




Python for scientists: Numpy, Scipy and Matplotlib

Anna Ivagnes, Federico Pichi, Gianluigi Rozza

September 30-October 1, 2025

*Slides inspired by previous courses given
by Marco Tezzele and Nicola Demo*



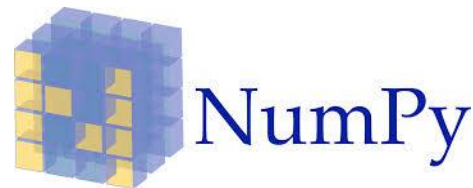


NumPy

Home: <http://www.numpy.org/>

Doc: <https://docs.scipy.org/doc/>

```
>>> import numpy
>>> a = numpy.array([1, 2, 3])
```



NumPy is a Python module implementing **N-dimensional arrays** (contiguous in memory!) and some linear algebra algorithms.

It became the standard for Python scientific programming.

Arrays

- NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called **axes**.
- NumPy's array class is called **ndarray (array)** and the most used attributes are
 - **ndim**: number of axes (dimensions) of the array
 - **shape**: a tuple of integers indicating the size of the array in each dimension
 - **size**: total number of elements of the array

```
>>> import numpy as np
>>> a = np.array([6, 7, 8])
>>> b = np.array([[ 1., 0., 0.],
                  [ 0., 1., 2.]])
```

Arrays Creation

- A frequent error consists in calling array with multiple numeric arguments

```
>>> a = np.array(1, 2, 3, 4)    # WRONG
>>> a = np.array([1, 2, 3, 4])  # RIGHT
```

- The function `zeros` creates an array full of zeros, the function `ones` creates an array full of ones

```
>>> np.zeros((3, 4))
>>> np.ones((2, 3, 4), dtype=np.int16)
```

- To create sequences of numbers the is the analogous of range: `arange`

```
>>> np.arange(3, 20, 4)
>>> np.arange(0, 2, 0.3)
```

Arrays - Some useful methods

- Many operations, such as computing the sum of all the elements in the array, are implemented as methods of the `ndarray` class

```
>>> a = np.random.random((2, 3))
>>> a.sum()
>>> a.min()
>>> a.max()
>>> b = np.arange(12).reshape((3, 4))
>>> b.min(axis=1)
```

- To access elements use the `[]` operator: `a[0, 2]` or `a[0][2]`

```
>>> a = np.arange(5)
>>> a[3] = 3
```

Arrays Slicing

- One-dimensional arrays can be indexed, sliced and iterated over, much like lists and other Python sequences

```
>>> a = np.arange(10)**2
>>> a[2:5]          # from index 2 to 5-1 (exclusive)
>>> a[:6:2] = -3     # change elements from 0 to 5 every 2 with -3
>>> a[::-1]         # reversed a
```

- Multidimensional arrays can have one index per axis.

```
>>> b = np.arange(12).reshape(3,4)
>>> b[2, 3]
>>> b[0:3, 1]      # each row in the second column (equivalent to b[:, 1])
>>> b[1:3, :]      # each column in the second and third row of b
>>> b[-1]          # the last row (equivalent to b[-1, :])
```

Shape Manipulation

- The shape of an array can be changed. The following three commands all return a modified array, but do not change the original array

```
>>> a = np.floor(10 * np.random.random((3, 4)))  
>>> a.ravel()           # returns the array, flattened  
>>> a.reshape(6, 2)     # returns the array with a modified shape  
>>> a.T                 # returns the array, transposed  
>>> a.reshape(3, -1)    # if a dimension is -1 the others are automatic
```

- The `reshape` function returns its argument with a modified shape, whereas the `ndarray.resize` method modifies the array itself

```
>>> a.resize((2, 6))
```

Basic Operations

- Arithmetic operators on arrays apply elementwise. A new array is created and filled with the result

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.arange(4)
>>> c = a-b
>>> b**2
>>> 10 * np.sin(a)
>>> a < 35
```

- Some operations, such as `+=` and `*=` act in place to modify an existing array rather than create a new one.

