`python` Intermediate

`Matplotlib` introduction and `NumPy` extras

Mauricio Sevilla

email= j.sevillam@uniandes.edu.co

01.04.19

During this week, we are going to learn to use one of the most used plotting libraries available on `python` , Matplotlib (https://matplotlib.org/).

As we are going to use some `array` -like structures, we are also going to keep working on NumPy (http://www.numpy.org/).

Firts of all, `matplotlib` is a huge library, so we are going to use (Not forever), just a *small* module, `pylab` . This package can be imported, but we are going to use a standard method, such that,

In [1]:
```
import numpy as np
import matplotlib.pylab as plt
```

Where all the `plt` functions belong to the plot library.

Second of all, we have to understand that `matplotlib` does not plot functions, it plots `array` -like structures or numbers, such as `numpy.array` s, `list` s, `int` s, `float` s and so on.

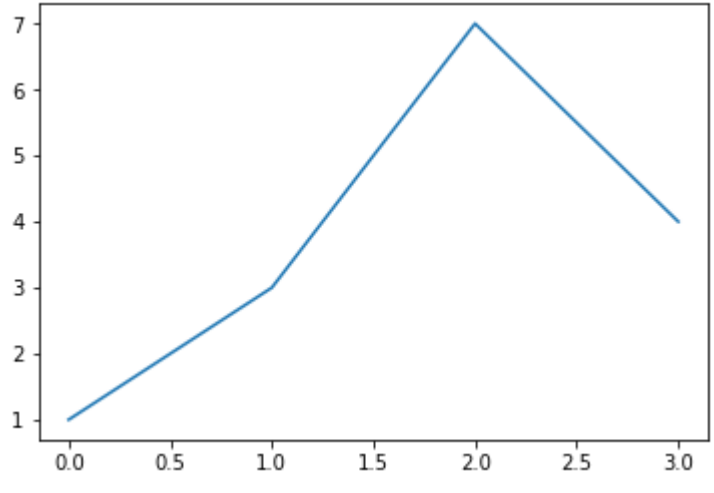
Let see some examples, consider the following list

In [2]:
```
x=[1,3,7,4]
```

we want to plot these points, easily we can use the `matplotlib.pylab.plot` function.

In [3]:
```
plt.plot(x)
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x10ec77d30>]
```

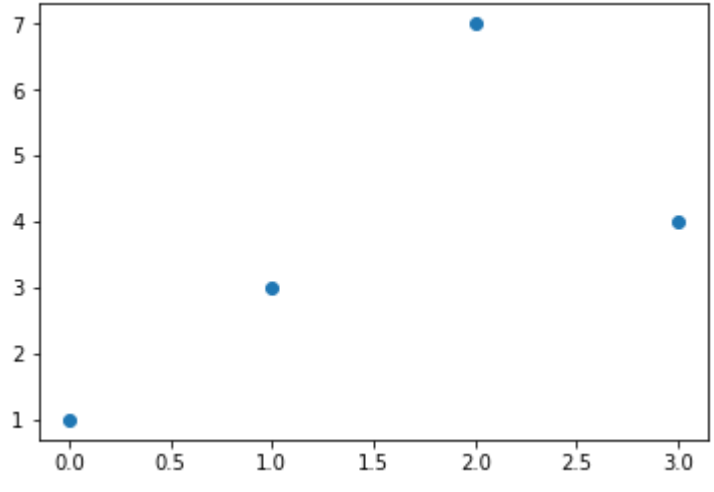


There are two relevant comments to do here,

1. We can plot a single `array` (`matplotlib` uses a `range(len(.))` for the x axis) on the latest `matplotlib` versions.
2. By default, it plots lines. (Can be changed)
3. On the notebook, we get the plot as the cell output, on a `python` script, we should also use `plt.show()`. Just for you not to get confused about it, we are going to use the line `plt.show()` in all the cells with plots.

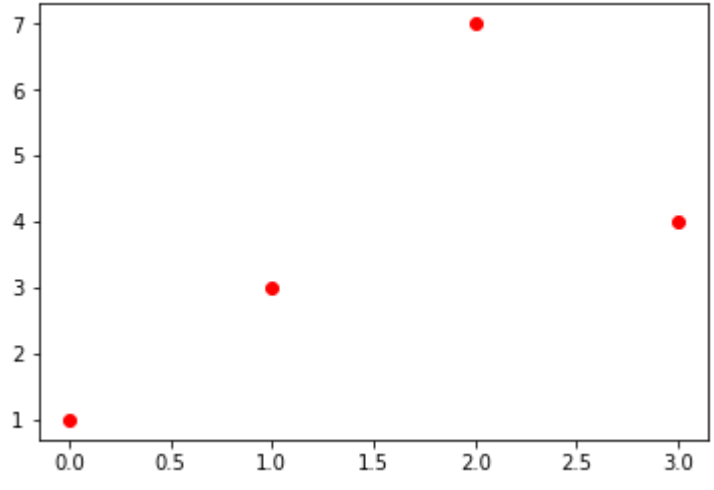
One can give the plot additional options, for example, let us use points instead of lines on our plot

In [4]:
```
plt.plot(x,'o')
plt.show()
```



what if, additionally I want the points to be red,

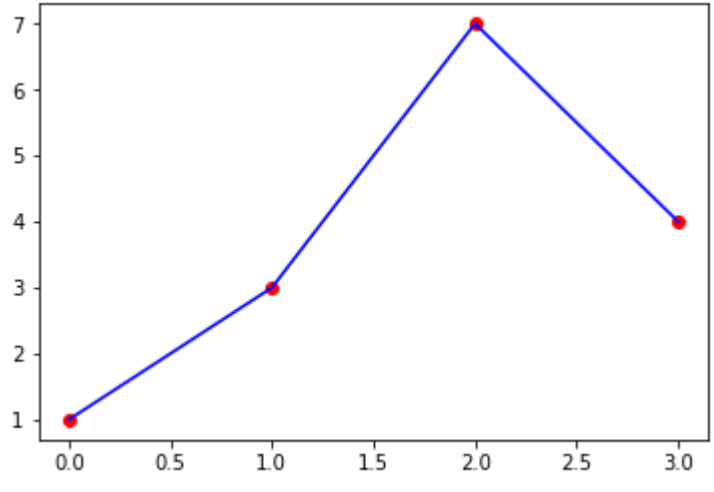
In [5]:
```
plt.plot(x,'ro')
plt.show()
```



if we want to have more than one plot, we use `matplotlib.pylab.plot` more than once before the `matplotlib.pylab.show()`.

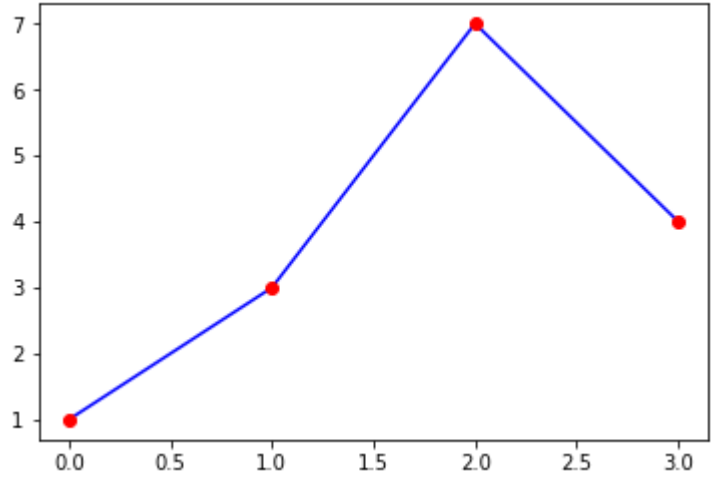
On the Jupyter notebook, we must use them on the same cell.

In [6]:
```
plt.plot(x,'ro')
plt.plot(x,'b')
plt.show()
```



Order is important!

