Python and MC applications

lecture 1 - python set-up and essential modules

ian.Postuma@pv.infn.it

What you need for the course

Basic python knowledge

Possibly a **linux** working environment

Some experience with 

terminals

What to expect from **lecture 1**

Python installation and environment management

Matplotlib & Numpy intro and basics

3D data handling for lecture 1 & 2

What is python™ ?

CrossPlatform programming language

Very High Level Language - **VHLL**

Object Oriented and easy to split into packages

Interactive interpreter, i.e. it is easy to experiment

Easily **extensible** through C

Named after “**Monty Python’s Flying Circus**” not the reptile !

British surreal sketch comedy TV show

1969 - 1974

Essentially

There are no ; at the end of a line

is the start of a comment line

Code looks always good, **indentation is mandatory**

Extensive libraries and build in features,
e.g. these are the same:

```
myList = []
for i  in range(10):
    myList.append(i)
```

```
range(10)
```

```
[i for i in range(10)]
```

Numbers

```
Python 3.8.11 (default, Aug  6 2021, 08:56:27)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # this is a comment
>>> # We start with numbers
>>> # define and assign variables a and b
>>> a = 17
>>> b = 3
>>> # a and b are defined as int values
>>> c = a / b # c is now a float 5.666666666666667
>>> d = a // b # floor division
>>> print(d)
5
>>> e = a % b # remainder of the division
>>> print(e)
2
>>> # floored quotient * divisor + remainder
>>> print( d * b + e )
17
```

Numbers and strings

```
>>> # simple power format
>>> a = 8**2
>>> print(a)
64
>>> # strings are arrays of characters
>>> s = "Python"
>>> print(s[2])
t
>>> print(s[-4])
t
>>> # Strings can be concatenated
>>> s = "hello " + s
>>> print(s)
hello Python
>>>
>>> # Portions of strings may be selected
>>> print(s[5:])
 Python
>>> len(s) # get the length of the string
12
```

Indices may be positive or negative integers. 0 being the first value, 1 the second and -1 the last ...

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Slice of an array, only the values after the first 5 indices are selected

Numbers and strings

```
>>> # simple power format
>>> a = 8**2
>>> print(a)
64
>>> # strings are arrays of characters
>>> s = "Python"
>>> print(s[2])
t
>>> print(s[-4])
t
>>> # Strings can be concatenated
>>> s = "hello " + s
>>> print(s)
hello Python
>>>
>>> # Portions of strings may be selected
>>> print(s[5:])
 Python
>>> len(s) # get the length of the string
12
```

Indices may be positive or negative integers. 0 being the first value, 1 the second and -1 the last …

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Slice of an array, only the values after the first 5 indices are selected

Numbers and strings

```
>>> # simple power format
>>> a = 8**2
>>> print(a)
64
>>> # strings are arrays of characters
>>> s = "Python"
>>> print(s[2])
t
>>> print(s[-4])
t
>>> # Strings can be concatenated
>>> s = "hello " + s
>>> print(s)
hello Python
>>>
>>> # Portions of strings may be selected
>>> print(s[5:])
 Python
>>> len(s) # get the length of the string
12
```

Indices may be positive
or negative integers.
0 being the first value,
1 the second and
-1 the last …

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Slice of an array,
only the values after
the first 5 indices are
selected

Lists

```
>>> # Python list is an array of values, like strings
>>> myList = range(10)
>>> print(myList)
[0 ,1 …, 9]
>>> # now we define a list of cube values through a for cycle
>>> cubes = [] # empty list
>>> for i in myList: # loop through myList
...     cubes.append(i**2) # append cube values to cubes
...
>>> print(cubes)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> # the same result can be achieved simply by
>>> cubes = [i**2 for i in myList]
>>> print(cubes)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> # this one line python list definition
>>> # makes code run faster an easier to read
```

Same outcome

User defined functions

```
>>> # definition of a function that returns a python list
>>> # of cube values in a range Xmin to Xmax
>>> # by defaul Xmin = 0
>>> def CubesList(Xmax, Xmin=0):
...     # check if Xmax > Xmin
...     if Xmax < Xmin:
...         # if Xmax < Xmin we return an error message
...         # and exit the function
...         print("Error Xmax < Xmin")
...         return 1
...     # check if values are integers
...     if not isinstance(Xmax,int) or not isinstance(Xmin,int):
...         # if not int we return an error message
...         # and exit the function
...         print("Error Xmax or Xmin are not integers")
...         return 2
...     # now we define the list
...     return [ i**2 for I in range(Xmin, Xmax)]
```

User defined functions

```
>>> # Use the function
>>> print(CubesList(-2))
Error Xmax < Xmin
1
>>> print(CubesList(2.4))
Error Xmax or Xmin are not integers
2
>>> print(CubesList(4,12))
Error Xmax < Xmin
1
>>> print(CubesList(12,4))
[16, 25, 36, 49, 64, 81, 100, 121]
>>> print(CubesList(12))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

Python working environment for this course

Notebook environment with comment and scripting block

Scientific computing tool

comprehensive library for creating static, animated, and interactive visualizations

Python working environment for this course

Notebook environment with comment and scripting block

Cross platform and OpenSource

Language independent

Live Code for fast scientific coding

jupyter Welcome to P

File Edit View Insert Cell

jupyter

Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

jupyter Lorenz Differential Equations (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

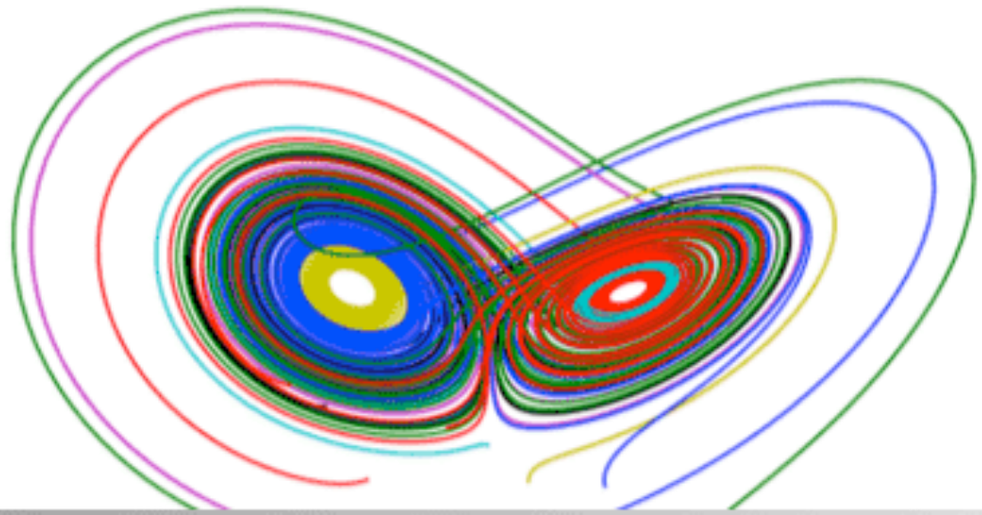
angle 308.2

max_time 12

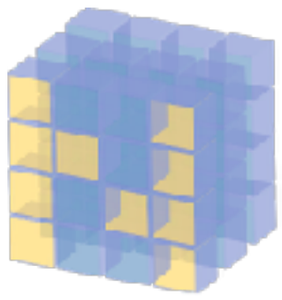
σ 10

β 2.6

ρ 28



Python working environment for this course



NumPy Scientific computing tool

