3.0, 1.5]

# list indexing, example [:}] returns all elements and [:-1] returns all except the last one [:-1]
x = np.linspace(0, 1, 1002)[1:-1]

for i in range(0,len(a)):
    dist = beta(a[i],b[i])
    plt.plot(x, dist.pdf(x),label=[a[i],b[i]])
plt.xlim(0, 1)
plt.ylim(0, 3)

plt.xlabel('$x$')
plt.ylabel(r'$p(x|a,b)$')
plt.title('Beta Distribution')
plt.legend()
plt.show()
```



We can see that as a gets larger, the distribution moves to high values of x (ie θ), while as b increases, the distribution moves to lower values of x (ie θ). If a and b get larger together, the beta distribution becomes narrower.

The Poisson Distribution

The Poisson Distribution is the limiting distribution that describes processes that are random but are governed by some underlying mean rate. Examples include monitoring the decay of radioactive materials, or in sociology, the rate of births/deaths over winter and so on. If football were a Poisson process with a mean number of goals per game of, say, 2 then we would expect football games to have 2 ± 1.4 (the square root of 2) goals, i.e. between about 0.6 and 3.4 goals per match. That is actually not far from what is observed and the distribution of goals per game in football matches is actually quite close to a Poisson distribution.

The Poisson Distribution is given by,

$$P_\mu(\nu) = e^{-\mu}\frac{\mu^\nu}{\nu!}$$

where μ is the mean count rate, and $P_\mu(\nu)$ stands for the the probability of ν successes (counts) in a specific interval.

The standard deviation of the Poisson Distribution is given by $\sigma_\nu = \sqrt{\mu}$.

This is neat result: if we measure a given number of counts $\hat{\nu}$ in a given interval, then the uncertainty on our count rate is simply $\sqrt{\hat{\nu}}$.

Note that the Poisson Distribution is *not* symmetric about the mean.

The Poisson distribution approaches the Binomial Distribution when the number of trials is very large (think atoms in a lump of Uranium), but the probability θ is very small (the chances of a single atom decaying in a hour).

The Poisson distribution approaches the normal distribution as $\mu \to \infty$.

Example

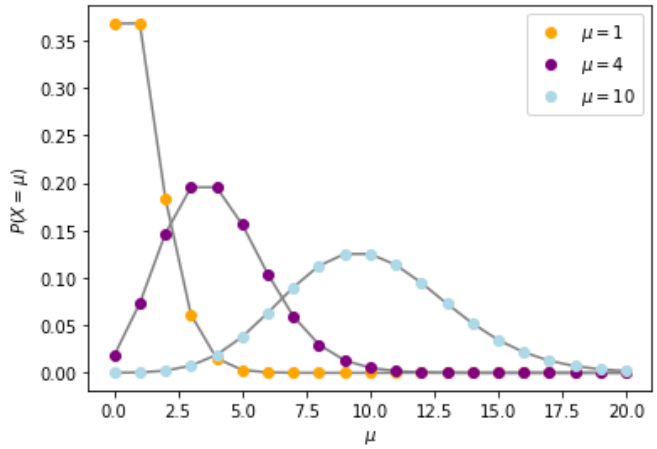
Let's plot some example Poisson distributions. Again we can use `scipy.stats` to help us use Poisson distribution, calling the function `poisson.pmf()` .

Solution

Click below to see the solution.

In [22]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as sp
from scipy.stats import poisson

X = np.arange(0,21)

# let's plot 3 curves with different mu values

col = {1: 'orange', 4: 'purple', 10: 'lightblue'} # define colours for lines
labels = {1: "$\mu=1$", 4: "$\mu=4$", 10: "$\mu=10$"} # define labels for lines

plt.figure(figsize=(6,4.2))

for mu in 1,4,10:
    P = poisson.pmf(X,mu)

    plt.plot(X, P, '-', color='grey')
    a = plt.plot(X, P, 'o', color=col[mu], label = labels[mu])
    plt.legend()

plt.xlabel("$\mu$")
plt.ylabel(r"$P(X=\mu)$")
plt.xlim(-1,21)
plt.show()
```



Example

Suppose that astronomers estimate that large meteorites (above a certain size) hit the earth on average once every 100 years, and that the number of meteorite hits follows a Poisson distribution. What is the probability of zero meteorite hits in the next 100 years?

Solution

Click below for the solution.

Poisson distribution given by

$$P_\mu(\nu) = e^{-\mu}\frac{\mu^\nu}{\nu!}$$

where $\nu = 0$ and $\mu = 1$

$$P(k = 0 \text{ meteorites hit in next 100 years}) = \frac{1^0 e^{-1}}{0!} = \frac{1}{e}$$

In [23]:
```
import numpy as np

mu = 1 # mean = once every 100 years
nu = 0 # number of successes

prob = (np.exp(-mu) * mu**nu) / 1.
print('The probability that no large meteorites hit the earth in the next 100 years is {:.2f}.'.format(prob))
```

```
The probability that no large meteorites hit the earth in the next 100 years is 0.37.
```

Exponential Distribution

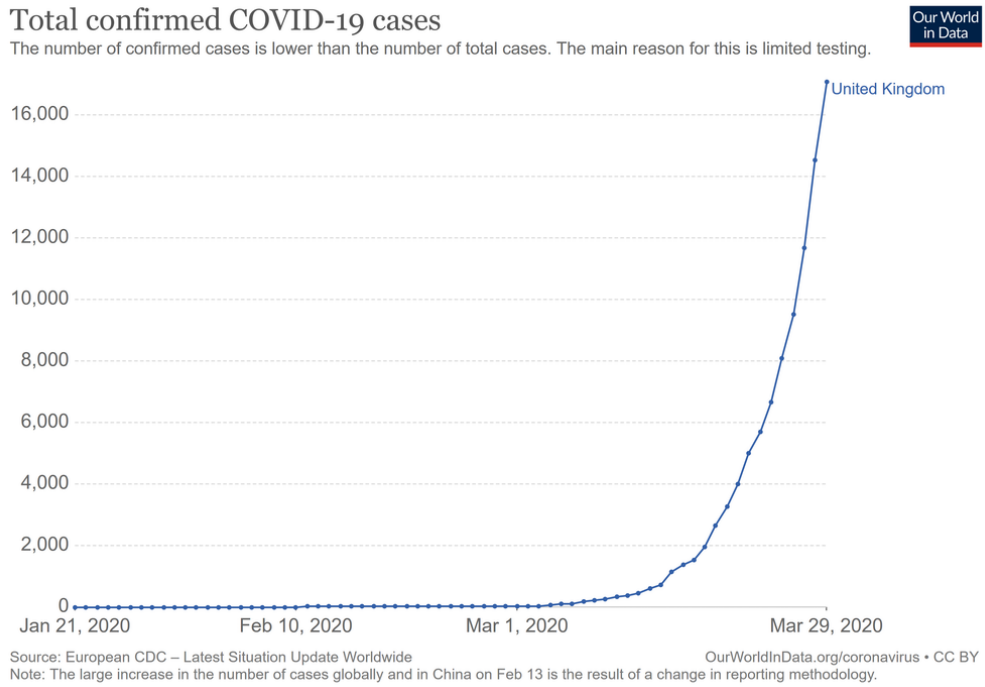
You may have noticed in the news that the exponential functions have been getting a lot of attention recently due to its description of the evolution of infection rates in pandemics.

In probability, an exponential probability distribution is effectively a "waiting time" distribution - how long do we have to wait before the next event ie bus, infection etc?

The form of the PDF for this distribution is $\lambda\exp(-\lambda t)$. In this case λ is the mean number of events over a given interval. We can see then that this function is the probability distribution of the *time between events* in a Poisson point process.

The mean is given by $\frac{1}{\lambda}$ and variance $\frac{1}{\lambda^2}$.

As an aside, this is different to the exponential growth function that has been in the news so much where $y = a\exp^{\lambda t}$. In the COVID-19 pandemic, it was quickly realised that without intervention, the number of new Covid-19 cases would double every three to four days. It is difficult for our brains to understand just how bad an exponential increase like this is as we're so used to thinking linearly.



The figure above shows the total number of confirmed cases of COVID-19 in the UK increasing exponentially during early 2020. *Credit: Max Roser, Hannah Ritchie and Esteban Ortiz-Ospina (2020) - 'Coronavirus Disease (COVID-19) – Statistics and Research'*

To see the real danger in an exponential infection rate, imagine if a 1000 people are infected on day 1 and then the number of infected people doubles every 2 days. How many would be infected in 60 days?

In [24]:
```
def exp_growth(a,l,t):
    return a*np.exp(l*t)

# doubling time is 2 days
t_d = 2.
# want to know time in 60 days
t = 60.

y_0 = 1. # initially (on day 0) only this many infected = a
y_2 = 2.   # in 3 days the rate doubles

# need to work out lambda using the fact we know that y = 2 for a time of 2 days
l = (np.log(y_2/y_0))/t_d

# now how many in 60 days?
y_60 = exp_growth(y_0,l,t)

print('The number infected after 60 days is {:.1f}'.format(y_60/1e6)+' million people')
```

```
The number infected after 60 days is 1073.7 million people
```

Example

Let's say that on average, the number of shoppers arriving in store are 50 customers per hour. The mean interval between events (one customer to another) is 0.83 customers per minute.

What is the probability that a customer will come into the store in the next minute?

Solution

Click below for the solution.

To do this we need to integrate the PDF (to get the CDF - remember PDF not in probability units but in probability density units) and plug in the values of 1 minute and $\lambda = 0.83$.

$$P(t < 1\text{min}) = \int_0^1 0.83\, e^{-0.83t} dt$$

$= \left[-e^{-0.83t}\right]_0^1$

$= 1 - e^{-0.83t}$

In [25]:

```
# function that calculates probability for exponential from the CDF (integral of PDF)
def cdf_exp(l,t):
    return 1-np.exp(-l*t)

t = 1.0 # time
l = 0.83 # mean rate of customers per minute

print('Probability that customer arrives in store in next minute is {:.2f}'\
      .format(cdf_exp(l,t)))
```

```
Probability that customer arrives in store in next minute is 0.56
```

Marginalisation

What do we do if the PDF is a function of more than one variable?

There may exist a function $p(x, y)$ that yields the probability of both x and y. This type of function may even be a combination of discrete and continuous variables and will have units of $1/xy$. But suppose we wanted to know just $p(x)$. How do we calculate this, given that we only have $p(x, y)$? If you consider the units of the problem, the answer becomes clear: we just need to integrate or sum over y, i.e.

$p(x) = \int p(x, y)dy$

if y is a continuous variable, or

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

if y is discrete. This process is called *marginalising over y*. We could do the same to get $p(y)$. Marginalisation is extremely important, since it allows us to deal with nuisance parameters, that is, those that we don't know very well (or that are not well constrained).

Example

Scientists have measured someone's happiness and also recorded the weather for in England, Wales and Scotland. We want to know if weather affects happiness.

We can write this mathematically as $P(\text{happiness}|\text{weather})$ but what we've measured is the following

$P(\text{happiness}, \text{country} \mid \text{weather})$

ie we can't simply just get rid of the fact that the happiness will be a function of both nationality and weather. How do we approach this?

Solution

Click below to see the solution.

Marginalisation tells us that we can calculate the quantity we want if we simply sum over all possibilities of countries ie

$P(\mbox{happiness} | \mbox{weather}) = P(\mbox{happiness}, \mbox{Wales | weather})

- P(\mbox{happiness}, \mbox{England | weather}) + P(\mbox{happiness}, \mbox{Scotland | weather}) $

Example from here.

Now you are ready to tackle the **Chapter 3 quiz** on Learning Central and the Chapter 3 yourturn notebook.

Chapter 4

Errors, Correlations and Hypothesis Testing

Please ensure you have watched the Chapter 4 video(s).

You will learn the following things in this Chapter

- How to combine (with weighting) measurements based on their errors.
- Covariance and correlation.
- Correlation statistics.
- How to interpret a correlation statistic.
- Classic hypothesis testing and confidence intervals.
- How to interpret a confidence interval.
- How to use Python programming to do the above.
- After completing this notebook you will be able to attempt CA 1 questions 2 and 3.

Revision of Errors

If the measurement of a particular quantity is subject to many, independent and random errors, then the *central limit theorem* allows us to use the normal distribution to model the quantity's errors.

Note that there are two types of errors:

- **statistical errors** - from random nature of measurement process, can be reduced by increasing the number of measurements and averaging over them.
- **systematic errors** - these arise from flawed measurements (eg a rogue voltmeter adding +2V to every measurement because its not properly calibrated). This is easy to spot as it remains even after repeating measurements multiple times.

Combining measurements with different errors

Suppose we have two students, let's call them A and B, who make a measurement of the length of our snake, x. Student A finds the length to be $x = x_A \pm \sigma_A$, while student B finds that $x = x_A \pm \sigma_A$. Given that both sets of data are valid estimates of the snake's length, we'd like to combine the results from the two experiments, to get a new, and hopefully improved result x_{AB}, with an associated uncertainty σ_{AB}.

How to proceed? It is tempting to simply average the two results, e.g. $x_{AB} = \dfrac{x_A + x_B}{2}$, but this feels a bit fishy if the two uncertainties σ_A and σ_B are not equal. Why should they have equal weighting, if one is less accurate (higher uncertainty) than the other? The answer is to weight the values according to their uncertainties, to produce a **weighted average**.

$$\hat{x_0} = \frac{w_A x_A + w_B x_B}{w_A + w_B}$$

where $\hat{x_0}$ denotes the weighted average, and $\hat{\sigma}_{x_0}$ is the standard deviation

$$\hat{\sigma}_{x_0} = \frac{1}{\sqrt{\sum w_i}}.$$

w_i denotes the individual weights of each component in the average and $w_A = 1/\sigma_A^2$ and $w_B = 1/\sigma_B^2$.

This type of weighting - also called optimal weighting - is extremely important in data analysis. Optimal weighting allows you to take account of all data points, with each point contributing to the final result in a way than depends on how well you trust the data (i.e. the variance of the point). The problem is, that you need to know something about the error in each point (not always the case).

Derivation: weighted errors

We are going to assume once again that the errors in our measurements are normally distributed, and the two experiments performed by students A and B were completely independent. In that case, the probability that the students would obtain their results is given by,

$$P_{x_0}(x_A) \propto \frac{1}{\sigma_A} e^{-\frac{(x_A - x_0)^2}{2\sigma_A}}$$

for student A and

$$P_{x_0}(x_B) \propto \frac{1}{\sigma_B} e^{-\frac{(x_B - x_0)^2}{2\sigma_B}}$$

for student B. Note that the probabilities depend on the unknown, but true value of the measurement x_0.

So the probability that **both** students found the lengths x_A and x_B is then simply,

$$P_{x_0}(x_A \cap x_B) = P_{x_0}(x_A, x_B)$$

$$= P_{x_0}(x_A) \times P_{x_0}(x_B) \propto \frac{1}{\sigma_A \sigma_B} e^{-\chi^2/2},$$

where we have introduced the notation χ^2 (chi-squared) as a shorthand for,

$$\chi^2 = \left(\frac{x_A - x_0}{\sigma_A}\right)^2 + \left(\frac{x_B - x_0}{\sigma_B}\right)^2.$$

Using the principle of maximum likelihood from Chapter 3, we can see that $P_{x_0}(x_A, x_B)$ has a maximum when χ^2 has a minimum. So we want to know the value of x_0 that would maximise the chances of A finding x_A {\bf and} B finding x_B. To do this, we need to differentiate χ^2 and set the derivative equal to zero,

$$2\frac{x_A - x_0}{\sigma_A} + 2\frac{x_B - x_0}{\sigma_B} = 0$$

The solution for x_0 is then simply,

$$\text{best estimate for } x_0 = \left(\frac{x_A}{\sigma_A^2} + \frac{x_B}{\sigma_B^2}\right) \Big/ \left(\frac{1}{\sigma_A^2} + \frac{1}{\sigma_B^2}\right)$$

If we define weights to have the form $w_A = \frac{1}{\sigma_A}^2$ and $w_B = \frac{1}{\sigma_B}^2$, then we can tidy this up to obtain,

$$\hat{x_0} = \frac{w_A x_A + w_B x_B}{w_A + w_B}$$

where $\hat{x_0}$ denotes the weighted average.

Covariance

During lab experiments, you will have learnt how to identify sources of error in your experiments, and how to propagate these errors through to the final result.

We will briefly discuss the maths underlying the error propagation here, since it provides the background for an important property in statistics: the covariance. Let's first assume that we have a function f that is dependent on some measured quantity x, and yields a value y that we are interested in knowing, such that $y = f(x)$. Now the measurements of x are associated with some random error, σ_x, and so the final value of y will also have an error σ_y. How do we calculate σ_y?

Assuming the errors in x are small, and are close to the true value $\hat{x}$, we can expand $f(x)$ around the point $\hat{x}$ and derive an expression for $y - \hat{y}$ which leads to:

$$\sigma_y^2 = \left(\frac{df}{dx}\right)^2_{\hat{x}} \sigma_x^2,$$

For two variables eg $z = f(x, y)$, we get the following:

$$\sigma_z^2 = \left(\frac{\partial f}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + 2\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\sigma_{xy}.$$

Ignoring the last term on the RHS for a moment, we see that the expression is the normal error propagation formula that you may have seen during your lab work (for independent errors). If σ_x and σ_y are not independent, then we need the last term! This is called the *covariance*.

$$\sigma_{xy} = \frac{1}{N}\sum(x - \hat{x})(y - \hat{y}).$$

In fact, this is called the population covariance.

The variance of a variable describes how much the values are spread. The covariance instead is a measure that tells the amount of dependency between two variables. A positive covariance means that the values of the first variable are large when the values of the second variables are also large. A negative covariance means the opposite: large values from one variable are associated with small values of the other. For truly independent variables σ_{xy} will be zero.

We know from Chapter 3 that our experiments are *sampling* from the underlying true population, so we need to define a sample covariance:

$$\sigma_{xy} = \frac{1}{N-1}\sum(x - \hat{x})(y - \hat{y}).$$

Sometimes it can be difficult to conclude where two sources of error (or two parameters) are indeed correlated – often this is the case when the number of data points is small. So what then? If we assume the errors are independent, can we find a way to get the upper limit on the error, to ensure that we don't miss anything?

Yes, we can use the **Schwarz inequality** to write $|\sigma_{xy}| \leq \sigma_x\sigma_y$ and therefore

$$\sigma_z \leq |\frac{\partial f}{\partial x}|\sigma_x + |\frac{\partial f}{\partial y}|\sigma_y.$$

Problems with covariance

Covariance keeps the scale of the variables x and y, and therefore can take on any value. This makes interpretation difficult and comparing covariances to each other impossible. For example, $\sigma_{XY} = 5.2$ and $\sigma_{ZQ} = 3.1$ tell us that these pairs are positively associated, but it is difficult to tell whether the relationship between X and Y is stronger than Z and Q without looking at the means and distributions of these variables. We can normalise the covariance to give us both direction and strength of the correlation between these parameters.

Derivation error equation and covariance

Let's first assume that we have a function f that is dependent on some measured quantity x, and yields a value y that we are interested in knowing, such that $y = f(x)$. Now the measurements of x are associated with some random error, σ_x, and so the final value of y will also have an error σ_y. How do we calculate σ_y?

Assuming the errors in x are small, and are close to the true value $\hat{x}$, we can expand $f(x)$ around the point $\hat{x}$,

$$f(x) = f(\hat{x}) + (x - \hat{x})\left(\frac{df}{dx}\right)_{\hat{x}} + \cdots$$

If we now identify $\hat{y} = f(\hat{x})$, then we can see that,

$$y - \hat{y} = f(x) - f(\hat{x}) \approx (x - \hat{x})\left(\frac{df}{dx}\right)_{\hat{x}}.$$

which gives us an expression for how the value of y derived from our measured value of x, relates to the true values of both y and x, which are given by $\hat{y}$ and $\hat{x}$. If we then take many measurements of x, we can use the expression above to write the standard deviation about the mean, as

$$\frac{1}{N}\sum_i^N (y_i - \hat{y})^2 = \left(\frac{df}{dx}\right)_{\hat{x}}^2 \frac{1}{N}\sum_i^N (x_i - \hat{x})^2$$

or simply,

$$\sigma_y^2 = \left(\frac{df}{dx}\right)_{\hat{x}}^2 \sigma_x^2.$$

For two variables x and y where $z = f(x, y)$, we expand our function f around the true values of $\hat{x}$ and $\hat{y}$ using Taylor's expansion, to get,

$$\hat{z} = \hat{f}(x, y) = f(\hat{x}, \hat{y}) + \left(\frac{\partial f}{\partial x}\right)_{\hat{x}} (x - \hat{x}) + \left(\frac{\partial f}{\partial y}\right)_{\hat{y}} (y - \hat{y}) + \cdots$$

Now write the bracket term in the variance for z:

$$(z - \hat{z})^2 = (f(x, y) - f(\hat{x}, \hat{y}))^2$$

$$(z - \hat{z})^{\approx} \left(\frac{\partial f}{\partial x}\right)^2 (x - \hat{x})^2 + \left(\frac{\partial f}{\partial y}\right)^2 (y - \hat{y})^2 + 2\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}(x - \hat{x})(y - \hat{y}),$$

which then gives us the result that,

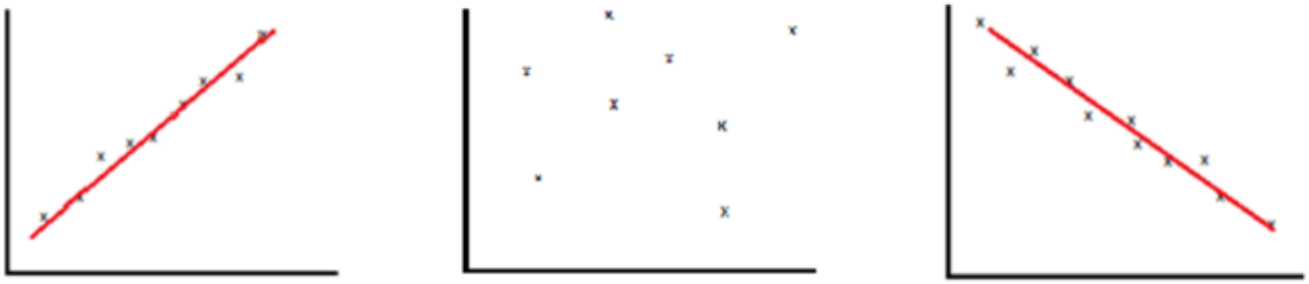
$$\sigma_z^2 = \left(\frac{\partial f}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + 2\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\sigma_{xy}.$$

$$\sigma_z^2 = \left(\frac{\partial f}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + 2\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\sigma_{xy}.$$

Correlation

So how do we measure the degree of correlation (the association between the observed values of two variables) in data? In almost any business, or modelling that happens, it is useful to express one quantity in terms of its relationship with others. For example, sales might increase when the marketing department spends more on TV advertisements. Often, correlation is the first step to understanding relationships between variables and subsequently building better business and statistical models.

Two variables may have a positive association, so that as the values for one variable increase, so do the values of the other variable. Alternatively, the association could be negative or neutral. Correlation quantifies this association, often as a measure between the values -1 to 1 for perfectly negatively correlated and perfectly positively correlated. The calculated correlation is referred to as the "correlation coefficient." This correlation coefficient can then be interpreted to describe the measures.



For a linear function, the extent to which data points $(x_1, y_1) \ldots (x_N, y_N)$ support a linear correlation is given by the *linear correlation coefficient* sometimes called the Pearson correlation coefficient,

$$r = \frac{\sigma_{xy}}{\sigma_x \, \sigma_y}$$

$$r = \frac{\sum(x - \hat{x})(y - \hat{y})}{\sqrt{\sum(x - \hat{x})^2 \sum(y - \hat{y})^2}}.$$

If r is close to ± 1, then we would say that the points are correlated. Completetly uncorrelated points would have $r = 0$. An illustration of this is seen in the image below:

So, why is correlation a useful metric?

- Correlation can help in predicting one quantity from another.
- Correlation can (but often does not) indicate the presence of a causal relationship.
- Correlation is used as a basic quantity and foundation for many other modeling techniques.

Example

Why does Pearson r correlation equal to ± 1 give us the strength of the correlation?

Solution

Click below to see the solution.

From the Schwarz inequality $|\sigma_{x,y}| \leq \sigma_x \sigma_y$.

Now imagine that all the points do indeed lie exactly on a straight line $y = A + Bx$. Since $y_i = A + Bx_i$ and $\hat{y} = A + B\hat{x}$, then $y_i - \hat{y} = B(x_i - \hat{x})$. Using this to remove the ys, we get,

$$r = \frac{B\sum(x_i - \hat{x})^2}{\sqrt{\sum(x_i - \hat{x})^2 B^2 \sum(x - \hat{x})^2}} = \frac{B}{|B|} = \pm 1.$$

However in the case that there's no correlation with x and y, then although the numerator will fluctuate $+/-$ve, the dominator will always be positive and drive r to zero as the number of points tend to infinity.

Correlation with Probabilities

So how close to 1 is close enough? It turns out it is actually possible to work out the probability that r will exceed a given value r_0 after a given number of uncorrelated data points are considered, i.e. $P_N(|r| \geq r_0)$.

If we look at standard probability tables (see table below), the probability of randomly finding a correlation between two variables with coefficient of $r \geq 0.7$ for a sample of $N = 3$ is 51% **even if 2 variables are uncorrelated**. Therefore we need to combine any r correlation value with some measure of the probability of getting that value just by random given the number of data points you have. We will return to this in Chapter 5.

Table 9.4. The probability $Prob_N(|r| \geqslant r_o)$ that N measurements of two uncorrelated variables x and y would produce a correlation coefficient with $|r| \geqslant r_o$. Values given are percentage probabilities, and blanks indicate values less than 0.05%.

					r_o						
N	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
3	100	94	87	81	74	67	59	51	41	29	0
6	100	85	70	56	43	31	21	12	6	1	0
10	100	78	58	40	25	14	7	2	0.5		0
20	100	67	40	20	8	2	0.5	0.1			0
50	100	49	16	3	0.4						0

Note that the correlation between two variables that each have a Gaussian distribution can be calculated using standard methods such as the Pearson's correlation but this procedure cannot be used for data that does not have a Gaussian distribution. Instead, we will see something at the end of the Block which can allow us to look for correlations without assuming a Gaussian distribution.

Example

Generate some fake data using the following

```
data1 = 20 * np.random.normal(0,1,1000) + 100
```

```
data2 = data1 + (10 *np.random.normal(0,1,1000) + 50)
```

Find the covariance of this dataset and the Pearson r correlation and comment on what you find.

The `random.normal(0,1,N)` generates data with normal distribution centred on 0, with width 1 and N data points.

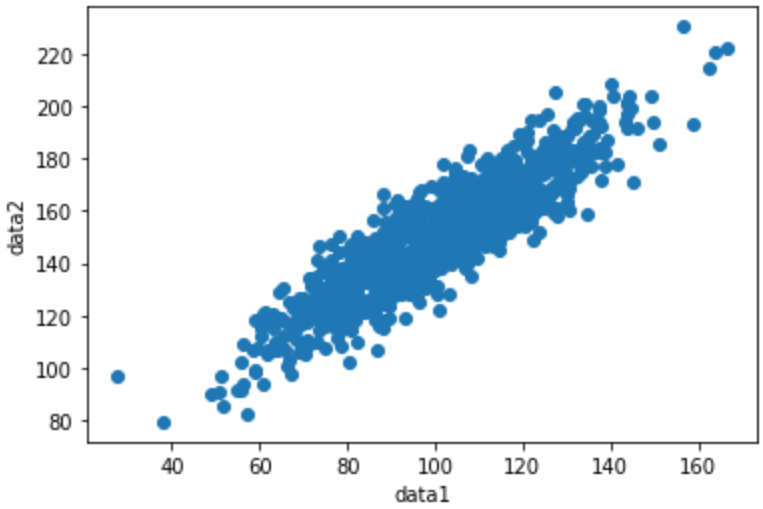
Solution

Click below to see the Solution.

In [3]:

```
import numpy as np
import pylab as plt

# the line below makes the plot appear in the jupyter notebook
%matplotlib inline

# let's generate some random data
data1 = 20 * np.random.normal(0,1,1000) + 100
data2 = data1 + (10 *np.random.normal(0,1,1000) + 50)
n = len(data1)

# define a function that estimates covariance
def cov(x,y,n):
    x_hat = np.mean(x)
    y_hat = np.mean(y)
    return np.sum((x-x_hat)*(y-y_hat))/(n-1)

# let's work out mean of the data
xhat = np.mean(data1)
yhat = np.mean(data2)

print('the mean of x is {:.2f}'.format(xhat))
print('the mean of y is {:.2f}'.format(yhat))

# covariance between the datasets
covar = cov(data1,data2,n)
print('the covariance between x and y is {:.2f}'.format(covar))

# plot
plt.scatter(data1, data2)
plt.xlabel('data1')
plt.ylabel('data2')
plt.show()
```

```
the mean of x is 101.20
the mean of y is 151.10
the covariance between x and y is 411.91
```



The data looks to be highly correlated. Now it's not too much more work to calculate the linear correlation coefficient r. Here we will see how to do this using the inbuilt python function from the `scip.stats` package. Many of the things we'll do in the course have inbuilt routines in python but part of the coursework will see you doing it from scratch to check understanding.