In [7]:
```
plt.plot(x,'b')
plt.plot(x,'ro')
plt.show()
```



One can use severad already defined colors

Character	Color
`'b'`	Blue
`'g'`	Green
`'r'`	Red
`'c'`	Cyan
`'m'`	Magenta
`'y'`	Yellow
`'k'`	Black
`'w'`	White

The point style can be choosen from,

Character	Description	Character	Description
`'.'`	Point	`'s'`	Square
`','`	Pixel	`'p'`	Pentagon
`'o'`	Circle	`'*'`	Star
`'v'`	Triangle Down	`'h'`	Hexagon 1
`'^'`	Triangle Up	`'H'`	Hexagon 2
`'<'`	Triangle Left	`'+'`	Plus
`'>'`	Triangle Right	`'x'`	x
`'1'`	Tri-Down	`'D'`	Diamond
`'2'`	Tri-Up	`'d'`	Thin Diamond
`'3'`	Tri-Left	`'|'`	Vertical Line
`'4'`	Tri-Right	`'_'`	Horizontal Line

The lines allow us to choose among,

Character	Description
`'-'`	Solid
`'--'`	Dashed
`'-.'`	Dash-dot
`':'`	Dotted

Some Examples

```
'b'    # blue markers with default shape
'ro'   # red circles
'g-'   # green solid line
'--'   # dashed line with default color
'k^:'  # black triangle_up markers connected by a dotted line
```

Let us use a bigger set of data, first we are going to generate some points as we did on the last homework, and then, we will load an external file.

In [8]:
```
def fun(x):
    return np.sin(5*x)*np.exp(-x/5)
```

The function `numpy.linspace` allows us to create an array with a certain amount of numbers on a given interval,

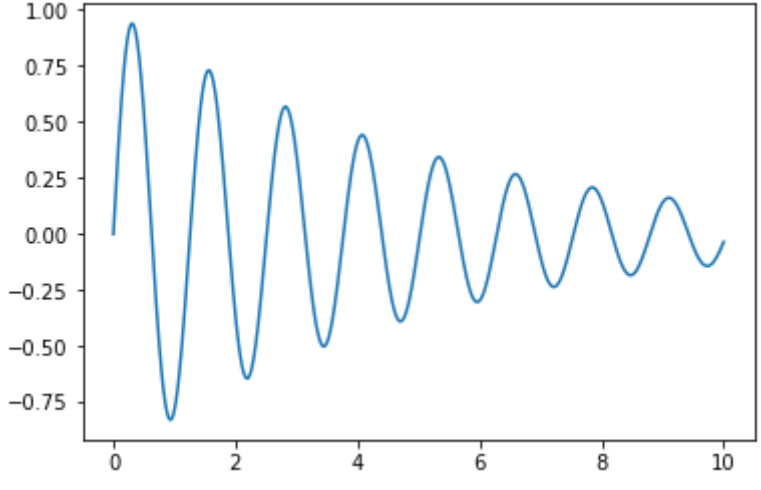
In [9]:
```
x=np.linspace(0,10,1000)
```

And, after that, create a new one applying a function to them,

In [10]:
```
y1=fun(x)
```

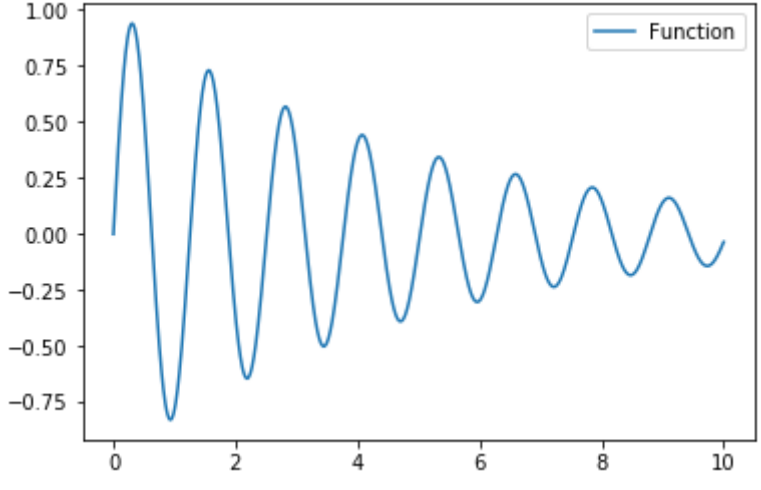
The plot, is done as simply as

In [11]:
```
plt.plot(x,y1)
plt.show()
```



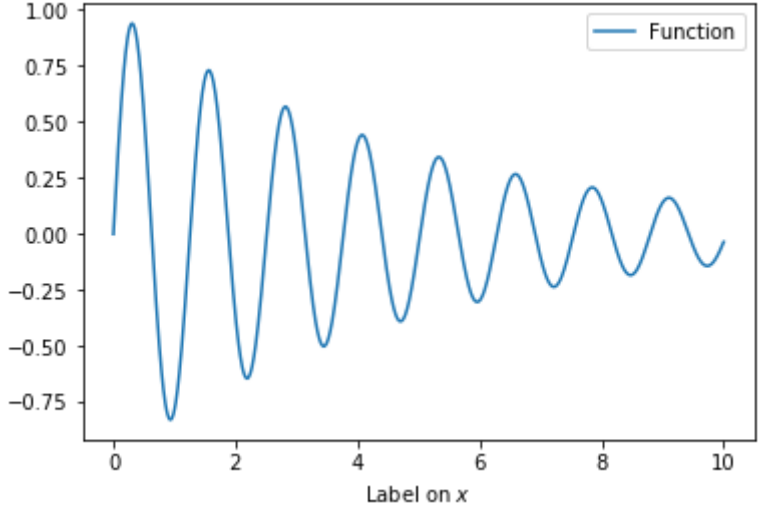
We can add a legend,

In [12]:
```
plt.plot(x,y1,label='Function')
plt.legend()
plt.show()
```



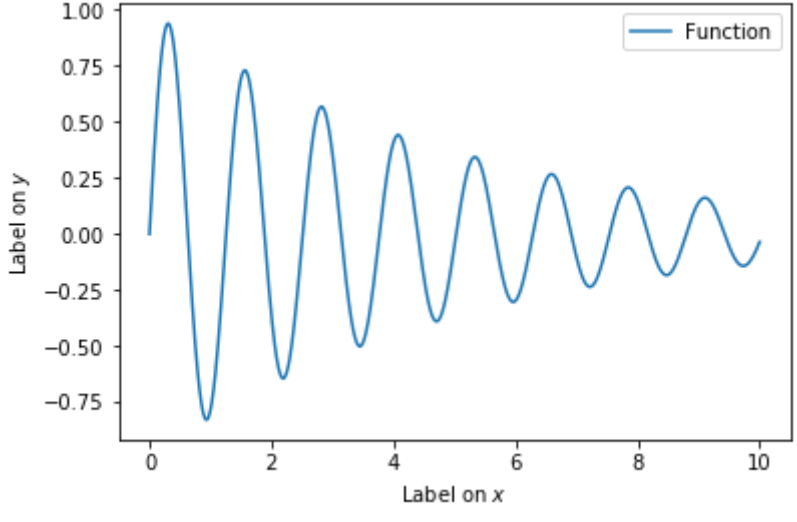
Or even labels, on x

In [13]:
```
plt.plot(x,y1,label='Function')
plt.xlabel('Label on $x$')
plt.legend()
plt.show()
```



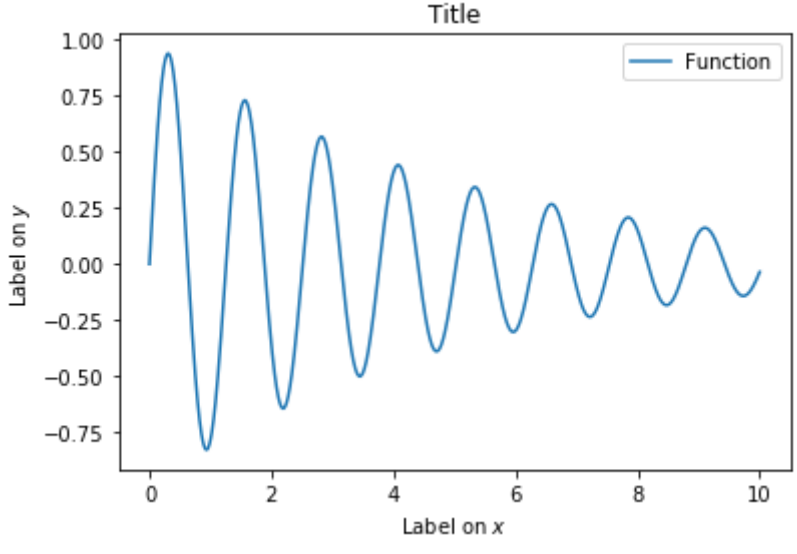
or in both y and x directions

In [14]:
```
plt.plot(x,y1,label='Function')
plt.xlabel('Label on $x$')
plt.ylabel('Label on $y$')
plt.legend()
plt.show()
```



And a title!