```
>>> a *= 3
>>> b += a
```


Arrays and Algorithms

- NumPy contains also lot of methods to handle arrays and utilities of linear algebra
- The NumPy methods are optimized (performance)
- It is extremely recommended to use the NumPy functions (or a combination of them) to compute the result, instead of implementing by yourself the algorithm
- Try to avoid to perform element-wise operation on a NumPy array (no **for**!)

```
import numpy
v = numpy.ones(1e5)
for i in range(v.shape[0]):
    v[i] += 1
```

```
import numpy
v = numpy.ones(1e5)
v += 1
```

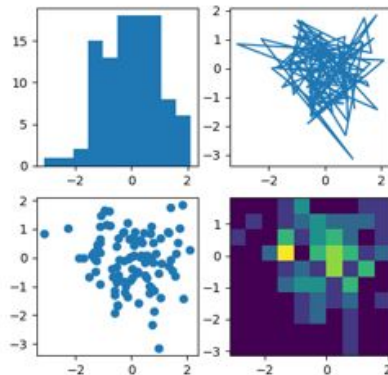
Matplotlib

Home: <https://matplotlib.org/>

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y)
>>> plt.show()
```

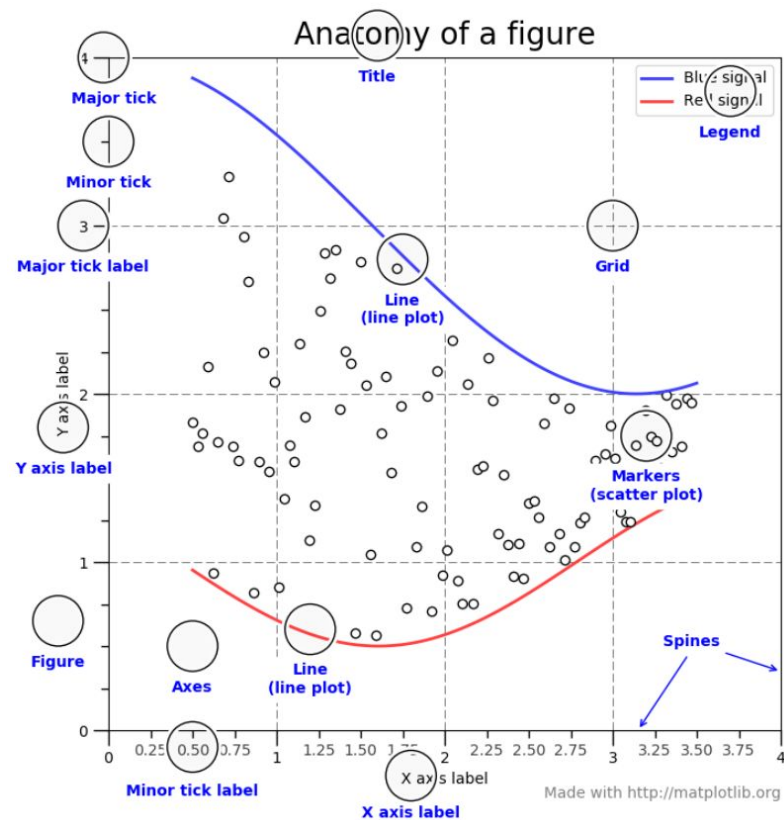


Matplotlib is a **plotting library** for the Python programming language and its numerical mathematics extension NumPy.