In [4]:
```
from scipy.stats import pearsonr

corr, _ = pearsonr(data1, data2)
print('Pearsons correlation is: %.3f' % corr)
```

```
Pearsons correlation is: 0.902
```

We know from our notes that this value of r indicates that the data is strongly correlated.

Null hypothesis Tests

Testing a hypothesis is one of the foundations of data analysis. Examples include: does this drug make people better? Is the die fair? Are an observed population of low-mass galaxies consistent with the predictions from ΛCDM? Did CERN really detect the Higg's Boson? We'll start this section by outlining the formal ideas behind hypothesis testing, and then look at some classic examples.

The most common form of hypothesis testing is involves trying to find the unknown parameter θ that is part of a model $f(\theta)$. Now you might have a best guess for the unknown parameter, and an associated uncertainty, so really we're not always testing if θ is an exact value, but more generally whether $\theta \in \Theta$, that is θ is part of some set of possible values Θ. From our best guess of θ, what we're trying to determine is whether $\theta \in \Theta_0$ or $\theta \in \Theta_1$, and where,

$\Theta_0 \cup \Theta_1 = \Theta$ and $\Theta_0 \cap \Theta_1 = 0$.

We then make a set of new observations of some outcome of the model $X = \{x_1, x_2, x_3, \ldots\}$, and we want to test whether they support the idea that, say, $\theta \in \Theta_1$. We also know the probability of the model predicting the data, which is given by $p(X, \theta)$.

This is **Null Hypothesis Significance Testing**, which we will abbreviate as NHST.

$$H_0 : \quad \theta \in \Theta_0 \ \text{the null hypothesis} \tag{1}$$
$$H_1 : \quad \theta \in \Theta_1 \ \text{the alternative hypothesis} \tag{2}$$

The null hypothesis assumes that nothing interesting happens/happened. The alternative hypothesis is where the action is i.e. some observation/ phenomenon is real (i.e. not a fluke) and statistical analysis will give us more insights on that. The null hypothesis is saying that one variable does not affect the other and anything we see in our results are due to chance and are not significant in terms of supporting the idea being investigated. Thus, the null hypothesis assumes that whatever you are trying to prove did not happen.

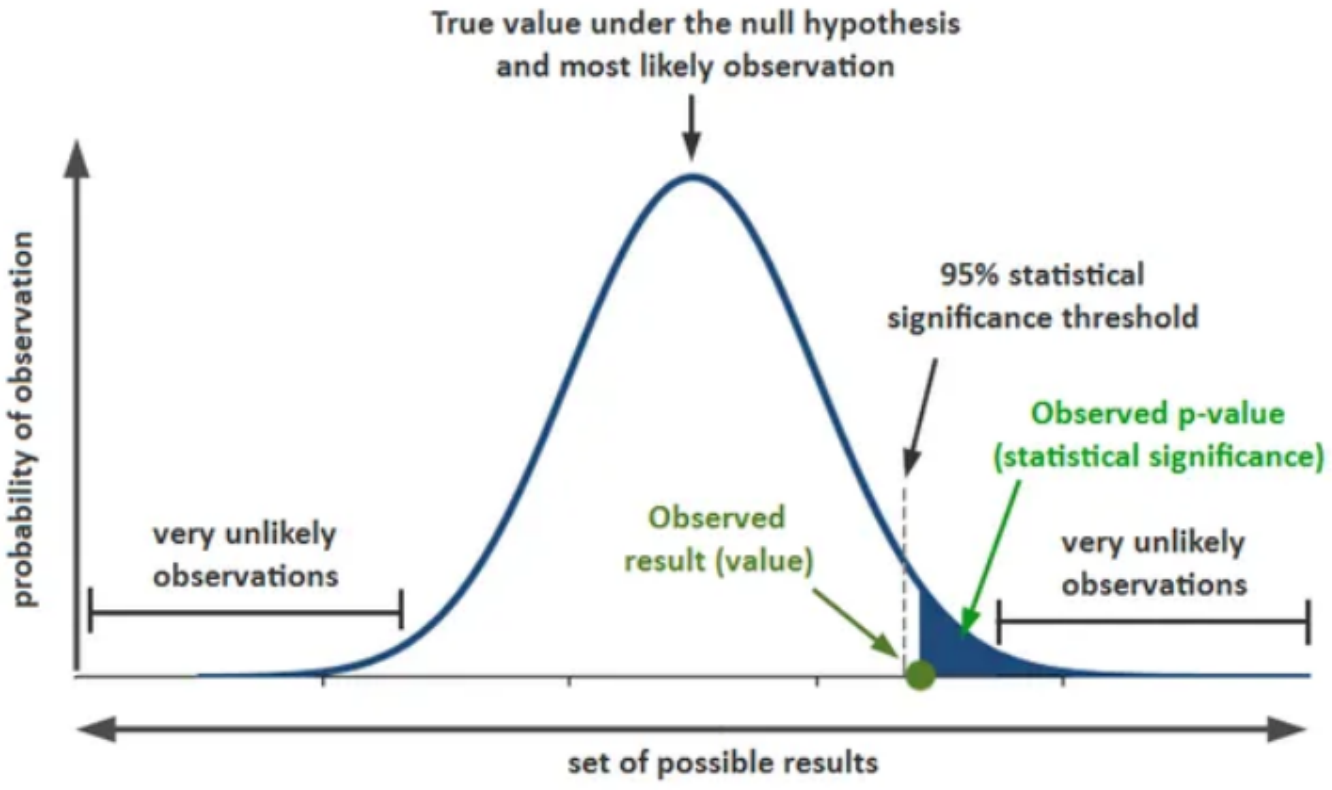
We then compute a statistic and then ask "What is the chance of observing the test-statistic for this sample (considering its size and the probability governing the system), purely randomly (ie if the null hypothesis were true)?"

This chance — the probability value of observing the test-statistic — is the so-called $p-$value. The $p-$value is the probability of getting a value of the test statistic at least as extreme as that actually observed value purely by chance, if the null hypothesis is true.

- $p < 0.05$: A p-value less than 0.05 for your result is considered by some scientists as statistically significant. It indicates strong evidence against the null hypothesis because what we're saying is that there less than a 5% probability the null is correct (and the results are random). Therefore, we reject the null hypothesis, and accept the alternative hypothesis.

- $p > 0.05$: A p-value higher than 0.05 is not statistically significant and indicates strong evidence for the null hypothesis. In this case we would keep the null hypothesis and reject the alternative hypothesis. Note that we **cannot accept the null hypothesis**, we can only reject the null or fail to reject it.

As an example, if a test gives the p-value, $p = 0.03$, the null hypothesis would be rejected at significance level (α) where $\alpha = 0.05$, but not at the more conservative significance level of $\alpha = 0.01$. Below is a schematic showing this idea:



So the kind of statistical statement you may read is something like the following, "Feeding chocolate to female chickens gives a proportion of male chicks that is significantly less than 50% (with $p = 0.001$)."

But beware:

- having a p-value below 0.05 does not mean that there is a 95% probability that the research hypothesis is true.
- Stating a low p value does not prove our research idea is correct because this would imply we have 100% certainty which is impossible.
- we would instead comment that our results provide support for a research theory.

Errors in NHST

It is possible to make two types of error in classical hypothesis testing when you reject a null hypothesis, and they have well defined names:

- Type I error is when you (for some reason) reject H_0 when it is true (eg experimental results are affected by randomness) - this results in a false positive.
- Type II error is when you decide not to reject H_0 when it is false.

For the typical $\alpha = 0.05$ or equivalently 5%, 1 experiment in every 20 will yield incorrect conclusions.

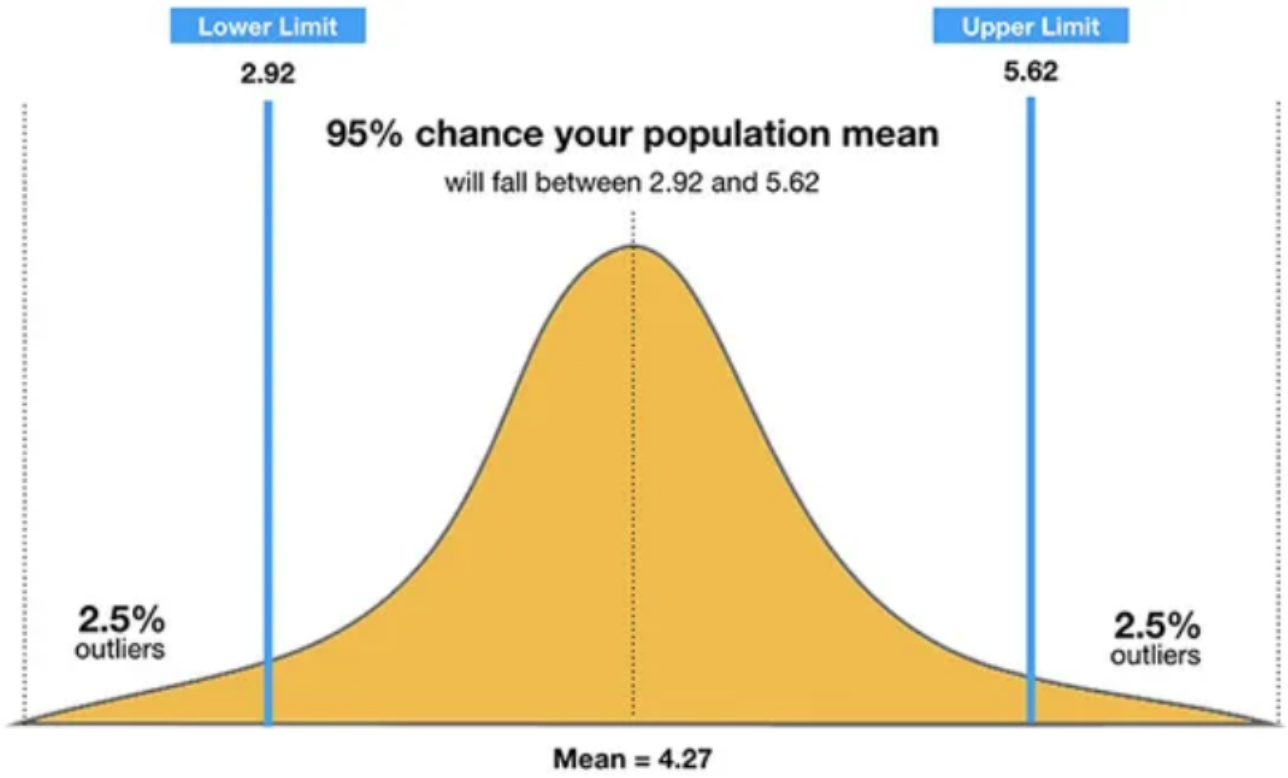
Remember, you cannot prove that something is correct in classical hypothesis testing, only prove that it is wrong. This is why the errors focus on H_0 - at best you can accept that H_0 is correct, and thus our hypothesis that $\theta \in \Theta_1$ is wrong.

Confidence Intervals

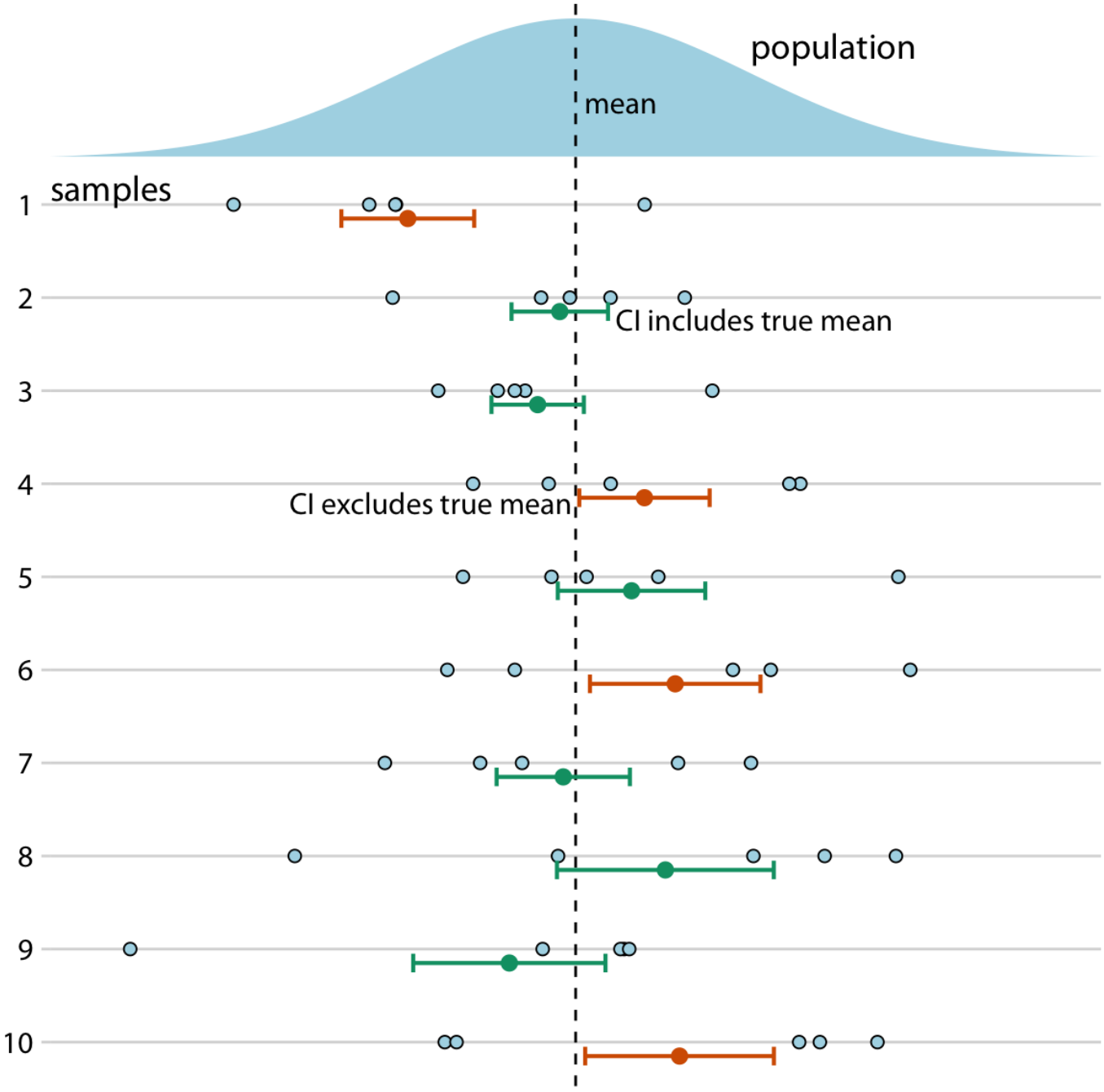
The p-value above gives the difference from a previously specified statistical level α (say probability of 0.05). In contrast, a confidence interval (CI) contains a parameter (like a population mean) with a certain confidence level. In other words, the CI provides a range of possible plausible values for the target population, as well as the probability with which this range covers the real value.

Often data analysts use a confidence interval of 95% to quote their numbers. A statement that you may see is "Feeding chocolate to female chickens produced 36.1% male chicks with a 95% confidence interval of 25.9 to 47.4%."

For example, your mean battery life might be 105 hours, and the 95% confidence interval in that number ranges from 100 to 110 hours. That means if you repeat your experiment millions of times, 95% of the time you repeat your experiment, the average battery life will fall into that range and the other 5% it will not. Take a look at the figure below to see this illustrated (taken from here).



What this is really telling us is that of the many 95% confidence intervals produced from multiple experiments in order to try and measure a variable, 95% are expected to contain the true value. The other 5% of experiments may completely fail! This is a bit awkward! See the image below (taken from Claus O. Wilke's Fundamentals of Data Visualisation book here) for an illustration of this for a confidence interval of 68% (ie for this variable, the confidence interval includes the true mean approximately 68% of the time - green and does not contain the true mean 32% of the time -orange).



Check out the incredible animation of confidence intervals here.

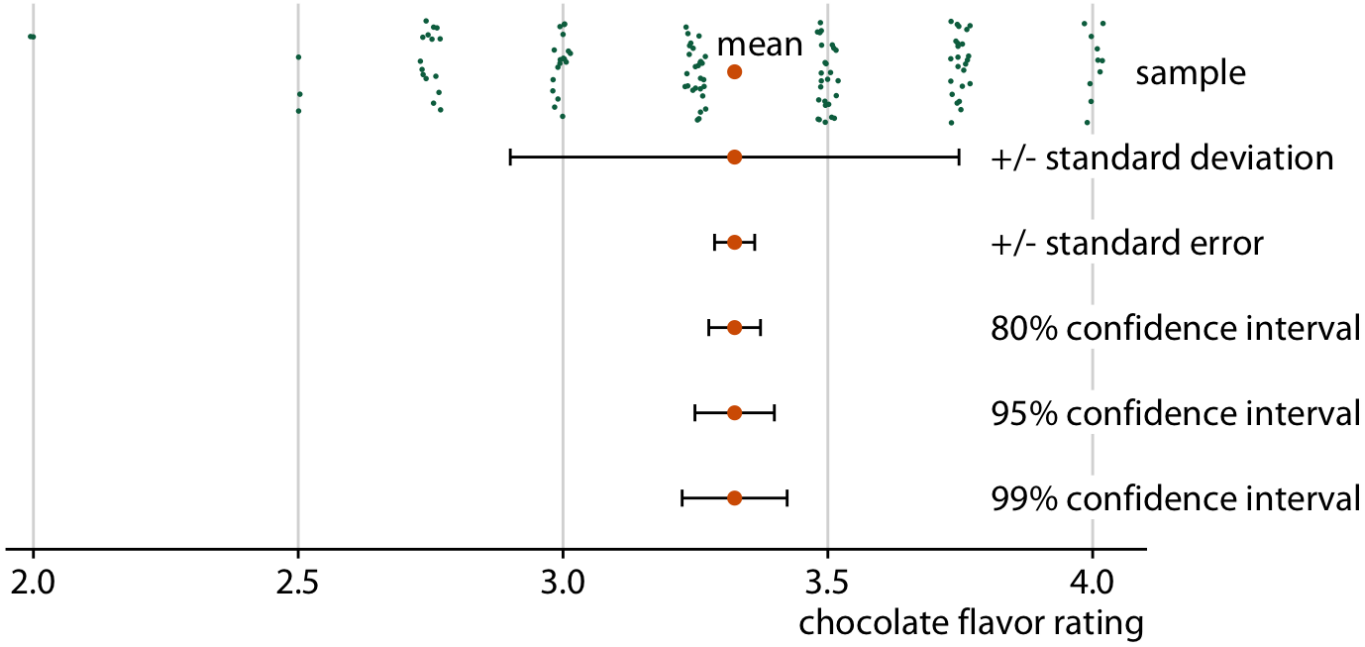
Comparing with our normal distribution error function from Chapter 3, we can see that a probability of 0.954 equates to a value that is $\pm 2\sigma$ from the mean value (to be exact it is $\pm 1.96\sigma$. As such we can also quote a 95% confidence interval as $\text{mean} \pm 1.96\sigma$, or a 99.7% confidence interval as $\text{mean} \pm 3\sigma$.

Tips:

- The narrower the confidence interval (upper and lower values), the more precise is our estimate.
- The confidence interval on your sample statistic should become more narrow as you make more measurements $N \rightarrow \infty$ as you approach your population value.
- If we want a confidence interval of say 99%, then our confidence interval increases.

Below we show a figure (again by Claus Wilke) that illustrates the relation between a sample of data, the sample mean, standard deviation, standard error, and confidence intervals, in an example of chocolate bar ratings. This figure is from this blog and is a sample of results of ratings from experts tasting 125

chocolate bars from Canadian chocolate makers. The scale 1 to 5 represents unpleasant to luxury. The large orange dot is the mean of the ratings. Data source: Brady Brelinski, Manhattan Chocolate Society.



Example

A poll given by the New York Times indicates that, in the next election, 34% of surveyed voters said they would vote for Biden and 31% said they would vote for Trump (with $\pm$ 3% margin of error). Note that polls typically use the default 95% confidence level. What does this mean?

Solution

Click below to see the Solution.

This means that, based on the voters sampled, there is a 95% chance that candidate X has the support of anywhere between 31% and 37% of all voters, while there is a 95% chance that candidate Y has the support of anywhere between 28% and 34% of all voters. Clearly, both intervals overlap, which make it difficult to conclude with certainty that candidate X would really be ahead of candidate Y, if we were to ask the question to the entire population of voters

Example

Fox news discusses a publication where a 95% confidence interval for the average amount of television watched by Americans was found to be (2.69, 6.04) hours. They state that this means that 95% of all Americans watch between 2.69 and 6.04 hours of television. Is this statement factual?

Solution

Click below to see the Solution.

This statement by Fox news is false. The correct statement would be that we are 95% confident that the average amount of television watched by Americans is between 2.69 and 6.04 hours.

Bayesian vs Frequentist Approaches

Frequentist Approach

- How likely is the data given the model?
- A probability is a measure of the frequency of repeated events, so the interpretation is that parameters are fixed (but unknown), and data are random.
- Probability Density Functions *quantify variability in a sequence of trials* such that $p(x)$ describes how the values of x would be distributed among infinte trials N.
- The confidence interval either contains the population mean or it does not, frequentists are 95% confident that the true value of the mean is contained in the quoted confidence interval as N gets large.
- This confidence interval is not a statement of the *sample, but rather the population*, 5% of confidence intervals won't contain the mean!

Bayesian Approach

- How likely is the model given the data?
- A probability is a measure of the degree of certainty about values, so the interpretation is that parameters are random and data are fixed.
- PDFs *quantify uncertainty in estimating the data* such that $p(x)$ describe how the probability is distributed over possible values of x that might have been measured in a single trial.
- Bayesians use credible intervals instead of confidence intervals (see Chapter 6) to ask "where do we expect the true parameter value to lie?"

Now you are ready to tackle the **Chapter 4 quiz** on Learning Central and the Chapter 4 yourturn notebook.

Chapter 5

Non parametric tests

Please ensure you have watched the Chapter 5 video(s).

You will learn the following things in this Chapter

- When to use a non parametric test on your data.
- How to calculate the Kendall's rank correlation.
- Understand how to use Spearman's rank to derive correlations.
- Understand when to use a Kilmogorov Smirnov test to compare distributions.
- How to use Python programming to do the above.
- After completing this notebook you will be able to finish CA 1 question 5.

A non parametric test (sometimes called a distribution free test) does not assume anything about the underlying distribution (for example, that the data comes from a normal distribution). Non-parametric tests are important particularly if the sample size is small or if there are large outliers in the data.

If we have a parameter that is normally distributed, then we would expect our sampling distribution to be normally distributed as well, characterised by a mean and standard deviation. What does this mean? Well next time we take data to measure the parameter, then we might expect that any future observation will lie in a given range dictated by the normal distribution.

The normal family of distributions all have the same general shape and are parameterized by mean and standard deviation. That means that if the mean and standard deviation are known and if the distribution is normal, the probability of any future observation lying in a given range is known. Here's a good example from Wikipedia.

Suppose that we have a sample of 99 test scores with a mean of 100 and a standard deviation of 1. If we assume all 99 test scores are random observations from a normal distribution, then we predict there is a 1% chance that the 100th test score will be higher than 102.33 (that is, the mean plus 2.33 standard deviations), assuming that the 100th test score comes from the same distribution as the others.

For *non-parametric tests*, we don't need to assume anything about the distribution of test scores to reason that before we gave the test, it was equally likely that the highest score would be any of the first 100 scores. Thus there is a 1% chance that the 100th score is higher than any of the 99 that preceded it.

Here we review some of the more common examples of non-parametric tests: (i) the univariate tests Kolmogorov-Smirnov and Anderson Darling and (ii) bivariate tests Kendall's τ and the Spearman's ρ_s rank. There are many other tests including Mann-Whitney etc.

Spearman's rank

As we saw in Chapter 4, correlation is a measure of how much two variables are related. This means that as one variable increases, so do the values of the other variable, or perhaps the other variable decreases as one increases. This is simple to estimate if both variables have a Guassian distribution, but if we do not know if this is the case, we need to use non parametric methods.

Spearman's rank correlation coefficient can be defined as a special case of the Pearson r applied to ranked (sorted) variables. Unlike Pearson, Spearman's correlation is not restricted to linear relationships. Instead, it measures monotonic association (only strictly increasing or decreasing, but not mixed) between two variables and relies on the rank order of values. In other words, rather than comparing means and variances, Spearman's coefficient looks at the relative order of values for each variable. This makes it appropriate to use with both *continuous and discrete* data.

This test is named after Charles Spearman and is denoted by greek letter ρ. Spearman's rank correlation is given by:

$$\rho_s = \frac{\sum_{i=1}^{N} R(x_i)\, R(y_i) - N(N+1)^2/4}{\sqrt{\sum_{i=1}^{N} R(x_i)^2 - N(N+1)^2/4}\sqrt{\sum_{i=1}^{N} R(y_i)^2 - N(N+1)^2/4}}$$

In the case of no ties (i.e. duplicates of x, y, etc), the above expression reduces to

$$\rho_s = 1 - \frac{\sum_{i=1}^{N} [R(x_i) - R(y_i)]^2}{N\left(N^2 - 1\right)}.$$

As you can possibly see from the equations above, Spearman's ρ is effectively calculating Pearson r correlation on the ordered/ranked values instead of the real values. So we are quantifying the degree to which the ordered/ranked variables are associated with a monotonic function (see below). The null hypothesis assumes that the samples are uncorrelated.

One nice feature of the Spearman test is that it is not so sensitive to outliers at the extremes of x and y as the Pearson correlation co-efficient. This is because the points are treated by their rank rather than their intrinsic value.

When can we use it?

- When the data is monotonic

When $N \sim 30$ and larger, ρ_s is a normal distribution with mean of 0 and variance $\frac{1}{N-1}$.

A nice easy way to rank the data in python is to use `from scipy.stats import rankdata` and `rank_x = rankdata(x)` .

Example

Use the following snippet of code to generate some fake data (the `random.seed` is to make sure we always get the same set of random data points each time we run our cell).

```
np.random.seed(100)

x = np.random.normal(0,1,30)

y = np.random.normal(0,1,30)
```

1. Estimate the Pearson r using in-built Python functions.
2. Let's see what one crazy outlier does to our correlations. Let's add an outlier data point using the snippet of code below. Re-evaluate the Pearson r. Also estimate Spearman's ρ on this dataset using in-built Python functions.

```
new_x = np.append(x, [20])

new_y = np.append(y, [20])
```

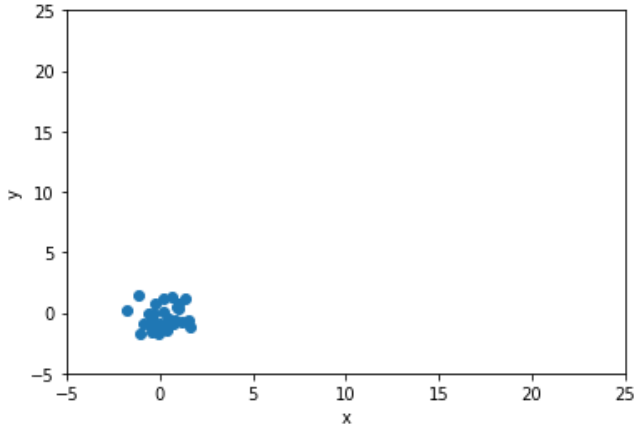
Solution

Click below to see the solution.

In [9]:

```
import numpy as np
from scipy.stats import pearsonr
from scipy.stats import spearmanr
import pylab as plt
%matplotlib inline

np.random.seed(100)

# data from qn 1
x = np.random.normal(0,1,30)
y = np.random.normal(0,1,30)

# let's take a look at the data
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-5,25)
plt.ylim(-5,25)
plt.scatter(x,y)

corr_pearson, p_pearson = pearsonr(x,y)
print('Pearson r correlation is {:.3f}'.format(corr_pearson))
```

```
Pearson r correlation is 0.106
```

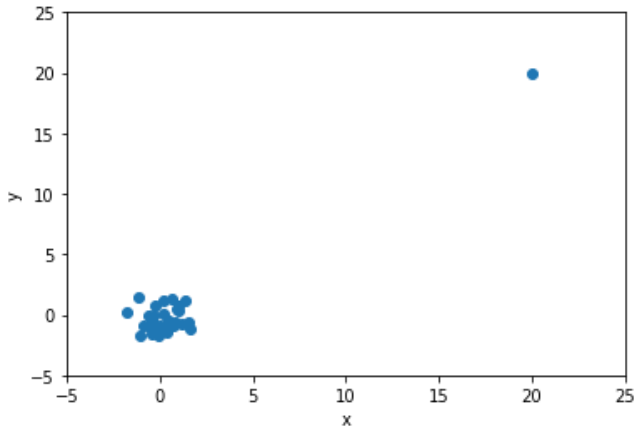


So it appears from Pearson r that the data is not correlated.