```
>>> # The standard way to import NumPy:
>>> import numpy as np

>>> # Create a 2-D array, set every second element in
>>> # some rows and find max per row:

>>> x = np.arange(15, dtype=np.int64).reshape(3, 5)
>>> x[1:, ::2] = -99
>>> x
array([[  0,   1,   2,   3,   4],
       [-99,   6, -99,   8, -99],
       [-99,  11, -99,  13, -99]])
>>> x.max(axis=1)
array([ 4,  8, 13])

>>> # Generate normally distributed random numbers:
>>> rng = np.random.default_rng()
>>> samples = rng.normal(size=2500)
```

Written in C = FAST

OpenSource

The standard for:

Scientific Domains

Array Libraries

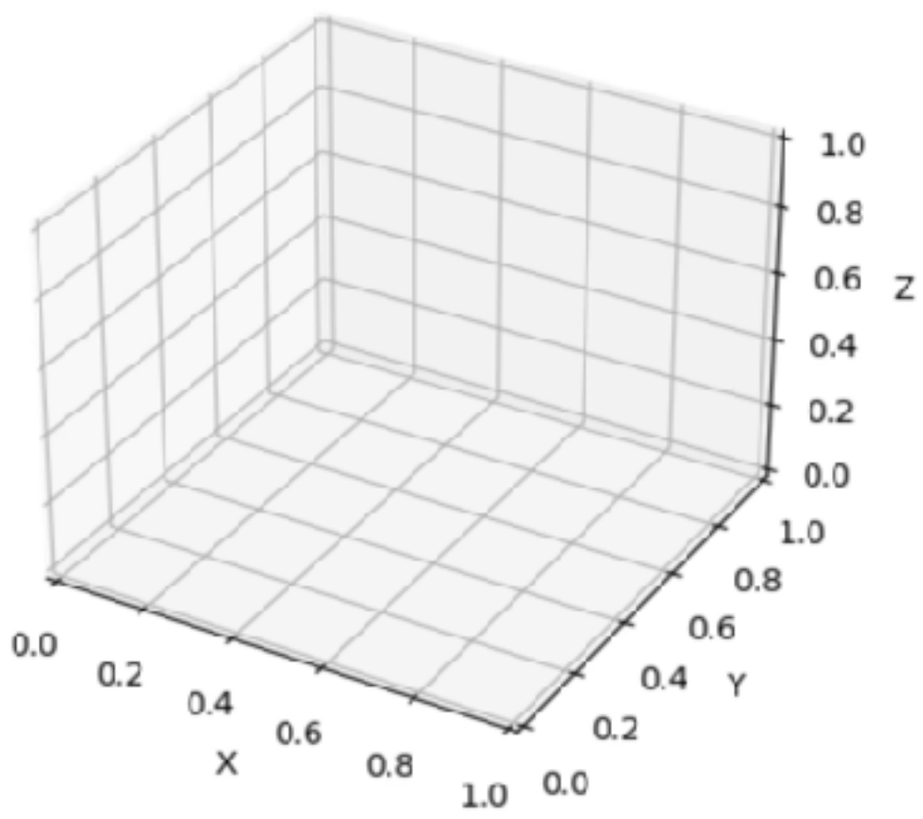
Data Science

Machine Learning

Visualisation

Python working environment for this course

matplotlib



Python working environment for this course

Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

```
conda create -y -n pythonlecture python=3.8

conda activate pythonlecture

conda install jupyter numpy matplotlib -y
```

Python working environment for this course

Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

Create an environment with the minimal working packages:

```
conda create -y -n pythonlecture python=3.8

conda activate pythonlecture

conda install jupyter numpy matplotlib -y
```

Python working environment for this course

Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

Create an environment with the minimal working packages:

```
conda create -y -n pythonlecture python=3.8
```

Conda manages environments this feature helps with package version control and encapsulation

This command creates the python lecture environment with python version 3.8

Python working environment for this course

Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

Create an environment with the minimal working packages:

```
conda create -y -n pythonlecture python=3.8

conda activate pythonlecture

conda install jupyter numpy matplotlib -y
```

Python working environment for this course

Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

Create an environment with the minimal working packages:

```
conda create -y -n pythonlecture python=3.8

conda activate pythonlecture

conda install jupyter numpy matplotlib -y
```

Download & install miniconda

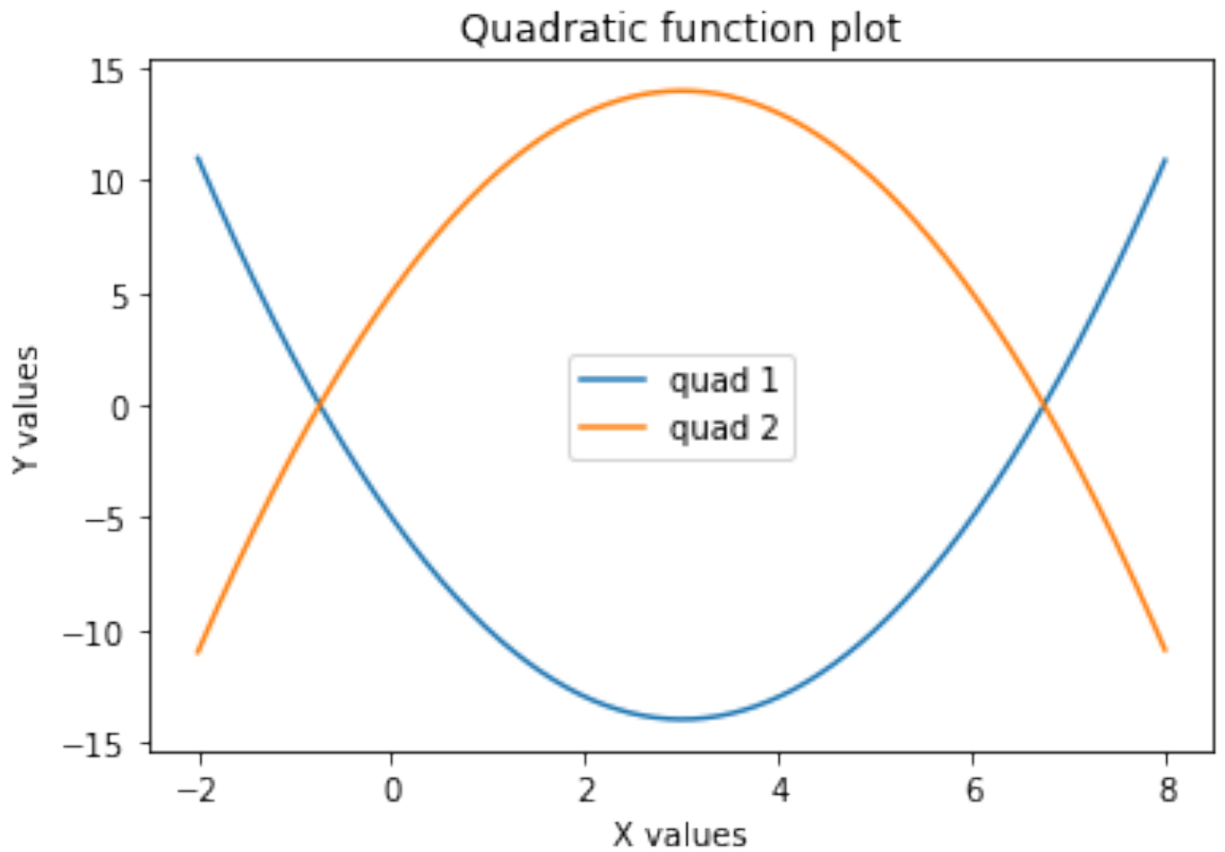
Package Manager : anaconda or miniconda
https://docs.conda.io/en/latest/miniconda.html

And try to **execute the** code **lines** in

```
Lab0_pythonIntro.py

10 minutes - please tell me if
the installation proceeded well
```

MatPlotLib - simple plot



MatPlotLib - import module

```
import matplotlib.pyplot as plt
```

As in C or C++ you start by uploading the necessary python modules. In this case we load the `pyplot` module of the `matplotlib` package. For simplicity we can define an alias, which in this case is defined as `plt`. So instead of using the module name `matplotlib.pyplot` we may use the module by calling `plt`.

MatPlotLib - data

Let us now prepare the data. For this example we define a quadratic function from which we will define the x and y axes. instead of using a for loop we will use the python list feature `[i for i in range(10)]`. But first we will define a function.

```
def quad(x,a,b,c):
    return a*x**2 + b*x + c
```

In python you define a function by starting the line with `def` followed by the function name and parameters `quad(x,a,b,c)`. In a function you may define many variables and algorithms. In this case we return the values of a quadratic functions. In the following line we will define the a list of x and y values.

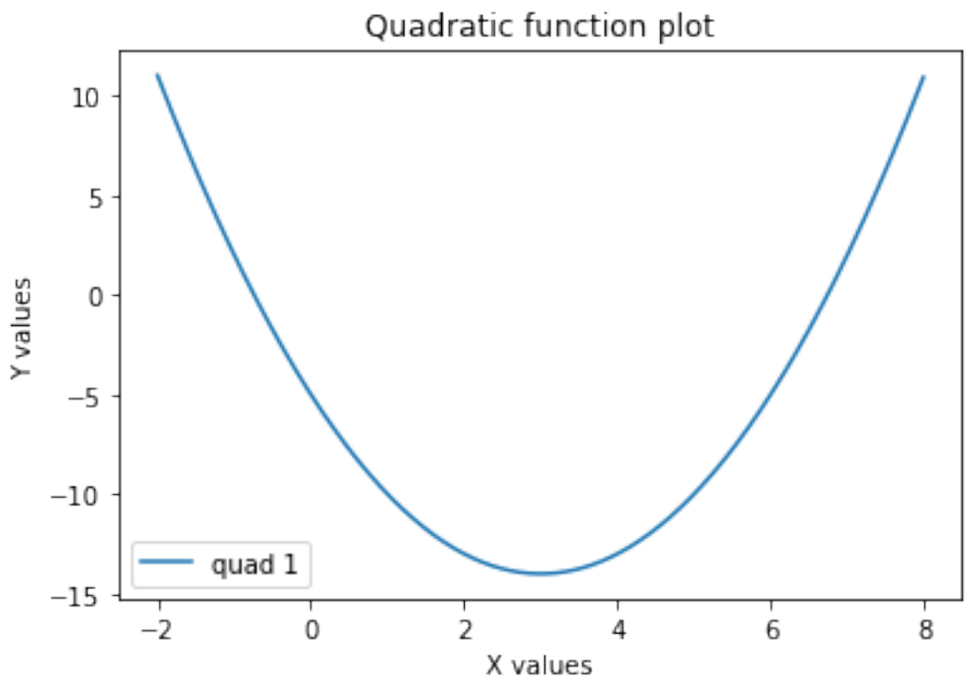
MatPlotLib - data