Parts of a Figure

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
```



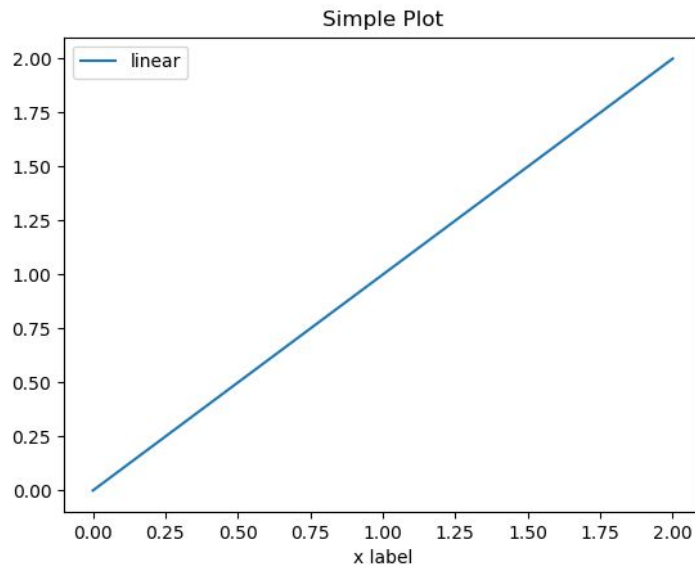
A Simple Plot

- Let's plot together some simple functions

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np

>>> x = np.linspace(0, 2, 100)
>>> plt.plot(x, x, label='linear')
>>> plt.xlabel('x label')
>>> plt.title("Simple Plot")
>>> plt.legend()
>>> plt.show()
```

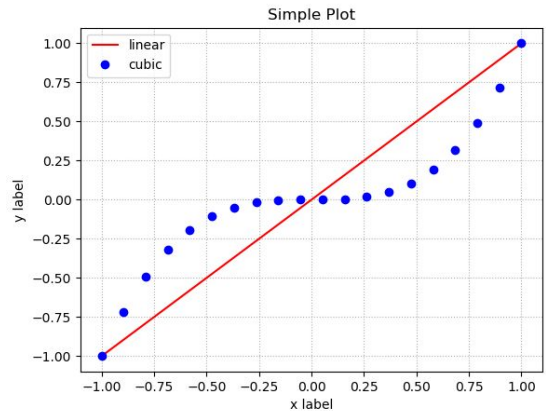
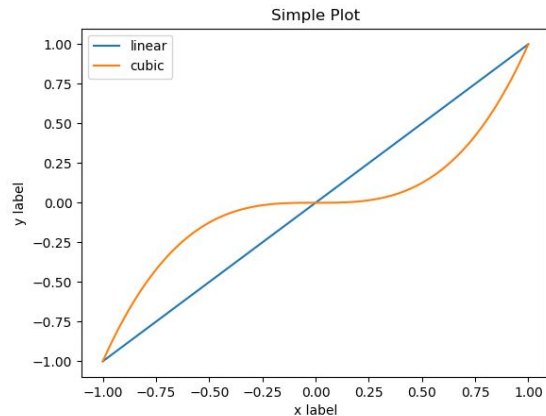
- Add a cubic function, change the x interval to [-1, 1], and add the label on the y axis



A Simple Plot

- You should have something like this plot
- Now let's add a grid, change the color, and make a scatter plot

```
>>> x = np.linspace(-1, 1, 20)
>>> plt.plot(x, x, 'r-', label='linear')
>>> plt.plot(x, x**3, 'bo', label='cubic')
>>> plt.legend()
>>> plt.grid(linestyle='dotted')
>>> plt.savefig('my_plot.pdf')
```