In [10]:

```
# now for qn 2
# let's add on that extra data point which we know is an outlier

new_x = np.append(x, [20])
new_y = np.append(y, [20])

# let's take a look at the data
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-5,25)
plt.ylim(-5,25)
plt.scatter(new_x,new_y)

corr_pearson, p_pearson = pearsonr(new_x,new_y)
print('Pearson r correlation is {:.3f}'.format(corr_pearson))
```

```
Pearson r correlation is 0.950
```



Now Pearson says we have strongly correlated data with correlation of 0.95!

Let's try Spearman now.

In [11]:

```
corr_spearman, p_spearman =  spearmanr(new_x,new_y)

print('Spearman rho correlation is {:.3f}'.format(corr_spearman)+' with p-value of {:.2f}'.format(p_spearman))
```

```
Spearman rho correlation is 0.220 with p-value of 0.23
```

Here we see that Spearman suggests there is no correlation (or rather it is very weak) in contrast to the extremely strong correlation value for Pearson's correlation coefficient. This reminds us to **look at our data** before we try doing any fancy statistics on it!

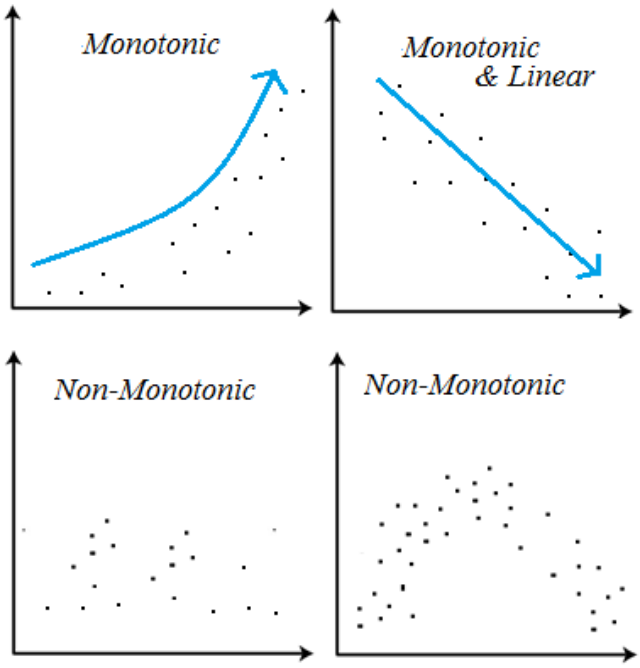
What do we mean by monotonic?

Monotonic relationships between data sets are where:

- one variable increases and the other increases or
- one variable decreases and the other decreases or

- one variable increases as the other decreases.

An example cartoon of monotonic, monotonic and linear, and non monotonic data is shown below (image from here).



Monotonic variables can increase (or decrease) in the same direction, but not always at the same rate. But linear variables increase (or decrease) in the same direction *at the same rate*.

Kendall's Tau

Unlike the Spearman ρ statistic, Kendalls' τ test does not take into account the difference between ranked data — only the directional agreement. Therefore, this coefficient is more appropriate for *discrete data*. Kendall's τ coefficient is defined as:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{N(N-1)/2}$$

- A concordant pair eg (x1, y1), (x2, y2) are pairs of values in which ranks coincide: $x1 < x2$ and $y1 < y2$ or $x1 > x2$ and $y1 > y2$.
- A discordant pair would be one that does not satisfy this condition, eg ($x = 1$,$y = 7$) and ($x = 2$,$y = 5$) would be a discordant pair.

Ties in y are assumed to account equally as 0.5 to both the number of concordant pairs and the number of discordant pairs. Ties in x are ignored. If N is large the distribution is given by a normal distribution with mean $E[\tau_x] = 0$ and variance

$$\sigma^2 = \frac{\sqrt{2(2N+5)}}{3\sqrt{(N(N-1))}}.$$

Note that the null hypothesis for Kendall τ would be that the variables are uncorrelated.

Chi-square Test

The chi-square goodness-of-fit test (or χ^2 test) can be applied to discrete distributions such as the binomial and the Poisson.

This test is used for *categorical data*.

In this type of hypothesis test, we try to determine whether the data fit a particular distribution or not. For example, we may suspect the unknown data fit a binomial distribution and use a chi-square test (meaning the distribution for the hypothesis test is chi-square) to determine if there is a good match or not.

This chi square statistic is obtained by calculating the difference between the observed number of cases and the expected number of cases in each **category**. This difference is squared and divided by the expected number of cases in that category to give the (general) chi-squared statistic:

$\chi^2 = \sum_i^N \left(\frac{O_i - E_i}{\sigma_i} \right)^2$.

We can compare the chi-squared statistic to probability tables using the number of degrees of freedom. In general, the number of degrees of freedom if you write your data as a table is (number of rows-1) $\times$ (number of columns-1). The null hypothesis is that there is no relationship between two categories.

If $\chi^2 \leq n_{\text{bin}}$ then we can say that there is a reasonable evidence that our assumption about the limiting function was correct. On the other hand, if $\chi^2 >> n_{\text{bin}}$ then we might want to reconsider.

Generalising chi-square

In the above example, our expected difference between the observed values and expected values was given by $\sqrt{E_k}$. However, in general we can simply write wherever we are comparing the expected values (from say a model) with data values (measured in our experiment)

$$\chi^2 = \sum_i^N \left(\frac{O_i - E_i}{\sigma_i} \right)^2$$

where σ_i is the standard deviation expected for each point that is being compared to the underlying distribution.

Derivation: chi-square test

Suppose we wish to know the range of gun. We could perform an experiment where we fire n projectiles from the gun, and measure the distance x at which the projectiles first make contact with the ground. In such case, due to the many, random uncertainties, it is natural to assume that the distribution x will be normally spread around some central value x_0, with a standard deviation of σ, such that the group of ranges $X = x_1, x_2, \ldots, x_n = N(x_0, \sigma)$

This is a new gun, and we know very little about it, so we have no prior knowledge of either x_0 or σ. Our first task is to estimate these from the measurements. We do this in the standard way, with,

$$\text{best estimate for } x_0 = \hat{x} = \frac{\sum x_i}{n}$$

and

$$\text{best estimate for } \sigma = \hat{\sigma} = \sqrt{\frac{\sum (x_i - \hat{x})^2}{n-1}}$$

We now want know: is our assumption that the distances are normally distributed correct? To do this, we have to compare the expected distribution to our actual distribution, to see whether our values of x are within those we expect for a limiting distribution of the form $N(x_0, \sigma)$, which we are approximating with $N(\hat{x}, \hat{\sigma})$.

The first problem is that because x is a continuous, rather than discrete variable, we need to split the distribution into intervals - remember that PDFs are not meaningful for a given x, but rather only make sense over some range of x. So let us split our data up into, say 4 bins, with boundaries chosen at $\hat{x} - \hat{\sigma}$, $\hat{x}$ and $\hat{x} + \hat{\sigma}$. We will refer to the number of bins as n_{bin}

We can now bin our data, counting the number of measurements O_k that we observe in each bin, k. We can then compare this number to the expected number of measurements we would have obtained in the bin, E_k, if our assumptions about the limiting distribution are true. To calculate E_k for each bin, we simply work out the probability of obtaining an value in the range of x covered by the bin, and multiply this by the total number of measurements made in the experiment,

$E_k = n \int_{x_{\min}}^{x_{\max}} N(\hat{x}, \hat{\sigma}) dx,$

where $x_{\min}$ and $x_{\max}$ are the lower and upper boundaries of bin k respectively. Clearly, the expected number E_k is only the number that we would expect in some average sense, and we do not expect any one experiment of finite n to yield O_k to be exactly E_k. However we do expect the deviations to be small, if our assumption about the underlying distribution is correct.

So how do we define small or large in this case? Well, the expected number of measurements in each bin can be thought of in terms of counting statistics -- i.e. the Poisson distribution that we encountered before. In this sense, E_k represents the mean number that we would expect from many experiments (each taking exactly n measurements). We know that the standard deviation around this mean is given by simply $\sqrt{E_k}$. So we would expect the deviations in our observed values from the expected values, i.e. $|O_k - E_k|$, to be close to the value $\sqrt{E_k}$, if our assumptions about the underlying distribution are correct.

That is, we expect,

$$\frac{|O_k - E_k|}{\sqrt{E_k}} \sim 1$$

If we sum up the deviations in all bins, we can define,

$$\chi^2 = \sum_{k=1}^{n_{bin}} \frac{(O_k - E_k)^2}{E_k}$$

Null hypothesis testing with chi-square

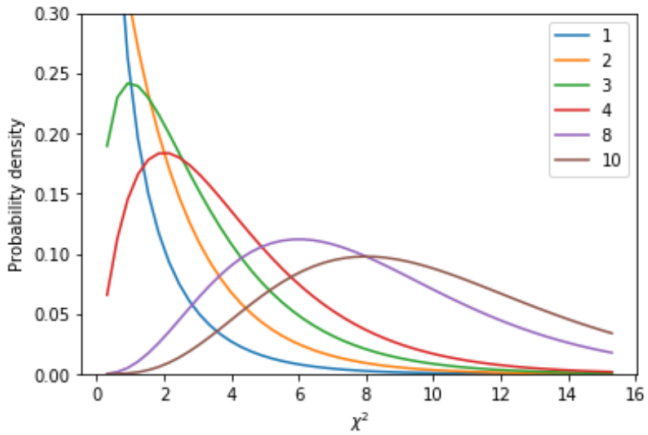
Let's say we measure a value of χ^2 for our experiment. How confident are we that we didn't just measure this statistic due to chance. Simply put, for the χ^2-square test, we have the following:

- H0: The data are consistent with a specified distribution.
- H1: The data are not consistent with a specified distribution

If the null hypothesis is true, the observed and expected values will be close in value and the χ-square statistic will be close to zero. If the null hypothesis is false, then the χ-square statistic will be large.

So then we compare with the probability density function for χ^2 and see if our χ^2 value is greater than or less than some critical value that depends both on the significance probability we want to test against (α, Chapter 4) and on the degrees of freedom d - $\chi^2_{\alpha,\mathrm{d}}$.

The distribution of χ^2 is shown below.



As $k \to \infty$, the expected value (mean) of this PDF tends to k and $\sigma \to \sqrt{2k}$. If $\chi^2_{\mathrm{data}} < \chi^2_{\alpha,\mathrm{d}}$ then we can not reject our null hypothesis.

The reduced chi-square statistic

Technically, one should compare the value of χ^2 with the number of degrees of freedom d, rather than the number of bins. In general,

$$d = n_{\mathrm{data}} - n_{\mathrm{params}}$$

where n_{data} is the number of data points, and n_{params} is number of parameters that have needed to be computed from the data - also known as the *constraints*. Let's say we have three constraints: the total number of data points n and then both the mean and variance were computed from the data. Now let's assume we ony have 4 bins of data, $n_{\mathrm{data}} = 4$. So we see that $d = 1$ for this example. Had we had fewer bins, the χ^2 test is basically not appropriate to use.

This suggests that a more convenient way to think about χ^2 is via the reduced χ^2, or χ^2 per degree of freedom, which we will denote as $\tilde{\chi}^2$, and define via

$$\tilde{\chi}^2 = \frac{\chi^2}{d}$$

We then see that for the case where our assumptions about the underlying distribution are correct,

expected value of $\tilde{\chi}^2 \approx 1$.

However, recent work by Andrae et al (2010) show that in practice, the degrees of freedom are in fact somewhere between $n_{\text{data}} - n_{\text{params}}$ and $n_{\text{data}} - 1$, where it is $n_{\text{data}} - n_{\text{params}}$ only if the model we are comparing data to is *linear* and *linearly independent* for the given data.

KS Test

The Kolmogorov-Smirnov, or K-S test, is one of the most commonly used tests in data analysis. The K-S is used to test whether two sets of data are drawn from the same underlying limiting distribution. It is based around the *cumulative distribution* of the data, and is restricted to continuous variables. This is a very common test in astronomy and medical science.

When would we use it? Let's assume that Group A take a placebo drug and Group B have medicine. Does the medicine have any effect? We can compare the distribution of results from Group A and B to see if they are different.

However, even if the medicine doesn't work, these numbers will never be the same due to randomness in experimental data, so *how different* must the outcomes be to say there is a difference or not in the medicine and the placebo?

The cumulative distribution $S_N(x)$ of a set of N data points $\{x_1, x2, \ldots, x_N\}$ is the fraction of data points with values less than a given x. The function is created by simply sorting the data and making a running, normalised, sum for each value of x. Clearly the function moves in steps of $1/N$, being constant between consecutive values of x.

The K-S test then measures the *maximum distance* between two cumulative distributions. If one is comparing the data set to some limiting distribution $f(x)$, with corresponding cumulative function $F(x)$, then the distance D between the two cumulative distributions is simply,

$$D = \max_{-\infty<x<\infty} |S_N(x) - F(x)|$$

or in the case where one is comparing two data sets,

$$D = \max_{-\infty<x<\infty} |S_{N_1}(x) - S_{N_2}(x)|$$

where $S_{N_1}(x)$ and $S_{N_2}(x)$ are the two data sets. One nice feature of using the distance in this way is that the K-S invariant to expansions and contractions in x -- it works just the same for x and $\log(\mathrm{x})$.

The null hypothesis in this case is that **the two distributions are the same**. So to put it more simply, the KS test uses the maximum deviation D between two distributions as a figure of merit to decide whether they match each other. Then we would infer that the closer the KS D statistic value is to zero, then it is more probable that 2 samples came from the same distribution. The closer D gets to 1 suggests that the 2 samples did not compe from the same underlying distribution, ie we could infer that they are statistically different from each other.

Disadvantages of the KS test

- The test is not distribution free, and so the data cannot be used to provide the parameters for the model $f(x)$, against which it is to be tested. For example, you cannot use the the K-S test to check whether the data has been drawn from a Gaussian, for which the mean and variance was derived from the data - the model parameters need to be known in advance.
- Although it has been used to examine multivariate distributions, this is generally not a good idea, due to the difficulties of defining cumulative functions in multivariate space.
- Finally, and probably most importantly, the KS is not good at picking up differences in the tails of functions. This is because the cumulative functions, which form the basis of the test, tend to 0 and 1 by definition. This results in the KS test being most sensitive around the median of the distributions (the 50th percentile), and progressively less sensitive as we move to the extremes.

Example

Data for recovery times (in mins) after a heart attack for a group of patients given a placebo and a group given a medicine are measured to be:

```
placebo_B=[1.26, 0.34, 0.70, 1.75, 50.57, 1.55, 0.08, 0.42, 0.50, 3.20, 0.15, 0.49, 0.95, 0.24,
1.37, 0.17, 6.98, 0.10, 0.94, 0.38]
```

treatment_B= [2.37, 2.16, 14.82, 1.73, 41.04, 0.23, 1.32, 2.91, 39.41, 0.11, 27.44, 4.51, 0.51, 4.50, 0.18, 14.68, 4.66, 1.30, 2.06, 1.19]

Find the expected value and associated statistics for this data.

Solution

Click below to see the solution.

In [12]:

```
import numpy as np

treatment_B=[1.26, 0.34, 0.70, 1.75, 50.57, 1.55, 0.08, 0.42, \
           0.50, 3.20, 0.15, 0.49, 0.95, 0.24, 1.37, 0.17, 6.98, 0.10, 0.94, 0.38]

placebo_B=  [2.37, 2.16, 14.82, 1.73, 41.04, 0.23, 1.32, 2.91, \
              39.41, 0.11, 27.44, 4.51, 0.51, 4.50, 0.18, 14.68, 4.66, 1.30, 2.06, 1.19]

mean_placebo = np.mean(placebo_B)
std_placebo = np.std(placebo_B,ddof=1)

mean_treatment = np.mean(treatment_B)
std_treatment = np.std(treatment_B,ddof=1)

print('The Placebo group')
print('The average recovery time is {:.2f} +/- {:.2f} mins'.format(mean_placebo,std_placebo))
print('min and max recovery times are {:.2f} and {:.2f} mins'\
      .format(np.min(placebo_B),np.max(placebo_B)))

print()
print()

print('The Medicine group')
print('The average recovery time is {:.2f} +/- {:.2f} mins'.format(mean_treatment,std_treatment))
print('min and max recovery times are {:.2f} and {:.2f} mins'.\
      format(np.min(treatment_B),np.max(treatment_B)))
```

```
The Placebo group
The average recovery time is 8.36 +/- 12.82 mins
min and max recovery times are 0.11 and 41.04 mins


The Medicine group
The average recovery time is 3.61 +/- 11.16 mins
min and max recovery times are 0.08 and 50.57 mins
```

So from these simple stats it appears that the recovery time is on average faster with the medicine compared to the treatment.

Example

Plot the cumulative distributions of these two groups.

Solution

Click below to see the solution.

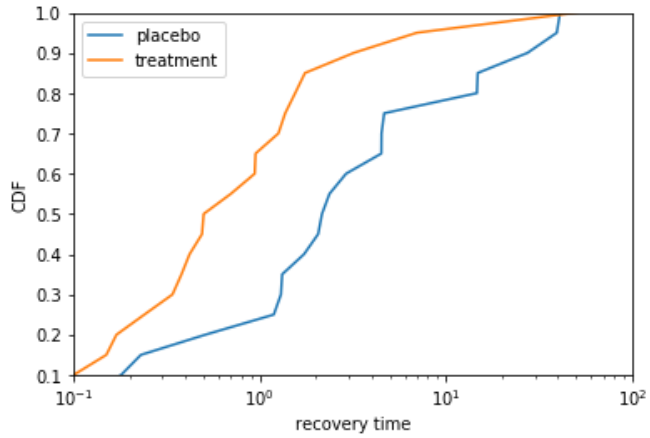
Let's look at how the data is distributed by first sorting it. To make a cumulative distribution from discrete data, all we need to do is sort the data and sum it at each step.

In [13]:

```
#make a function to plot the cumulative distribution function for the sample
def cdf(x):
    xs = np.sort(x)
    # this basically makes a cumulative distribution function
    ys = np.arange(1, len(xs)+1)/float(len(xs))
    return xs, ys

p_t,p_cdf = cdf(placebo_B)
t_t,t_cdf = cdf(treatment_B)

plt.semilogx(p_t,p_cdf,label='placebo')
plt.semilogx(t_t,t_cdf,label='treatment')

plt.xlim(0.1,100)
```

```
plt.ylim(0.1,1)
plt.legend()
plt.ylabel('CDF')
plt.xlabel('recovery time')
```

Out[13]: Text(0.5, 0, 'recovery time')



But for most of the x values, the fraction of the group given medicine that is less than x is clearly less than the fraction of the placebo group that is less than x. That is, by-and-large, the group with the medicine have shoter recovery times than the control/placebo values for the same cumulative fraction.

For example, the median (cumulative fraction = 0.5) recovery time for the treatment group is clearly less than one minute whereas the median for the placebo is more than 1min.

Example

Use the KS test to decide if these distributions are significantly different. You may use inbuilt functions.

Solution

Click below to see the solution.

In [14]:
```
from scipy.stats import ks_2samp
results = ks_2samp(treatment_B,placebo_B)

print('D is {:.3f}'.format(results[0]))
print('The p value is {:.4f}'.format(results[1]))
```

```
D is 0.450
The p value is 0.0335
```

The p-value is below our significance level of 0.05 so we can reject the null hypothesis that the two samples are the same.

Example

Above we saw the KS test being applied to two different samples. Here we will compares a data sample with a distribution.

We measure the current running through a circuit over a period of 8 hours. Each hour, we record the following currents (in amps)

1.41, 0.26, 1.97, 0.33, 0.55, 0.77, 1.46, 1.18

Is there any evidence that the current is drawn from a uniform distribution between 0 and 2 amps?

Solution

Click below to see the solution.

The probability density function of a uniform random variable X between 0 and 2 is $f(x) = 0.5$ for $0 < x < 2$. Therefore, the probability that X is less than or equal to x is:

$$P(X \leq x) = \int_0^x \frac{1}{2} dt = \frac{1}{2} x$$

We are interested in testing:

the null hypothesis H_0 :

$F(x) = F_0(x)$

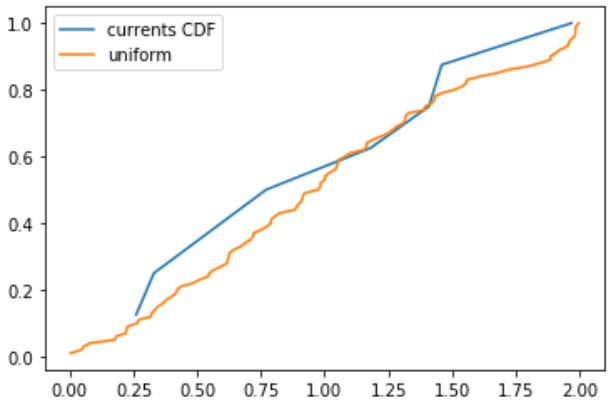
against the alternative hypothesis $H_1:

$F(x) \neq F_0(x)$.

Let's compare the cumulative distributions for the current data and a randomly generated uniform distribution from $0 < x < 2$.

In [15]:
```
import numpy as np
import pylab as plt

%matplotlib inline

#make a function to plot the cdf for the sample
def ecdf(x):
    xs = np.sort(x)
    ys = np.arange(1, len(xs)+1)/float(len(xs))
    return xs, ys

# generate random uniform distribution
s = np.random.uniform(0,2,100)

# you could also use the following that automatically calculates the function
# from statsmodels.distributions.empirical_distribution import ECDF

current = [1.41, 0.26, 1.97, 0.33, 0.55, 0.77, 1.46, 1.18]

xs, ys = ecdf(current)
xss, yss = ecdf(s)
plt.plot(xs, ys, label="currents CDF", markerfacecolor='none')
plt.plot(xss, yss, label="uniform", markerfacecolor='none')
plt.legend(loc='upper left')
```

Out[15]:
```
<matplotlib.legend.Legend at 0x1a16573690>
```



We can also use scipy.stats inbuilt KS routine and tell it we are comparing data to a uniform distribution.

In [16]:
```
# import package
from scipy.stats import kstest
import numpy as np
import pylab as plt

%matplotlib inline

current = [1.41, 0.26, 1.97, 0.33, 0.55, 0.77, 1.46, 1.18]

# run test of data against a uniform distribution
# ks test outputs the $D$ statistic and a $p value$.

kstest(current,'uniform',args=[0,2])
```

Out[16]:
```
KstestResult(statistic=0.14500000000000002, pvalue=0.9960120762325497)
```

The function returns D, the maximum difference between the cumulative distribution (CDF) of the input data and the model (the

uniform distribution). The second is the p-value, meaning the probability that if the data had actually been drawn from the proposed CDF, the resulting value of D would have been as large or larger than the one we measured.

This value of p tells us that 99.6% of the time, we would expect a value of D as large as the one measured even if the data had been drawn from the uniform sample. Thus we *can not reject the null hypothesis that the measured currents are drawn from a uniform distribution*.

Derivation: KS test

To derive the KS test involves modelling the random walk of data, analogous to Brownian motion. The function that models this is given by

$$K(\lambda) = 2\sum_{j=1}^{\infty}(-1)^{j-1}e^{-2j^2\lambda^2}$$

The significance level of an observed value of D is given approximately by,

$$\text{Prob}\left(D > \text{observed}\right) = K\left(\left[\sqrt{N_e} + 0.12 + 0.11/\sqrt{N_e}\right]D\right)$$

where N_e is the effective number of data points: $N_e = N$ for the case where we have a single set of data points, but for two distributions this is given by

$$N_e = \frac{N_1 N_2}{N_1 + N_2}$$

Anderson Darling

This is an improvement on the problems of the K-S test in the tails of the distributions. The Anderson-Darling test weights the data points away from the median. The statistic is given by:

$$A^2_{AD,N} = N\sum_{i=1}^{N}\frac{[i/N - S_N(x_i)]^2}{S_N(x_i)(1 - S_N(x_i))}$$

which should be computed for *both* the observed data set and the null distribution. There is unfortunately no analytic form for the A-D test, and one instead needs to resort to numerical computation: one simply creates a suite of random draws of N points from the null distribution to build up a pdf of $A^2_{AD,N}$. The observed value of $A^2_{AD,N}$ from the data set can then be compared to the probability of drawing the same value from the null, to decide wether to retain or reject the null hypothesis that the distributions are the same.

Now you are ready to tackle the **Chapter 5 quiz** on Learning Central and the Chapter 5 yourturn notebook.

Chapter 6

Back to Bayes-ics

Please ensure you have watched the Chapter 6 video(s).

You will learn the following things in this Chapter

- The idea of a conjugate prior and why they are useful.
- The interplay between priors and new data.
- Use Bayes Theorem on standard forms of probability density functions.
- Use this to do hypothesis testing in the Bayes framework.
- How to use Python programming to do the above.
- After completing this notebook you will be able to start CA 2.

Previously (Chapter 2) we used Bayes Theorem to find the probability of A | B using discrete data points, but here we will see that we can also use it to find the probability density function that some model or model parameter explains the data given the data we measure and some previously measured values or estimates at the parameter (the prior). Let's remind ourselves that:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}.$$

Let's recap the terms:

- $P(\theta|D)$ the **posterior** - the probability of model parameter θ being true, given the data
- $P(D|\theta)$ the **likelihood** - given model parameter θ what is likelihood of obtaining the data
- $P(\theta)$ the **prior** - the probability of the model parameter θ being 'true'
- $P(D)$ the **evidence** - the probability of getting the data, give all possible model parameter values (θ and others!)

In this Chapter we'll take a look at some classic examples where we can (easily) analytically solve the equations to get the model parameters out (which for us are the mean and standard deviation) out of the data we observe. From above we can see that the posterior probability is what we want to calculate, the prior is what we think we already know about the population, and the likelihood is the likelihood of obtaining our data given the measured mean and standard deviation.

We will look at 2 common situations:

- a likelihood that can be described by a normal distribution
- a likelihood that can be described by a bernoulli/binomial distribution

Before we do this, we also define the term **conjugate prior**. This is one that has the same functional form as the posterior. If the likelihood is normally distributed, then the choice of a normal prior will ensure that the posterior is also a normal, and as such, our prior can be said to be conjugate. This is generally a good thing!

Even if the true functional form of the posterior is *not* conjugate, it is common practise to approximate the prior distribution with a function that is, simply because it makes the mathematics easier, and it makes the functions behave! *Note that the prior is only conjugate when we consider likelihoods of a certain functional form.*

Normal likelihood - normal prior

Analytical expressions

Suppose that we have a set of observations $X = x_1, x_2, \ldots, x_n$ of some quantity that we believe to have been drawn from a normal distribution. We can then write,

$$p(x|\theta) = N(\theta, \sigma)$$

where θ is the mean value of the distribution (unknown) and σ describes the width (known).

We can then also write the probability of the mean of the sample, $\hat{X}$, as,

$p(\hat{X}|\theta) = N(\theta, \sigma/\sqrt{n})$

Notice that now we do not use σ as the error since as we make more measurements, our estimate of the mean gets better (recall the difference between the error and the standard error on the mean!).

Now suppose that we have some data from a previous study that reports a value for θ of μ_0, with an associated error, σ_0. This study was also subject to random errors, so we are free to write,

$p(\theta) = N(\mu_0, \sigma_0)$

Bayes theorem allows us to combine this information to determine the PDF of θ. Since the denominator is the integral over all θ, it is just a constant, and so it doesn't affect the shape of the posterior, only the height. As such, we can ignore it for the moment, and focus on the numerator (i.e. the likelihood $\times$ prior).