```
x = [ i/100 - 2 for i in range(1000)]
y = [ quad(i,1,-6,-5) for i in x]
```

Once the x and y values are defined we may plot the outcome. Simply by usig the `plot` function of the `plt` module. The first twor arguments of the `plot` function are the x and y lists respectively. The following arguments must be anticipated by the argument name, for example we use the `label` argument. Other than `plot` we use other methods such as: `title`, `xlabel`, `ylabel`, `legend`, show and `close`.

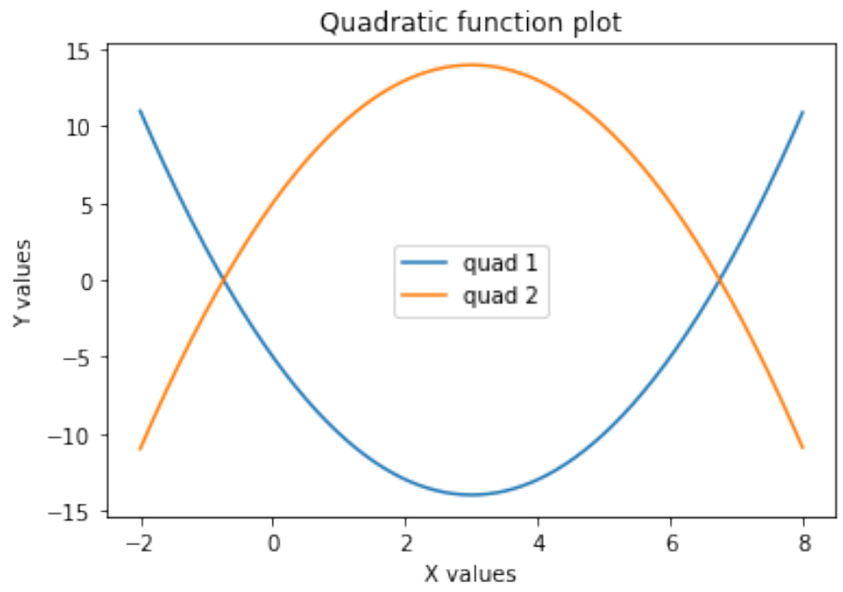
MatPlotLib - plot

```
plt.title("Quadratic function plot")
plt.plot(x,y,label="quad 1")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend()
plt.show()
plt.close()
```



MatPlotLib - plot

```
x2 = [ i/100 - 2 for i in range(1000)]
y2 = [ quad(i,-1,6,5) for i in x]
plt.title("Quadratic function plot")
plt.plot(x,y,label="quad 1")
plt.plot(x2,y2,label="quad 2")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend()
plt.show()
plt.close()
```

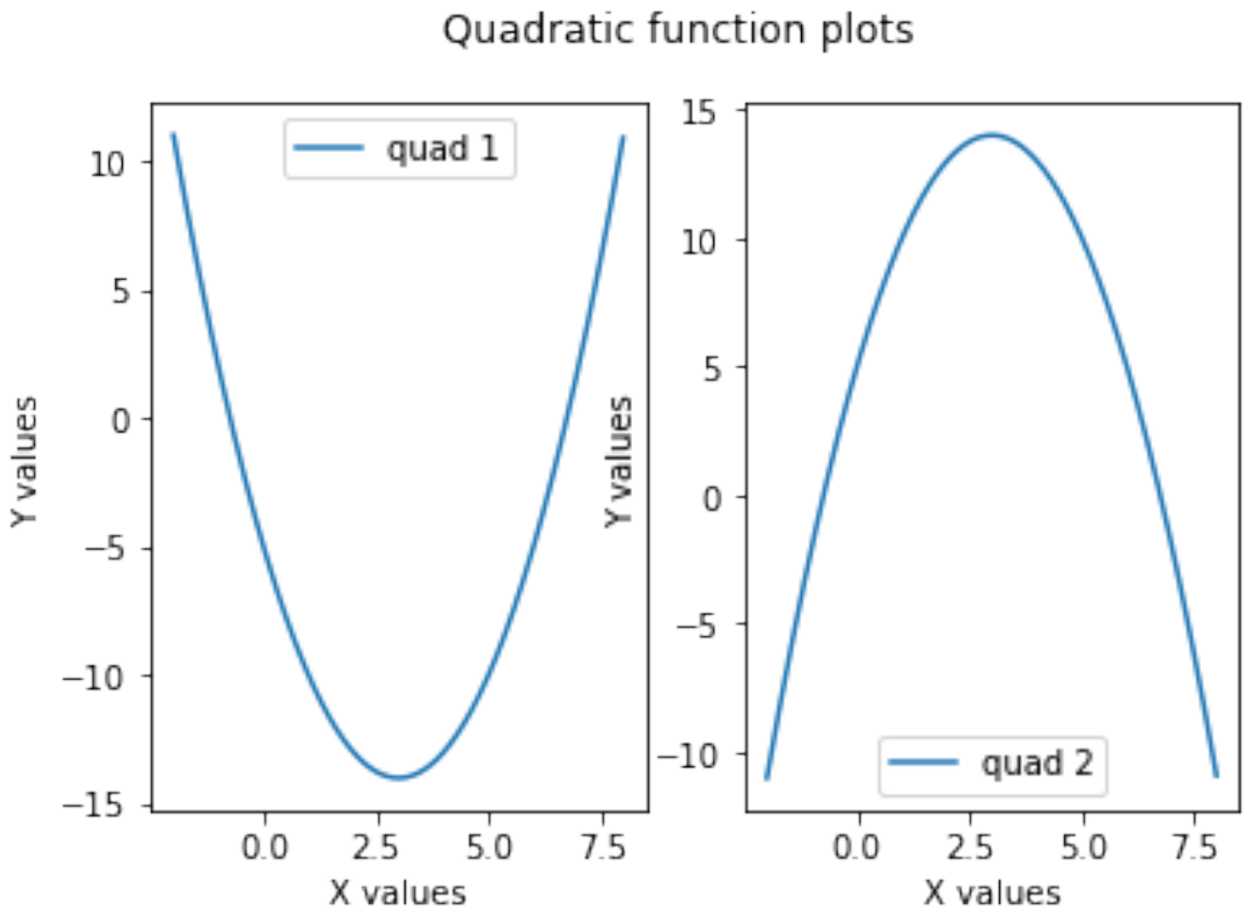


MatPlotLib - double plot