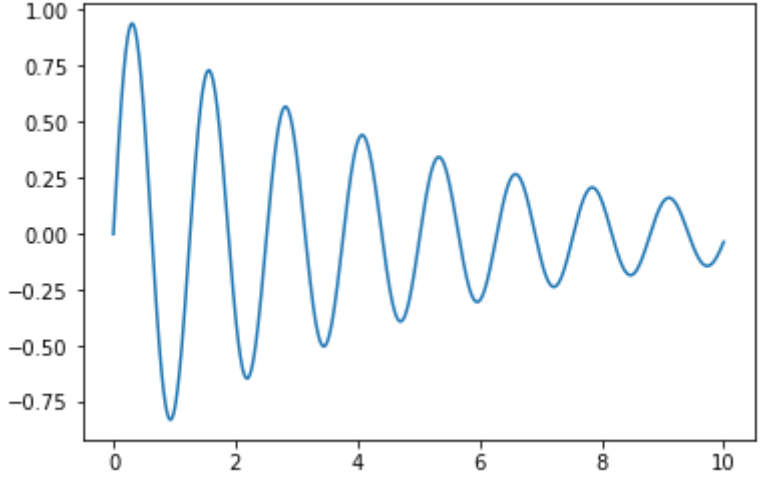
In [15]:
```
plt.plot(x,y1,label='Function')
plt.title('Title')
plt.xlabel('Label on $x$')
plt.ylabel('Label on $y$')
plt.legend()
plt.show()
```



Figures

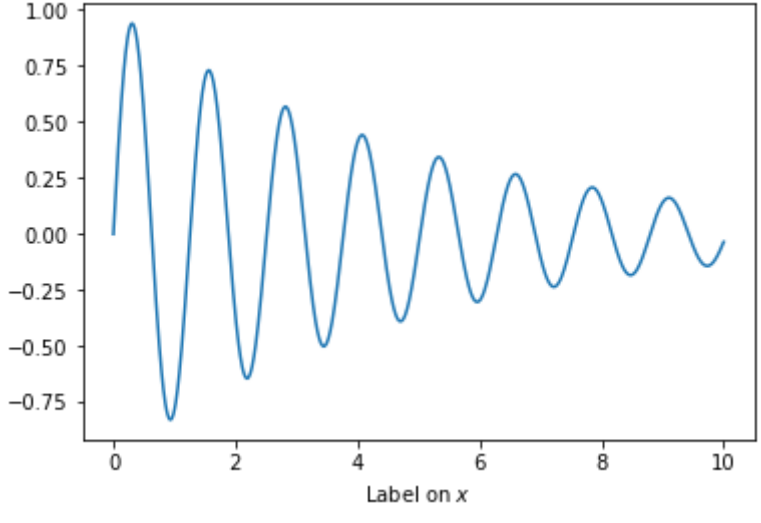
There is a better way to do it, at the beginning it looks longer, but it will help us later when creating multiplots, it is using `matplotlib.pylab.figure` s.

In [16]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(x,y1)
plt.show()
```



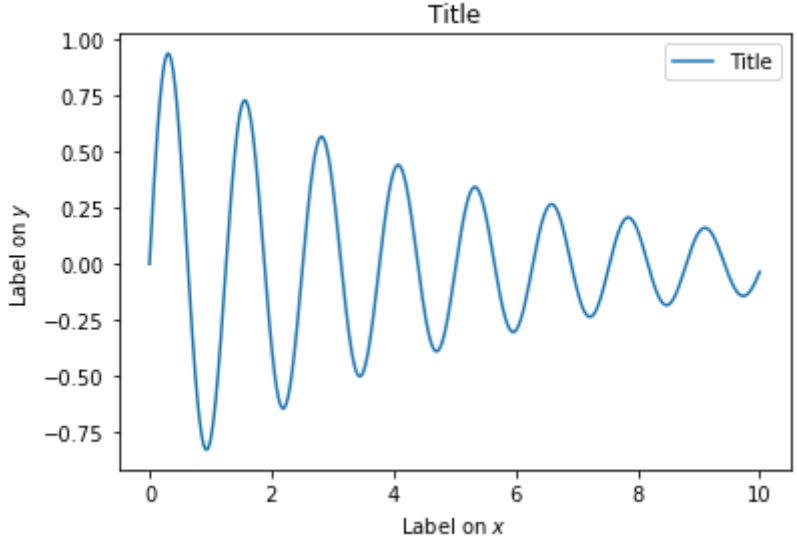
Here, the labels are added a little bit different

In [17]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(x,y1)
ax.set_xlabel('Label on $x$')
plt.show()
```



and so on,

In [18]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(x,y1,label='Title')
ax.set_xlabel('Label on $x$')
ax.set_ylabel('Label on $y$')
ax.set_title('Title')
plt.legend()
plt.show()
```



Spectra

Let us use several experimental data on external files, you will be able to see the spectrum of a Deuterium and Tungsten (Wolfram) lamp, measued at the Optics Laboratory at the Universidad Nacional de Colombia.

On the following link you will find a folder with the data,

In [19]:
```
url='https://raw.githubusercontent.com/jmsevillam/Herramientas-Computacionales-Uni
Andes/master/Data/Spectra/'
```

We are going to open the files by using the `numpy.genfromtxt` function

In [20]:
```
Deuterium=np.genfromtxt(url+'D.txt')
Tungsten=np.genfromtxt(url+'W.txt')
Both=np.genfromtxt(url+'W+D.txt')
```

This is one of the most important thins when working with data as we are,

In [21]:
```
print(Deuterium)
```

```
[[ 1.97750921e+02  4.96420000e-04]
 [ 1.97952761e+02  3.73730000e-04]
 [ 1.98154619e+02  1.24860000e-04]
 ...
 [ 1.02398952e+03  3.67800000e-05]
 [ 1.02423379e+03 -1.89400000e-05]
 [ 1.02447807e+03  2.32180000e-04]]
```

In [22]:
```
print(np.shape(Deuterium))
```

```
(3648, 2)
```

There are several ways to get the columns of the file, first

In [23]:
```
print(np.shape(Deuterium.T))
```

```
(2, 3648)
```

In [24]:
```
print(Deuterium.T)
```

```
[[ 1.97750921e+02  1.97952761e+02  1.98154619e+02 ...  1.02398952e+03
   1.02423379e+03  1.02447807e+03]
 [ 4.96420000e-04  3.73730000e-04  1.24860000e-04 ...  3.67800000e-05
  -1.89400000e-05  2.32180000e-04]]
```

Second, using a slicing strucure to extract the nth column,

In [25]:
```
print(Deuterium[:,0])
```

