Scientific Programming in Python

Francesco Mannella

Institute of Cognitive Sciences and technologies - CNR

Outline

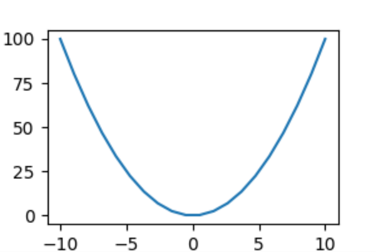
1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

NumPy and Matplotlib - hands-on code

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

x = np.linspace(-10, 10, 20)
y = x**2

plt.plot(x, y)
plt.show()
```




```
>>> print(x)
[-10.          -8.94736842  -7.89473684  -6.84210526  -5.78947368
  -4.73684211  -3.68421053  -2.63157895  -1.57894737  -0.52631579
   0.52631579   1.57894737   2.63157895   3.68421053   4.73684211
   5.78947368   6.84210526   7.89473684   8.94736842  10.        ]
>>>
>>> print(y)
array([100.        ,  80.05540166,  62.32686981,  46.81440443,
        33.51800554,  22.43767313,  13.5734072 ,   6.92520776,
         2.49307479,   0.27700831,   0.27700831,   2.49307479,
         6.92520776,  13.5734072 ,  22.43767313,  33.51800554,
        46.81440443,  62.32686981,  80.05540166, 100.        ])
```

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Arrays

Numpy arrays are *enhanced* lists, you can do arithmetic element-wise operations on them.

```
a = np.array([2.1, 3.4])
b = np.array([5., 7.])
c = 3.7

d = a + b*c    # d == [20.6, 29.3]
```

They can have many dimensions:

```
a = [[1, 3, 0],[4, 5, 2], [0, 2, 1],
    [4, 1, 4]]
# a[1][2] == 2

b = np.array(a)
# b[1, 2] == 2
```

You can ask arrays for enhanced math operations:

```
m = b.mean()
# m == 2.25

colMeans = b.mean(0) # each col is a
    group
# colMeans == array([2.25, 2.75, 1.75])

rowMeans =  b.mean(1) # each row is a
    group
# rowMeans ==  array([1.33333333,
    3.66666667, 1.        , 3.
     ])

sd = b.std()
# sd == 1.6393596310755
```

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Arrays: Hands-on code

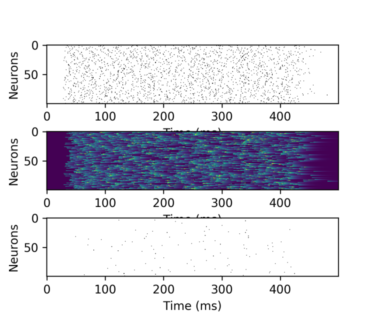
Building an Integrate-and-fire model of neuron

```python
from __future__ import print_function
from __future__ import division

import numpy as np
import matplotlib.pyplot as plt

# parameters
n_neurons = 100
dt = 1.0   # ms
tau = 10.0  # ms
v_r = 0.0   # mV
v_th = 15.0 # mV

sim_time = 500

# arrays
v = np.zeros([n_neurons, sim_time])
s = np.zeros([n_neurons, sim_time])
inps = np.zeros([n_neurons, sim_time])

# fills input array with spikes
for n in range(n_neurons):
    spiketimes = np.random.poisson(
      range(40, 440, 20))
    inps[n, spiketimes] = 80.0

# compute neurons' activations
for t in range(1, sim_time):
    v[:, t] = v[:, t - 1] + (dt / tau)
      * (- v[:, t - 1] + inps[:, t -1])
    ths = np.where(v[:, t] >= v_th)
    s[ths, t] = 1
    v[ths, t] = v_r
```

Arrays: Hands-on code

Building an Integrate-and-fire model of neuron

```python

plt.subplot(311)
plt.imshow(inps, cmap=plt.cm.binary)
plt.ylabel("Neurons")
plt.xlabel("Time (ms)")
plt.subplot(312)
plt.imshow(v)
plt.ylabel("Neurons")
plt.xlabel("Time (ms)")
plt.subplot(313)
plt.imshow(s, cmap=plt.cm.binary)
plt.ylabel("Neurons")
plt.xlabel("Time (ms)")
plt.show()
```



Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Creation functions

Creating arrays with ones:

```
a = np.ones(3)
b = np.ones([5, 2])
```

Creating arrays with zeros:

```
a = np.zeros(3)
b = np.zeros([5, 2, 4])
```

Creating arrays with a fixed value:

```
a = np.ones([3, 4])*5.0
b = np.full([3, 4], 5.0)
```

Sequencies:

```
a = np.arange(10)   # works as range but builds numpy arrays
a = np.linspace(-10, 10, 9)
# a == np.array([-10.0, -7.5,
#                 -5.0, -2.5,
#                  0.0,  2.5,
#                  5.0 , 7.5,
#                 10.0 ])
```

Load from file:

```
a = np.loadtxt('data.txt')
```

where data.txt contains:

```
2.27 0.55
0.70 2.08
5.93 5.19
3.36 3.43
```

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Arithmetic operators

All arithmetic operators can be used. They act in an elementwise manner.

```
a = np.array([[3, 4, 5],[2, 3, 1]])
b = 4.0
c = np.array([ [2, 1, 2], [3, 2, 4]])

d = a + b * c + a * b
```

```
>>> print(d)
[[23. 24. 33.]
 [22. 23. 21.]]
```

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. **Manipulation functions**
7. Slices

Manipulation functions

Concatenating arrays:

```
a = np.linspace(0, 1, 5)
b = np.ones(3) * 0.5
c = np.hstack([a, b])
# c == np.array([0.0, 0.25, 0.5, 0.75,
      1.0, 0.5, 0.5, 0.5])
```

Concatenating arrays by row:

```
a = np.ones(3) * 2
b = np.ones(3) * 7
c = np.vstack([a, b])
# c == np.array([[2, 2, 2],
#                [7, 7, 7]])
```

Creating a 2D grid from two 1D Arrays:

```

x = np.arange(5)

# creates two grids
X, Y = np.meshgrid(x, x)
```

```
>>> print(X)
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
>>> print(y)
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]]
```

Outline

1. NumPy and Matplotlib
2. Arrays
3. Arrays: Hands-on code
4. Creation functions
5. Arithmetic operators
6. Manipulation functions
7. Slices

Slices

Slices are almost similar to those of lists:

```
a = np.array([[3, 4, 5],
              [2, 3, 1],
              [2, 2, 6],
              [4, 1, 2]])
```

```
>>> print(a[:, 1])
[4 3 2 1]
>>> print(a[:1, :])
[[3 4]
 [2 3]
 [2 2]
 [4 1]]
```

Operations on slices modify the original array:

```
a[:1, 0] = -99
```

```
>>> print(a)
[4 3 2 1]
>>> print(a[:1, :])
[[3 -9 5]
 [2 -9 1]
 [2 -9 6]
 [4 -9 2]]
```

```
a[a > 5] = 0
```

```
print(a[:1, :])
[[0 0 5]
 [0 0 0]
 [0 0 6]
 [0 0 0]]
```