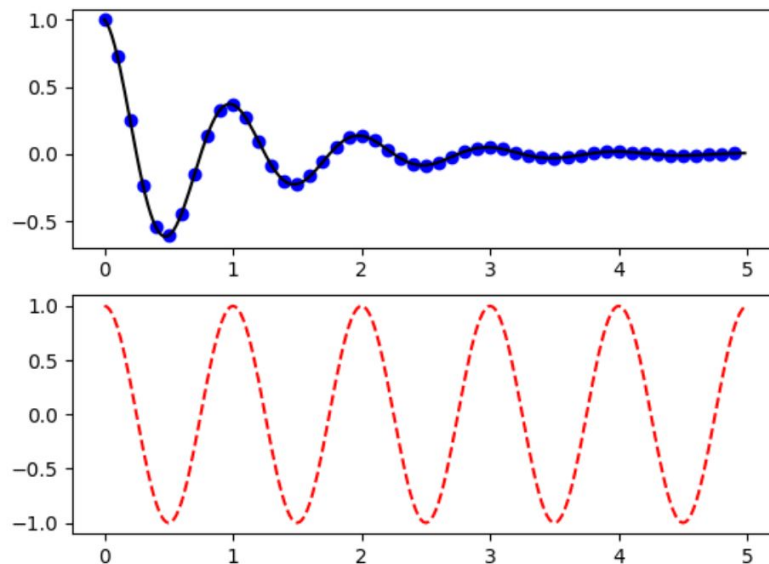


Multiple Figures and Axes

```
>>> def f(t):  
>>>     return np.exp(-t) * np.cos(2*np.pi*t)
```

```
>>> t1 = np.arange(0.0, 5.0, 0.1)  
>>> t2 = np.arange(0.0, 5.0, 0.02)
```

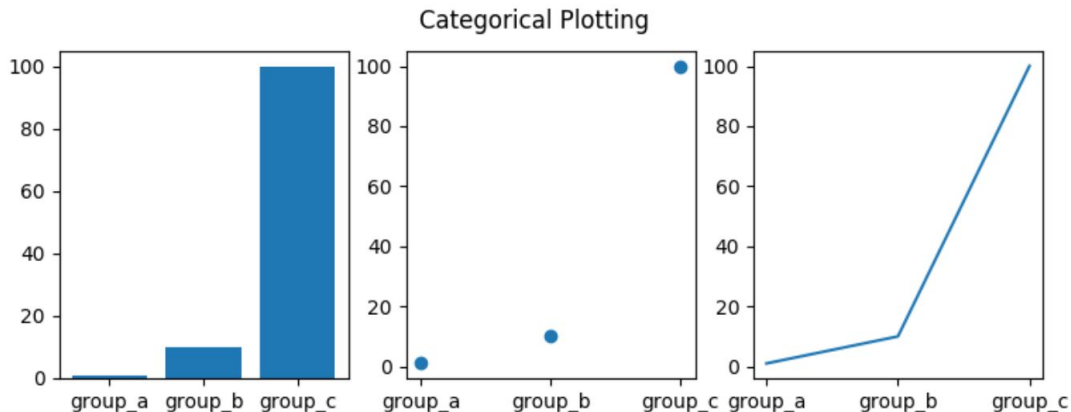
```
>>> plt.figure()  
>>> plt.subplot(211)  
>>> plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
>>> plt.subplot(212)  
>>> plt.plot(t2, np.cos(2*np.pi*t2), 'r--')  
>>> plt.show()
```



Categorical Variables

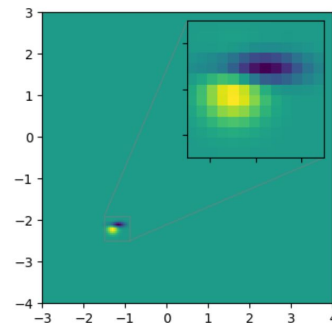
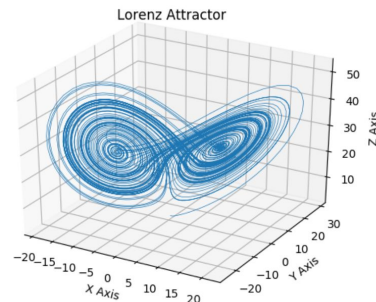
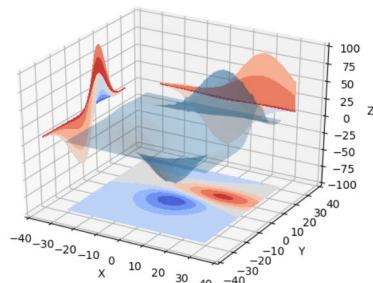
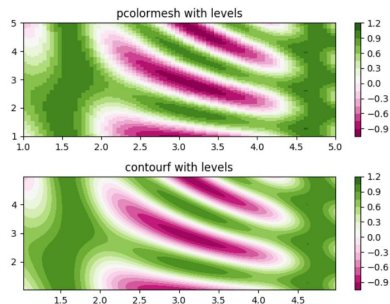
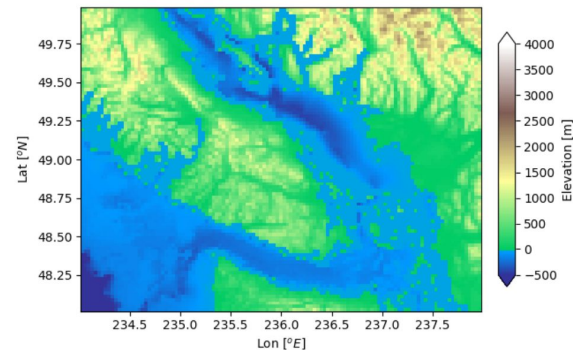
```
>>> names = ['group_a', 'group_b', 'group_c']  
>>> values = [1, 10, 100]  
>>> plt.figure(figsize=(9, 3))  
>>> plt.subplot(131)  
>>> plt.bar(names, values)
```

```
>>> plt.subplot(132)  
>>> plt.scatter(names, values)  
>>> plt.subplot(133)  
>>> plt.plot(names, values)  
>>> plt.suptitle('Categorical Plotting')  
>>> plt.show()
```