```
[ 197.75092074  197.952761    198.15461887 ... 1023.98952476 1024.23379367
 1024.47806826]
```

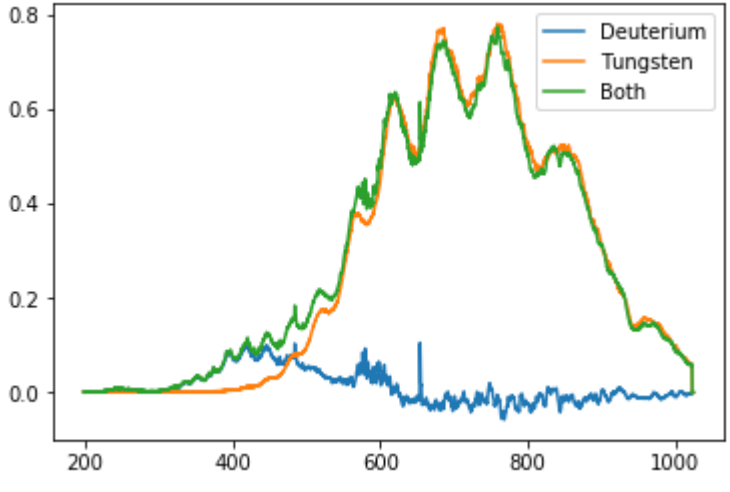
Thus,

In [26]:
```
xD=Deuterium[:,0]
yD=Deuterium[:,1]
xW=Tungsten[:,0]
yW=Tungsten[:,1]
xB=Both[:,0]
yB=Both[:,1]
print(yD)
```

```
[ 4.9642e-04  3.7373e-04  1.2486e-04 ...  3.6780e-05 -1.8940e-05
  2.3218e-04]
```

Again, the multiple plot goes as,

In [27]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(xD,yD,label='Deuterium')
ax.plot(xW,yW,label='Tungsten')
ax.plot(xB,yB,label='Both')
ax.legend()
plt.show()
```



What if, we would like to have the three plots separated, here is where the figures and axes structure becomes useful, we may define more than one axes on each figure,

```
ax=fig.add_subplot(nml)
```

with `n`, `m` and `l` numbers. `n` is the number of columns, `m` is the number of rows and `l` is the position.

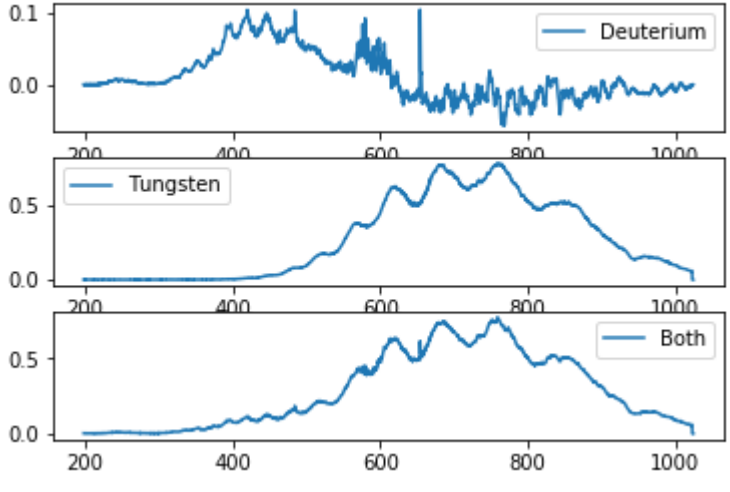
or equivalent

```
ax=fig.add_subplot(n,m,l)
```

This last one is needed when `l>9`

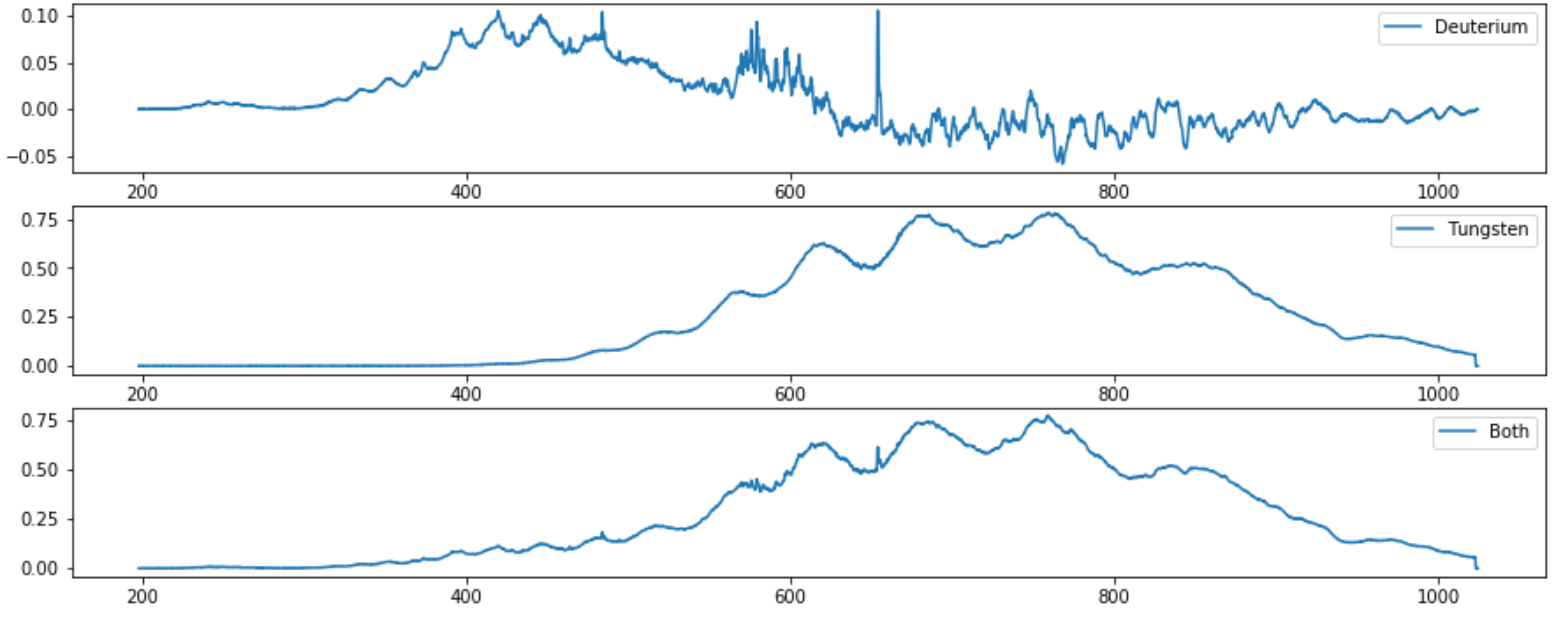
For example,

In [28]:
```
fig=plt.figure()
ax1=fig.add_subplot(311)
ax2=fig.add_subplot(312)
ax3=fig.add_subplot(313)
ax1.plot(xD,yD,label='Deuterium')
ax1.legend()
ax2.plot(xW,yW,label='Tungsten')
ax2.legend()
ax3.plot(xB,yB,label='Both')
ax3.legend()
plt.show()
```