Since both the likelihood and prior are normal distributions, the posterior must also be a normal distribution, this leads to the following expressions for the mean and standard deviation of the posterior.

$$\hat{\theta} = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2/n}\hat{X} + \frac{\sigma^2/n}{\sigma_0^2 + \sigma^2/n}\mu_0$$

$$\hat{\sigma}^2 = \frac{\sigma_0^2\sigma^2/n}{\sigma_0^2 + \sigma^2/n}.$$

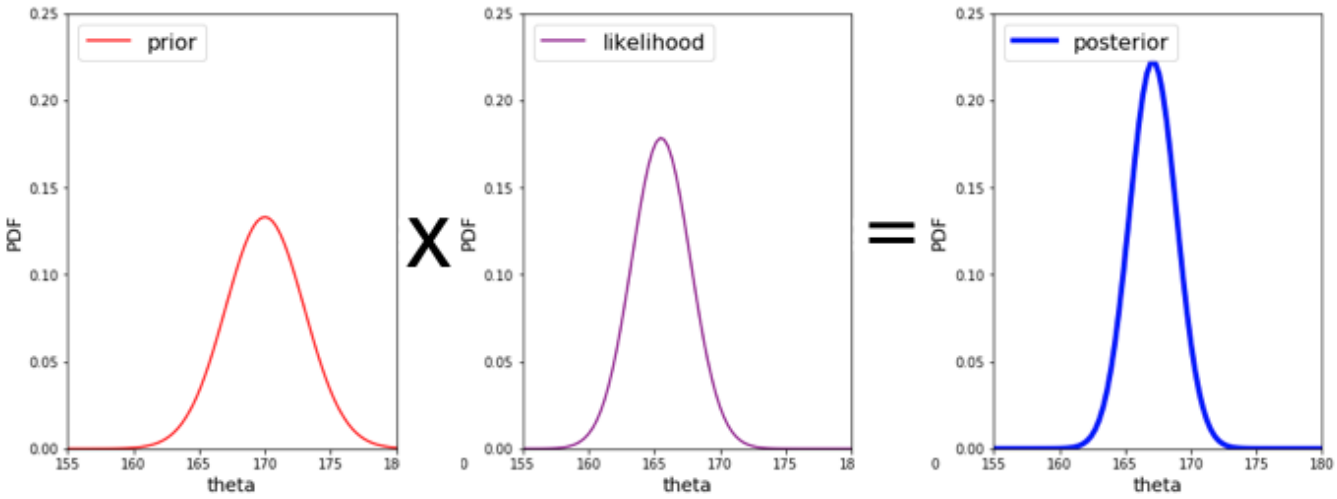
A nice way of thinking about this interplay between the prior and likelihood, is to imagine that the prior is simply adding another data point. If this data point is good (i.e. small variance, such that it is strongly peaked around the mean), then the prior pulls the posterior towards it. If the prior is vague, then it behaves like a data point with a big error, and the posterior relies on the data to provide the underlying shape. This is same behaviour, and indeed very similar maths, to the idea of *weighted averages* that we discussed in Chapter 5.

Good and Bad Data

When the data is good (eg if σ is small, and/or n is large), the equation for the mean above shows that the 1st term will tend to one and the second term to zero, the *posterior will favour the data over the prior*.

When the data is bad (eg if σ is large, and/or n is small), the equation for the mean above shows that the 1st term will tend to one and the second term to zero, the *posterior will favour the data over the prior*.

So to recap, the figure below summarises what we need to do to get the Bayesian probability density function (posterior) from the prior and likelihood, and how we may then get out statistics such as mean, variance etc.



Example

Ten 20-year old students have their heights measured (in cm) with variance of 50cm resulting in the following data:

heights = 169.6, 166.8, 157.1, 181.1, 158.4, 165.6, 166.7, 156.5, 168.1, 165.3

A previous set of (normally distributed) measurements said the mean and standard deviation of heights is 170 ± 3cm.

Assuming that the heights are drawn from a normal distribution, state what shape the posterior distribution has and derive the mean and standard deviation of the posterior distribution.

Plot the data you observe as a histogram. Also plot the PDFs for the likelihood and the prior.

Plot the posterior PDF and overplot the mean of the posterior (tip: `plt.axvline(value)` plots a vertical line). Briefly discuss what you see.

Solution

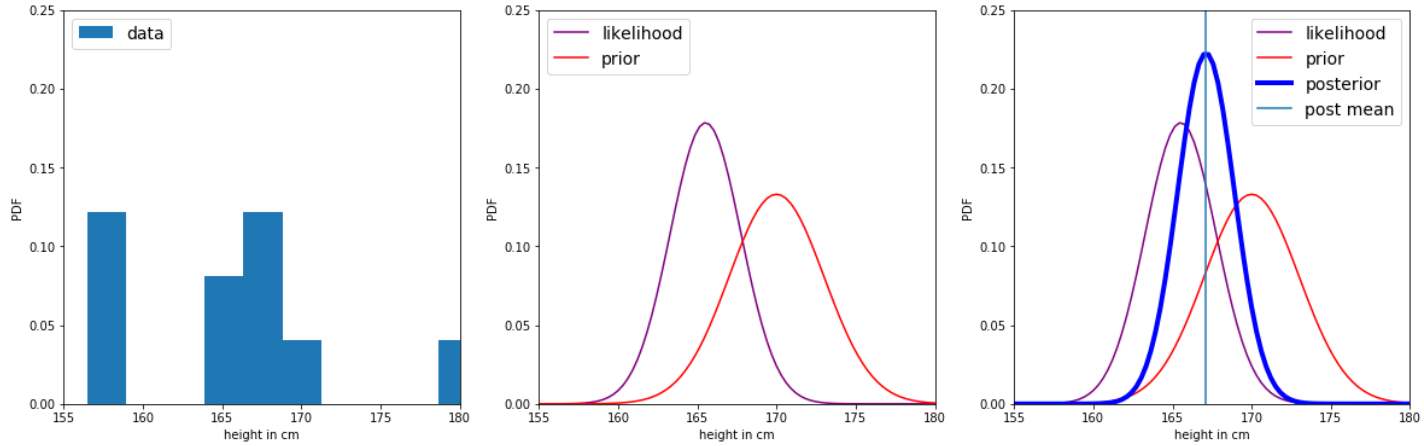
Click below to see the solution.

In [9]:

```
import numpy as np
import pylab as plt
from scipy.stats import norm

%matplotlib inline

# function for mean of posterior if posterior normal
def post_mean(var_0,var,n,X,mu_0):
    result = (var_0/ (var_0 + (var/n)) ) * X + ((var/n)/(var_0+(var/n)))*mu_0
    return result

# function for width (standard dev) of posterior if posterior normal
def post_std(var_0,var,n):
    result = np.sqrt((var_0*(var/n)) / (var_0 + (var/n)))
    return result

ht = [169.6,166.8,157.1,181.1,158.4,165.6,166.7,156.5,168.1,165.3]
n = len(ht)

var = 50.
err_mean = np.sqrt(var/n)

data_mean = np.mean(ht)
data_var = var
prior_var = 3**2.
prior_mean = 170.

# set up an array
d = np.linspace(150,180,100)

# these are normal distributions so
# can use the analytical equations from the notes for posterior
# mean and standard deviation
posterior_mean = post_mean(prior_var,data_var,n,data_mean,prior_mean)

posterior_std = post_std(prior_var,data_var,n)

print(r'likelihood mean and error are {:.3f} +/- {:.3f} cm'.format(data_mean,err_mean))

print('posterior mean and std are {:.3f} +/- {:.3f} cm'.format(posterior_mean,posterior_std))

plt.figure(figsize=(20,6))
# for plotting

plt.subplot(131)
plt.hist(ht,density=True,label='data')
plt.legend(loc='upper left',fontsize=14)
plt.xlim(155,180)
plt.ylim(0,0.25)
plt.xlabel('height in cm')
plt.ylabel('PDF')

plt.subplot(132)
plt.plot(d,norm.pdf(d,data_mean,err_mean),label='likelihood',c='purple') #likelihood
plt.plot(d,norm.pdf(d,prior_mean,np.sqrt(prior_var)),label='prior',c='red') # prior distribution
plt.legend(loc='upper left',fontsize=14)
plt.xlim(155,180)
```

```
plt.ylim(0,0.25)
plt.xlabel('height in cm')
plt.ylabel('PDF')

plt.subplot(133)
plt.plot(d,norm.pdf(d,data_mean,err_mean),label='likelihood',c='purple') #likelihood
plt.plot(d,norm.pdf(d,prior_mean,np.sqrt(prior_var)),label='prior',c='red') # prior distribution
plt.plot(d,norm.pdf(d,posterior_mean,posterior_std),label='posterior',c='blue',lw=4)
plt.xlim(155,180)
plt.ylim(0,0.25)
plt.axvline(posterior_mean,label='post mean')
plt.xlabel('height in cm')
plt.legend(loc='upper right',fontsize=14)
plt.ylabel('PDF')
plt.show()
```

```
likelihood mean and error are 165.520 +/- 2.236 cm
posterior mean and std are 167.120 +/- 1.793 cm
```



What does this tell us? We can see that the prior is quite far from data. The data has a few peaks and is not really Gaussian! but we do expect the population to be Gaussian. The peaks at heights <160cm caused the likelihood PDF to peak at a lower mean than the previous measurement (the prior). The posterior (given the prior information) favours the second group of data points but has a wider distribution.

Derivation: Mean and Variance from the Normal Distribution

How did we get the analytical expressions for the mean and variance of a normally distributed posterior?

$$p(\theta|\hat{X}) \propto p(\hat{X}|\theta)p(\theta)$$

$$\propto \exp\left[-\frac{(\hat{X}-\theta)^2}{2\sigma^2/n}\right]\exp\left[-\frac{(\theta-\mu_0)^2}{2\sigma_0^2}\right]$$

$$\propto \exp\left[-\frac{1}{2}\left(\frac{\hat{X}^2-2\theta\hat{X}+\theta^2}{\sigma^2/n}+\frac{\theta^2-2\theta\mu_0+\mu_0^2}{\sigma_0^2}\right)\right]$$

$$\propto \exp\left[-\frac{1}{2}\left(\frac{\theta^2}{\sigma^2/n}-\frac{2\theta\hat{X}}{\sigma^2/n}+\frac{\theta^2}{\sigma_0^2}-\frac{2\theta\mu_0}{\sigma_0^2}\right)\right]$$

$$\propto \exp\left[-\frac{1}{2}\left(\theta^2\left(\frac{1}{\sigma^2/n}+\frac{1}{\sigma_0^2}\right)-2\theta\left(\frac{\hat{X}}{\sigma^2/n}+\frac{\mu_0}{\sigma_0^2}\right)\right)\right]$$

Now, if we consider that θ were a normal distribution, then

$$f(\theta) \propto \exp\left[-\frac{(\theta-\hat{\theta})^2}{2\hat{\sigma}^2}\right]$$

$$\propto \exp\left[-\frac{1}{2}\left(\frac{\theta^2}{\hat{\sigma}^2}-\frac{2\theta\hat{\theta}}{\hat{\sigma}^2}\right)\right]$$

which has the same form as the expansion of the product of the likelihood and prior above. So the posterior must also be a normal distribution, and we can match up the terms containing θ^2 and 2θ from the relations above. From matching the θ^2 terms, we get,

$$\hat{\sigma}^2 = \left(\frac{1}{\sigma^2/n} + \frac{1}{\sigma_0^2} \right)^{-1},$$

which after a little algebra, gives:

$$\hat{\sigma}^2 = \frac{\sigma_0^2 \sigma^2/n}{\sigma_0^2 + \sigma^2/n}.$$

Similarly, we can match up the 2θ terms to get,

$$\frac{\hat{\theta}}{\hat{\sigma}^2} = \frac{\hat{X}}{\sigma^2/n} + \frac{\mu_0}{\sigma_0^2}$$

$$\hat{\theta} = \frac{\sigma_0^2 \sigma^2/n}{\sigma_0^2 + \sigma^2/n} \left(\frac{\hat{X}}{\sigma^2/n} + \frac{\mu_0}{\sigma_0^2} \right)$$

$$\hat{\theta} = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2/n} \hat{X} + \frac{\sigma^2/n}{\sigma_0^2 + \sigma^2/n} \mu_0$$

The expressions above give, respectively the mean ($\hat{\theta}$) and variance ($\hat{\sigma^2}$) of the **posterior** that describes the probability of θ. More importantly, we can see that our mean of the posterior, $\hat{\theta}$, depends on *both the mean and variance of the data*, $\hat{X}$ and σ^2, and *the mean and variance of the prior*, μ_0 and σ_0^2.

Binomial/Bernoulli Likelihood - beta prior

Analytical expressions

After the normal likelihood, Binomial/Bernoulli distributions are probably the second most common form, since it covers a wide range of problems, such the ski test we saw earlier in the course, drug trials, etc. The conjugate prior to this is is a beta function determined by shape parameters a and b.

The posterior mean is then given by,

$$\hat{\theta} = \frac{\nu}{N} \frac{N}{N+a+b} + \frac{a}{a+b} \frac{a+b}{N+a+b}$$

and the variance is

$$\sigma^2 = \frac{\hat{\theta}(1-\hat{\theta})}{\nu + a + N - \nu + b + 1}.$$

We can roughly associate a with the number of successes you previously observed and $(a+b) \sim n$ with the number of trials you observed. So if you have a guess for the prior mean m, you can try

$a = mn$

$b = 1 - m$

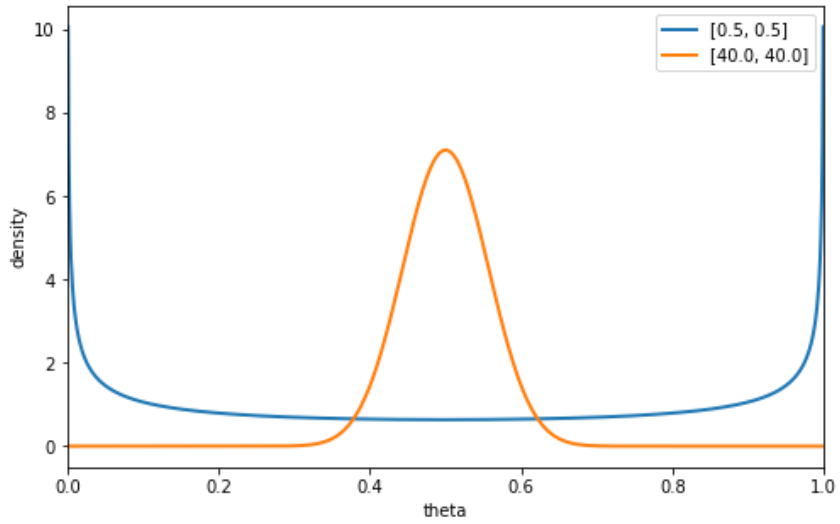
eg if you have a coin but do not know if it is biased towards heads and tails then $a = b = 0.5$ is a safe assumption. If you repeat the experiment 6 times and coin seems to be fair then a good choice might be $a = b = 3$. Below are some examples of priors using beta functions.

In [10]:

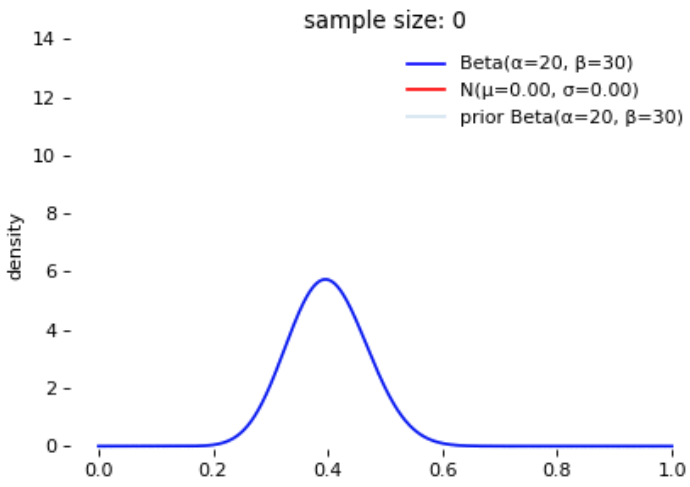
```
from scipy.stats import beta

plt.figure(figsize=(8,5))

a = [0.5,40.]
b = [0.5,40.]
```

```
x = np.linspace(0, 1, 1002)[1:-1]

for i in range(0,len(a)):
    dist = beta(a[i],b[i])
    plt.plot(x, dist.pdf(x),label=[a[i],b[i]],lw=2)

plt.xlim(0, 1)
plt.legend()
plt.ylabel('density')
plt.xlabel('theta')
```

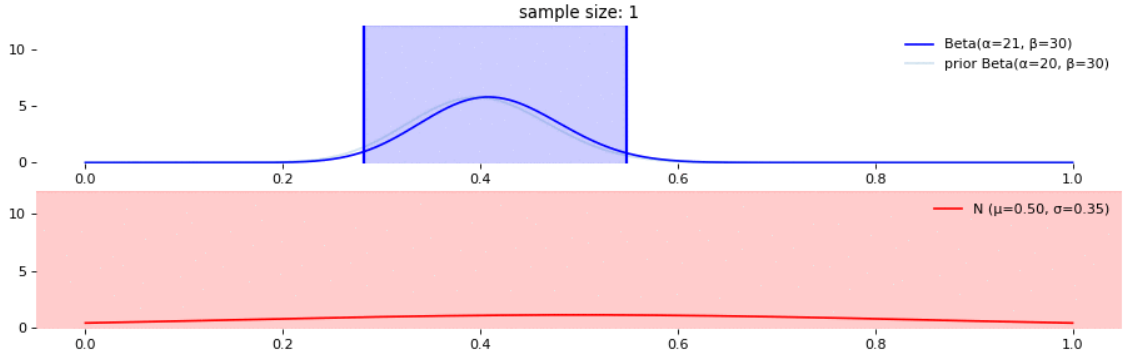
Out[10]: Text(0.5, 0, 'theta')



In practice let's say we're interested in whether a coin is fair. Someone else has previously thrown the coin 50 times and recorded 20 heads, therefore we already know something about the coin. We then do our own sampling by flipping it 10 times and record 7 heads. What we're doing is combining the prior information with our data to get the parameter we want. The animation below shows the probability density of the prior (light blue), likelihood (red) based on our measurements, and the posterior (dark blue) if the sampling numbers are increased. You can see that as the sampling number increases, the posterior peak rises and narrows, getting closer to the likelihood.



The difference between a credible interval and a confidence interval decreases as the number of data taken increases. The advantage of using a credible interval is most clear when the sample size is small or if it's difficult to take more data, eg giving patients different types of medicine. Below the animation shows the change in the credible interval (blue) and the confidence interval (red) for the coin as the sample size increases.



These animations are from this excellent post.

Derivation: Mean and Variance from Binomial Distributions

How did we derive the analytical approximations to the posterior mean and variance for the Bernoulli/Binomial case?

Remember that Bernoulli and Binomial distributions both have the form,

$$p(\nu|N,\theta) \propto \theta^{\nu}(1-\theta)^{(N-\nu)}.$$

In the case of the Bernoulli distribution, which focuses on a single outcome, the leading constant is 1, while in the case of the Binomial distribution, which accounts for any combination of the ν successes in N trails, the leading constant is given by the binomial factor $\binom{N}{\nu} = \dfrac{N!}{\nu!(N-\nu)!}$.

Remember that the functional family that has the same form as Bernoulli and Binomial distributions are called **beta distributions**. The mean and variance of the beta distribution are given by,

$$\hat{\theta}_B = \frac{a}{a+b}$$

$$\sigma_B^2 = \frac{\hat{\theta}(1-\hat{\theta})}{a+b+1}.$$

One can consider the prior as if it is reporting previously observed data. In this case, we can equate the $a-1$ and $b-1$ in the beta distribution to the ν and $N-\nu$ terms in the Bernoulli and Binomial distributions. This can the be used to guide your choice of a and b.

For example, if you think the coin is a joke coin, but do not know whether it is biased towards heads or tails, then $a = b = 0.5$ is perhaps a good choice. If you think the coin is probably fair, based on a low number (around 5-6) of observations in the past, then $a = b = 4$ might be a good choice, or larger values of a and b if you had a larger sample in the past. Given the number of events in the previous trial, and its outcome, you can use this to calculate your a and b for the Bayes calculation.

The full form of a Bayesian analysis of a Bernoulli likelihood with beta prior is,

$$p(\theta|\nu,N) = \frac{p(\nu,N\,|\,\theta)\,p(\theta)}{p(\nu,N)}$$

$$p(\theta|\nu,N) = \frac{\theta^{\nu}\,(1-\theta)^{(N-\nu)}\,\theta^{(a-1)}\,(1-\theta)^{(b-1)}}{B(a,b)\,p(\nu,N)}$$

$$p(\theta|\nu,N) = \frac{\theta^{(\nu+a)-1}(1-\theta)^{(N-\nu+b)-1}}{B(\nu+a,\ N-\nu+b)}.$$

Although the powers of θ and $(1-\theta)$ probably made sense there, you are probably wondering where the magic on the denominator came from. The clue is in the way we wrote the powers in the numerator, where you can see that we have deliberately written them in the form of a beta distribution, where

$$a \equiv \nu + a,$$

and,

$b \equiv N - \nu + b$.

This allows us to replace the complicated integral in the denominator with the standard normalisation for a beta distribution of the form beta$(\nu + a,\ N - \nu + b)$, which is simply $B(\nu + a,\ N - \nu + b)$.

Important: you must include the beta normalising denominator (from the beta distribution - see Chapter 3) in your posterior calculations, because it changes the shape of the beta function!

Once again, we can look at the interplay between the mean predicted by the data (via the likelihood), and the mean predicted by prior. The prior mean is $a/(a+b)$.

The mean of the posterior can then be found by substituting $a_{\rm post} \equiv \nu + a$ and $b_{\rm post} \equiv N - \nu + b$ into the same equation. Wen can then rearrange the posterior mean $\hat{\theta}$ to get,

$$\hat{\theta} = \frac{\nu + a_{\rm post}}{N + a_{\rm post} + b_{\rm post}} = \frac{\nu}{N}\,\frac{N}{N+a+b} + \frac{a}{a+b}\,\frac{a+b}{N+a+b}$$

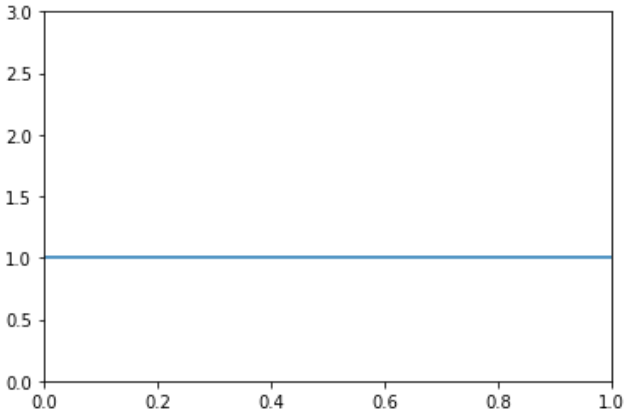
where we see once again that we weight the data and the prior, by their uncertainty.

What if we know nothing about the prior?

In the absence of any other information, a **uniform** (or constant) prior is often assumed. This can be interpreted as meaning that all possible values are equally likely, or you have no prior information and you cannot distinguish between possible values. Note that a uniform prior can be used by setting the prior to 1 over the range required, or by using a beta prior with $a = b = 1$. Here is an example of this below.

In [11]:
```
a = 1.0
b = 1.0

x = np.linspace(0, 1, 1002)[1:-1]

dist = beta(a,b)
plt.plot(x, dist.pdf(x),label='a=1,b=1')
plt.xlim(0, 1)
plt.ylim(0, 3)
```

Out[11]: (0, 3)



Although it sounds like it would make no difference, it turns out that a uniform prior can still add information and change the probabilities, particularly if your experiment has low success rates or low numbers of trials (for Binomial events say). Think about it this way: using a uniform prior in a Bernoulli experiment is equivalent to adding two observations to the data: one success and one fail (or one heads, one tails) etc.

Example

Suppose we have an experiment running N Bernoulli trials where ν successes are recorded. The unknown probability of success is p. A suitable uniform prior would be $p(\theta) = 1$ for all values. One might think we do not gain anything from a uniform prior, but using a uniform prior in a Bernoulli experiment is equivalent to adding two observations to the data, one success and one fail (or one heads, one tails). Here we will prove it can make a difference.

- Compare expressions for the expectation value expected from the data (the mean of the Bernoulli distribution) with that from the posterior.

- Calculate these for small N and ν and large N and ν.

Solution

Click below to see the solution.

For ν successes and N trials, the Bernoulli mean is given by $p = \nu/N$.

The prior is a beta function with $a = 1, b = 1$.

The posterior mean for this family, with a beta prior is given by

$$\text{experiment mean} = \frac{\nu}{N}$$

so

$$\text{posterior mean} = \frac{\nu}{N}\frac{N}{N+a+b} + \frac{a}{a+b}\frac{a+b}{N+a+b}$$ for a and b from priors.

Or we can write

$$\text{posterior mean} = \frac{\nu + a_{\text{post}}}{N + a_{\text{post}} + b_{\text{post}}}$$ where a_{post} and b_{post} are given by $\nu + a$ and $N - \nu + b$.

In [12]:
```
def post_mean_bin(n,nu,a,b):
    a_post = nu + a
    b_post = n - nu + b
    return (nu+a)/(n+a+b)

def bern_mean(n,nu):
    return nu/n

# let's test some values of nu_test,N_test
nu_test = [2.,8.,80.,800.]
N_test = [10.,10.,100.,1000.]

#
a_prior = 1.
b_prior = 1.

for i in range(0,len(nu_test)):
    print("N=",N_test[i],"nu=",nu_test[i])
    print('ratio between values from data + posterior is \
    {:.3f}'.format(bern_mean(N_test[i],nu_test[i])/post_mean_bin(N_test[i],nu_test[i],a_prior,b_prior)))
```

```
N= 10.0 nu= 2.0
ratio between values from data + posterior is     0.800
N= 10.0 nu= 8.0
ratio between values from data + posterior is     1.067
N= 100.0 nu= 80.0
ratio between values from data + posterior is     1.007
N= 1000.0 nu= 800.0
ratio between values from data + posterior is     1.001
```

So for the same N, the lower the success rate ν, the larger the difference between the posterior and the data values are - ie the uniform prior does affect the estimate of θ from the posterior. Difference is smaller for larger N.

The Evidence term

At this point, you are probably wondering why we've neglected the evidence term!

The evidence term is a number - it does not affect the shape of the distribution, just the absolute values of probability. The evidence term is just a normalisation for the problem. Often, we can ignore the evidence. For example when we just care the mean and variance of the posterior, the normalisation is not important. Also, if we simply want to the know the ratio of θ taking

two values, we again don't need to know the normalisation of the posterior, so we can ignore the evidence. Similarly, if we are just interested in finding the overall shape of the posterior.

However, we've established that for the case of a normal likelihood, and a normal prior, the posterior is also normally distributed. Also, we've already worked out the width parameter of the normal, so in this case, it is trivial to normalise the posterior, and thus create a true PDF.

For problems where we really do want to know the actual probabilities, but where the maths is tricky, we can still work without evaluating the evidence in many cases. For example, one can get a crude approximation to a normalised posterior by first making a *histogram* of the posterior, and then numerically integrating to find the total area under the histogram. If one then divides the original histogram by this area, the result is a normalised histogram of the posterior, this is an approximation to the posterior's underlying PDF.

Credible Intervals

The outcome of Bayes Theorem gives us the posterior which in these cases provides us with the probability density function for the parameter θ given the data and how likely you think this is. So if the parameter is outside the credible interval we say it's not credible, if it lies within 95% of the credible interval we say it is credible.

So in Bayesian statistics, a credible interval is an interval within which an unobserved parameter value falls with a particular probability.

Recall that in Bayesianism, the probability distributions reflect our degree of belief. So when we compute the credible region it is equivalent to saying

- "Given our observed data, there is a 95% probability that the true value of the mean falls within the CR" - Bayesians

In frequentism, on the other hand, μ is considered a fixed value and the data (and all quantities derived from the data, including the bounds of the confidence interval) are random variables. So the frequentist confidence interval is equivalent to saying

- "There is a 95% probability that when I compute confidence intervals from data of this sort, the true mean will fall within the confidence interval."- Frequentists

Here's another way again:

Suppose we read that the observed data D_{obs} support conclusion C. How do the two parties see this?

- "C was selected with a procedure that is right 95% of the time over a set of D that includes D_{obs}."- Frequentist
- "The strength of the chain of reasoning from the model and D_{obs} to concluding C has probability 0.95." - Bayesian

Credible intervals for Gaussians produce similar results to confidence intervals, but this is simply because of the properties of Gaussians.

Note that credible intervals are also known as high density intervals. These are used in many areas of science, in particular in gravitational wave astronomy.

Example

Suppose you are told "Here is a confidence interval from the Large Hadron Collider experiment." What does it mean?

Solution

Click below to see the solution.

There are two ways you could answer the question:

1. “There is 95% probability/plausibility/likelihood that the population parameter lies in the interval.”
2. “If we repeat the experiment infinitely many times, 95% of the experiments will capture the population parameter in their confidence intervals.”

The first answer is wrong. The first statement is interpreting a *Bayesian credible interval*. The second is the correct interpretation of a frequentist confidence interval.

Example

We are given a coin at random, and asked to perform 50 flips. We find that heads comes up 35 of the 50 times. We want to know if the coin is fair. Plot the likelihood, prior and posterior distributions on one plot.

Solution

Click below to see the solution.

The probability of obtaining the resulting series of heads (success) and tails (fail) is given by the Binomial distribution (since it is one coin flipped in a sequence of events/successes.

$p(\nu = 35|N = 50, \theta) = \theta^N(1-\theta)^{(N-\nu)}$

Since we were given the coin at random and we know nothing about it, we will adopt a uniform prior.