Want more?

- As soon as you know what kind of plot you need just find it in the matplotlib gallery!
- <https://matplotlib.org/gallery/index.html>





SciPy

Home: <https://www.scipy.org/>

Doc: <https://docs.scipy.org/doc/>

```
>>> from scipy.interpolate import (  
>>>     interp1d)  
>>> f = interp1d(x, y)
```



SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.

It contains modules for **optimization, interpolation, integration**, etc.

Power on Top of NumPy



- Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world, by constructing on top of NumPy special objects and classes.
- Possible applications include:
 - integration
 - optimization
 - interpolation
 - signal processing
 - fast fourier transform
 - statistics
 - image processing

Function Interpolation

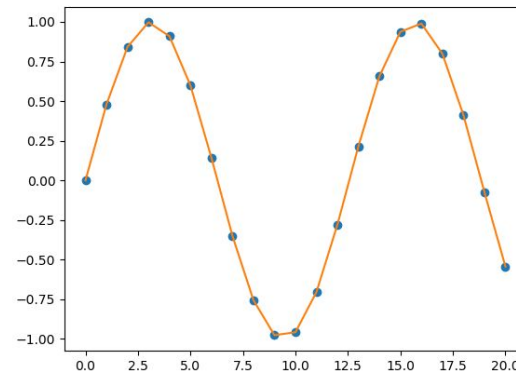
- The `scipy.interpolate` is a sub-package for objects used in interpolation
- Let's try to interpolate $y = \sin(x/2)$ in the interval $[0, 20]$ using the method `interp1d`

```
>>> import matplotlib.pyplot as plt
>>> from scipy import interpolate

>>> # train the interpolator with this points
>>> x = np.arange(0, 21)
>>> # test the interpolator over
>>> points = np.arange(0, 20.1, 0.1)
>>> # make a scatter plot of y against x
>>> # make a plot of the predicted y for points
```

Table of Contents

- [Interpolation \(scipy.interpolate\)](#)
 - [Univariate interpolation](#)
 - [Multivariate interpolation](#)
 - [1-D Splines](#)
 - [2-D Splines](#)
 - [Additional tools](#)



Function Interpolation

- Interpolate $y = \sin(x/2)$ in the interval $[0, 20]$ using the method `interpolate.interp1d()`

```
>>> # train the interpolator with this points
>>> x = np.arange(0, 21)
>>> # test the interpolator over
>>> points = np.arange(0, 20.1, 0.1)
>>> # make a scatter plot of y against x
>>> # make a plot of the predicted y for points
```

`scipy.interpolate.interp1d`

```
class scipy.interpolate.interp1d(x, y, kind='linear', axis=-1, copy=True, bounds_error=None, fill_value=nan, assume_sorted=False)
```

Interpolate a 1-D function.