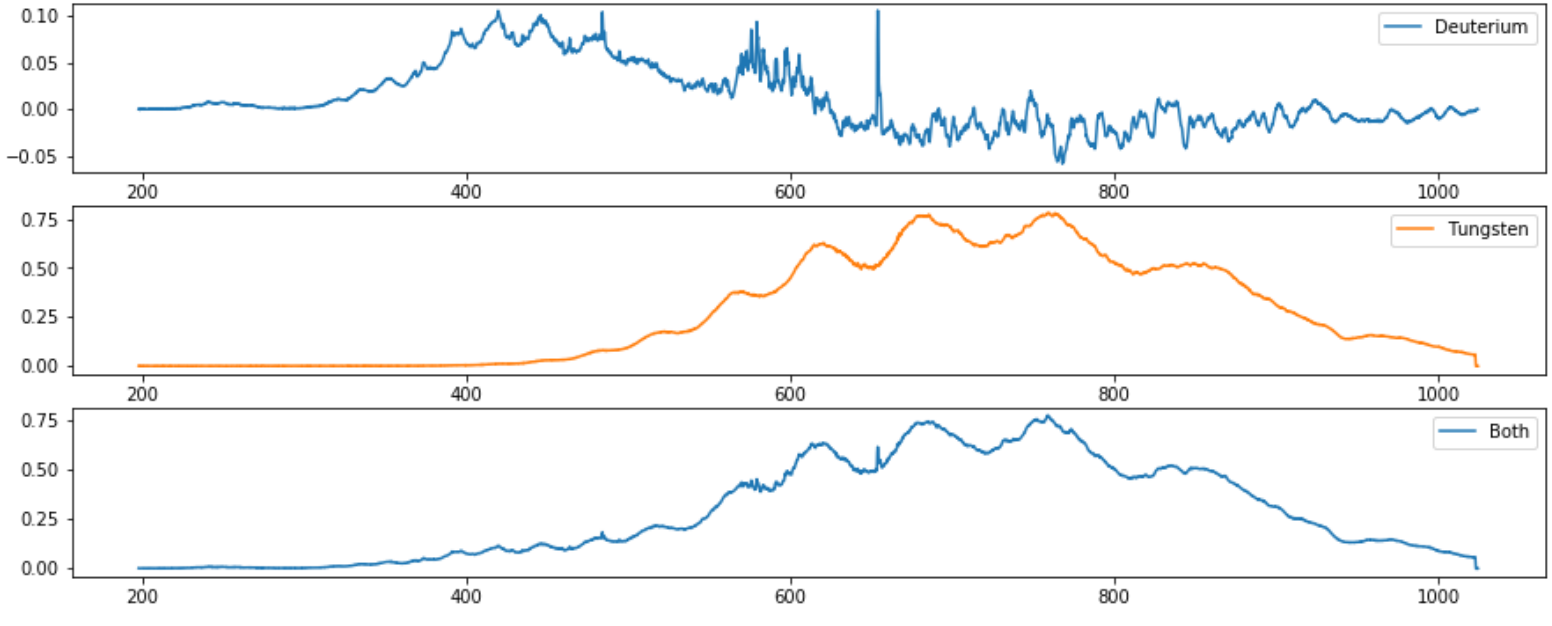
But here we can see that, the space between the plots is not enough, one possible solution for this is changing the size of the figure,

In [29]:
```
fig=plt.figure(figsize=(15,6))
ax1=fig.add_subplot(311); ax2=fig.add_subplot(312); ax3=fig.add_subplot(313)
ax1.plot(xD,yD,label='Deuterium'); ax1.legend()
ax2.plot(xW,yW,label='Tungsten');ax2.legend()
ax3.plot(xB,yB,label='Both'); ax3.legend()
plt.savefig('plot1.pdf')
plt.show()
```



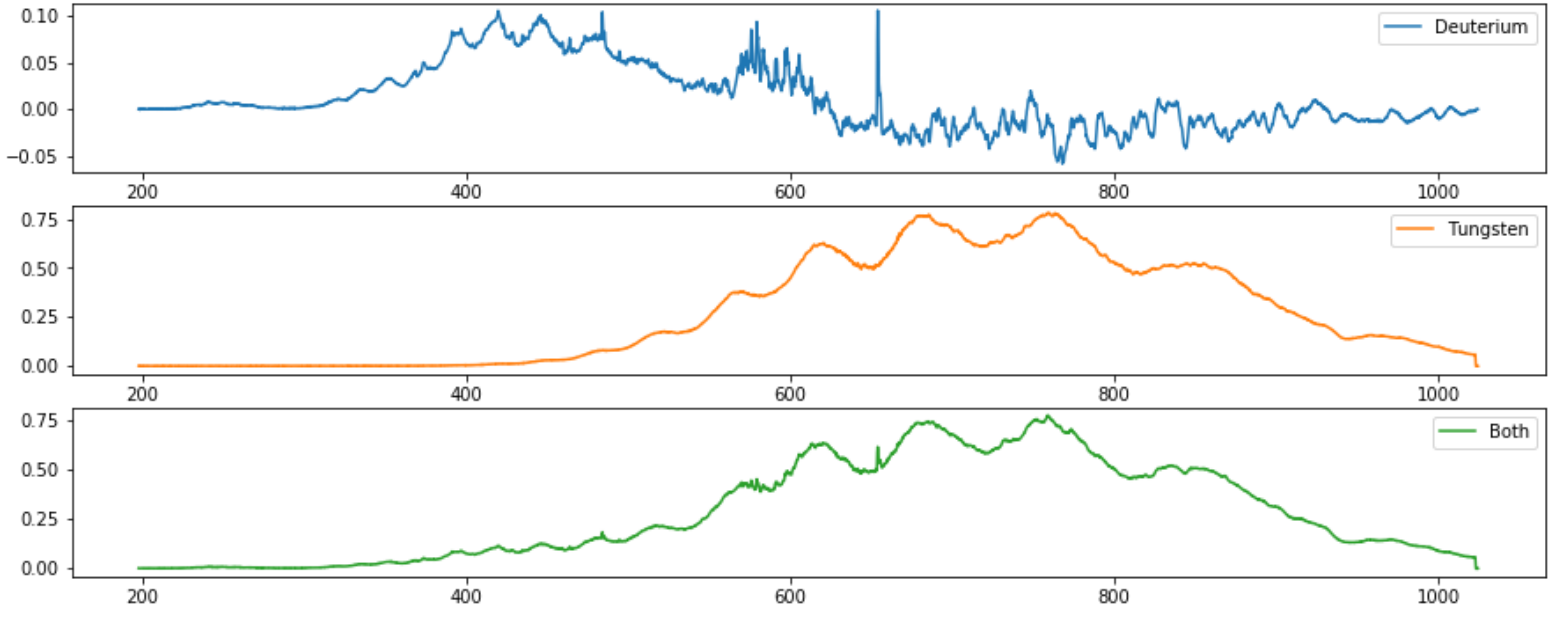
We can also change the color of the plots, for the default colors,

In [30]:
```
fig=plt.figure(figsize=(15,6))
ax1=fig.add_subplot(311); ax2=fig.add_subplot(312); ax3=fig.add_subplot(313)
ax1.plot(xD,yD,label='Deuterium'); ax1.legend()
ax2.plot(xW,yW,c='C1',label='Tungsten'); ax2.legend()
ax3.plot(xB,yB,label='Both'); ax3.legend()
plt.savefig('plot2.pdf')
plt.show()
```