```
fig, ax = plt.subplots(1,2)
fig.suptitle("Quadratic function plots")
ax[0].plot(x,y,label="quad 1")
ax[1].plot(x2,y2,label="quad 2")
for a in ax:
    a.set_xlabel("X values")
    a.set_ylabel("Y values")
    a.legend()
plt.show()
plt.close()
```

To set more than one plot on a canvas (eg. 2) the `subplots` is the method to call. It separates the canvas in rows and columns, the first argument is the number of rows while the second argument is the number of columns. It returns two variables, a Figure object and an axes object. Now lets try to see how to plot the previous curves side by side

MatPlotLib - double plot



MatPlotLib - image plot

For this example we will load a jpg image with the build in function `imread` of the `matplotlib.image` module. The method returns an RGB array containing the pixel data of the image.

```
import matplotlib.image as mpimg
img = mpimg.imread('stinkbug.png')
print(img.shape)
```

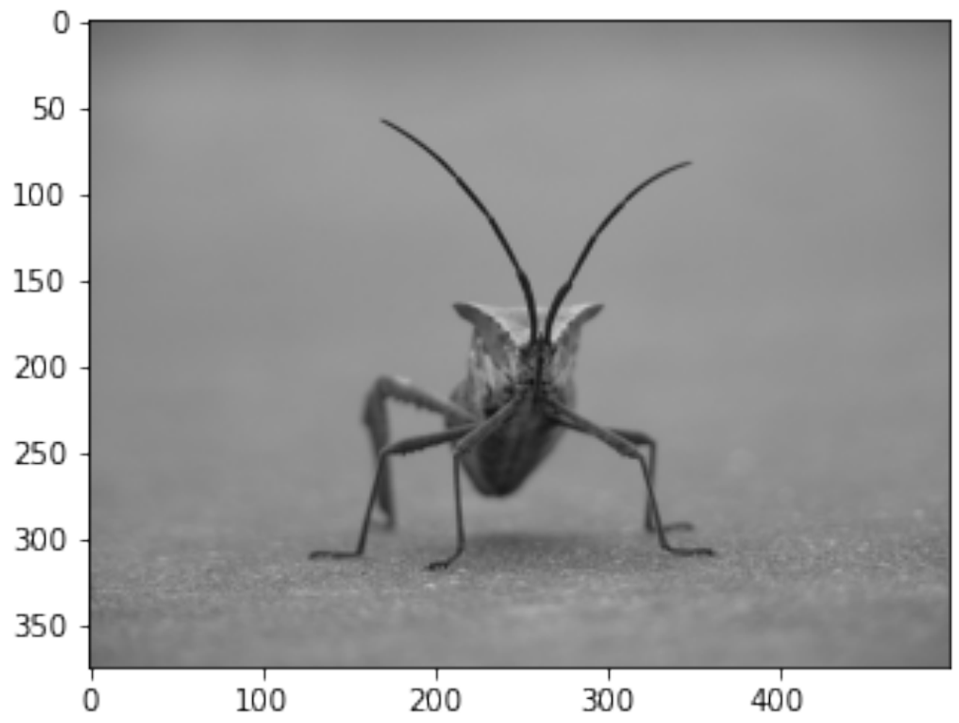
(375, 500, 3)

Which is the shape of the uploaded image, 375 pixels high, 500 pixels long and 3 channels for the red, green and blue.

MatPlotLib - image plot

Image plotting is performed with the `imshow()` function of the `matplotlib.pyplot` module

```
plt.imshow(img)
plt.show()
plt.close()
```



MatPlotLib - image plot

With the pixel information we can extract the mean intensity and its standard deviation, by using the functions in the math package. Another interesting thing is to plot the histogram `hist()` function of the `matplotlib.pyplot` module

```
import math as m
r_channel = img[:,:,0]
mean = 0
for i in r_channel.ravel():
    mean += i
mean /= len(r_channel.ravel())

std = 0
for i in r_channel.ravel():
    std += (i-mean)**2
std /= len(r_channel.ravel()) - 1
std = m.sqrt(std)

print("{:1.4f} +- {:1.4f}".format(mean,std))
```

```
0.5511 +- 0.0787
```

MatPlotLib - image plot

With the pixel information we can extract the mean intensity and its standard deviation, by using the functions in the math package. Another interesting thing is to plot the histogram `hist()` function of the `matplotlib.pyplot` module

```
import math as m

mean = 0
for i in r_channel.ravel():
    mean += i
mean /= len(r_channel.ravel())

std = 0
for i in r_channel.ravel():
    std += (i-mean)**2
std /= len(r_channel.ravel()) - 1
std = m.sqrt(std)

print("{:1.4f} +- {:1.4f}".format(mean,std))
```

0.5511 +- 0.0787

MatPlotLib - image plot