[\[source\]](#)

x and y are arrays of values used to approximate some function f : $y = f(x)$. This class returns a function whose call method uses interpolation to find the value of new points.

Note that calling `interp1d` with NaNs present in input values results in undefined behaviour.

Parameters:

x : $(N,)$ array_like

A 1-D array of real values.

y : $(...,N,...)$ array_like

A N-D array of real values. The length of y along the interpolation axis must be equal to the length of x .

Scikit-learn

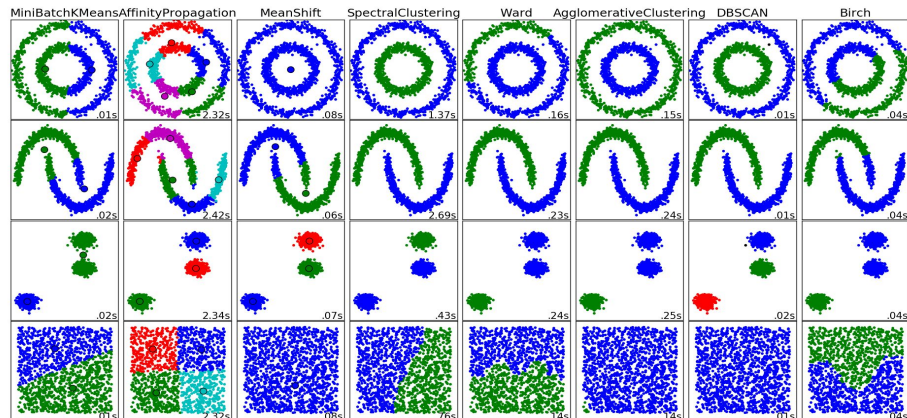
Home: <https://scikit-learn.org>

```
>>> from sklearn.cluster import KMeans
>>> kmeans = KMeans(n_clusters=2)
```



Scikit-learn is a **machine learning** library for the Python programming language.

It implements methods for **regression, clustering, classification**, etc.



Utilities for Python



PyPI / PIP

Home: <https://pypi.org/>

```
$ pip install numpy
$ pip search mypackage
$ pip install --upgrade pip
```

The Python Package Index (**PyPI**) is a repository of software for the Python programming language.

PIP is a package manager for Python software using the PyPI repo.

BE CAREFUL TO THE PIP VERSION!

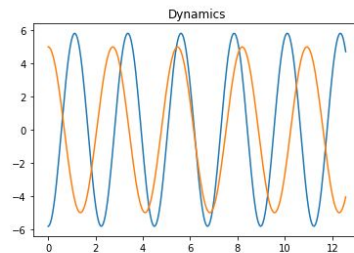
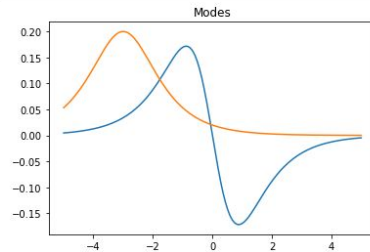
IPython Notebook

It provides a browser-based notebook interface with support for **code**, **text**, mathematical expressions, inline plots and other media.

We can plot the modes and the dynamics:

```
In [6]: for mode in dmd.modes.T:
        plt.plot(x, mode.real)
        plt.title('Modes')
        plt.show()

        for dynamic in dmd.dynamics:
            plt.plot(t, dynamic.real)
            plt.title('Dynamics')
            plt.show()
```



Finally, we can reconstruct the original dataset as the product of modes and dynamics. We plot the evolution of each mode to emphasize their similarity with the input functions and we plot the reconstructed data.



Other useful tools

- **Pylint**: a fully customizable source code analyzer.
- **CONDA**: cross-platform, Python-agnostic binary package manager
- **virtualenv**: A tool to create isolated Python environments
- **yapf** : Python code formatter



Important links

1. PEP8 Style Guide: <https://www.python.org/dev/peps/pep-0008/>
2. Awesome Python: <https://github.com/vinta/awesome-python>
3. Bug Solver*: <https://google.com>, <https://chatgpt.com/>