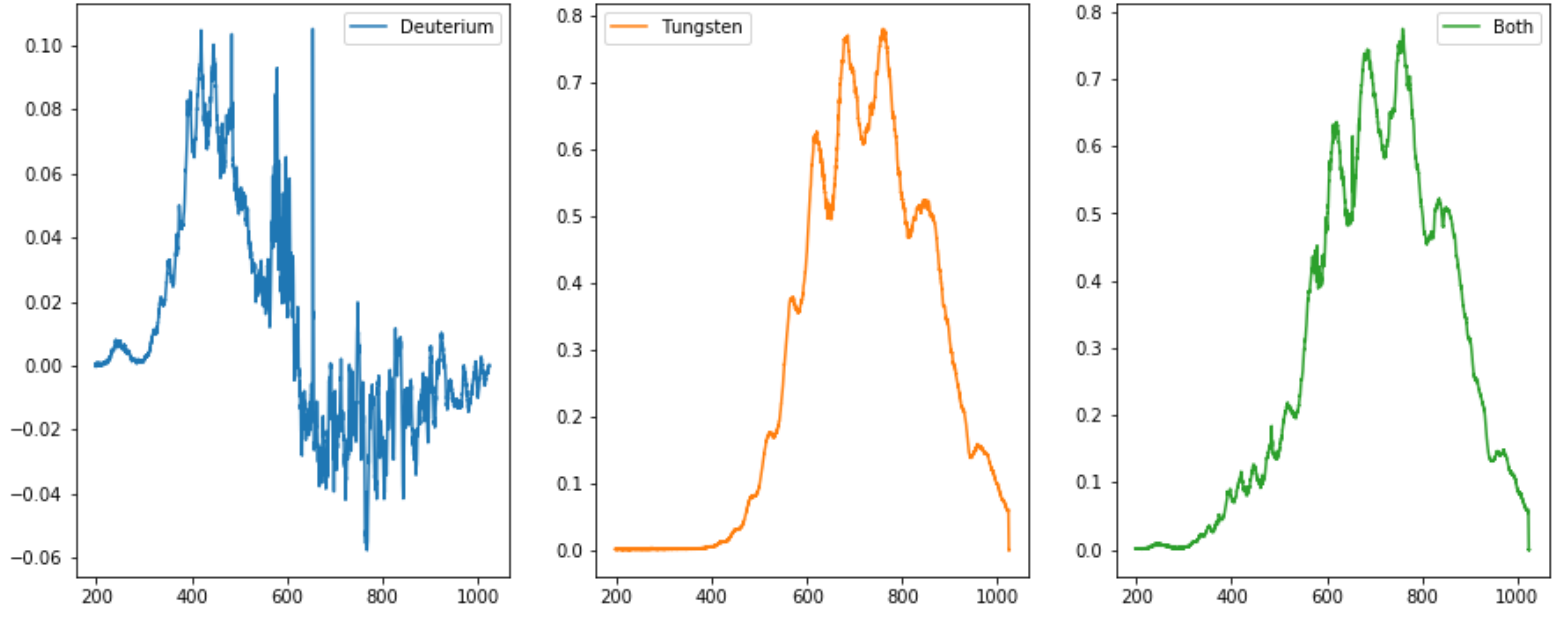
The numeration starts on 0

In [31]:
```
fig=plt.figure(figsize=(15,6))
ax1=fig.add_subplot(311); ax2=fig.add_subplot(312); ax3=fig.add_subplot(313)
ax1.plot(xD,yD,c='C0',label='Deuterium'); ax1.legend()
ax2.plot(xW,yW,c='C1',label='Tungsten'); ax2.legend()
ax3.plot(xB,yB,c='C2',label='Both'); ax3.legend()
plt.savefig('plot3.pdf')
plt.show()
```



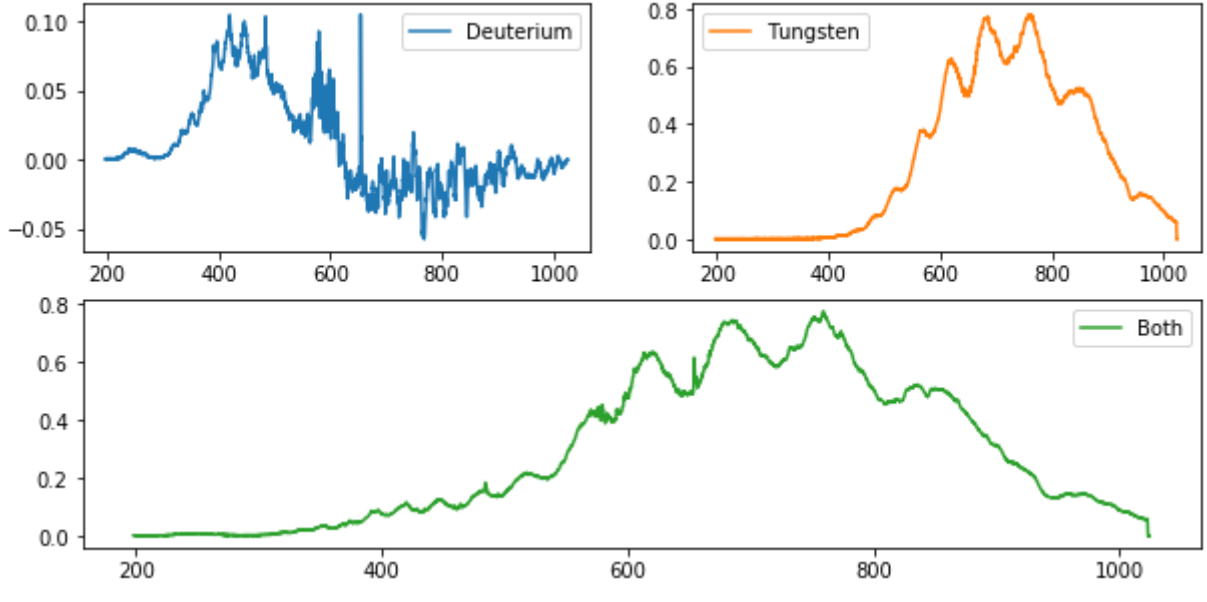
A different example, is getting three columns and just one row

In [32]:
```
fig=plt.figure(figsize=(15,6))
ax1=fig.add_subplot(131); ax2=fig.add_subplot(132); ax3=fig.add_subplot(133)
ax1.plot(xD,yD,label='Deuterium'); ax1.legend()
ax2.plot(xW,yW,c='C1',label='Tungsten'); ax2.legend()
ax3.plot(xB,yB,c='C2',label='Both'); ax3.legend()
plt.savefig('plot4.pdf')
plt.show()
```



We can arange the plots differently,

In [33]:
```
fig=plt.figure(figsize=(10,5))
ax1=fig.add_subplot(221); ax2=fig.add_subplot(222); ax3=fig.add_subplot(212)
ax1.plot(xD,yD,label='Deuterium'); ax1.legend()
ax2.plot(xW,yW,c='C1',label='Tungsten'); ax2.legend()
ax3.plot(xB,yB,c='C2',label='Both'); ax3.legend()
plt.savefig('plot5.pdf')
plt.show()
```



Lorentz System

Is a system of differential equations which becomes very famous, because it exhibits chaotic behavior with certain combination of paramters,

Named after Edward Lorenz, which during 1963 was developing a simplified mathematical model for atmospheric convection.

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned}$$

Where σ, ρ and β are the model parameteres.

The code that solves those equations can be found at the repository on the examples folder link (https://github.com/jmsevillam/Herramientas-Computacionales-UniAndes/blob/master/examples/Lorentz%20Model/Lorentz.py).

On the following link there is a file with the data, on three columns

x	y	z

We load just as before,

In [34]:
```
url='https://raw.githubusercontent.com/jmsevillam/Herramientas-Computacionales-Uni
Andes/master/examples/Lorentz%20Model/'
data=np.genfromtxt(url+'Lorentz.dat')
```

The data has the exact same structure as before,

In [35]:
```
print(data)
```

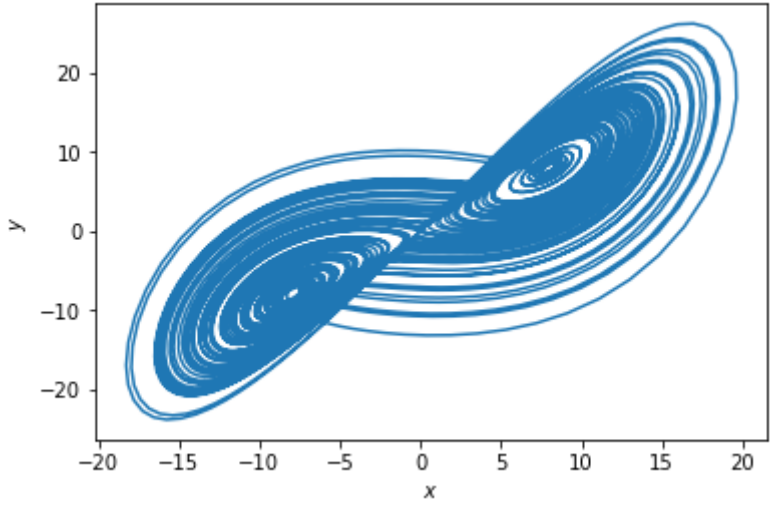
```
[[0.00000000e+00 1.00000000e+00 0.00000000e+00]
 [1.00000000e-01 9.90000000e-01 0.00000000e+00]
 [1.89000000e-01 1.00510000e+00 9.90000000e-04]
 ...
 [5.62775790e+00 6.07865401e+00 2.05607171e+01]
 [5.67284751e+00 6.26769956e+00 2.03545232e+01]
 [5.73233271e+00 6.46855338e+00 2.01672930e+01]]
```