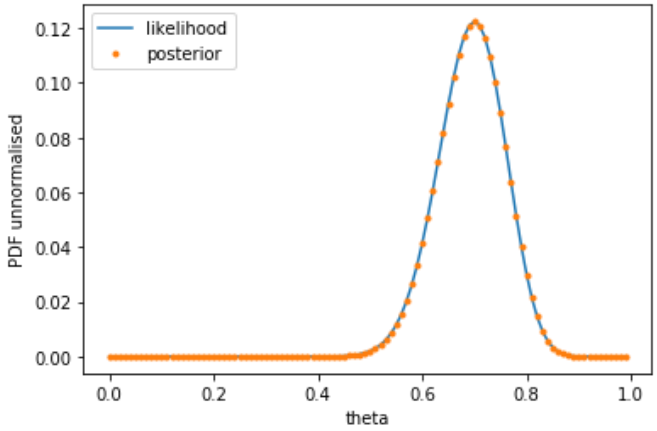
$p(\theta) = 1$ f or $0 < \theta < 1$.

Note that in this example I will write this simple as $p = 1$ for all x but a uniform prior for Bernoulli/Binomial family is equivalent to a beta function with $a = 1, b = 1$, so we could also write it as a beta function (we can also be written in the format = Beta(1,1)).

Our null hypothesis is that the coin is fair ($\theta = 0.5$).

In [13]:
```
import numpy as np
import pylab as plt
import math

%matplotlib inline

nu = 35
N=50

# for plotting
nsteps=100

h = np.arange(0,1,1./nsteps)

likelihood_coin = (math.factorial(N)/(math.factorial(nu)*(math.factorial(N-nu))))*h**nu * (1 - h)**(N - nu)

# prior is constant = 1 for all x
prior_coin = [1.0 for i in range(0,len(h))]

# posterior = likelihood x prior
posterior_coin = likelihood_coin*prior_coin

#let's save an array of x and y for working out credible intervals later

plt.plot(h,likelihood_coin,label='likelihood')
plt.plot(h,posterior_coin,'.',label='posterior')

#plt.xlim(0,1)
plt.ylabel('PDF unnormalised')
plt.xlabel('theta')
plt.legend(loc='upper left')
```

Out[13]:
```
<matplotlib.legend.Legend at 0x1a22fe4cd0>
```



Example

Now estimate the 95% credible intervals and the value we would expect for a fair coin $\theta = 0.5$. For a ROPE we could set this to something like $\theta = 0.5 \pm 0.025$.

Solution

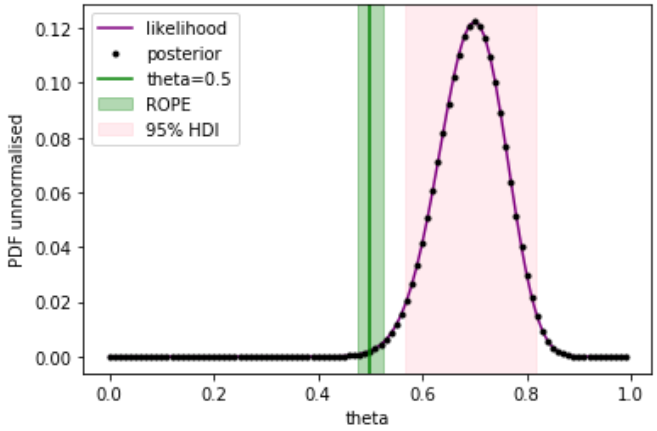
Click below to see the solution.

So is it fair? Let's assume we're approaching gaussian distribution for $N = 50$ and simply use the fact that the mean $\pm 1.96\sigma$ gives 95% probability intervals.

To do this we need to calculate the mean and standard deviation of the posterior. We can do this using the analytic expressions listed above for a bernoulli likelihood with beta prior.

In [14]:
```
# posterior_mean
nu =35.
N=50.

# uniform prior = beta(1,1) ie a=1, b=1
a = 1.
b = 1.

# posterior mean + std from analytical approx in lecture notes
posterior_mean_coin = np.float((nu+a)/(N+a+b))
posterior_std_coin = np.sqrt( posterior_mean_coin*(1-posterior_mean_coin)/(nu+a+N-nu+b+1))

# set up credible intervals = mean + / - 1.96 x sigma
HDI_a = posterior_mean_coin+1.96*posterior_std_coin
HDI_b = posterior_mean_coin-1.96*posterior_std_coin

plt.plot(h,likelihood_coin,label='likelihood',c='purple')
plt.plot(h,posterior_coin,'.',label='posterior',c='black',lw=2)

# this makes nice shaded regions to show off where the ROPE is
plt.axvspan(0.5-0.025,0.5+0.025,color='green',label='ROPE',alpha=0.3)
plt.axvspan(HDI_b,HDI_a,color='pink',label='95% HDI',alpha=0.3)

plt.axvline(0.5,color='green',label='theta=0.5')
plt.ylabel('PDF unnormalised')
plt.xlabel('theta')
plt.legend(loc='upper left')
```

Out[14]:
```
<matplotlib.legend.Legend at 0x1a22fac450>
```



The fair coin ($\theta = 0.5$) falls outside the 95% HDI, so we can reject the null hypothesis that the coin is fair. (Note it could also be that our ROPE was too stringent).

Note that in python we could determine credible intervals if we assume that the posterior were normal by (i) using `scipy.stats norm.interval(0.95,mean,std)` function or (ii) numpy's `percentile(data,95)` function.

Example

Let's return to the question earlier about the student heights. Now we are going to assume the null hypothesis is that the height of a 20 year old student is > 170 cm. Determine the 95% credible intervals for the posterior. Plot this range over your posterior distribution. You can do this using lines axvline or axvspan to colour the entire range.

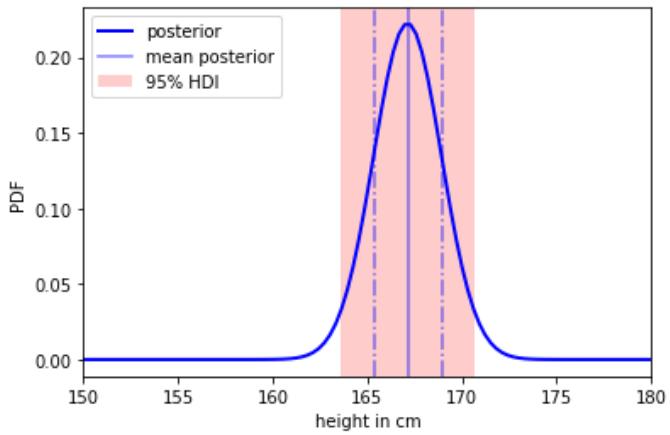
Solution

Click below to see the solution.

In [15]:
```
# can use norm.interval to get the interval out for 95%
# note that 95% is roughly equivalent to 2xstandard deviations so they can use that
HDI_1,HDI_2 = norm.interval(0.95,posterior_mean,posterior_std)
print('95% confidence intervals are {:.3f} and {:.3f} cm'.format(HDI_1,HDI_2))

plt.plot(d,norm.pdf(d,posterior_mean,posterior_std),label='posterior',c='blue',lw=2)
plt.axvline(posterior_mean,label='mean posterior',color='blue',alpha=0.5)
plt.axvline(posterior_mean+posterior_std,color='blue',ls='-.',alpha=0.5)
plt.axvline(posterior_mean-posterior_std,color='blue',ls='-.',alpha=0.5)
plt.axvspan(HDI_1,HDI_2,facecolor='red', alpha=0.2, label = '95% HDI')
plt.legend(loc='upper left')
plt.xlim(150,180)
#plt.ylim(0,0.3)
plt.xlabel('height in cm')
plt.ylabel('PDF')
```

```
95% confidence intervals are 163.606 and 170.634 cm
```

Out[15]:
```
Text(0, 0.5, 'PDF')
```



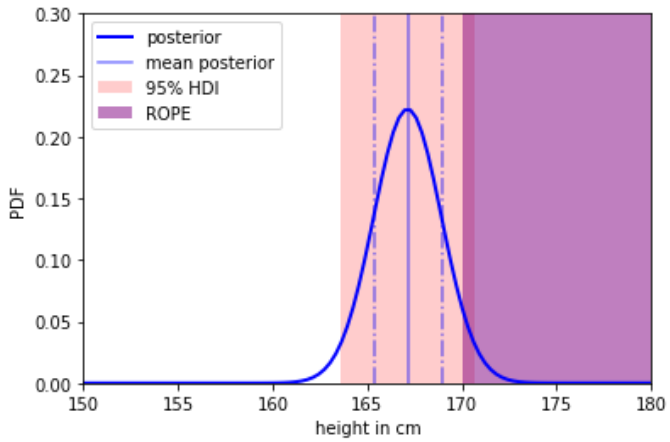
We want to test null hypothesis that height of a 20 year old student is > 170cm. Can we reject the null hypothesis based on our data?

Let's use everything > 170cm as a reasonable region (our ROPE).

In [16]:
```
%matplotlib inline

# can use norm.interval to get the interval out for 95%
HDI_1,HDI_2= norm.interval(0.95,posterior_mean,posterior_std)
print('95% confidence intervals are {:.3f} and {:.3f} cm'.format(HDI_1,HDI_2))

plt.plot(d,norm.pdf(d,posterior_mean,posterior_std),label='posterior',c='blue',lw=2)
plt.axvline(posterior_mean,label='mean posterior',color='blue',alpha=0.5)
plt.axvline(posterior_mean+posterior_std,color='blue',ls='-.',alpha=0.5)
plt.axvline(posterior_mean-posterior_std,color='blue',ls='-.',alpha=0.5)
plt.axvspan(HDI_1,HDI_2,facecolor='red', alpha=0.2, label = '95% HDI')
plt.axvspan(170,170+10,facecolor='purple', alpha=0.5, label = 'ROPE')

plt.legend(loc='upper left')
plt.xlim(150,180)
plt.ylim(0,0.3)
plt.xlabel('height in cm')
plt.ylabel('PDF')
```

```
95% confidence intervals are 163.606 and 170.634 cm
```

Out[16]:
```
Text(0, 0.5, 'PDF')
```

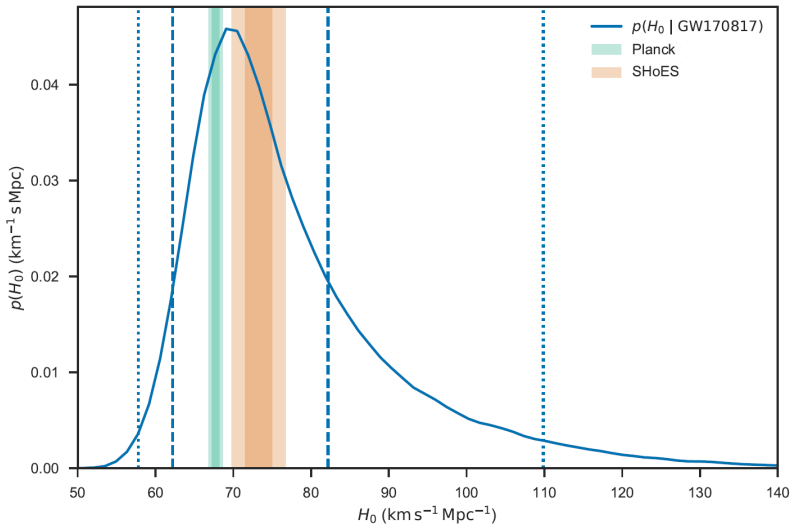


The ROPE is just within the 95% confidence interval, so although we cannot reject the null hypothesis that students heights are > 170cm as it lies *just within our credible region*, the data does not give strong evidence for the null hypothesis.

Some real life examples

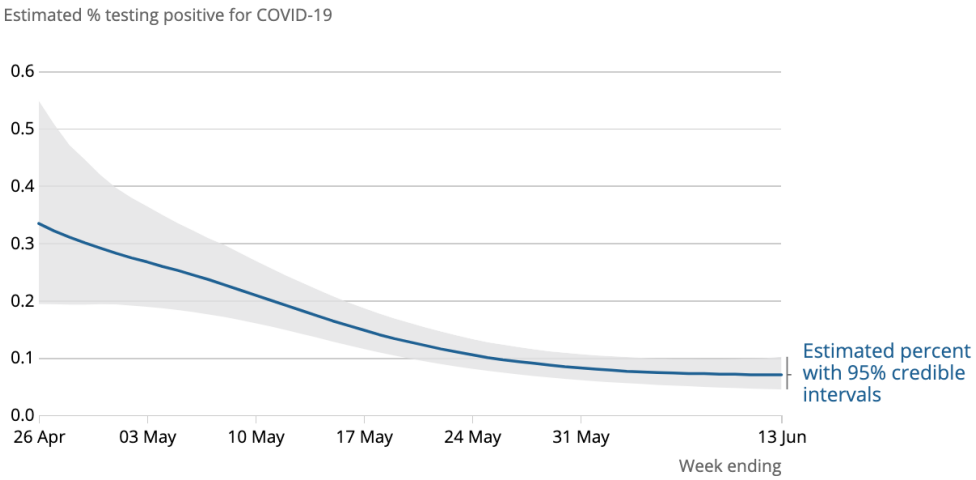
1. The incredible discovery of gravitational waves existing in the Universe in 2017 led to the award of a Nobel Prize. The gravitational waves were from a binary neutron star merger, also observed using electromagnetic telescopes sensitive to light across the entire electromagnetic spectrum. As well as discovering a new kind of physics, the observations allowed scientists to combine the galaxy distance measured from the gravitational-wave data with radial velocity measurements from the electromagnetic data. In doing so, they were able to make an entirely independent measurement of an important quantity in cosmology: the present-day expansion rate of the Universe. The figure below shows the measurement of the Hubble constant, H_0 (the expansion rate of the Universe) determined using the gravitational wave signals. The relative probability of different values of H_0 is represented by the solid blue curve- with a peak at $70\,\mathrm{km\,s^{-1}Mpc^{-1}}$.

   Dashed and dotted blue lines show the 68.3% and 95.4% credible intervals for H_0. For comparison the green and orange bands represent measurements of H_0 from two experiments using electromagnetic data: the Planck satellite and the SHoES analysis (exploding stars and twinkling lights). The darker and lighter coloured bands indicate 68.3% and 95.4% credible intervals for these values. Scientists were able to conclude from this figure that the Planck and SHoES measurements of the expansion rate of the Universe are not in agreement with each other at the 95.4% probability level. However, the gravitational-wave result is, consistent with both the Planck and SHoES values. *Figure reproduced from this site*.



1. The Office of National Statistics uses a Bayesian model and quotes 95% credible intervals for their infection rates from COVID-19 in England. They state that:

> *During the most recent week of the study, we estimate that 27,100 people in England had the coronavirus (COVID-19) (95% credible interval: 19,300 to 36,700).1 This equates to 0.05% (95% credible interval: 0.04% to 0.07%) of the population in England or around 1 in 2,000 people (95% credible interval: 1 in 2,800 to 1 in 1,500). This is based on statistical modelling of the trend in rates of positive nose and throat swab results.* - Source



Source: Office for National Statistics – COVID-19 Infection Survey

---

Now you are ready to tackle the **Chapter 6 quiz** on Learning Central and the Chapter 6 yourturn notebook.

Chapter 7

Random Numbers and Monte Carlo Markov Chains

Please ensure you have watched the Chapter 7 video(s).

You will learn the following things in this Chapter

- How to generate random data
- Understand the basic idea behind Monte Carlo Markov Chains and why they are useful
- How to construct a basic Metropolis algorithm
- The issues surrounding convergence, and how to test for it
- How to use MCMC to draw random points from a Bayesian posterior
- How to use Python programming to do the above.
- After completing this notebook you will be able to start CA 2.

Bayes Theorem with complicated prior distributions

In Chapter 6 we derived some analytical expressions to get the mean and varaince of some parameter using a sample, a prior and Bayes theorem. But what if our prior and likelihood distributions weren't so well-behaved as the ones discussed previously? Sometimes it is most accurate to model our data or our prior beliefs using distributions which don't have convenient shapes. What if our likelihood were best represented by a distribution with two peaks, and for some reason we wanted to account for some really wacky prior distribution, impossible to solve for analytically.

It turns out we can use something called a Monte Carlo Markov Chain (MCMC) to "sample" our posterior distribution without knowing the underlying distributions.

Monte Carlo

A Monte Carlo (MC) simulation is a computer model where we generate many different data sets (either from a model or the probability distribution of the real data) and analyse them like the real data.

The many different data sets are generated using random number generators

Imagine we're in a well known burger chain and want to guarantee that the super burger on offer has a particular mean weight. It's not feasible to weigh every super burger produced, so we use sampling techniques to randomly choose 100 super burgers from the shop. We calculate the mean of these 100 burgers and say that the population mean falls within a margin of error from what the mean of our sample is.

The uses of MC are incredibly wide-ranging, and have led to a number of groundbreaking discoveries in the fields of physics, game theory, and finance.

Origin of Monte Carlo

The Monte Carlo concept was invented by the mathematician Stanislaw Ulam working on the Manhattan Project in World War II. He used the tools of random sampling to model likelihoods of outcomes, originally applied to a card game (Monte Carlo Solitaire). This is what he thought:

What if we wanted to find the probability of getting blackjack (an ace and ten card)? Normally we would find the probability by counting the number of possible combinations of cards that would give Blackjack and dividing by all combinations of cards. Ok sounds good.

But what if there are more than 52 cards in the deck, or we don't know how many cards are in the deck. Right, so another option to get the probability is to sit down and play a 100/1000 games and record the outcomes and how many times we get Blackjack. But this would take a long time if we needed larger samples/better statistics.

What to do? Since we know from Chapter 3 that "As the number of identically distributed, randomly generated variables increases, their sample mean (average) approaches their theoretical mean" he realised we could use random sampling of the 100 games to make a data set much bigger and get closer to the true (population) value!

Generating Random Numbers

The most random way of constructing a simple point process is to assume that each event happens independently of every other event. We also need to assume that there is a constant probability per unit time of an event happening. (You may recognise this as a Poisson process).

But when you ask people to pick a number at random, it turns out that the distribution isn't quite as random as we might expect - it is well known amongst magicians that if you ask people to pick a number between 1 and 10, far more people choose one number over any other number.

When looking at your data to test if it is random or not can be very difficult to tell by eye. What is needed is a rigorous statistical analysis to confirm if the distribution in your image or data looks random by chance. Note also that even simple random processes, coincidences happens more frequently than one would naively expect.

In python we can use `from numpy import random` . We can generate random integers, and generate random numbers from a distribution for example, Binomial, Poisson, Gaussian etc. The `numpy` function `random.normal` allows you to do this, or we can use the `scipy.stats` function `norm` .

Tip: When using code to generate N random numbers, try it out on a small number first to check your code is doing what you want it to do. It can take a long time to run in Colab.

Example

Generate 5000 random datasets from a gaussian distribution with mean $\mu = 3$ and standard deviation $\sigma = 1.0$. Plot the datasets.

Solution

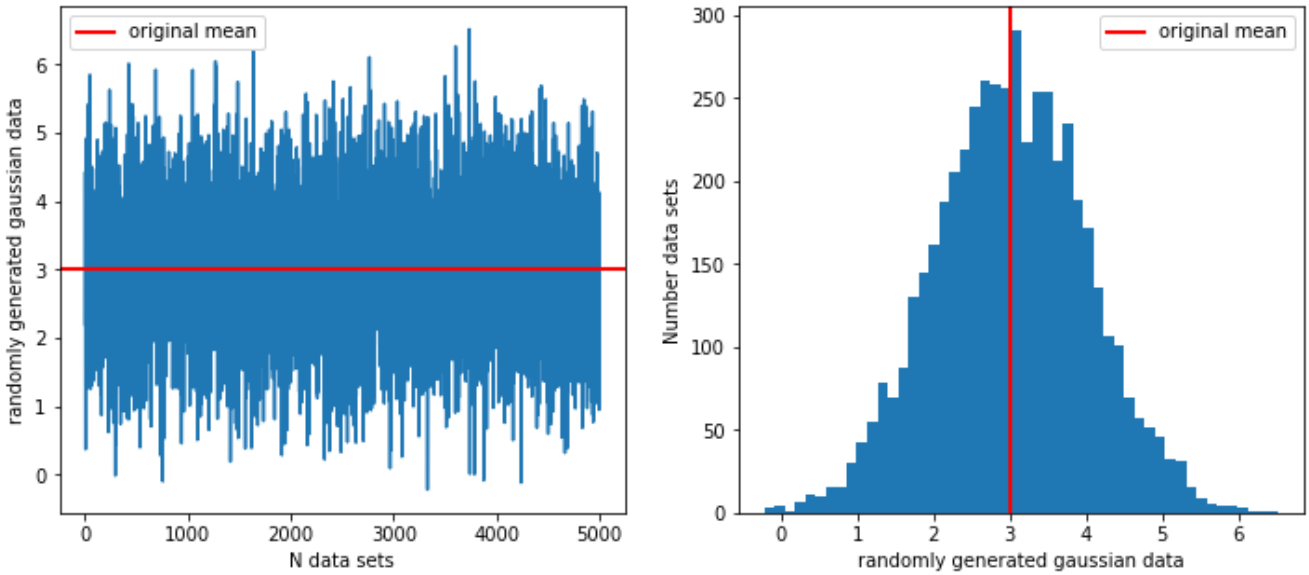
Click below to see the solution.

In [1]:

```
import numpy as np
import pylab as plt

%matplotlib inline

# set up our mean and sigma
mu = 3.
sigma = 1.0
nPoints=5000  # nice large number so that we can start to approximate the PDF

randomG_data = np.random.normal(mu,sigma,size=nPoints)

# what does this data look like - let's plot it!
plt.figure(figsize=(12,5))
plt.subplot(121)
plt.plot(randomG_data)
plt.xlabel('N data sets')
plt.ylabel('randomly generated gaussian data')
plt.axhline(3.0,label='original mean',c='r',lw=2)
plt.legend()
```

```
plt.subplot(122)
plt.hist(randomG_data,bins=50)
plt.axvline(3.0,label='original mean',c='r',lw=2)
plt.xlabel('randomly generated gaussian data')
plt.ylabel('Number data sets')
plt.legend()
```

Out[1]: <matplotlib.legend.Legend at 0x11a16d910>



Example

Generate some random data drawn from a normal distribution with 50 data points and 1000 data points. Generate random data from an exponential with 1000 data points, and plot the results.

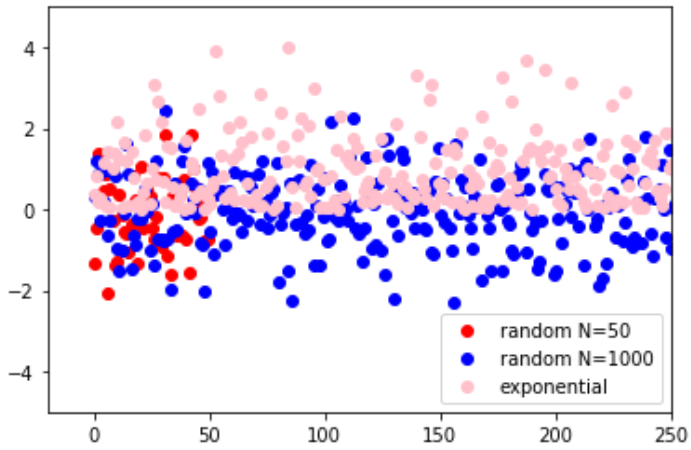
Solution

Click below to see the solution.

In [2]:

```
from scipy import stats
import pylab as plt # for plotting
# the line below makes the plot appear in the jupyter notebook
%matplotlib inline


# Draw 50 random data points from a normal (Gaussian) distribution
test_b = np.random.normal(size=50)

# Draw 1000 random data points from a normal (Gaussian) distribution
test_c = np.random.normal(size=1000)

# Draw 200 random data points from an exponential distribution
test_d = np.random.exponential(size=1000)

# generate some x arrays with same length of y data sets

x_c = np.arange(1000) # 1000 data points
x_b = np.arange(50) # 50 data points

plt.scatter(x_b,test_b,color='red',label='random N=50')
plt.scatter(x_c,test_c,color='blue',label='random N=1000')
plt.scatter(x_c,test_d,color='pink',label='exponential')

plt.xlim(-20,250)
plt.ylim(-5,5)
```

```
plt.legend(scatterpoints=1,loc='lower right')
plt.show()
```



Example

Generate 1000 sets of random data to represent flipping a coin 10 times and getting 5 heads. Plot the distribution.

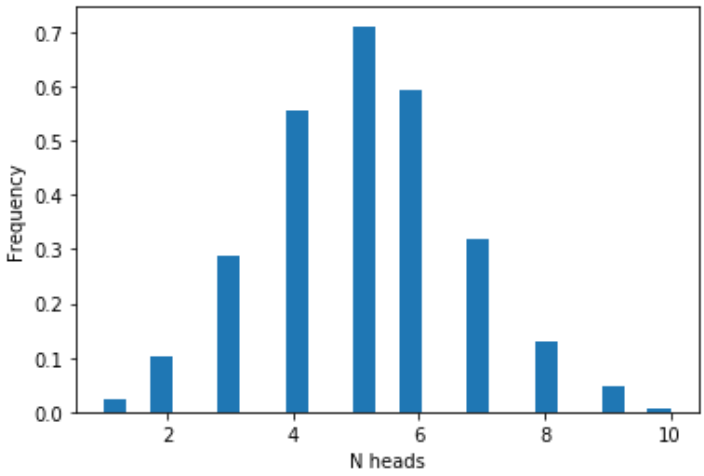
Solution

Click below to see the solution.

Flipping a coin is a Binomial distribution with $N = 10$ and $\nu = 5$.

In [3]:
```
N=10.
nu=5.

p = nu/N

data = np.random.binomial(N,p,1000)

plt.hist(data,bins=25,density=True)

plt.xlabel('N heads')
plt.ylabel('Frequency')
```

Out[3]:
```
Text(0, 0.5, 'Frequency')
```



Example

A company drills 9 wild-cat oil exploration wells, each with an estimated probability of success of 0.1. All nine wells fail. What is the probability of that happening? Tip: generate some random data.

Solution

Click below to see the solution.

Let's do 20,000 trials of the model, and count the number that generate zero positive results. This is a binomial distribution (pass/fail) with N=9 and $\mu = 0.1$.

In [4]:
```
N=9
p = 0.1
N_trials = 20000

# count the number of trials with binomial outcome = 0
n_fails = sum(np.random.binomial(N,p,N_trials)==0)

prob = n_fails/np.float(N_trials)

print('Number of times that all 20000 wells fail is {:.3f}'.format(n_fails))
print('Probability wells fail is n(fail)/N(trials) {:.3f}'.format(prob))
```

```
Number of times that all 20000 wells fail is 7683.000
Probability wells fail is n(fail)/N(trials) 0.384
```

Example

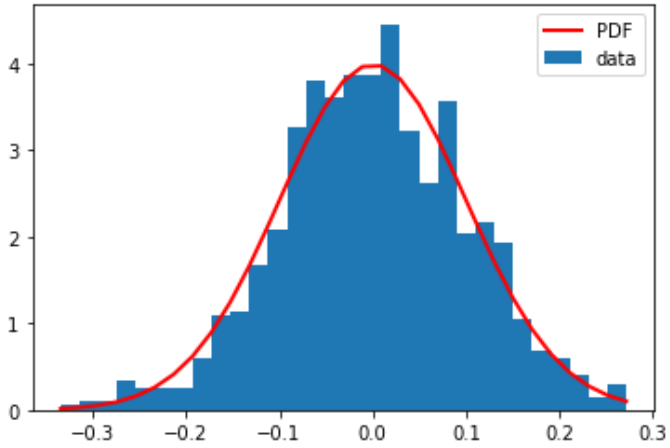
Generate 1000 random data points from a gaussian with mean = 0 and sigma = 0.1. Plot a histogram.

Overlay the gaussian distribution with the same mean and standard deviation.

Solution

Click below to see the solution.

In [5]:
```
from scipy import stats
import pylab as plt # for plotting
# the line below makes the plot appear in the jupyter notebook
%matplotlib inline

# a function that outputs the probability density function
# of a normal distribution given a mean and standard deviation
def pdf_func(mean,std,x):
    result = 1./(std * np.sqrt(2 * np.pi)) * np.exp(- (x - mean)**2 / (2 * std**2) )
    return result

# generate data from random distribution with mean = 0 and standard deviation = 0.1
mean_data = 0
std_data = 0.1
random_data = np.random.normal(mean_data,std_data, 1000)

# plot the histogram of data generated above and produce bins for pdf function
count, bins, ignored = plt.hist(random_data, 30, density=True,label='data')
# now
plt.plot(bins,pdf_func(mean_data,std_data,bins), linewidth=2, color='r',label='PDF')

plt.legend()
plt.show()
```



Monte Carlo Markov Chain

A Markov chain is a sequence of events that are related probabalistically, where each outcome determines what happens next.

- Each future step in the sequence is related only to the present step
- It should have no memory of the steps before

The main goal of MCMC is to provide an approximation to a probability distribution. This is extremely useful. Although we have random number generators for uniform, normal, and sometimes binomial, Bernoulli or Poisson distributions, these can only get us so far. For example, the kind of pdfs that commonly appear in Bayesian analysis are often complicated functions, which are hard, or even impossible, to put in an analytic pdf form. MCMC allows us to get around these problems.