```
plt.imshow(r_channel)
plt.axis("off")
plt.show()
plt.close()
```

MatPlotLib - Exercise

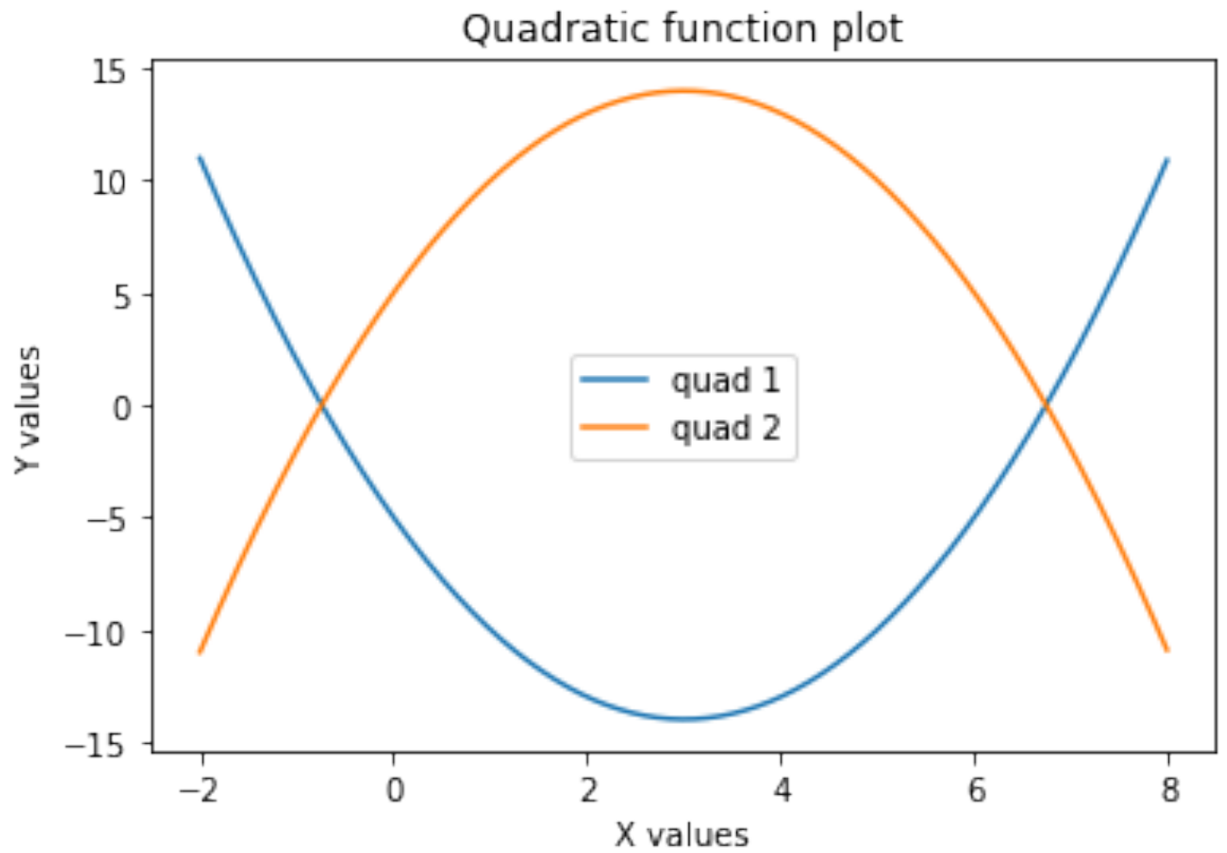
Plot blue filter image and histogram side by side

Set image and histogram title and take away the axes ticks from the image

Extra: calculate average intensity and set a vertical line in the histogram plot

TIP : search examples online -> https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html?highlight=subplots#matplotlib.pyplot.subplots

Numpy - intro


Quadratic function plot
15
10
5
0
−5
−10
−15
−2
0
2
4
6
8
Y values
X values
quad 1
quad 2

Numpy - import module

As for the `pyplot` module of the `matplotlib` package, we import the numpy package by assigning the `np` alias.

```
import numpy as np
```

Why using numpy instead of a python list ?

Numpy arrays are homogeneous, i.e. they can contain one type of data at the time; while python lists may contain more than one data type.
This makes numpy mathematical operations more efficient.
Numpy is faster and less memory greedy.

Numpy - array

Arrays are grid of values, everything rotates around this data structure of the numpy library. The array contains raw data, methods to locate and element and to interpret data. The elements are all of the same type, referred to as the array `dtype`

An array can be initialised from a python list

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(a[1])
print(a[1,3])
```

```
[5 6 7 8]

8
```

Numpy - array

ndim: will tell you the number of axes, or dimensions, of the array.

size: will tell you the total number of elements of the array. This is the product of the elements of the array's shape.

shape: will display a tuple of integers that indicate the number of elements stored along each dimension of the array. If, for example, you have a 2-D array with 2 rows and 3 columns, the shape of your array is (2, 3).

```
print(a.ndim)
print(a.size)
print(a.shape)
```

2

12

(3, 4)

Numpy - basic arrays

As shown, arrays can be created from python lists but also through numpy functions such as :

```
np.zeros(2)                                  array([0., 0.])
np.ones(2)                                   array([1., 1.])
np.arange(4)                                 array([0, 1, 2, 3])
np.arange(2, 9, 2)                           array([2, 4, 6, 8])
np.linspace(0, 10, num=5)    array([ 0. ,  2.5,  5. ,  7.5, 10. ])
np.random.random(2)                  array([0.21183343, 0.84038339])
np.ones(2, dtype=np.int64)                   array([1, 1])
```

Functions `zeros` and `ones` will create an array with specified dimensions filled with zeros or ones respectively.

Numpy - basic arrays

As shown, arrays can be created from python lists but also through numpy functions such as :

```
np.zeros(2)                                  array([0., 0.])
np.ones(2)                                   array([1., 1.])
np.arange(4)                                 array([0, 1, 2, 3])
np.arange(2, 9, 2)                           array([2, 4, 6, 8])
np.linspace(0, 10, num=5)    array([ 0. ,  2.5,  5. ,  7.5, 10. ])
np.random.random(2)                array([0.21183343, 0.84038339])
np.ones(2, dtype=np.int64)                               array([1, 1])
```

The `arange` function creates an array with a range of elements. And even an array that contains a range of evenly spaced intervals. To do this, you will specify the **first number**, **last number**, and the **step size**.