So we must split it into the three columns

In [36]:
```
x=data[:,0]
y=data[:,1]
z=data[:,2]
```

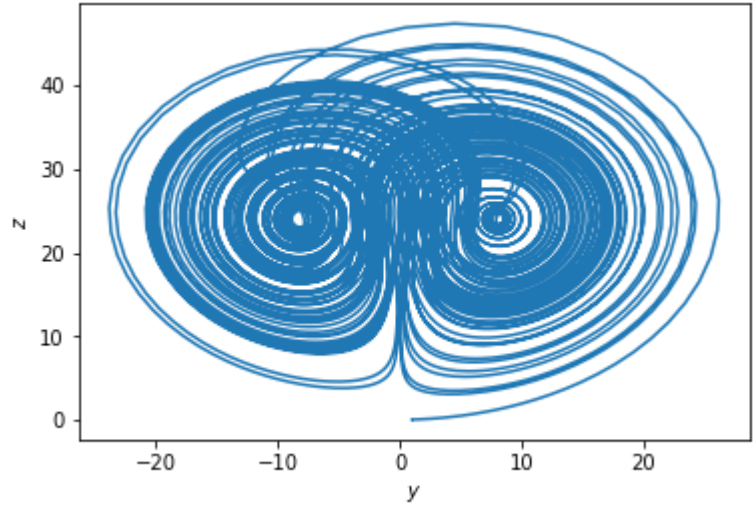
As we have three coordinates, we can plot the projections,

In [37]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.plot(x,y)
plt.show()
```



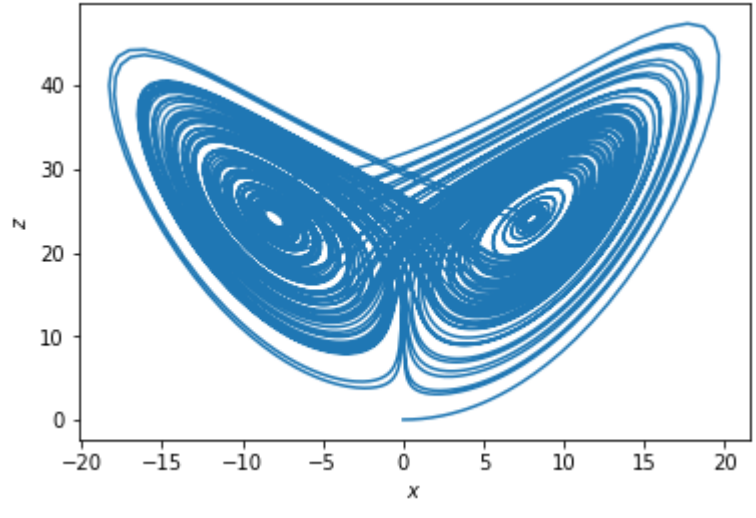
On a different axis,

In [38]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlabel('$y$')
ax.set_ylabel('$z$')
ax.plot(y,z)
plt.show()
```



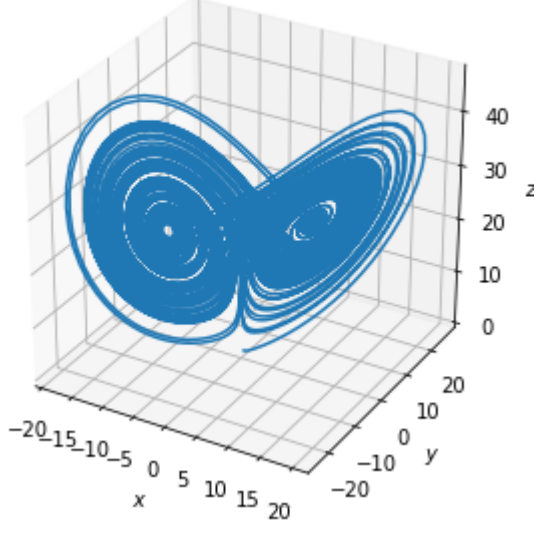
And on the last one!

In [39]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlabel('$x$')
ax.set_ylabel('$z$')
ax.plot(x,z)
plt.show()
```



If we want to visualize this on the three dimensions, by using another part of `matplotlib`,

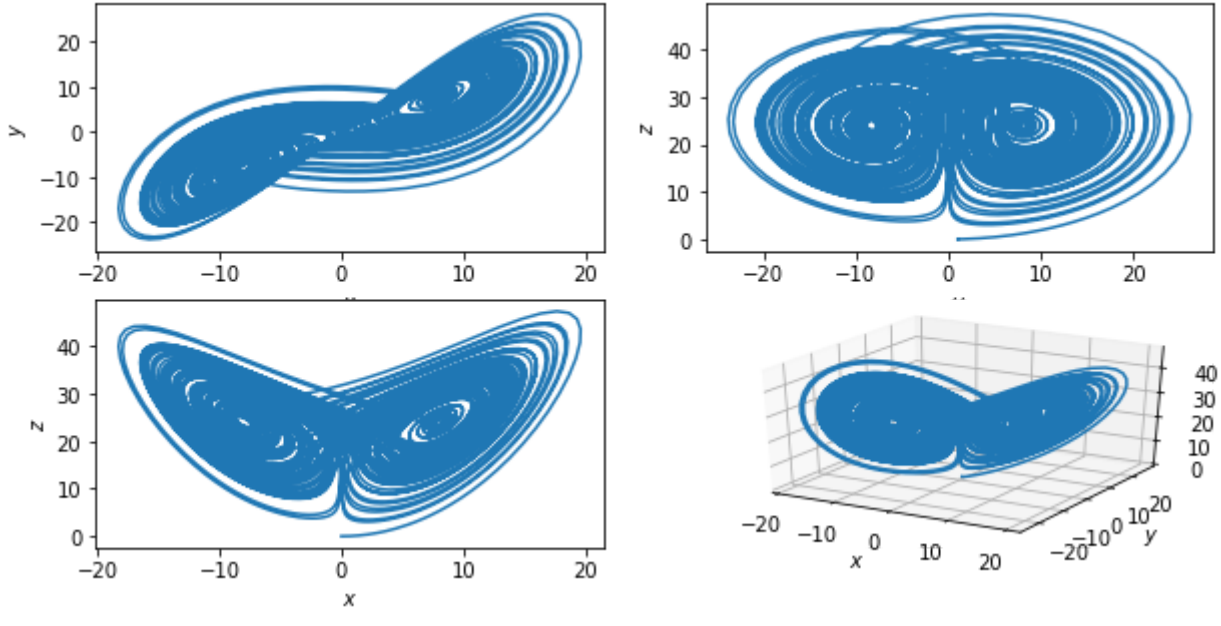
In [40]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(111,projection='3d')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$z$')
ax.plot(x, y, z)
plt.show()
```



The plots do not have to be of the same kind when plotting the axes,

In [41]:
```
fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(221); ax1.set_xlabel('$x$'); ax1.set_ylabel('$y$')
ax1.plot(x, y)
ax2 = fig.add_subplot(222); ax2.set_xlabel('$y$'); ax2.set_ylabel('$z$')
ax2.plot(y, z)
ax3 = fig.add_subplot(223); ax3.set_xlabel('$x$'); ax3.set_ylabel('$z$')
ax3.plot(x, z)
ax4 = fig.add_subplot(224,projection='3d'); ax4.set_xlabel('$x$'); ax4.set_ylabel(
'$y$'); ax4.set_zlabel('$z$')
ax4.plot(x, y, z)
```