MCMCs stochastically explore the parameter space in such a way that the histogram of their samples produces the target distribution. Here we will introduce the **Metropolis algorithm**, the first version of the MCMC and is one of the top algorithms used in computational analysis. All other MCMCs tend to be a special case of the Metropolis so we'll use this in our course.

One of the most common uses of MCMCs is to obtain a representative sample of points from the posterior,

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

In this case we make $p(D|\theta)p(\theta)$ our target distribution for the MCMC, and the chain of points that results from our converged MCMC run is then a random sample from the Bayesian posterior. This is especially important if our posterior is multivariate (i.e. θ is not one variable, but rather represents a whole host of variables), since then the simpler grid approximations become numerically unwieldy. And once again, we need not worry about trying to normalise $p(D|\theta)p(\theta)$, since MCMC does not care if the function that it is evaluating is normalised or not (only that it is non-negative).

Once one has a representative sample from the posterior, there are many things that we can do! For example, we can calculate the mean (expectation value) from the points, the variance, etc. That is often enough, which is why we do not always need to produce a normalised posterior.

Even if a normalised posterior is required, it can often be done simply by creating a normalised histogram of the resulting MCMC points. If the multivariate, this can be done on each variable in turn -- that is, first making a histogram of all points that have θ_1, then another with all points that have θ_2, etc. By doing this, you are essentially *marginalising* over the other variables. Once you have your normalised posterior, you can then use it to calculate the HDI of the various marginalised posteriors, which can be useful for hypothesis testing.

The Metropolis Hastings Algorithm

We are trying to draw numbers from a target distribution that we will denote as $P(\theta)$. The following steps outline the algorithm

Make a first guess at the dependent variable (or variables) $\theta_{current}$. We then propose to make a random step in the variables to a new location

$$\theta_{proposed} = \theta_{current} + \Delta\theta.$$

If the value of the function at $\theta_{proposed}$ is greater than that at $\theta_{current}$ -- i.e.

$$P(\theta_{proposed}) > P(\theta_{current})$$

then we accept a move to the point $\theta_{proposed}$. However, if

$$P(\theta_{proposed}) < P(\theta_{current})$$

then we consider the probability of the move to $\theta_{proposed}$ as

$$p_{move} = \frac{P(\theta_{proposed})}{P(\theta_{current})}.$$

All this can actually be tidied up by simply writing

$$p_{move} = \min\left(\frac{P(\theta_{proposed})}{P(\theta_{current})}, 1\right)$$

If $p_{move} < 1$, then we still have a decision to make. We now use draw a *uniform random number* between 0 and 1 which we will denote as u_{rnd}. If

$$u_{rnd} \leq p_{move}$$

then the proposed move is accepted. If not, we reject the move and stay where we are.

Once the decision is made to accept/reject the point, we update $\theta_{current}$ with the new position (which could be the same position if the move was rejected!) and store its value.

Repeat by going back to the first step, only now we don't have to guess since we have just moved to $\theta_{current}$.

Key features of MCMC

One of the key features of MCMC is that the decision to move is dependent on the ratio of the target distribution,

$$p_{move} = \min\left(\frac{P(\theta_{proposed})}{P(\theta_{current})}, 1\right)$$

This means that the function that holds the shape of the target distribution does not need to be normalised. In fact, that is often the point. The functions for which we use MCMC are often those which are difficult or impossible to normalise - this is really useful for the denominator of Bayes.

The only real condition on $P(\theta)$ is that it is positive (negative probabilities are not a good idea).

To perform an MCMC, we must be able to do several things. We list them here for clarity.

- Generate a random value from the proposal distribution (i.e. create $\theta_{proposed}$).
- Evaluate the target distribution $P(\theta)$ at any point θ
- Draw from a uniform random distribution

In the list above we used the term proposal distribution, which refers to the θ-space from which we are drawing new locations. In most MCMC algorithms, this is done by drawing $\theta_{proposed}$ from a **normal distribution** centred on

$\theta_{current}$, with some width $\Delta\theta$ -- i.e. $N_{\theta_{current},\Delta\theta}(\theta)$. In principle, we are free to chose any function for the proposal.

Which bit is Monte Carlo and which bit Markov Chain?

We can see from the steps above that MCMC is a Monte Carlo process, in that the algorithm depends on random variables. A Markov Chain is any series of steps in which each new step has no memory of the step before. Above we see that the Metropolis algorithm is just such a process - the decision to move to $\theta_{proposed}$ depends only on $\theta_{current}$ and $\theta_{proposed}$. After the move, this is all forgotten, and a new proposal is made, and so on.

Example

We will return to the heights example from Chapter 6 where we saw that the prior $\times$ likelihood resulted in a normally-distributed posterior probability. Here we will prove that an MCMC can recover an underlying posterior distribution.

We have 10 measurement of student heights in cm = [169.6,166.8,157.1,181.1,158.4,165.6,166.7,156.5,168.1,165.3]

The variance in the measurement is 50cm. The prior data is normally distributed with mean = 170cm and standard deviation of 3cm.

1. Define a function in python to calculate the posterior probability for a value theta, given your likelihood and prior data.
2. Convert the description above for the Metropolis MCMC into a code. You will need to calculate the probability for the current value of theta at each step by calling your function from a.
3. Plot your MCMC values of theta to check if your MCMC has converged.
1. Plot the posterior distribution derived from the analytic form and compare them.

Solution

Click below to see the solution.

How do we approach this?

For 2 gaussians (the prior and the likelihood are both gaussian) we simply need to mulitply two normal distributions together such that

$$\text{prob} = \text{norm}(\text{likelihood mean}, \text{likelihood err}) \times \text{norm}(\text{prior mean}, \text{prior err}).$$

As we're interested in only getting the mean and std out of the MCMC, we don't need to worry about normalising the numerator of Bayes.

We do need to think about how many random steps we want to take. Let's first try 100.

Tip: Best to scribble down the method on paper before trying to code the MCMC.

In [6]:

```
from scipy.stats import norm
%matplotlib inline
# from last week - analytical values

posterior_mean= 167.120
posterior_std= 1.793

# define a function for the probability
# prob(posterior) = normal(likelihood) x normal(prior)
def posterior(theta):
    data=[169.6,166.8,157.1,181.1,158.4,165.6,166.7,156.5,168.1,165.3]
    X_hat=np.mean(data)
```

```
    n = len(data)
    var = 50.
    sig = np.sqrt(var)/np.sqrt(10.)
    mu_0 =170.
    s_0 = 3.
    A = 1./(sig*np.sqrt(2*np.pi))
    B= (theta - X_hat)**2./(2*sig**2.)
    A_prior = 1./(s_0*np.sqrt(2*np.pi))
    B_prior= (theta - mu_0)**2./(2*s_0**2.)
    val = norm.pdf(theta,X_hat,sig)*norm.pdf(theta,mu_0,s_0)
    return val


# set up MCMC step paramaters
N_mcmc = 100
theta_current = np.zeros(N_mcmc+1)
# use our prior as a guess
theta_current[0] = 160.

# choose a value for width of normal distribution to get the step in height
# this is between the prior and the likelihood values
sigma_mcmc = 2

for i in range(N_mcmc):
    p_current = posterior(theta_current[i]) # put current value in posterior equation
    dtheta = np.random.normal(0,sigma_mcmc) # randomally draw a value of theta to trial
    theta_proposed = theta_current[i] + dtheta #get new proposed theta (random theta + stepsize)
    p_proposed = posterior(theta_proposed) # calculate posterior p for proposed theta

    #  keep this value if probability proposed theta greater than the current prob
    if p_proposed > p_current:
        theta_current[i+1] = theta_proposed
    else:
    # if probability lower
    # use the ratios of probability to define probability of whether we move to that value or not
        p_new_move = p_proposed/p_current
        # generate random number for probability
        u_random = np.random.uniform(0,1)
        # if u_random < p_new_move, then accept, if not, reject
        if u_random <= p_new_move:
            theta_current[i+1] = theta_proposed
        else:
            theta_current[i+1] = theta_current[i]

# get mean + std from mcmc generated samples
mean_mcmc=np.mean(theta_current)
std_mcmc=np.std(theta_current)

print('The mean height from the MCMC is {:.2f} +/- {:.2f} cm'.format(mean_mcmc,std_mcmc))
```

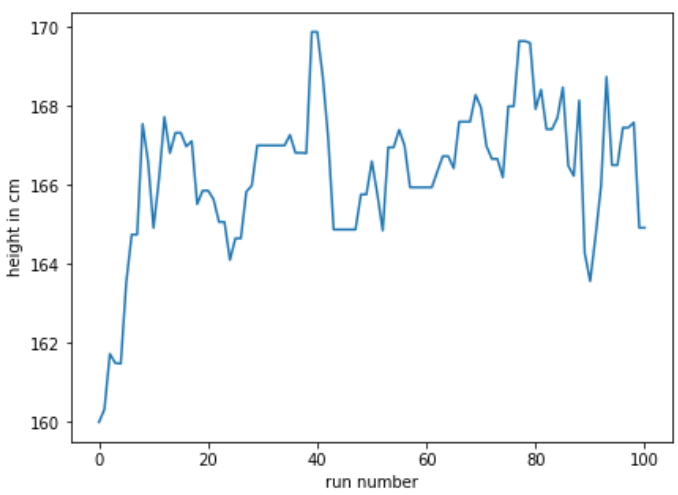
```
The mean height from the MCMC is 166.33 +/- 1.82 cm
```

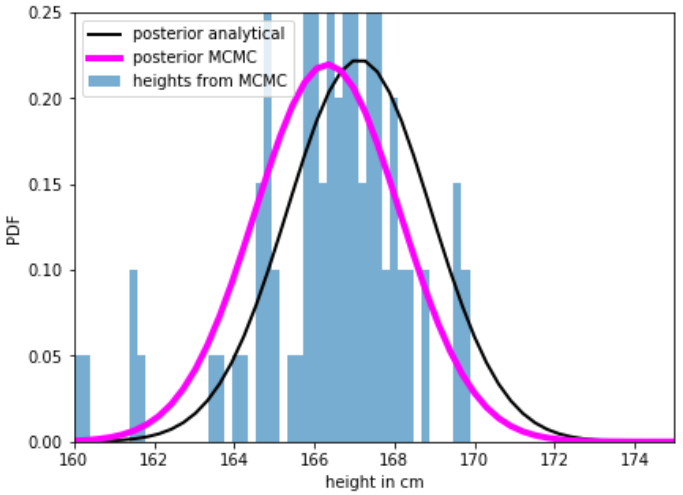
In [7]:

```
# tip : put the plotting routines outside the cell of the
# MCMC otherwise you'll need to do your N runs everytime you change a plotting feature
# generate x for plotting

plt.figure(figsize=(15,5))
plt.subplot(121)
plt.plot(theta_current)
plt.xlabel('run number')
plt.ylabel('height in cm')

x = np.linspace(150,180,100)
posterior_mcmc=norm.pdf(x, mean_mcmc, std_mcmc)

plt.subplot(122)
```

```
#plots the analytic posterior
plt.plot(x,norm.pdf(x,posterior_mean,posterior_std),label='posterior analytical',c='black',lw=2)

# plots the mcmc generated posterior
plt.hist(theta_current,bins=50,density=True,label='heights from MCMC',alpha=0.6)
plt.plot(x,posterior_mcmc,label = 'posterior MCMC',c='magenta',lw=4)
plt.legend(loc='upper left')
plt.xlim(160,175)
plt.ylim(0,0.25)
plt.xlabel('height in cm')
plt.ylabel('PDF')
```

Out[7]: `Text(0, 0.5, 'PDF')`





You can see that the above plots are not great - *left*- it looks like our MCMC is not behaving very well and is very clumpy, *right* - we expect the posterior to be a normal (see Chapter 6) and yet the values of heights we've gotten out of the MCMC are not very normal. The analytical posterior (black) is also not very close to the MCMC posterior...

Let's instead try and N of 100,000 and see what happens. This will take much longer to run but will be worth the wait!

In [8]:

```
N_mcmc = 100000
theta_current = np.zeros(N_mcmc+1)
# use our prior as a guess
theta_current[0] = 160.

# choose a value for width of normal distribution to get the step in age
# this is between the prior and the likelihood values
sigma_mcmc = 2

for i in range(N_mcmc):
    p_current = posterior(theta_current[i]) # put in posterior equations
    dtheta = np.random.normal(0,sigma_mcmc) # randomally draw a value of theta to trial
    theta_proposed = theta_current[i] + dtheta #get new theta
    p_proposed = posterior(theta_proposed)

    #  keep value if probability proposed theta greater than the current prob
    if p_proposed > p_current:
        theta_current[i+1] = theta_proposed
    else:
    # if probability lower
    # use the ratios of probability to define probability of the move
        p_new_move = p_proposed/p_current
        # generate random number for probability
        u_random = np.random.uniform(0,1)
        # if u_random < p_new_move, then accept, if not, reject
        if u_random <= p_new_move:
```

```
            theta_current[i+1] = theta_proposed
        else:
            theta_current[i+1] = theta_current[i]

# get mean + std from mcmc generated samples
mean_mcmc=np.mean(theta_current)
std_mcmc=np.std(theta_current)

print('The mean height from the MCMC is {:.2f} +/- {:.2f} cm'.format(mean_mcmc,std_mcmc))
```

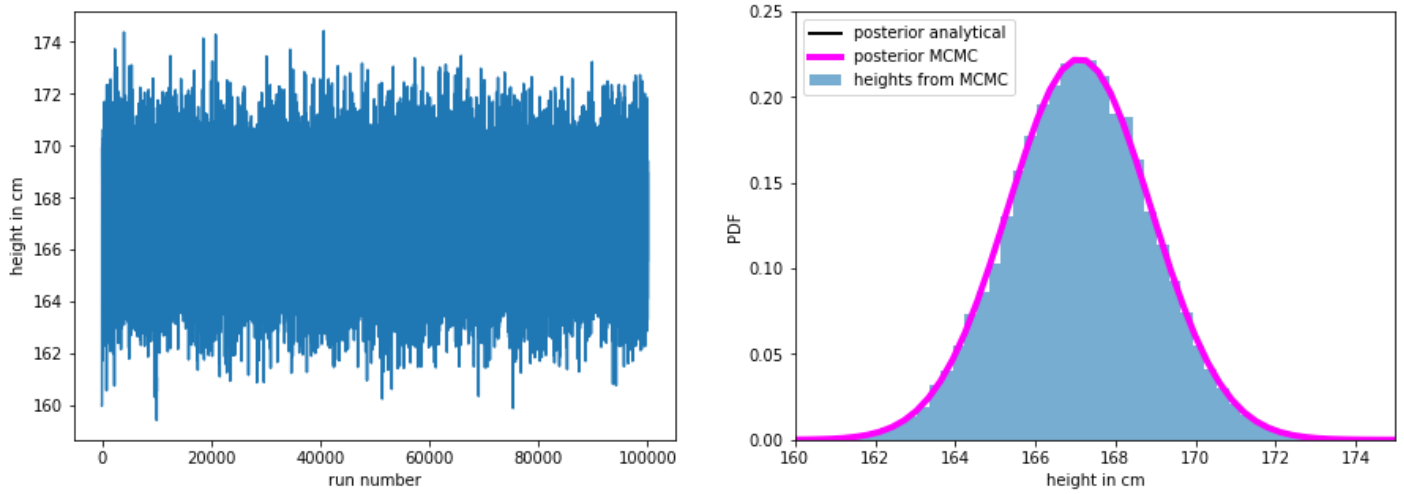
```
The mean height from the MCMC is 167.10 +/- 1.79 cm
```

In [9]:

```
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.plot(theta_current)
plt.xlabel('run number')
plt.ylabel('height in cm')

x = np.linspace(150,180,100)
posterior_mcmc=norm.pdf(x, mean_mcmc, std_mcmc)

plt.subplot(122)
#plots the analytic posterior
plt.plot(x,norm.pdf(x,posterior_mean,posterior_std),label='posterior analytical',c='black',lw=2)

# plots the mcmc generated posterior
plt.hist(theta_current,bins=50,density=True,label='heights from MCMC',alpha=0.6)
plt.plot(x,posterior_mcmc,label = 'posterior MCMC',c='magenta',lw=4)
plt.legend(loc='upper left')
plt.xlim(160,175)
plt.ylim(0,0.25)
plt.xlabel('height in cm')
plt.ylabel('PDF')
```

Out[9]:

```
Text(0, 0.5, 'PDF')
```



This is much better - things look like they're more stable now and now we can see the MCMC posterior data returns a gaussian and agrees well with the analytical posterior for a normal likelihood and prior.

Pitfalls

1. As you can imagine, it is possible to unknowingly start your MCMC a long way from the underlying peak of the target distribution, and so the first n points in the chain are usually rejected, since their location may be biased. This period of n skipped positions is known as the **burn in**. Given that modern computers are extremely fast, it is common to throw away many thousands of points to ensure that the remaining sample is

not biased by the initial conditions. Deciding how many points to skip depends on how well you think the MCMC is doing at sampling your target distribution. We will discuss this more further down. Generally speaking, if you start close to the peak of the target distribution, your burn-in will be shorter than if you start in a region of low (and flat) probability.

1. Clearly the value of $\Delta\theta$ is also something that needs to considered carefully. If $\Delta\theta$ is too small, then the MCMC will explore the parameter space very slowly. It will be accurate, and because $P(\theta_{proposed})/P(\theta_{current}) \sim 1$, the proposals will often be accepted, but clearly it take a very long time to fully sample the distribution. This is a particular problem if you also happen to start your MCMC far away from the peak.

1. On the other hand, if you make $\Delta\theta$ too large, the ratio of $P(\theta_{proposed})/P(\theta_{current})$ will often be very small, and so there will be very little chance of accepting the proposed move. The result is that you will have a large number of duplicate values in your MCMC sample. One benefit of the large $\Delta\theta$ is that it can often move itself out from regions of low probability. However it can also overshoot peaks!

1. Another problem associated with the choice of $\Delta\theta$ is the clustering of data points. This tends to happen for cases in which the proposal distribution is Normal, and $\Delta\theta$ is too small. The cluster arises from the probabilities distribution of the Normal -- although it is centred narrowly around $\theta_{current}$ there is a non-zero probability of making a big jump (several σ away). So the MCMC chain tends to loiter in a spot for a long time, before making a big jump to a new patch of the parameter space.

1. Another problem with MCMC is that it can get stuck in local maxima. For example, imagine a case with a strong prior and conflicting data. In this case there will be (at least) two peaks, one from the data and one from prior. The MCMC walk might loiter around one of the these peaks for a while, before venturing down the slope to lower probability regions of the parameter space. What that occurs, there is a chance that the walk then wander up the other, unexplored peak, but there's also a chance that it simply goes back up the peak that it just came from! Note that this well-explored peak might not be the global maximum - it could be a small, relatively insignificant peak at an overall low probability, but one that we just happened to stumble upon due to our choice of initial starting position. We then have to wait a very long time before the points start to represent the overall distribution.

What all this means is it takes a bit of trial and error to get the standard Metropolis algorithm to converge in the way you want and it can be *very sensitive to your initial guess*. However, it has the advantage that it is very easy to code, and that computers are now fast enough to allow you explore the parameter space and fine-tune the sampler to hone in on the optimal values. We can also do some tests to determine if the MCMC is acting sensibly.

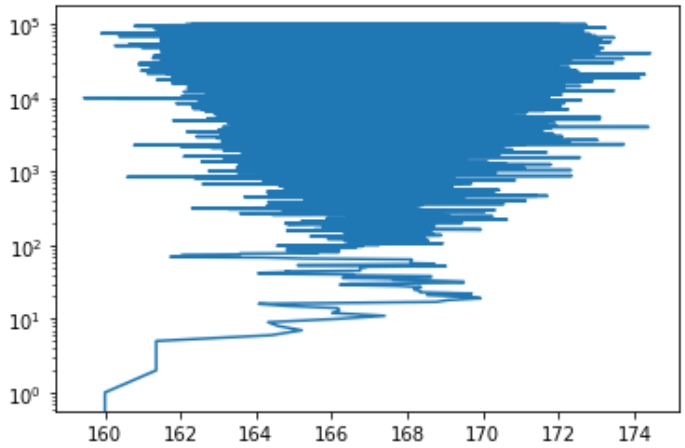
Testing for convergence

The simplest way to assess convergence is to monitor the properties of your sample until any trends have disappeared, such as an increasing mean, or oscillating skewness, etc. It is also good monitor this over different lengths of N within your chain. For example, after 1 million points have been generated, you could try comparing the various moments of the first 1000 points, with the last. This will also start to give you a feel for where the burn-in period ends. Or you plot the mean, std, variance for each 100 points, and see at which point in the chain these values start to settle down.

We can derive an acceptance ratio for our data based on the number of proposed values that are accepted versus the total number of steps: $N_{\rm accepted}/N_{\rm steps}$, which should be ~ 0.4 for 1-dimensional cases like the ones here.

In [10]:
```
# let's generate the walk for plotting
x_mcmc=np.linspace(0,N_mcmc,N_mcmc+1)

plt.semilogy(theta_current,x_mcmc)
```

Out[10]: [<matplotlib.lines.Line2D at 0x1a1f7a2510>]



In code speak - this would look like

```
burnin=100
burnt_sample = sample[burnin:]
```

This would take out the first 100 entries in your MCMC output.

A more worrying problem is the case in which your well sampled chain has become stuck in a not very important peak. Thankfully, a simple solution is at hand: do several chains with different starting positions and/or step sizes, and allow them to independently explore the pdf. Again you can compare the properties of the various chains at different stages, to see how well they are doing. If one chain has a very different mean, say, than the others, then it is likely that it has become stuck in far-off maximum.

For cases where the MCMC results look clumpy even for a sensible number of walks ($N > 10000$), you can perform *thinning*, whereby you only accept each nth value from the chain and throw away the rest. To decide on n , you can either simply look at the data.

To thin the data in a code, this would look like `length=20`
`independentsamples = sample[::length]`

This would pull out numbers every 20 points+initial guess such that you would return 6 independent samples if your length is 20 and your MCMC was only 100 trials. If you did 1000 trials, this would then return 51 values from your original 1000.

You could also perform some sort of correlation statistic on sets of n points, to see where the correlations end. To do this we want to check how correlated a sample i is to sample $i-1$, $i-2$ etc. Matplotlib has a library for this `plt.accor(data–mean(data),maxlags=10)` , see example below. This checks the differences in a sample of data ahead of the current sample X_{i+k}, where the current sample is X_i. We refer to the lag k as the range ahead of the current sample:

$$r_k = \frac{\sum(X_i - \hat{X})(X_{i+k} - \hat{X})}{\sum(X_i - \hat{X})^2}.$$

Another convergence test which can be rather powerful is testing what you get if you re-run your MCMC with different initial conditions or step sizes. If they have converged, they should give you the same result.

Important

The values returned from MCMC are not strictly the same as those that you would get from, say, from a real random process, due to the pseudo-random number generates we have in computers.

Example

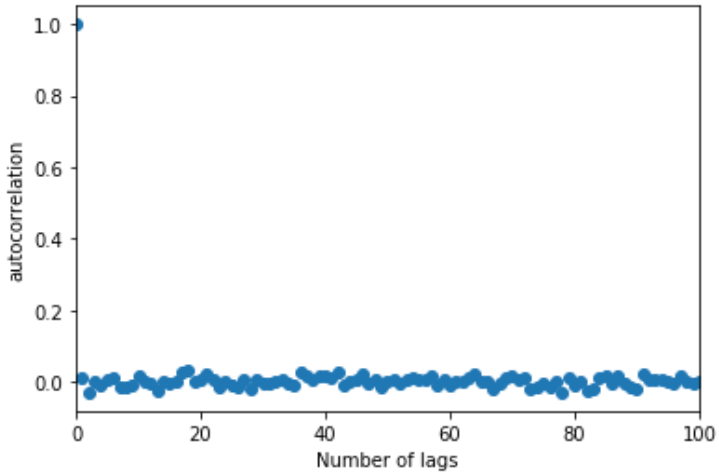
Plot the autocorrelation in the random sample from the top of this notebook.

Solution

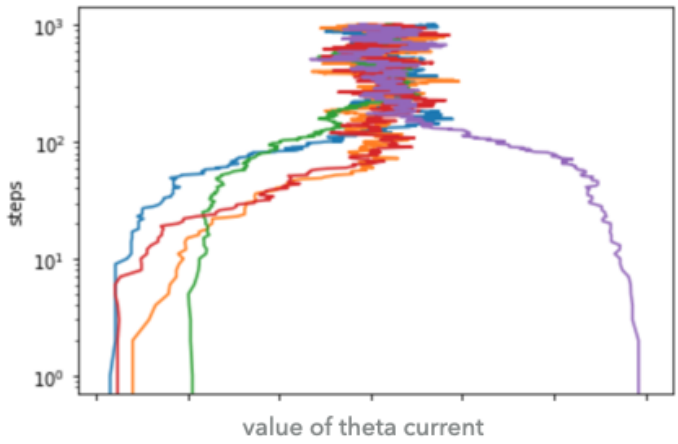
Click below to see the solution.

In [11]:
```
#plot autocorrelation for the random data generated above
plt.acorr(randomG_data-np.mean(randomG_data), maxlags=100, normed=True, usevlines=False);
plt.xlim(0,100)
plt.ylabel('autocorrelation')
plt.xlabel('Number of lags')
```

Out[11]:
```
Text(0.5, 0, 'Number of lags')
```



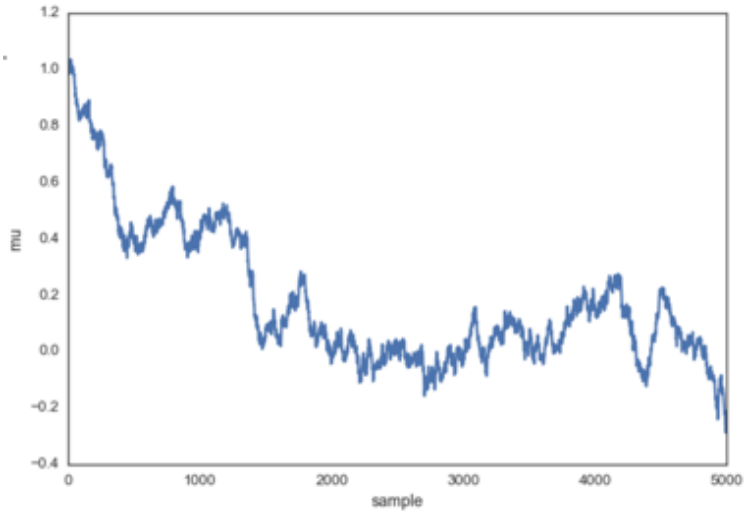
What about running different starting positions/initial guesses or different step sizes. This can be done by running the MCMC over a number of different theta_current values for example.

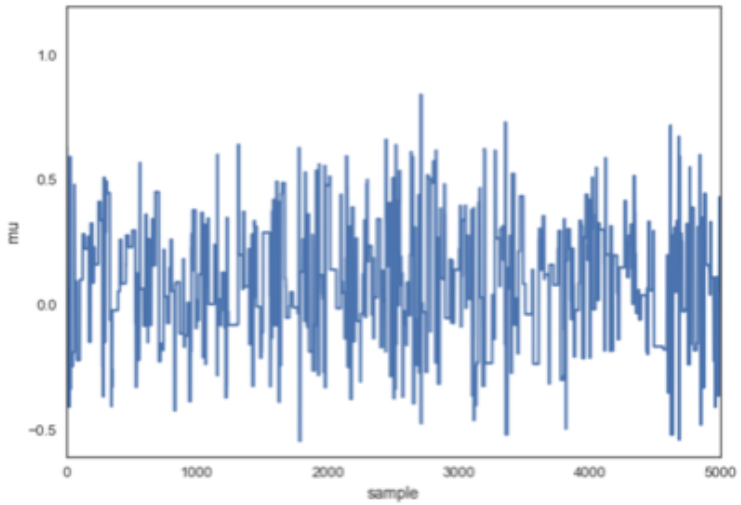


Notice how they all converge.

Below we shown an illustration of the resulting MCMC for a large number of trials but with too small (left) and too large (right) a step size. If $\Delta\theta$ is too small, then the MCMC will explore the parameter space very slowly. It will be accurate, and because $P(\theta_{proposed})/P(\theta_{current}) \sim 1$, the proposals will often be accepted, but clearly it take a very long time to fully sample the distribution. This is a particular problem if you also happen to start your MCMC far away from the peak. If you make $\Delta\theta$ too large, the ratio of $P(\theta_{proposed})/P(\theta_{current})$ will often be very small, and so there will be very little chance of accepting the proposed move. The result is that you will have a large number of duplicate values in your MCMC sample. It can also overshoot peaks!