Numpy - basic arrays

As shown, arrays can be created from python lists but also through numpy functions such as :

```
np.zeros(2)                                  array([0., 0.])
np.ones(2)                                   array([1., 1.])
np.arange(4)                                 array([0, 1, 2, 3])
np.arange(2, 9, 2)                           array([2, 4, 6, 8])
np.linspace(0, 10, num=5)    array([ 0. ,  2.5,  5. ,  7.5, 10. ])
np.random.random(2)                  array([0.21183343, 0.84038339])
np.ones(2, dtype=np.int64)                               array([1, 1])
```

You can also use `np.linspace` to create an array with values that are spaced linearly in a specified interval.

Numpy - basic arrays

As shown, arrays can be created from python lists but also through numpy functions such as :

```
np.zeros(2)                                  array([0., 0.])
np.ones(2)                                   array([1., 1.])
np.arange(4)                                 array([0, 1, 2, 3])
np.arange(2, 9, 2)                           array([2, 4, 6, 8])
np.linspace(0, 10, num=5)    array([ 0. ,  2.5,  5. ,  7.5, 10. ])
np.random.random(2)                  array([0.21183343, 0.84038339])
np.ones(2, dtype=np.int64)                   array([1, 1])
```

By calling `np.random.random` it creates and array with the specified shape containing uniformly distributed random variables between 0 and 1.

Numpy - basic arrays

As shown, arrays can be created from python lists but also through numpy functions such as :

```
np.zeros(2)                                  array([0., 0.])
np.ones(2)                                   array([1., 1.])
np.arange(4)                                 array([0, 1, 2, 3])
np.arange(2, 9, 2)                           array([2, 4, 6, 8])
np.linspace(0, 10, num=5)    array([ 0. ,  2.5,  5. ,  7.5, 10. ])
np.random.random(2)                  array([0.21183343, 0.84038339])
np.ones(2, dtype=np.int64)                   array([1, 1])
```

While the default data type is floating point `np.float64`, you can explicitly specify which data type you want using the `dtype` keyword.

Numpy - reshape

Using `arr.reshape` will give a new shape to an array without changing the data. Just remember that when you use the reshape method, the array you want to produce needs to have the same `size` as the original array. If you start with an array with 12 elements, you'll need to make sure that your new array also has a total of 12 elements.

```
>>> a = np.arange(12)
>>> print(a)
[ 0  1  2  3  4  5  6  7  8  9 10 11]

>>> b = a.reshape((2,2,3))
>>> print(b)
[[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
```

Numpy - Indexing

You can index and slice NumPy arrays in the same ways you can slice Python lists. But, If you want to select values from your array that fulfil certain conditions, it's straightforward with NumPy.

```
a = np.arange(1,13,1).reshape(3,4)
print(a)

print(a[a<5])

five_up = a>=5
print(a[five_up])

print(a[a%2==0])

print(a[(a > 2) & (a < 11)])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Numpy - Indexing

You can index and slice NumPy arrays in the same ways you can slice Python lists. But, If you want to select values from your array that fulfil certain conditions, it's straightforward with NumPy.

```
a = np.arange(1,13,1).reshape(3,4)
print(a)

print(a[a<5])

five_up = a>=5
print(a[five_up])

print(a[a%2==0])

print(a[(a > 2) & (a < 11)])
```

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

Numpy - Indexing

You can index and slice NumPy arrays in the same ways you can slice Python lists. But, If you want to select values from your array that fulfil certain conditions, it's straightforward with NumPy.

```
a = np.arange(1,13,1).reshape(3,4)
print(a)

print(a[a<5])

five_up = a>=5                                  [ 5  6  7  8  9 10 11 12]
print(a[five_up])

print(a[a%2==0])

print(a[(a > 2) & (a < 11)])
```

Numpy - Indexing

You can index and slice NumPy arrays in the same ways you can slice Python lists. But, If you want to select values from your array that fulfil certain conditions, it's straightforward with NumPy.

```
a = np.arange(1,13,1).reshape(3,4)
print(a)

print(a[a<5])

five_up = a>=5                                    [ 2  4  6  8 10 12]
print(a[five_up])

print(a[a%2==0])

print(a[(a > 2) & (a < 11)])
```

Numpy - Indexing

You can index and slice NumPy arrays in the same ways you can slice Python lists. But, If you want to select values from your array that fulfil certain conditions, it’s straightforward with NumPy.

```
a = np.arange(1,13,1).reshape(3,4)
print(a)

print(a[a<5])

five_up = a>=5                                  [ 3  4  5  6  7  8  9 10]
print(a[five_up])

print(a[a%2==0])

print(a[(a > 2) & (a < 11)])
```

Numpy - shallow copy

NumPy functions, as well as operations like indexing and slicing, will return views (a shallow copy) whenever possible. This saves memory and is faster (no copy of the data has to be made). Modifying data in a view also modifies the original array!

```
b1 = a[0, :]
b1[0] = 99
print(b1)
print(“”)
print(a)
```

```
[99  2  3  4]

[[99  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Using the copy method will make a complete copy of the array and its data (a deep copy).

```
b2 = a.copy()
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**2)
print(a**(-1))
```

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

[[1. 1.]
 [1. 1.]]

