Scientific Programming Practical 9

Introduction

Luca Bianco - Academic Year 2017-18
luca.bianco@fmach.it

Numpy

Numpy is a fundamental library for **high performance scientific computations**. It provides fast and memory efficient data structures like `ndarray` with broadcasting capabilities, standard mathematical functions that can be applied on the arrays avoiding loops, linear algebra functions, I/O methods and it is well integrated with programming languages like C.

```
import numpy as np
```

ndarray

Numpy ndarray is an N-dimensional array object designed to contain homogeneous data (i.e. all data must have the same type)

They have two information:

a **shape**

and

a **dtype**

```
import numpy as np

A = np.array([[1,2.1,3], [4, 4.3, 5.1]])

print(A)
print(type(A))
print(A.shape)
print(A.dtype)
```

```
[[ 1.   2.1  3. ]
 [ 4.   4.3  5.1]]
<class 'numpy.ndarray'>
(2, 3)
float64
```

ndarray

Numpy ndarray is an N-dimensional array object designed to contain homogeneous data (i.e. all data must have the same type)

```python
import numpy as np
import math


mysqrt = [math.sqrt(x) for x in range(0,5)]
mycrt = [x**(1/3) for x in range(0,5)]
print("Original lists:")
print(mysqrt)
print(mycrt)
print("")
npData = np.array(mysqrt)
print("Numpy ndarray:")
print(npData)
print("")
print("The shape:", npData.shape)
print("The dimensionality:", npData.ndim)
print("The type:", npData.dtype)
print("")
twoDarray = np.array([mysqrt, mycrt])
print("The 2D array:")
print(twoDarray)
print("")
print("The shape:", twoDarray.shape)
print("The dimensionality:", twoDarray.ndim)
print("The type:", twoDarray.dtype)
```

```
Original lists:
[0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0]
[0.0, 1.0, 1.2599210498948732, 1.4422495703074083, 1.5874010519681994]

Numpy ndarray:
[ 0.          1.          1.41421356  1.73205081  2.        ]

The shape: (5,)
The dimensionality: 1
The type: float64

The 2D array:
[[ 0.          1.          1.41421356  1.73205081  2.        ]
 [ 0.          1.          1.25992105  1.44224957  1.58740105]]

The shape: (2, 5)
The dimensionality: 2
The type: float64
```

ndarray

Zeros, ones and diagonals...

1. Array: `np.zeros(N)` or matrix: `np.zeros((N,M))`
2. Array: `np.ones(N)` or matrix: `np.ones((N,M))`
3. Matrix: `np.eye(N)`

```
Zero array (1x3)
[ 0.  0.  0.]

Zero matrix (4x3)
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]

Ones array (1x3)
[ 1.  1.  1.]

Ones matrix (3x2)
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]

Diagonal matrix
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]

Range 0-4
[0 1 2 3 4]
A diagonal matrix:
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 0]
 [0 0 0 0 4]]
It's shape:
(5, 5)
```

```python
import numpy as np

zeros = np.zeros(3)
zMat = np.zeros((4,3))
ones = np.ones(3)
oMat = np.ones((3,2))
diag = np.eye(4)
rng = np.arange(5) #5 excluded!

print("Zero array (1x3)")
print(zeros)
print("")
print("Zero matrix (4x3)")
print(zMat)
print("")
print("Ones array (1x3)")
print(ones)
print("")
print("Ones matrix (3x2)")
print(oMat)
print("")
print("Diagonal matrix")
print(diag)
print("")
print("Range 0-4")
print(rng)
print("A diagonal matrix:")
dm = np.diag(rng)
print(dm)
print("It's shape:")
print(dm.shape)
```

ndarray

Zeros, ones and diagonals...

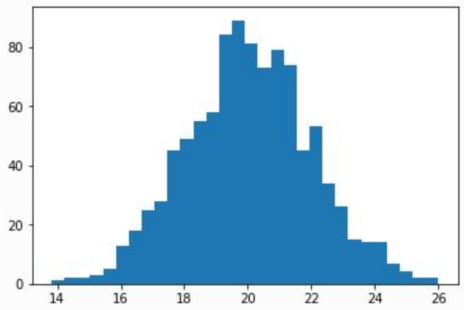
Numpy has its own **range** method that is called `np.arange(N)`. Evenly spaced values in a range can be obtained also with `np.linspace(S,E, num=N, endpoint=True/False)` to obtain N linearly spaced values from S to E (included, unless endpoint = False is specified).