A too small stepsize can lead to the MCMC not converging due to a very slow exploration of the probability space over a small range (top figure) or even too large (bottom figure) a stepsize can result in the MCMC jumping too much and potentially missing locations of probability peaks.





How does this magic work?

Quote from Stack Exchange answer

> Imagine you want to find a better strategy to beat your friends at the board game Monopoly. Simplify the stuff that matters in the game to the question: which properties do people land on most? The answer depends on the structure of the board, the rules of the game and the throws of two dice. One way to answer the question is this. Just follow a single piece around the board as you throw the dice and follow the rules. Count how many times you land on each property (or program a computer to do the job for you). Eventually, if you have enough patience or you have programmed the rules well enough in you computer, you will build up a good picture of which properties get the most business and probability of landing there. This should help you win more often. Where you land next only depends on where you are now, not where you have been before and the specific probabilities are determined by the distribution of throws of two dice. This is basically a combination of the monte carlo (playing the game lots of times and markov chain - next step only depends on current location).

From above, the MCMC uses

$$p_{\text{move}} = \frac{P(\theta_{\text{proposed}})}{P(\theta_{\text{current}})}$$

we get an acceptance probability. You can already see that if p_{proposed} is larger, that probability will be > 1 and we will definitely accept. However, if p_{current} is larger, say twice as large, there will be a 50% chance of moving there.

We can show this by computing the acceptance ratio over the normalized posterior and seeing how it's equivalent to the acceptance ratio of the unnormalized posterior (lets say θ_{current} is our current position, and θ_{proposed} is our proposal),

$$\frac{P(\theta_{\text{proposed}}|x)}{P(\theta_{\text{current}}|x)} = \frac{\frac{P(x|\theta_{\text{proposed}})P(\theta_{\text{proposed}})}{P(x)}}{\frac{P(x|\theta_{\text{current}})P(\theta_{\text{current}})}{P(x)}}$$

$$\frac{P(\theta_{\text{proposed}}|x)}{P(\theta_{\text{current}}|x)} = \frac{P(x|\theta_{\text{proposed}})P(\theta_{\text{proposed}})}{P(x|\theta_{\text{current}})P(\theta_{\text{current}})}$$

In words, dividing the posterior of proposed parameter setting by the posterior of the current parameter setting, $P(x)$ - that nasty quantity we can't compute - gets canceled out. So we're actually dividing the full posterior at one position by the full posterior at another position (no magic here). That way, we are visiting regions of high posterior probability relatively more often than those of low posterior probability.

From Hogg et al 2018, we can see this the following way:

$1 = \int p(\theta) d\theta$

Given this pdf $p(\theta)$, we can define the expectation value $E_{p(\theta)}[\theta]$ for θ as

$E_{p(\theta)}[\theta] = \int \theta p(\theta) d\theta$

and we can do the same for any quantity expressed as a function of θ (eg $g(\theta)$):

$E_{p(\theta)}[g(\theta)] = \int g(\theta) p(g(\theta)) d\theta.$

We've already seen in earlier Chapters that the expectation value is given by the mean value of the PDF of θ. If we then sample the PDF a number k times from $1 \ldots K$ ie $\{\theta_k\}_{k=1}^{K}$, then the integrals get replaced with the sums over each sample θ_k and

$E_{p(\theta)}[\theta] = \frac{1}{K} \sum_{k=1}^{K} \theta_k.$

Similarly for $g(\theta)$

$E_{p(\theta)}[g(\theta)] = \frac{1}{K} \sum_{k=1}^{K} g(\theta_k)$

which is the definition of the mean, thus repeated sampling of a PDF of θ also gives the expectation value.

Now you are ready to tackle the **Chapter 7 quiz** on Learning Central and the Chapter 7 yourturn notebook.

Chapter 8

Fitting models to Data

Please ensure you have watched the Chapter 8 video(s).

You will learn the following things in this Chapter

- Analytic expressions of fitting a straight line to data: linear regression.
- Understand how errors need to be accounted for.
- How to use Python functions to fit curves or non linear data.
- How to evaluate the goodness of your fit.
- Be able to use an MCMC to fit data with a model.
- Understand how bootstrapping can give us sampling statistics.
- How to use Python programming to do the above.
- After completing this notebook you will be able to finish CA2.

Explaining our data with models

Here we will review one of the most important aspects of day-to-day data analysis - given data and a model, what the values for the parameters in that model. In other words, **parameter estimation**, one of the most important goals of data analysis.

Fitting straight line models to data - linear regression

Probably one of the most common things scientists do, is fit a straight line to data. To do this we want to express the fit as a linear relation in the form of $y = A + Bx$ where A is the y-intercept and B is the gradient ie the true value of y_i is given by $y_i = A + Bx_i$. However we have to be careful with regards to errors in y or x.

Normally what happens is that the experimenters make a scatter plot of x and y, and notice that there's some sort of linear-looking relationship, and decide it would be good to have a mathematical form for this relationship, and so try to fit a straight line. They may even have calculated the covariance or correlation statistics from a data set, and realised that two of the variables have a strong correlation. How do we determine the best fit line to find the values of A and B parameters that describe our data and give us the “true y”?

Derivation: y=A+Bx parameters

Once again, we are going to look towards the principle of maximum likelihood for help. If we knew what A and B were, we could, for a given value of x_i, compute the corresponding value of y_i,

$$\text{true value for } y_i = A + Bx_i.$$

The measured values of y_i are therefore drawn from a normal distribution of width σ_y that is centred on the true value y_i. So we can write the probability of obtaining any given measurement y_i as,

$$p_{A,B}(y_i) \propto \frac{1}{\sigma_y} e^{-(y_i - A - Bx_i)^2 / 2\sigma_y^2}.$$

The subscripts A and B indicate that this probability depends on the unknown values of A and B.

Taking this a stage further, we can now write the probability of obtaining our entire set of measurements as

$$p_{A,B}(y_1, y_2, \ldots, y_N) = p_{A,B}(y_1) \cdots p_{A,B}(y_N)\; p_{A,B}(y_1, y_2, \ldots, y_N) \propto \tfrac{1}{\sigma_y^N} e^{-\chi^2/2}$$

where χ^2 is our old friend from Chapter 5,

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - A - Bx_i)^2}{\sigma_y^2}.$$

As before, we are interested in recovering the values of A and B that maximise the probability of obtaining the set of the observed measurements, (x_i, y_i). The joint probability of the measurements is obtained when χ^2 is a minimum, so we need to differentiate with respect to our unknowns A and B, and set the differentials equal to zero, as we did before:

$$\frac{\partial \chi^2}{\partial A} = \frac{-2}{\sigma_y^2} \sum_{i=1}^{N} (y_i - A - Bx_i) = 0$$

$$\frac{\partial \chi^2}{\partial B} = \frac{-2}{\sigma_y^2} \sum_{i=1}^{N} x_i (y_i - A - Bx_i) = 0.$$

This results in a pair of simultaneous equations for A and B,

$$AN + B \sum x_i = \sum y_i,$$

and

$$A \sum x_i + B \sum x_i^2 = \sum x_i y_i,$$

where we have dropped the limits on the sumation for sake of clarity.

These equations can then be solved for A and B to get,

$$A = \frac{\sum x^2 \sum y - \sum x \sum xy}{N \sum x^2 - (\sum x)^2}$$

$$B = \frac{N \sum xy - \sum x \sum y}{N \sum x^2 - (\sum x)^2}.$$

Together, these equations allow us to calculate the best fit to the line $y = A + Bx$. The resulting line called the least squares fit, or line of regression of y on x. Now that we have A and B, we naturally want to estimate the uncertainties in these values. But first we need to discuss the uncertainty in y.

Errors

Errors in y - constant error

In the analysis above, we assumed that the values of y_i were distributed around the true values with spread of σ_y. You will notice however, that our estimates of A and B do not actually depend on this number (or numbers, in the case where σ_y is different for each point). However the true values of y_i are predicted by the line which A and B describe, and so the deviations of our measured values y_i should be depend on A and B. This immediately suggests that a good way to estimate σ_y is from

$$\sigma_y = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - A - Bx_i)^2}$$

which is the usual root-mean-square deviation from the mean, where in this case, the mean is defined by the best estimate of the line we have just fitted (i.e. the values A and B). However, the equations above have a problem, it violates the degrees of freedom condition. Since both A and B have been defined from the data already, we need to replace the $\frac{1}{N}$ with $\frac{1}{N-2}$ to get,

$$\sigma_y = \sqrt{\frac{1}{N-2} \sum_{i=1}^{N} (y_i - A - Bx_i)^2}.$$

This makes sense: if you have only two data points, then the best guess for the line will have A and B such that the line goes exactly through both points - there is no scatter! As soon as we have 3 data points, then at least 1 will not sit on the line, and the idea of σ_y becomes meaningful.

How does this propogate into the uncertainty in A and B? Note that both A and B are well-defined functions of y_i. This means that we can use the standard error propagation formula that we encountered Chapter 5. Using this, we get,

$$\sigma_A = \sigma_y \sqrt{\frac{\sum x^2}{N \sum x^2 - (\sum x)^2}}$$

and

$$\sigma_B = \sigma_y \sqrt{\frac{N}{N \sum x^2 - (\sum x)^2}}.$$

These kind of errors are known as homoscedastic errors.

Errors in both x and y

What if there are uncertainties on both x and y? For the special case of straight line, the uncertainties in both x and y make very little difference. To see this imagine that there is no error in y, but x has an error of δx. This point lies off the line, at a distance δy so that ultimately it produces an equivalent error in y. For the case of a straight line, the relationship is,

$$\delta y(\text{equiv}) = \frac{dy}{dx} \delta x.$$

The standard deviation σ_x is the root-mean-square value of δx that would result from repeating this measurement for each of our points, and so,

$$\sigma_y(\text{equiv}) = \frac{dy}{dx} \sigma_x.$$

This is true for any function that has no second derivative, but in the special case of a straight line, $\frac{dy}{dx}$ is simply our fit parameter, B. Thus the problem of fitting a line with uncertainties in x but none in y is the same as the problem of fitting a line with uncertainties in y but none in x.

In the case where there are uncertainties in both x and y, we simply translate the σ_x to σ_y. Accounting for the intrinsic errors in y, this yields,

$$\sigma_y(\text{equiv}) = \sqrt{\sigma_y^2 + (B\sigma_x)^2}$$

where we have combined the errors in quadrature in the usual way. Clearly, if we are not interested in fitting a straight line, then this expression may be more complicated.

Heteroscedastic errors

What about the case in which the errors on y_i are not constant? (These are called heteroscedastic errors.) In this case we need to use the idea of **weighted least squares** (see Chapter 5). By carrying the weights $w_i = 1/\sigma_i^2$ (these are on y_i) through the analysis, one ends up with the following expressions,

Weight the probabilities by the errors where $w = 1/\sigma^2$

$$A = \frac{\sum wx^2 \sum wy - \sum wx \sum wxy}{\sum w \sum wx^2 - (\sum wx)^2}$$

$$B = \frac{\sum w \sum wxy - \sum wx \sum wy}{\sum w \sum wx^2 - (\sum wx)^2}.$$

The associated uncertainties are,

$$\sigma_A = \sqrt{\frac{\sum wx^2}{\sum w \sum wx^2 - (\sum wx)^2}}$$

and

$$\sigma_B = \sqrt{\frac{\sum w}{\sum w \sum wx^2 - (\sum wx)^2}}.$$

We basically see that every term in the homoscedastic errors has an extra w and N is replaced by $\sum w$. Just like a weighted average, points contribute to the end sum where less weight is given to the less precise measurements and more weight given to more precise measurements. This approach assumes that w is the exact weight which it is not (worse if derived from small samples) and this approach can still be sensitive to outliers. The concept of weighting can be used in both linear and non linear fitting, though the expressions above are only for the linear case.

Example

Netflix managers poll their subscribers to see how the fraction of users that watched She-Ra in October 2019 x compares with the age of the viewer y before they decide to commission a new series of the show. This data is available at this link and is called DataAnalysis_datafile1.dat. Don't forget to upload it to your Google Colab.

1. Write functions from scratch to fit a straight line to the data quoting all the fit parameters and their errors.
2. Comment on the uncertainties.

Solution

Click below to see the solution.

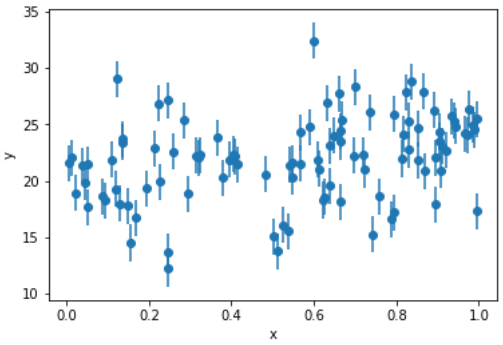
This questions involves fitting a straight line to set of points with homoscedastic errors on y.

Since the errors are homoscedastic, the scatter around the line should be notably larger than the $\sigma = 1.6$ that one might expect.

Let's open the datafile and take a look

In [19]:
```
import numpy as np
data = np.genfromtxt('DataAnalysis_datafile_1.dat',names=True)

# print out the columns to see what's in the file
print(data.dtype.names)

import numpy as np
import pylab as plt

%matplotlib inline

data = np.genfromtxt('DataAnalysis_datafile_1.dat',names=True)

plt.errorbar(data['x'],data['y'],yerr=data['sigma_y_estimate'],fmt='o')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
('x', 'y', 'sigma_y_estimate')
```



The scatter in the data is clearly much larger than the errors in y quoted! Note though, to orthogonalise the model we need to either use the above equations and **quote the covariance** or use:

$$y = A + B(x_i - \hat{x})$$

throughout, where $\hat{x}$ is the mean. Here we will show the 1st option.

The errors are homoscedastic since the errors on y are all the same.

In [20]:
```
# functions to estimate A and B based on notes and equations above
def best_fit_a(x,y,N):
    result = (np.sum(x**2.)*np.sum(y) - np.sum(x)*np.sum(x*y))/ ((N*np.sum(x**2.)) - (np.sum(x))**2. )
    return result

def best_fit_b(x,y,N):
    result = ((N*np.sum(x*y))- (np.sum(x)*np.sum(y))) / ((N*np.sum(x**2.)) - (np.sum(x))**2. )
    return result

A = best_fit_a(data['x'],data['y'],len(data['x']))
```

```
B = best_fit_b(data['x'],data['y'],len(data['x']))

print('Best fit line equation paramaters: A = {:.2f} age in years'.format(A),'and B= {:.2f} age in years/fraction'.format(B))
```

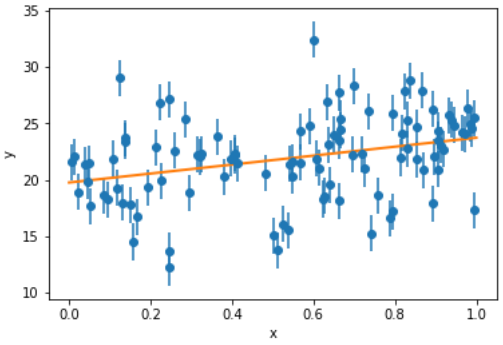
```
Best fit line equation paramaters: A = 19.76 age in years and B= 3.97 age in years/fraction
```

In [21]:

```
# to plot the straightline
def straightline(a,b,x):
    result = a + b*(x)
    return result

x = np.linspace(0,1,100)
fit = straightline(A,B,x)

plt.errorbar(data['x'],data['y'],yerr=data['sigma_y_estimate'],fmt='o')
plt.plot(x,fit,lw=2)
plt.xlabel('x')
plt.ylabel('y')
```

Out[21]:

```
Text(0, 0.5, 'y')
```



For errors in A and B we need to use the standard deviation of the y data:

$$\sigma_y = \sqrt{\frac{1}{N}\sum_{i-1}^{N}(y_i - A - Bx_i)^2}$$

where here N is the *number of degrees of freedom* ie in this case $N = \text{number of data points} - 2$. The $N-2$ appears because we already need to know A and B to work out σ_y.

We need to do this because the errors on the fit depends not just on the error bars in y but also on the fit itself, and by using the above equation we can estimate the true error in y (which should be larger than then 1.6 errors given to the students for each datapoint).

In [22]:

```
# let's estimate sigma_y using our A+B values

def sigma(N,A,B,x,y):
    sigma = np.sqrt( (1./(N-2))* np.sum((y - A - B*x)**2.0)  )
    return sigma

N=len(data['x'])
x= data['x']
y= data['y']

sigma_y = sigma(N,A,B,x,y)

print('The error in y (sigma_y) from our best fit line is {:.2f}.'.format(sigma_y))
```

```
The error in y (sigma_y) from our best fit line is 3.61.
```

Thus the σ_y of the data is much larger than the 1.6 on each data point quoted in the original data file.

Calculating σ_A and σ_B:

In [23]:

```
def error_in_a(x,sig_y,N):
    result = sig_y* np.sqrt( np.sum(x**2.) / ((N*np.sum(x**2.)) - (np.sum(x))**2.  )  )
    return result

def error_in_b(x,sig_y,N):
    result = sig_y* np.sqrt( N / ((N*np.sum(x**2.)) - (np.sum(x))**2.  )  )
    return result

error_A = error_in_a(data['x'],sigma_y,len(data['x']))
error_B = error_in_b(data['x'],sigma_y,len(data['x']))

print('A is {:.2f} age in years'.format(A),'with error {:.2f}'.format(error_A))
print('B is {:.2f} age in years/per fraction'.format(B),'with error {:.2f}'.format(error_B))
```

```
A is 19.76 age in years with error 0.75
B is 3.97 age in years/per fraction with error 1.19
```

So this is not a good fit if we take the error in y from the original data, but the fit vs model is better when accounting for the larger error in y compared to the true fit σ_y. But this is simply because the error is so large the model can be seen as an ok fit

One can also try the Spearman rank correlation to see if a straight line fit is appropriate:

In [24]:
```
from scipy.stats import rankdata
# let's rank the data
r_x = rankdata(data['x'])
r_y = rankdata(data['y'])

# need to set up equation in notes
def rho_s(rank_x,rank_y,N):
    top = np.sum(rank_x*rank_y) - (N*(N+1)**2./4)
    bottom_1 = np.sqrt( np.sum(rank_x**2.) - (N*(N+1)**2./4)   )
    bottom_2 = np.sqrt( np.sum(rank_y**2.) - (N*(N+1)**2./4)   )
    rho = top / (bottom_1*bottom_2)
    return rho

spearman = rho_s(r_x,r_y,len(data['x']))

print('the Spearman rank statistic is {:.2f}'.format(spearman))
```

```
the Spearman rank statistic is 0.37
```

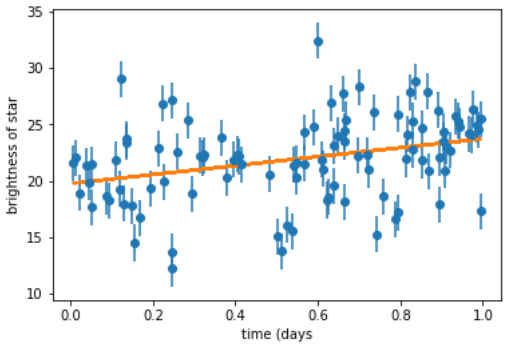
Thus a linear relationship between the two does not make a good model for this data since the Spearman rank statistic is less than 0.6. As the model chosen to fit the data is flawed the error on the fit parameters found from the equations above are in fact too small and does not accurately reflect the uncertainty in the A and B values given the quality of the fit.

We did all of the above by hand to make sure we could work through the mathematics, but we can also simply use Python to make things easier and quicker. For example, we could fit a straight line using the functions `poly1d` or `curve_fit` .

In [25]:
```
from scipy import optimize

# this is a function that we think may explain the data
def straightline_func(x,a,b):
    return a + b*x

data['x'],data['y']

params, params_covariance = optimize.curve_fit(straightline_func, data['x'],data['y'],\
                                               sigma=data['sigma_y_estimate'],absolute_sigma=True)

print('Fit parameters A and B are {:.3f} and {:.3f}'.format(params[0],params[1]))
print()
print('covariance matrix',params_covariance)

# plot the data
plt.errorbar(data['x'],data['y'],yerr=data['sigma_y_estimate'],fmt='o',label='data')
plt.xlabel('time (days')
plt.ylabel('brightness of star')

# plot the fit
plt.plot(data['x'],straightline_func(data['x'],params[0],params[1]),lw=2,label='best fit')
```

```
Fit parameters A and B are 19.764 and 3.967

covariance matrix [[ 0.10966227 -0.15270869]
 [-0.15270869  0.27741274]]
```

Out[25]:
```
[<matplotlib.lines.Line2D at 0x1a20e8ed10>]
```



In [26]:
```
# by hand
print('Best fit line equation paramaters by hand: A = {:.2f} age in years'.format(A),\
          'and B= {:.2f} age in years/fraction'.format(B))

# using Python's curve fit

print('Best fit line equation paramaters using functions: A = {:.2f} age in years'.format(params[0]),'and B= {:.2f} age in years/fraction'.form
```

```
Best fit line equation paramaters by hand: A = 19.76 age in years and B= 3.97 age in years/fraction
Best fit line equation paramaters using functions: A = 19.76 age in years and B= 3.97 age in years/fraction
```

How Good a Fit is Our Model?

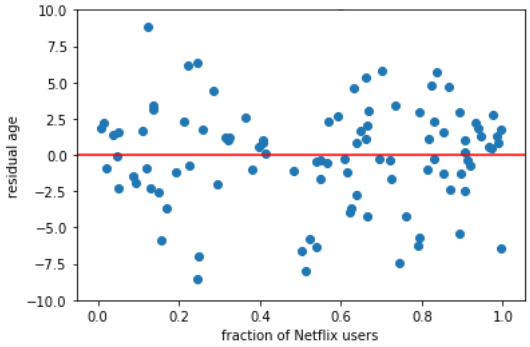
Residuals

Let's take the Netflix data above and take a look at how *different* our model predictions are (in this case our straight line with A and B) compared with the original data - this difference is called the *residual*.

In [27]:
```
# residual = difference between data and model
```

```
residual = data['y'] - straightline(A,B,data['x'])

plt.scatter(data['x'],residual)

plt.ylim(-10,10)
plt.ylabel('residual age')
plt.xlabel('fraction of Netflix users')
plt.axhline(0,c='red')
```

Out[27]: <matplotlib.lines.Line2D at 0x1a20fea410>



We can see just by looking at the residuals between the data and what would be predicted by our model, that there is a lot of scatter above and below the zero line (which is what we would get if our model matched our data perfectly). The scatter does not appear randomly distributed either.

Reduced chi-square test

Looking at the plots we've made above, the straight line does not look like a good fit to the huge scatter around the data - indeed the straight line does not even pass though many of the error bars on the data points. This suggests that either there are unaccounted sources of error or that the data is poorly modelled with a straight-line. We can quantify whether our fit is good by using the information provided by the residuals and estimating the chi-square statistic (Chapter 5).

The reduced chi-square test to evaluate the goodness of fit is given by

$$\bar{\chi}^2 = \frac{\chi^2}{N_d}$$

where N_d is the number of degrees of freedom = $N_{\rm data} - N_{\rm params}$.

Looking at the plots we've made above, the straight line does not look like a good fit to the huge scatter around the data - indeed the straight line does not even pass though many of the error bars on the data points. This suggests that either there are unaccounted sources of error or that the data is poorly modelled with a straight-line.

If the model describes the data and we have a good estimate of the uncertainty of each data point, then the reduced chi-square value would be approximately equal to one.

There might be fluctuation around one due to random variations about the mean value, so we should use an additional criteria to decide whether to reject the null hyopthesis (that the model describes the data and any deviations are explained due to random variation). Given that the mean of a chi-square distribution is $\chi^2_{\rm mean} = N_d$ and the variance is $2N_d$:

$$N_d \pm 2\sqrt{2N_d} \leq \chi^2_{\rm mean} \leq N_d \pm 2\sqrt{2N_d},$$

ie is your value for χ^2 within $\pm 2\sigma$ of the mean?

Confidence intervals on the chi-square statistic

We can also estimate confidence limits for chi-square parameters. The region of confidence (significance level α) is defined by

$$\chi^2_\alpha = \chi^2_{\rm min} + \Delta$$

The following table shows the value of Δ (difference above value of χ^2) for N fit parameters for a given confidence interval (1, 2 and 3 σ).

probability	N=1	N=2	N=3
0.680	1.00	2.30	3.53
0.954	4.00	6.17	8.02
0.997	9.00	11.8	14.20

Example

In the question above, we fit the Netflix data with a straight line. Derive a measure of the goodness of fit of this model.

$N_{\rm data} - N_{\rm params}$ is $100 - 2$ since we need A and B (which are not independent) and σ_y to determine χ^2. We would expect reduced χ^2 to equal close to 1 to denote a good fit.

In [28]:
```
def eqn(a,b,z):
    return a + b*z

def chi_sq(y,x,sig_y,a,b):
    result = np.sum(  (y - a-b*data['x'])**2. /sig_y**2. )
    return result

sig_y = 1.6 # quoted as error in the data
```

```
N_d = len(data['x']) - 1 # degrees of freedom: need to calculate A+B + sigma
chi_sq = chi_sq(data['y'],data['x'],sig_y,A,B)

red_chi_sq  = chi_sq / N_d
err_red_chi_sq = np.sqrt(2./len(data['x']))

print('chi-square is {:.3f} +/- {:.3f}'.\
      format(chi_sq,np.sqrt(2*N_d)))

print()
print('reduced chi-square is {:.3f} +/- {:.3f}'.\
      format(red_chi_sq,np.sqrt(2/err_red_chi_sq)))

# check if it's within 2sigma of the mean value of chi
chi_sq_mean = N_d
print()
print('chi-square mean is {:.3f}'.\
      format(chi_sq_mean))
crit = 2*np.sqrt(2*N_d)

print()
print('chi-square mean +/- 2sigma is {:.3f} to {:.3f}'.\
      format((chi_sq_mean-crit),(chi_sq_mean+crit)))
```

```
chi-square is 497.621 +/- 14.071

reduced chi-square is 5.026 +/- 3.761

chi-square mean is 99.000

chi-square mean +/- 2sigma is 70.858 to 127.142
```

Therefore this qualitively proves that our straight line model is not a good fit, and the chi-square statistic is *not* within 2σ of the mean.

Monte Carlo Model fitting

Suppose we have good reason to think that nature generates values of some physical quantity y in a way that depends on some other physical quantity x, in the manner of some model, let's say: $y(x) = Ax + B$. We would like to find out the values y and ranges of A and B by taking a number of measurements of $y(x)$ to trace the relationship. Suppose also that there is some uncertainty in each of the measurements.

To explore how our measurements might behave, we can simulate a large number of hypothetical experiments by using Monte Carlo methods and assess the range of parameters returned from these simulated or "fake" experiments.

We can approach this by simulating how much each measured y_i we've generated, deviates from the "true" value g_i which our model (in the example above, the straight line) predicts.

Example

Use the Monte Carlo method to generate random data from the Netflix data in DataAnalysis_datafile1.dat, and derive new (straight line) fit parameters A and B from this.

Solution

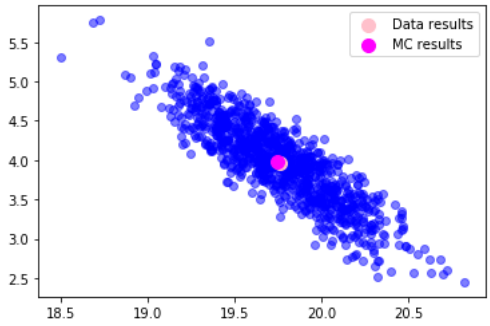
Click below to see the solution.

In [29]:

```
# generate n sets of data from the datafile by adding gaussian noise within error
n_sample = 1000
n = len(data['x'])

new_y =np.zeros([n,n_sample])
new_x =np.zeros([n,n_sample])

for i in range(n_sample):
    new_y[:,i] = np.random.normal(data['y'],data['sigma_y_estimate'])
    new_x[:,i] =  data['x']

A_mc = np.zeros(n_sample)
B_mc = np.zeros(n_sample)

# fit a line to each set of data
for j in range(n_sample):
    A_mc[j] = best_fit_a(new_x[:,j],new_y[:,j],n)
    B_mc[j] = best_fit_b(new_x[:,j],new_y[:,j],n)

# get average values of A and B from all the samples
mean_A_mc = np.mean(A_mc)
mean_B_mc = np.mean(B_mc)

print('Best fit line from earlier: A = {:.2f} age in years'.format(A),\
          'and B= {:.2f} age in years/fraction'.format(B))
print('Best fit line from MC: A = {:.2f} age in years'.format(mean_A_mc),\
          'and B= {:.2f} age in years/fraction'.format(mean_B_mc))

plt.scatter(A_mc,B_mc,marker='o',c='blue',alpha=0.5)
plt.scatter(A,B,c='pink',label = 'Data results',s=100)
plt.scatter(mean_A_mc,mean_B_mc,c='magenta',label = 'MC results',s=100)
plt.legend()
```

```
Best fit line from earlier: A = 19.76 age in years and B= 3.97 age in years/fraction
Best fit line from MC: A = 19.75 age in years and B= 3.99 age in years/fraction
```

Out[29]: <matplotlib.legend.Legend at 0x1a2109a110>



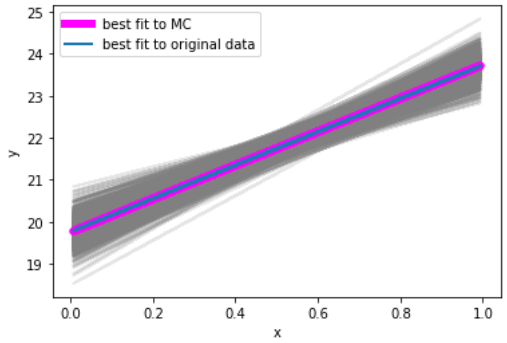
The best fit values from the straight line fit to the original data and the straight line fit to the MC data are rather similar. But the great thing about the MC is that we can use the standard deviation to derive an uncertainty in A and B.