[1. 1.]
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
```

```
[[1 2]   + [[1. 1.]   = [[2. 3.]
 [3 4]]     [1. 1.]]     [4. 5.]]
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
```

```
[[1 2]   -  [[1. 1.]   =   [[0. 1.]
 [3 4]]     [1. 1.]]       [2. 3.]]
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**2)
print(a**(-10))
```

```
[[1 2]   *  [[1. 1.]     =    [[1 2]
 [3 4]]     [1. 1.]]          [3 4]]
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**2)
print(a**(-10))
```

```
[[1 2]   /  [[1. 1.]    =   [[1 2]
 [3 4]]      [1. 1.]]        [3 4]]
```

Numpy - Operations

`+`, `-`, `*` and `/` operators works on NumPy arrays, but be aware that they are performed element-wise and that in some occasions broadcasting is performed.

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a*4)
print(a**10)
```

```
[[1 2]   *  4   =   [[ 4  8]
 [3 4]]              [12 16]]
```

Broadcasting

```
[[ 4  4]
 [ 4  4]]
```

Numpy - Operations

+, -, * /

broadcasting

```
a = np.arange(1,5,1).reshape((2,2))
print(a)
b = np.ones((2,2))
print(b)
c = np.ones((2))
print(c)

print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a*4)
print(a*(c+10))
```

```
[[ 11  11]
 [ 11  11]]
```

Broadcasting

```
[[1 2]  *([1. 1.] + 10) = [[11. 22.]
 [3 4]]                    [33. 44.]]
```

Broadcasting

```
[10  10]
```

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)

print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())

print(a.max(axis=0))
print(a.max(axis=1))
```

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

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)

print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())

print(a.max(axis=0))
print(a.max(axis=1))
```

min([[1 2]
 [3 4]])= 1

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)


print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())


print(a.max(axis=0))
print(a.max(axis=1))
```

```
min( [[1 2]
      [3 4]] )= 4
```

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)


print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())


print(a.max(axis=0))
print(a.max(axis=1))
```

```
10
```

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)


print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())


print(a.max(axis=0))
print(a.max(axis=1))
```

```
2.5
```

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)

print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())

print(a.max(axis=0))
print(a.max(axis=1))
```

```
1.11803…
```

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)

print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())

print(a.max(axis=0))
print(a.max(axis=1))
```

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

Numpy - Operations

Other useful array operations are: min, max, sum, mean, std ...

```
print(a)

print(a.min())
print(a.max())
print(a.sum())
print(a.mean())
print(a.std())

print(a.max(axis=0))
print(a.max(axis=1))
```

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

Numpy - flatten

There are two popular ways to flatten an array: `.flatten()` and `.ravel()`. The primary difference between the two is that the new array created using `ravel()` is actually a reference to the parent array (i.e., a “view”). This means that any changes to the new array will affect the parent array as well. Since ravel does not create a copy, it’s memory efficient.

```
a = np.arange(1,13,1).reshape((2,2,3))

print(a)
```

```
[[[ 1  2  3]
  [ 4  5  6]]
 [[ 7  8  9]
  [10 11 12]]]
```

```
print(a.ravel())
```

```
[ 1  2  3  4  5  6  7  8  9 10
11 12]
```

Numpy - math functions

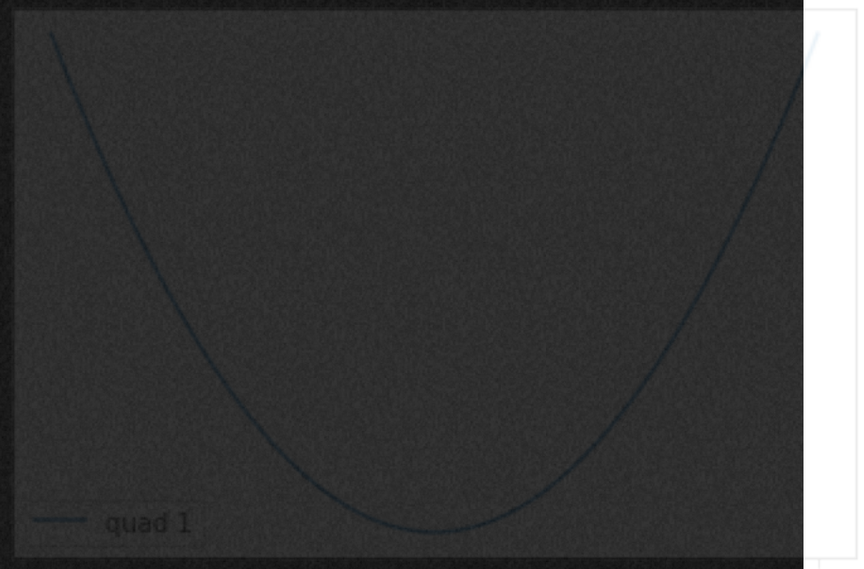
Defining mathematical formulas with NumPy arrays is easier than with python lists. Generally with NumPy one can avoid using python loops, instead one can use predefined function and operations

```
import matplotlib.pyplot as plt

def quad(x,a,b,c):
    return a*x**2 + b*x + c

x = np.linspace(-2,8, num=1000)
y = quad(x,1,-6,-5)

plt.title("Quadratic function plot")
plt.plot(x,y,label="quad 1")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend()
plt.show()
plt.close()
```



NumPy - math functions

Defining mathematical formulas with NumPy arrays is easier than with python lists. Generally with NumPy one can avoid using python loops, instead one can use predefined function and operations

```
import matplotlib.pyplot as plt

def quad(x,a,b,c):
    return a*x**2 + b*x + c

x = np.linspace(-2,8, num=1000)
y = quad(x,1,-6,-5)

plt.title("Quadratic function plot")
plt.plot(x,y,label="quad 1")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend()
plt.show()
plt.close()
```

NumPy - Exercise

Plot separately Red, Green and Blue channels of the baozi.jpg” image together with its histogram one on top of the other

Set image and histogram title and take away the axes ticks from the image

With NumPy: calculate average intensity and set a vertical line in the histogram plot

TIP : use what you have done in the previous exercise

NumPy - image masks

Masks are just arrays of bool values that are used as indexes to define which part of an image one wants to act upon. With masks we can open an image and select those pixels that have an intensity greater or smaller than a certain value.

```
import matplotlib.image as mpimg

img  = mpimg.imread('stinkbug.png')
mask = np.zeros(img.shape)
mask[img < 0.4] = 1

plt.imshow(mask)
plt.axis("off")
plt.show()
plt.close()
```

NumPy - Exercise

Plot Red, Green and Blue channels of the “baozi.jpg” image

Below each channel plot the mask for intensities greater than 150, 125 and 100 respectively for each channel

Extra: in the third row of plots, plot the filtered image, i.e. the mask over the image

TIP : use what you have done in the previous exercise