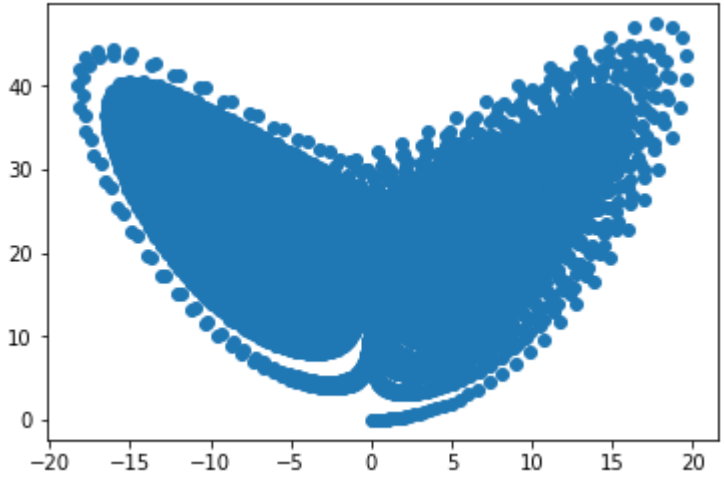
Out[41]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x10edfb4e0>]
```



Scatter

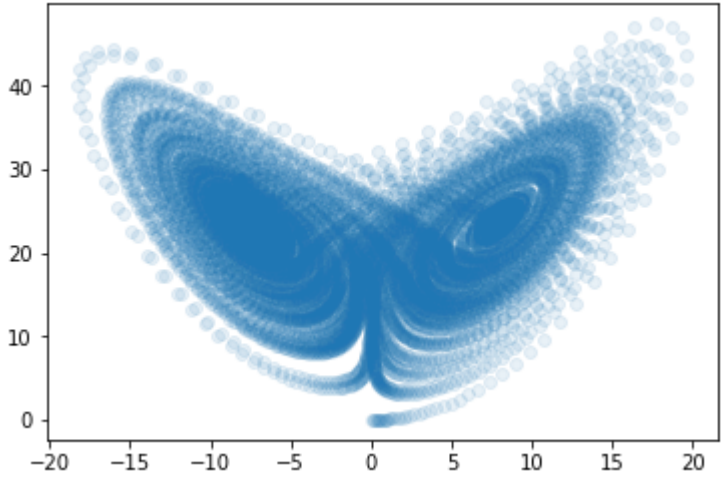
On `matplotlib` , there are several styles of plotting, a different than `plot` , is the `scater` , which looks like plotting with circles at first sight, but it will allows us to use many options,

In [42]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.scatter(x,z)
plt.show()
```



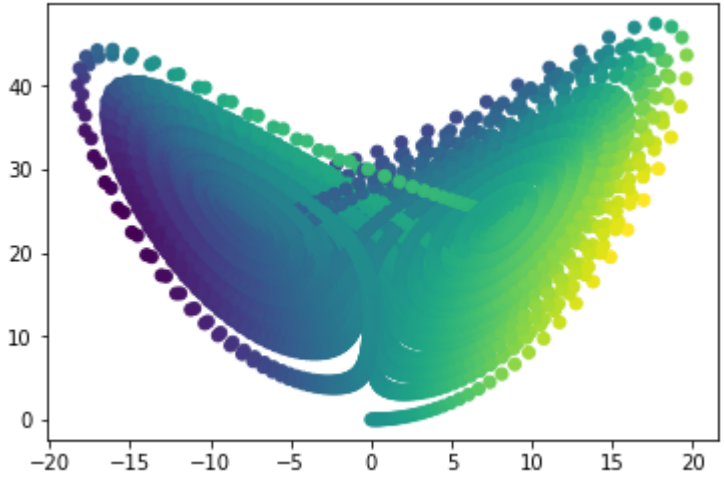
One option is the transparence,

In [43]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.scatter(x,z,alpha=0.1)
plt.show()
```



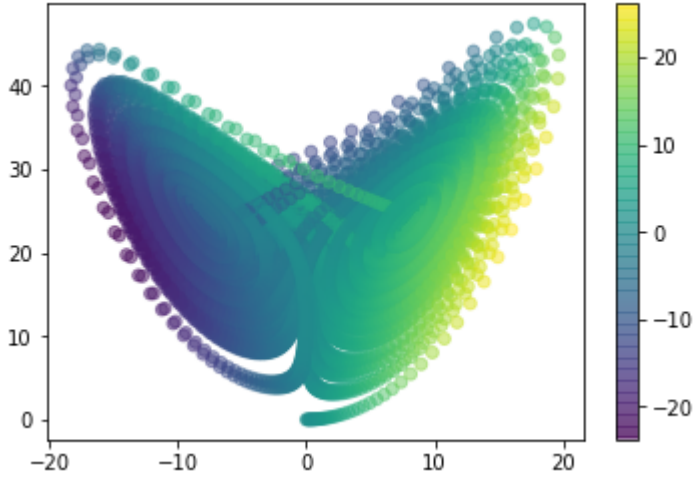
another one, is that we can use a pallete with the same or different array,

In [44]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.scatter(x,z,c=y)
plt.show()
```



They can be combinated, and even we can generate a colorbar from there,

In [45]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
pl=ax.scatter(x,z,c=y,alpha=0.5)
plt.colorbar(pl)
plt.show()
```



Surface plots

we are going to explore how to use two methods to plot surfaces, the first one, with the `Axes3d` we have already used, and the other as if we are looking from above and the high is meant to be represented with color, just as a contourplot.

It is necessary that we understand the `numpy.meshgrid` function, to do so, let us create a samall test

In [46]:
```
x=np.array([1,2])
y=np.array([3,4])
print(np.meshgrid(x,y))
```

```
[array([[1, 2],
       [1, 2]]), array([[3, 3],
       [4, 4]])]
```

It returns two arrays, so we have to separate them as,

In [47]:
```
X,Y=np.meshgrid(x,y)
print(X)
print(Y)
```

```
[[1 2]
 [1 2]]
[[3 3]
 [4 4]]
```

So, if we think this as the pairs formed from `(X[i,j],Y[i,j])`, they are all the possible combinations of the values of `x` and `y`.

For the surface plotting, first, we define a function we want to plot. (This example is thought to be on polar coordinates).

In [48]:
```
def f(r):
    return np.exp(-r/5)*np.cos(3*r)
```

We have to define the points where we are going to generate the plot.

In [49]:
```
x=np.linspace(-5,5,1000)
y=np.linspace(-5,5,1000)
```

Create a grid, and from there, we create another grid with the radious information (Polar coordinates),

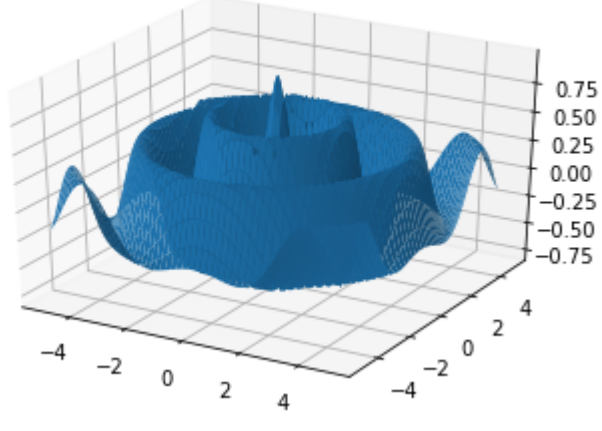
In [50]:
```
X,Y=np.meshgrid(x,y)
R=np.sqrt(X**2+Y**2)
```

We use the function we want to test,

In [51]:
```
Z=f(R)
```

Finally we can plot these matrices with the `plot_surface` method,

In [52]:
```
fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
surf = ax.plot_surface(X, Y, Z)
plt.show()
```



The other way, only needs one parameter, and uses `range` s for the coordinates

In [53]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.imshow(Z)
plt.show()
```