In [30]:
```
mean_A_mc_err = np.std(A_mc,ddof=1)
mean_B_mc_err = np.std(A_mc,ddof=1)

print('Best fit line equation paramaters from original data: A = {:.2f} +/- {:.2f} age in years'.format(A,error_A),\
          'and B= {:.2f} +/- {:.2f} age in years/fraction'.format(B,error_B))
print()
print()
print('Best fit line equation paramaters from MC: A = {:.2f} +/- {:.2f} age in years'.format(mean_A_mc,mean_A_mc_err),\
          'and B= {:.2f} +/- {:.2f} age in years/fraction'.format(mean_B_mc,mean_B_mc_err))
```

```
Best fit line equation paramaters from original data: A = 19.76 +/- 0.75 age in years and B= 3.97 +/- 1.19 age in years/fraction


Best fit line equation paramaters from MC: A = 19.75 +/- 0.33 age in years and B= 3.99 +/- 0.33 age in years/fraction
```

In [31]:
```
# let's plot all our best fit lines to take a look at them
for i in range(n_sample):
    fit_new = straightline(A_mc[i],B_mc[i],x)
    plt.plot(x,fit_new,c='grey',alpha=0.2)

# straight line fit from mean A and B from MC analysis
fit_mc = straightline(mean_A_mc,mean_B_mc,x)
fit_data = straightline(A,B,x)
# straight line fit from original data

plt.plot(x,fit_mc,lw=6,c='magenta',label='best fit to MC')
plt.plot(x,fit_data,lw=2,label='best fit to original data')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x1a210758d0>



Fitting curves to data

An easy way to fit curves is via `the scipy optimize.curve_fit` . This returns an array with the best fitting parameters for the function it is given, based on minimising the differences in the `func(x,fit parameters)-y` squared.

It also returns an array with the covariance of the fit parameters (see Chapter 4). The diagonals of this array also provides the variance of the parameter estimates.

Example

1. Generate some fake data and fit a curve to it using the `scipy optimize curvefit` function and an equation of the form $y = A\sin(bx) + C$. Perturb each y value by adding on some Δy to create new "noisy" data. Obtain errors in y using a random normal distribution with normalisation of 20% of y.

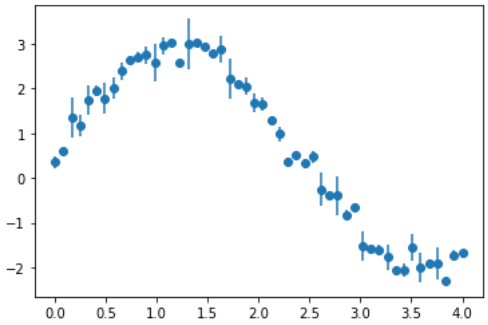
2. Derive the reduced chi-square value for your fit.

Solution

Click below to see the solution.

In [32]:
```
from scipy import optimize
import numpy as np
import pylab as plt
%matplotlib inline

# this is a function that we think may explain the data
def test_func(x, a,b,c):
    return a * np.sin(b * x)+c

# generate some fake data
# start off with x array
xdata = np.linspace(0, 4, 50)
# get y from the function above
y = test_func(xdata, 2.5, 1.3, 0.5)

# initialise the random_seed for reproducability
# ie we will get the same initial random seed every time
np.random.seed(1729)
# perturb y by this noise
y_noise = 0.2*np.random.normal(size=xdata.size)
# make new y array that is noisy
ydata = y + y_noise

# set up y error
y_sigma = 0.2*(0.5+np.random.normal(size=xdata.size))

# plot the data
plt.errorbar(xdata, ydata,yerr=y_sigma,fmt='o',label='data')
```

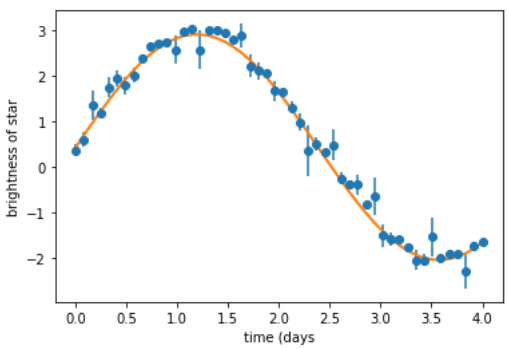
Out[32]:
```
<ErrorbarContainer object of 3 artists>
```



In [33]:
```
# get scipy to do the fitting for us
# need to include y errors
params, params_covariance = optimize.curve_fit(test_func, xdata, ydata,sigma=y_sigma,absolute_sigma=True)
print('Fit parameters A, B and C are {:.3f}, {:.3f} and {:.3f}'.format(params[0],params[1],params[2]))
print()
print('covariance matrix',params_covariance)

# plot the data
plt.errorbar(xdata, ydata,yerr=y_noise,fmt='o',label='data')
plt.xlabel('time (days')
plt.ylabel('brightness of star')

# plot the fit
plt.plot(xdata,test_func(xdata,params[0],params[1],params[2]),lw=2,label='best fit')
```

```
Fit parameters A, B and C are 2.485, 1.319 and 0.430

covariance matrix [[ 1.23722500e-04  1.45537330e-05  1.29967869e-05]
 [ 1.45537330e-05  4.12609818e-06 -1.42942807e-08]
 [ 1.29967869e-05 -1.42942807e-08  4.46367627e-05]]
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1a21ff5a10>]
```



In [34]:
```
import scipy.stats
# equation to get reduced chi^2
def chi_sq_red(x,y,y_error):
    # number of degrees of freedom
    n = len(x) - 3 #3 parameters already need to be calculated : a, b + c
    result = (1./n)*np.sum(((y-test_func(x,params[0],params[1],params[2]))/y_error)**2.0)
    return result
```

```
# calculate chi^2 of the fit:
print('The reduced chi-squared for the fit is {:.3f}'.format(chi_sq_red(xdata,ydata,y_sigma)))
```

```
The reduced chi-squared for the fit is 6.133
```

Approaches for non linear relationships or non-normal errors

The propogation of errors in complex systems can quickly get complicated. Sometimes the error terms are non-linear in the fit parameters, and cannot be analytically solved to give the best fitting model. To get around this we can use two powerful computatonal techniques to deal with this problem, and that can return reliable results - the MCMC and the bootstrap.

Using MCMC to derive the best fitting model parameters between model and data

To do this, we **make chi-square the distribution of space we need to sample**.

Here your likelihood is the chi-square function - best to use so-called "log likelihood" given the exponential term can get tricky with computers. The log likelihood for chi-square is given by

$$\text{log likelihood} = -0.5\frac{\sum(y_i - y_{\text{model}})}{\sigma^2}.$$

Derivation: the likelihood for a chi-square

If we have a chi-square distribution with a Gaussian error distribution, the probability of each point for chi-square is written as:

$$p(x_i|\alpha) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{(x_i - \mu_i(\alpha))^2}{2\sigma_i^2}\right)$$

so we can write the likelihood function as,

$$\mathcal{L}(\alpha) = e^{-\chi^2/2} \times \prod_{i=1}^{N}\frac{1}{\sigma_i} \times (2\pi)^{-N/2}.$$

We can now take the natural log of this to get,

$$-2\ln\mathcal{L} = \chi^2 + 2\sum_{i=1}^{N}\ln\sigma_i + N\ln(2\pi).$$

To maximise $\mathcal{L}(\alpha)$, we then need to minimise $\chi^2 + 2\sum_{i=1}^{N}\ln\sigma_i$. The maximum likelihood α_{ML} is then the one that satisfies,

$$\frac{\partial}{\partial\alpha}[-2\ln\mathcal{L}(\alpha)] = 0$$

and the variance of α_{ML} this given by:

$$\text{var}(\alpha_{ML}) \approx \frac{2}{\frac{\partial^2}{\partial\alpha^2}[-2\ln L(\alpha)]_{\alpha=\alpha_{ML}}}$$

If we're only interested in the shape of the distribution to estimate statistics such as the mean value of a parameter, we can ignore the additional terms and therefore

$-2\ln\mathcal{L} = \chi^2$ ie

$\ln\mathcal{L} \propto -0.5\chi^2.$

Bootstrapping

You want to ask a question from a population but you can't. You use a sample instead. Previously we took samples from the population using Monte Carlo by assuming a distribution of the sample eg Normal or Binomial. What if you are not happy with assuming a distribution?

So we just sample from the sample instead....it is a part of the population after all, just a small discrete part. Given a set of N data points, we randomly draw N numbers from this set:

$X = x_1, x_2, x_3, x_4, x_5, x_6, x_7$

$X_{\text{new}} = x_3, x_6, x_7, x_4, x_5, x_1, x_1.$

This means that some numbers will be duplicated, and some will be missing entirely (so-called sampling with replacement). Combining all the bootstrap datasets, we can now get a new estimate of the mean from this new, sampled data set or create N datasets to fit N straight lines or other functions to.

So the steps are:

- Choose a number of bootstrap samples to perform
- Choose a sample size
- For each bootstrap sample
  - Draw a sample with replacement with the chosen size
  - Calculate the statistic on the sample
  - Calculate the mean of the calculated sample statistics,

et voila!

Example

A population of data is randomly generated using `pop=np.random.randint(0,500, size=1000)` . Let's say this is the number of people getting flu twice in one year in the University.

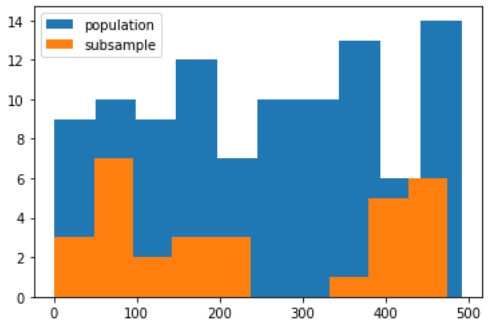
Generate a subsample of data 300 points by randomly drawing from this population. This is equivalent to mimicking an experiment where someone works out the average number of people getting flu twice in one year by asking 300 people.

- Plot the data and estimate the mean of the population and the mean of the subsample.
- Use the bootstrap method to estimate the mean parameter of the population (the mean number of people getting flu twice in one year) and plot your bootstrap sample. What do you notice?

Solution

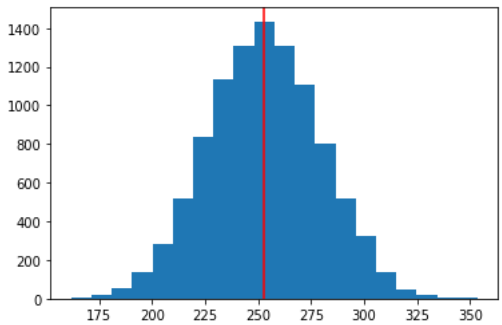
Click below to see the solution.

In [35]:
```
import numpy as np

np.random.seed(42)
pop = np.random.randint(0,500 , size=100)

# draw a sample from population
sample = np.random.choice(pop, size=30)

#plt the histograms
plt.hist(pop,label='population')
plt.hist(sample,label='subsample')
plt.legend()
plt.show()

# estimate means
print('The mean of the population is {:.2f}'.format(pop.mean()))
print()
print('The mean of the subsample is {:.2f}'.format(sample.mean()))
```




```
The mean of the population is 252.70

The mean of the subsample is 228.07
```

In [39]:
```
# make sure we have the same random seed
np.random.seed(42)

# simulated sampling distribution - the bootstrap
sample_props = []
for _ in range(10000):
    sample = np.random.choice(pop, size=30)
    sample_props.append(sample.mean())

plt.hist(sample_props,bins=20)
plt.axvline(np.mean(sample_props),c='red')
plt.show()

print('The mean of the bootstrapped sample (ie mean of the means) is {:.2f}'.format(np.mean(sample_props)))
```




```
The mean of the bootstrapped sample (ie mean of the means) is 252.83
```

Now the recovered population from the bootstrap looks like a Gaussian and not like the original population which was uniform. This is what we expect however from the Central Limit Theorem in Chapter 3:

> When independent random variables are added, their sum tends toward a normal distribution for large N even if the original variables themselves are not normally distributed.

The mean of the bootstrap sample is much closer to the mean of the original population indicating that the bootstrap technique is really powerful even when we dont know the original distribution. It also shows how far off the mean of the subsample of 100 people we used for our experiment is from the original population - clearly the subsample was not a good reflection of the original data.

Now you are ready to tackle the **Chapter 8 quiz** on Learning Central and the Chapter 8 yourturn notebook.

Chapter 9

CA 2 - A Data Analysis "project"

Analysing gravitational wave signals

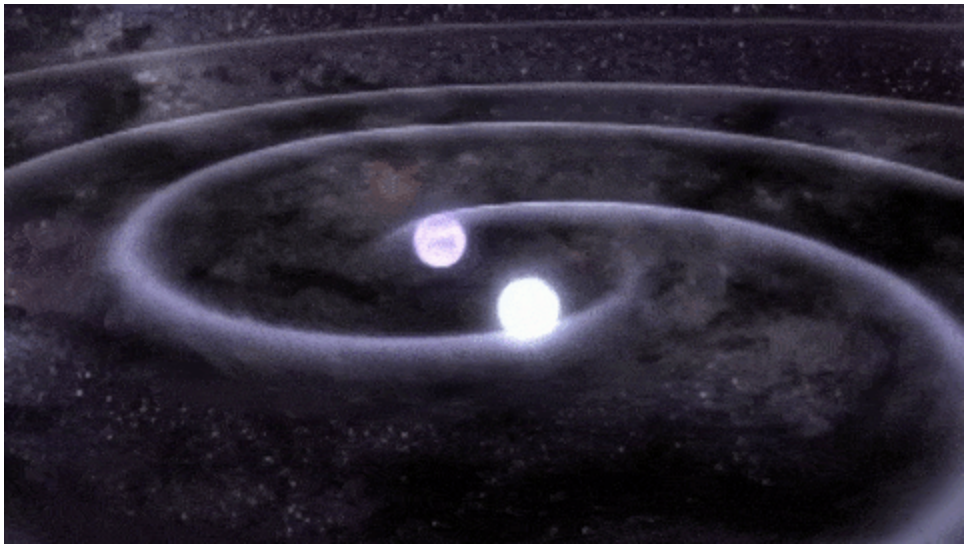
Please ensure you have watched the Chapter 9 video(s). This notebook introduces some of the theory behind your Data Analysis "project" in CA 2. In this project you will analyse some mock gravitational wave data from two unknown astrophysical objects merging together and coelescing. We will use a Monte Carlo Markov Chain (MCMC) to compare a scaled model that predicts how the wave changes depending on the total mass of the merging objects and their distance from us to the observed waveform. This will allow us to determine the nature of the orbiting objects that merged to form the gravitational wave, whether for instance they could be originating from merging white dwarfs, neutron stars or black holes.

You will learn the following things in this Chapter

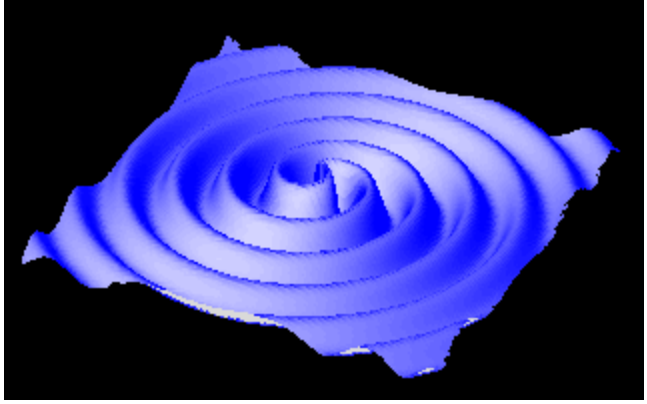
- What are gravitational waves
- How do we analyse signals from gravitational waves
- How to use Python programming to do the above.
- After completing this notebook you will be able to finish CA2.
- You may need to do some additional research into this subject. You may find it useful to look at the following publication from the LIGO consortium here.

What are gravitational waves?

Gravitational waves are disturbances in the curvature of spacetime, generated by accelerated masses, that propagate as waves outward from their source at the speed of light. They are predicted in General Relativity and other theories of gravity and since 2017, they have now been observed!



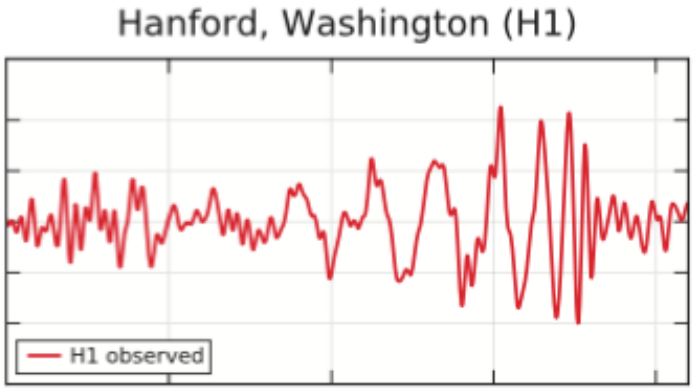
The animation above is from NASA. A 2 dimensional representation of the waveform genertaed by two neutron stars is shown below.



The animated gif above is from Wikipedia.

Gravitational Waves - Observed

Below is an image of the first gravitational wave signal (captured by the Hanford detector in Washington) observed by the LIGO/Virgo collaboration.



B. P. Abbott et al., *Phys. Rev. Lett.* 116, 061102 (2016)

We can describe GW waveforms by their strain (the stretching and squeezing of spacetime) as the two compact, dense astrophysical objects coalesce. The strain describes the amplitude of the waveform above. The system is parameterised by the masses of the merging objects, M_1 and M_2, and their distance from the observer D.

Other useful parameters that help when trying to describe the waveforms include:

The mass ratio $q = M_2/M_1$, with convention that $M_1 \geq M_2$ and so $q \leq 1$.

The "Chirp mass", which is a quantity used in general relativity, is given by:

$$M_{ch} = \frac{(M_1 M_2)^{3/5}}{(M_1 + M_2)^{1/5}}.$$

To derive parameters such as the masses of the coelescing objects and so on, we need to fit a model waveform to our data waveform.

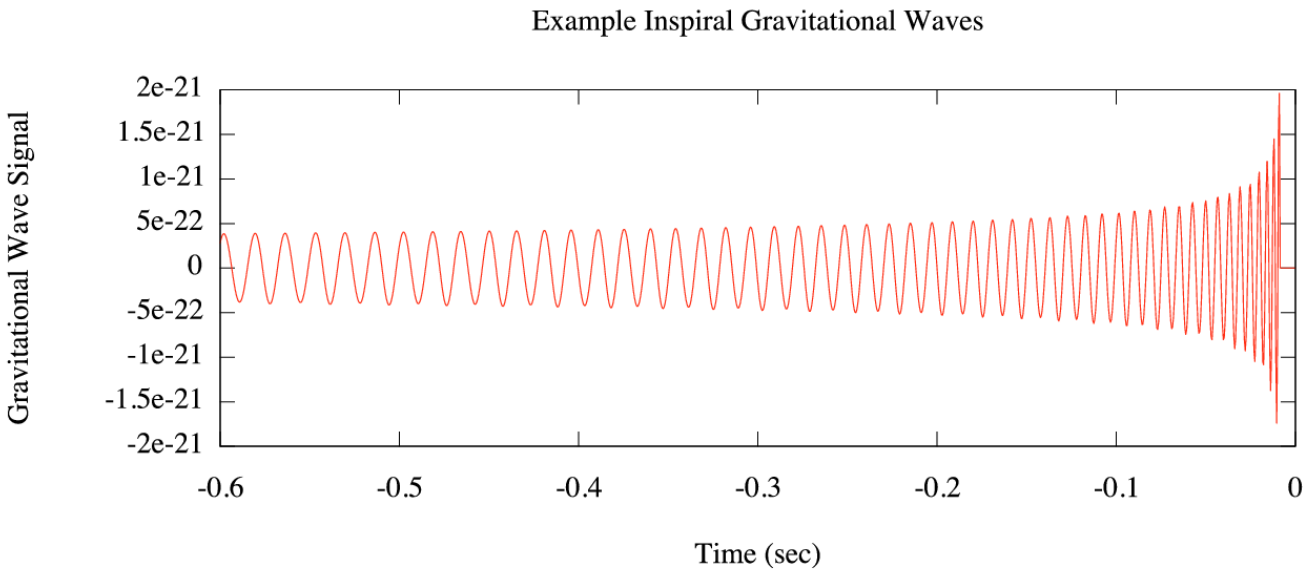
Gravitational waves - Models

Now that we have some data, we ideally want to try and compare predicted waveforms for a merger with

our observed waveform and use this to derive the mass and distance to the merger.

Model waveforms are known as **templates** in the gravitational wave community. Scientists attempt to produce a waveform for different masses and distances using a reference waveform and scaling it by its change in distance and mass and then see which one best fits the data observed.

The LIGO collaboration has created an animated gif that shows three examples of gravitational-wave signals from different types of inspiraling binary systems:



The red signal comes from a system of two black holes of mass $10M_\odot$ and $5M_\odot$ that are not spinning. The green signal is from black holes with the same masses, but now with each of the two components spinning. The blue signal comes from a system of two neutron stars each with mass equal to $1.4M_\odot$, that are not spinning.

Mathematically, if we have an equal-mass system (i.e $q = 1$) with total mass $M = M_1 + M_2$ at a distance D (not spinning), then we can scale the strain of a waveform, $h(t, M, D)$ from a reference waveform with M', D' as:

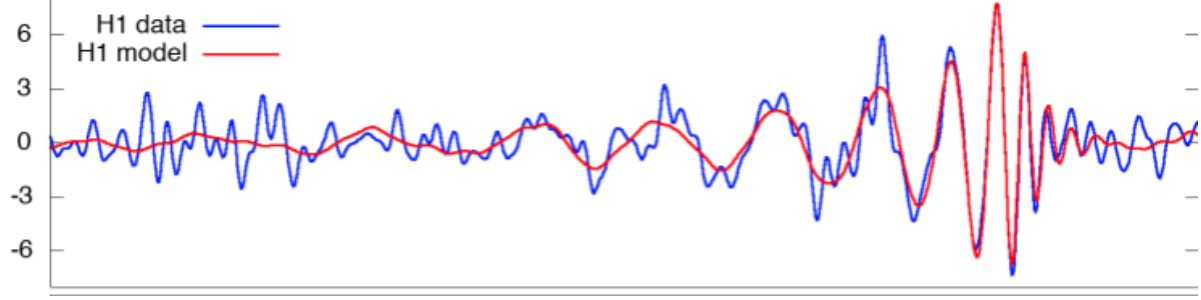
$$h(t, M, D) = \left(\frac{M}{M'}\right)\left(\frac{D'}{D}\right)\ h(t')$$

where:

$$t' = \left(\frac{M'}{M}\right) t$$

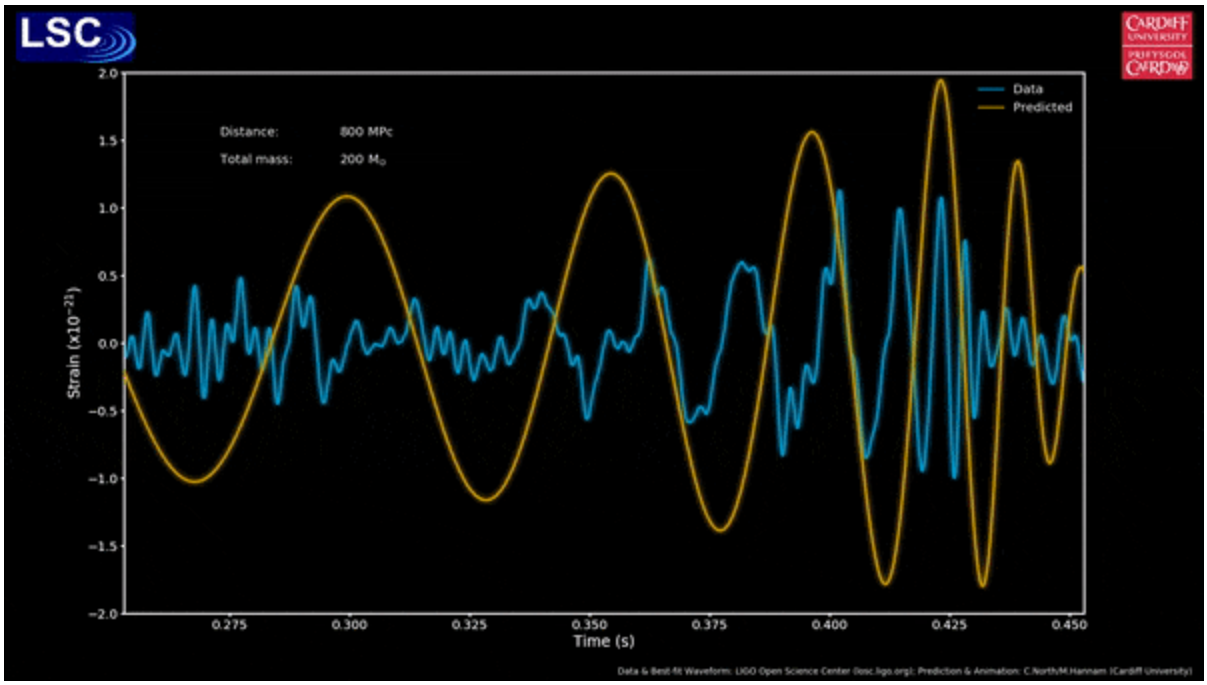
and $t = 0$ is defined as being the time at which the merger occurred.

Below we show the comparison of the Hanford data with the best fit model from Abbott et al. 2020.



So once a gravitational wave signal has been detected, the next step is to measure the properties of the system by comparing the observed signal to millions of different model waveforms predicted by general relativity and seeing which match the data.

The animation below illustrates how this works for two of the properties of the first gravitational wave detection shown in the Hanford waveform above: the total mass of the two black holes, and the distance from Earth. The model strain is produced by inspiralling (non-spinning, equal mass) black holes. The strain changes as the black holes inspiral and merge, the initial guess of the total mass of the system and distance is 200 solar masses and 800Mpc (2.5 billion years), then these two parameters are changed to get closer and closer to the best fit values.



There is a fantastic interactive version made by Cardiff University LIGO scientists here which allows you to choose different values to see the changes in the output waveforms.

Best fit parameters

So what made the gravitational waves that were seen by LIGO in the Hanford Observatory? LIGO found that the best model waveform to explain the discovery observations suggests the binary system had a total mass of ~ 65 solar masses. They used Bayesian inference to determine that this waveform most likely resulted from two black holes with masses of $35^{+5}_{-3}\,M_{\odot}$ and $33^{+3}_{-4}\,M_{\odot}$ at a distance of 440^{+160}_{-180} Mpc from Earth (Abbott et al. 2016). The errors quoted here are from the **90% credible intervals** derived using Bayesian inference (comparing the model with the data using a chi-square likelihood as we saw in Chapters 6 and 8).

The estimated pre-merger masses of the two components in the gravitational wave event recorded in the Hanford plot above, make a very strong argument that they are both black holes (neutron stars would not be massive enough). Also the enormous velocity and tiny separation of the two components required to explain the wafeform show that (a) vels have to be significant fractions of the speed of light and (b) approximate separation just a few times the characteristic size of a black hole, known as its Schwarzschild radius (ie they were only a few hundred kilometers apart just before they merged).

How does one calculate approximate separation before merging event?

If the objects are not spinning and they are in circular Keplerian orbits until the merger then we can note that the orbital separation R of the two bodies just before merger (peak amplitude of waveform) would be:

$$R = \left(\frac{G(M_1 + M_2)}{\omega_{\max}^2} \right)^{1/3}.$$

We can estimate ω from the waveforms since ω is related to the period of the gravitational waves. Note that after half a revolution, the two stars have just swapped their positions. The second half of the orbit repeats the motion pattern of the first half, and the gravitational wave signal is therefore repeated accordingly. One orbital period in the graviational waves observed therefore comprises two periods of the gravitational wave signal.

Now you are ready to tackle the **Chapter 9 quiz** on Learning Central and CA 2.