ndarray

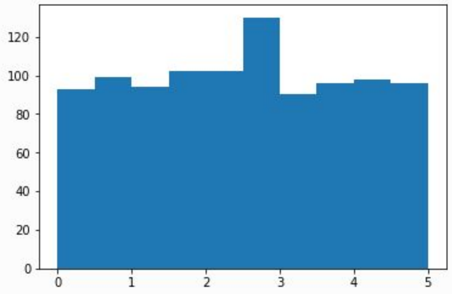
Create a random array of 1000 values drawn from: 1. a gaussian distribution with $\sigma = 20$ and $\mu = 2$ 2. a uniform distribution from 0 to 5 3. a binomial distribution with $p = 0.5$ and $n = 12$

Random values

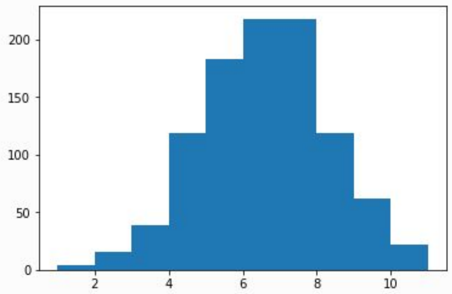
Gaussian(20,2):



Uniform(0,5):



Binomial(12,0.5):




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

#get the gaussian random array
g = np.random.normal(20,2, 1000)
#get the uniform random array with values in [0,5]
u = np.random.uniform(0,5, 1000)
#get the binomial random array
b = np.random.binomial(12,0.5, 1000)

print("Gaussian(20,2):")
plt.hist(g, bins = 30)
plt.show()
#print("")
print("Uniform(0,5):")
plt.close()
plt.hist(u, bins = 10)
plt.show()
print("")
print("Binomial(12,0.5):")
plt.close()
plt.hist(b, bins = 10)
plt.show()
```

random seed

Random values

```
import numpy as np

u = np.random.uniform(0,1,3)
u1 = np.random.uniform(0,1,3)
u2 = np.random.uniform(0,1,3)

print(" u: {}\n u1:{}\n u2:{}".format(u,u1,u2))

print("")
print("With random seed reinit.")
np.random.seed(0)
u = np.random.uniform(0,1,3)
np.random.seed(0)
u1 = np.random.uniform(0,1,3)
np.random.seed(0)
u2 = np.random.uniform(0,1,3)

print(" u: {}\n u1:{}\n u2:{}".format(u,u1,u2))
```

```
 u: [ 0.1855235   0.98513018  0.91982263]
 u1:[ 0.91378198  0.44407375  0.78036964]
 u2:[ 0.64288027  0.32220978  0.42408156]

With random seed reinit.
 u: [ 0.5488135   0.71518937  0.60276338]
 u1:[ 0.5488135   0.71518937  0.60276338]
 u2:[ 0.5488135   0.71518937  0.60276338]
```

Numpy ← → Pandas

```
{'two': 2, 'three': 3, 'one': 1, 'four': 4}

<class 'pandas.core.series.Series'>

<class 'numpy.ndarray'>

Numpy matrix
[[0 0 0 0 0 0]
 [0 1 0 0 0 0]
 [0 0 2 0 0 0]
 [0 0 0 3 0 0]
 [0 0 0 0 4 0]
 [0 0 0 0 0 5]]

Pandas DataFrame
   0  1  2  3  4  5
0  0  0  0  0  0  0
1  0  1  0  0  0  0
2  0  0  2  0  0  0
3  0  0  0  3  0  0
4  0  0  0  0  4  0
5  0  0  0  0  0  5

Reindexed DataFrame
   a1  b2  c3  d4  e5  f6
A   0   0   0   0   0   0
B   0   1   0   0   0   0
C   0   0   2   0   0   0
D   0   0   0   3   0   0
E   0   0   0   0   4   0
F   0   0   0   0   0   5
```

```python
import pandas as pd
import numpy as np

myDict = {"one" : 1, "two" : 2, "three" : 3, "four" : 4}

mySeries = pd.Series(myDict)

print(myDict)
print("")
print(type(mySeries))
print("")
print(type(mySeries.values))
print("")

myMat = np.diag(np.arange(6))
myDF = pd.DataFrame(myMat)
print("Numpy matrix")
print(myMat)
print("")
print("Pandas DataFrame")
print(myDF)
print("")
print("Reindexed DataFrame")
myDF = pd.DataFrame(myMat, index = list("ABCDEF"),
                    columns = ['a1','b2','c3','d4','e5','f6'])
print(myDF)
```

Reshaping

ndarrays can be reshaped...

