# 3. Deep learning tools

## 3.4 SciPy

Manel Martínez-Ramón
Meenu Ajith
Aswathy Rajendra Kurup

# Introduction

- Built on top of NumPy.
- Optimized to process multidimensional arrays.
- A tool to work with
  - images and signal processing
  - large data sets.
- This section contains some examples of the capabilities of the library.

# Data input and output

- SciPi is used to work with several data types as .mat, .wav and others, with specific commands for each data type.
- In this example we create, save and load a .mat file, that can be also read in MatLab.

```
1 import scipy.io as sio
2 import numpy as np
3
4 arr = np.array([1,2,3,4])  #create an array
5 #save the array by the name 'arr_samp'
6 sio.savemat('sample_data.mat', {'arr_samp': arr})
7 #load the array into the variable from mat file
8 sample_arr = sio.loadmat('sample_data.mat')
9
10 print(sample_arr['arr_samp'])
```

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

# Clustering methods

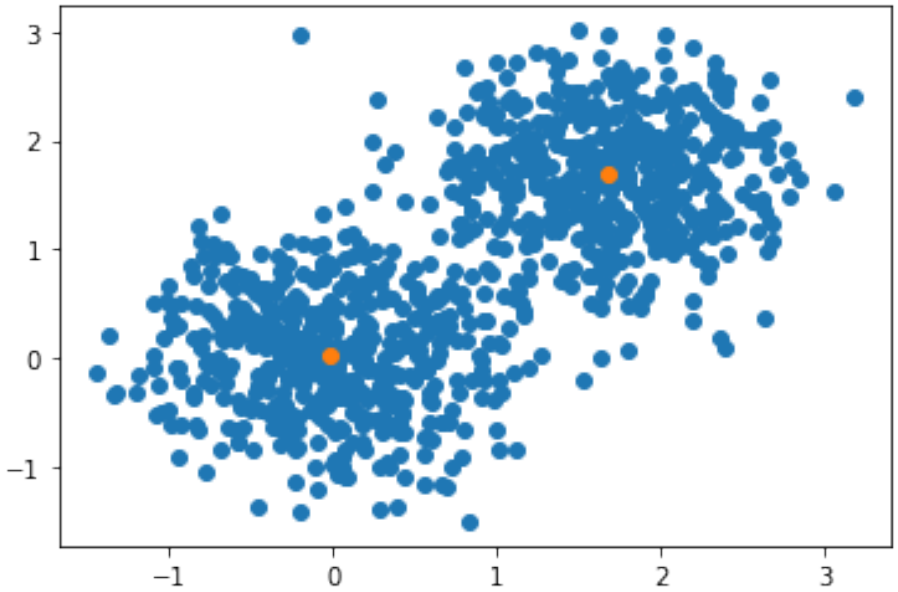
- Clustering is a procedure that is intended to split a set of data in different points with the purpose of obtaining an interpretation of the data structure.
- A popular rather limited clustering method is K-means, and SciPy has it implemented

```
1 from scipy.cluster.vq import kmeans, vq, whiten #importing
      the packages required for Kmeans clustering
2 import numpy as np
3 import matplotlib.pyplot as plt
4 data1 = np.random.randn(500,2) # Random 2D data
5 data2 = np.random.randn(500,2) + np.array([3,3])
6 data = np.vstack((data1,data2)) # stacking the two data
7 whit = whiten(data) # whiten the data
8 [center,_] = kmeans(whit, 2) # Apply Kmeans clustering
```

# Clustering methods

- This is the representation

```
1 out = vq(data, center) #assigns labels to data
2 plt.scatter(whit[:,0], whit[:,1]) #plot the scatterplots
3 plt.scatter(center[:,0], center[:,1]) # plot the centroids
4 plt.show()
```



# Constants

- *scipy.constants* gives access to many constants.
- They can be used in mathematical expressions.

```
1 import scipy.constants
2 from scipy.constants import find
3
4 print(find('light')) # find all constants with keyword 'light'
5 print(scipy.constants.physical_constants['speed of light in
      vacuum'])
6 print(scipy.constants.pi)
```

---

```
['speed of light in vacuum']
(299792458.0, 'm s^-1', 0.0)
3.141592653589793
```

# Linear algebra

- *scipy.linalg* performs linear algebra operations in Python. *F*aster than BLAS and LAPACK libraries.

```
1 from scipy import linalg
2 import numpy as np
3
4 A = np.array([[1,2],[3,2]]) # create a square matrix
5
6 print(linalg.det(A))     # compute the determinant of a
7 print(linalg.inv(A))        # compute the inverse of the matrix
```

```
-4.0
[[-0.5    0.5 ]
 [ 0.75 -0.25]]
```

# Linear algebra

```
1 val, vect = linalg.eig(A)  # compute the svd of a
2 print('\neigenvalues =')
3 print(val)
4 print('\neigenvectors =')
5 print(vect)
6 b = np.array([2,4])        #  2D vector
7 print(linalg.solve(a,b))   # Solution of equation Ax=b
```

```
eigenvalues =
[-1.+0.j  4.+0.j]

eigenvectors =
[[-0.70710678 -0.5547002 ]
 [ 0.70710678 -0.83205029]]

solution of equation Ax=b:
[1.  0.5]
```

# Numeric integrals

- The *integrate* package is used to compute numeric integrals with various methods.
- Example:

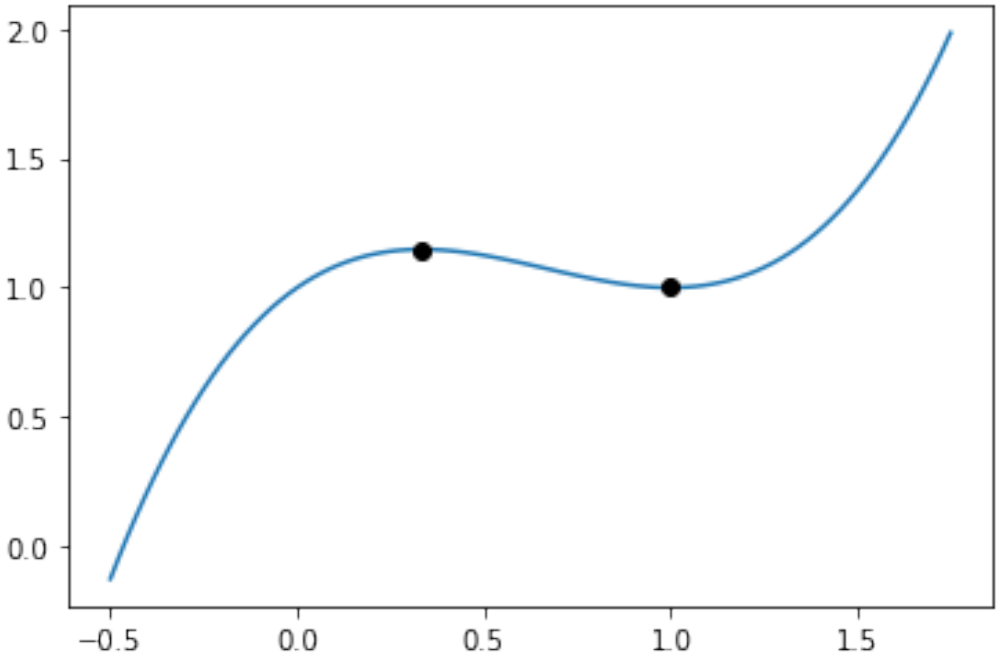
$$\int_{y=0}^{\frac{1}{2}} \int_{x=0}^{1-2y} xydxdy = \frac{1}{96} \approx 0.0104167$$

```
1 from scipy import integrate
2 def f(x, y):
3     return x*y
4 def bounds_y():
5     return [0, 0.5]
6 def bounds_x(y):
7     return [0, 1-2*y]
8
9 integrate.nquad(f, [bounds_x, bounds_y])
```

```
(0.010416666666666668, 4.101620128472366e-16)
```

# Optimization

- Minimizing or maximizing a function with constrained and unconstrained minimization problems.
- The package has the most common optimization approaches such as least squares (*least_squares()*) and curve fitting techniques (*curve_fit()*).



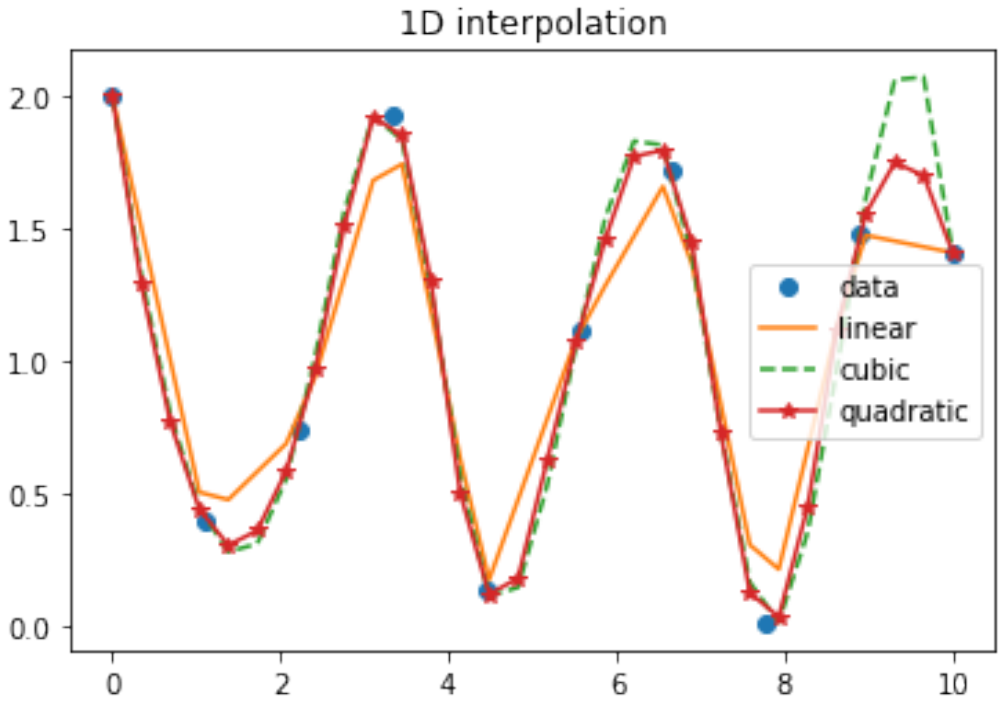

$$f(x) = x^3 - 2x^2 + x$$
$$f'(x) = 3x^2 - 4x + 1$$
$$x_0 = \frac{1}{3},\ 1$$

```python
from scipy import optimize
import numpy as np

def func(x):
        return  x**3 -  2*x**2 + x  + 1  # The function

optimize.minimize_scalar(func) # Find the minimum
```

```
     fun: 1.0
    nfev: 12
     nit: 8
 success: True
       x: 1.0
```

# Other stuff

- The package *interpolate* is also useful. It can interpolate using several methods.




- The package *scipy.ndimage* can does image processing operations:display, geometric transformations, filtering, edge detection...
- Library *MISC* is used as a sample to test operations.