```
import numpy as np

myA = np.arange(10)

myB = myA.reshape(2,7)
print(myB)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-2936ce504f51> in <module>()
      3 myA = np.arange(10)
      4
----> 5 myB = myA.reshape(2,7)
      6 print(myB)

ValueError: cannot reshape array of size 10 into shape (2,7)
```

```
import numpy as np

myA = np.arange(10)

print("The array:")
print(myA)
print("")
myB = myA.reshape((2,5))
print("Reshaped:")
print(myB)

myC = myB.ravel()
print("")
print("Back to array:")
print(myC)
```

```
The array:
[0 1 2 3 4 5 6 7 8 9]

Reshaped:
[[0 1 2 3 4]
 [5 6 7 8 9]]

Back to array:
[0 1 2 3 4 5 6 7 8 9]
```

Looping through arrays

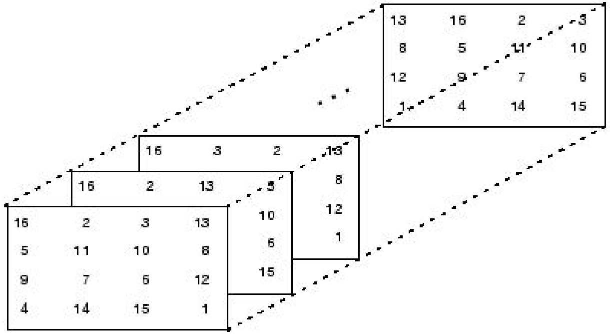
`np.ndarray.flat`

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]

Matrix:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Looping through elements:
Element:  0
Element:  1
Element:  2
Element:  3
Element:  4
Element:  5
Element:  6
Element:  7
Element:  8
Element:  9
Element:  10
Element:  11
Looping row by row:
Row:   [0 1 2]
	el:  0
	el:  1
	el:  2
Row:   [3 4 5]
	el:  3
	el:  4
	el:  5
Row:   [6 7 8]
	el:  6
	el:  7
	el:  8
Row:   [ 9 10 11]
	el:  9
	el:  10
	el:  11
```

```python
import numpy as np

myA = np.arange(12)
print(myA)
print("")
print("Matrix:")
myA = myA.reshape((4,3))
print(myA)

print("Looping through elements:")
for el in myA.flat:
    print("Element:", el)

print("Looping row by row:")
for el in myA:
    print("Row: ", el)
    for j in el:
        print("\tel:", j)
```

N-Dimensions

Note that `np.ndarray[0,:,:]` is the whole first matrix. `np.ndarray[:,0,:]` is all the first rows, while `np.ndarray[:,:,0]` is all the first columns. Regarding slicing and indexing, the same reasoning applies to n-dimensional matrices. For example, `myB` below is a 3x3x3 matrix.




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

myA[2,2] =  8
myA[1,3] =  7
myA[0,3] =  3
second row: [5 2 4 7]

3D matrix
 - shape: (3, 3, 3)
[[[6 7 7]
  [8 1 5]
  [9 8 9]]

 [[4 3 0]
  [3 5 0]
  [2 3 8]]

 [[1 3 3]
  [3 7 0]
  [1 9 9]]]

myB[0,2,2] =  9
Second matrix:
[[4 3 0]
 [3 5 0]
 [2 3 8]]
Third row of second matrix:
[2 3 8]
Second column of second matrix:
[3 5 3]
```

```python
import numpy as np


myA = np.random.randint(0,10, size = (3,4))
print(myA)
print("")
print("myA[2,2] = ", myA[2,2])
print("myA[1,3] = ", myA[1,3])
print("myA[0,3] = ", myA[0,3])
print("second row:", myA[1,:])
print("")
print("3D matrix ",)
myB = np.random.randint(0,10, size = (3,3,3))
print(" - shape:", myB.shape)
print(myB)
print("")
print("myB[0,2,2] = ", myB[0,2,2])
print("Second matrix:")
print(myB[1,:,:])
print("Third row of second matrix:")
print(myB[1,2,:])
print("Second column of second matrix:")
print(myB[1,:,1])
```

Operator broadcasting

```
import numpy as np

A = np.random.randint(0,10, size = (3,2))
B = np.random.randint(0,10, size = (3,3,3))
C = np.random.randint(0,10, size = (3,1))
print("Matrix A 3x2")
print(A)
print("")
print("Matrix B 3x3x3")
print(B)
print("")
print("Matrix C 3x1")
print(C)
print("")
print("A squared")
print(A**2)
print("")
print("A square-rooted")
print(np.sqrt(A))

print("")
print("B square-rooted")
print(np.sqrt(B))

print("A + C ")
print(A + C)
print("")

print("B + C ")
print(B  + C)

print("")
print("B")
print(B)
print("Sub array B - 20")
B[:, 0:2 , 0:2 ] -= 20
print(B)
```

```
Matrix A 3x2
[[0 4]
 [7 3]
 [2 7]]

Matrix B 3x3x3
[[[2 0 0]
  [4 5 5]
  [6 8 4]]

 [[1 4 9]
  [8 1 1]
  [7 9 9]]

 [[3 6 7]
  [2 0 3]
  [5 9 4]]]

Matrix C 3x1
[[4]
 [6]
 [4]]

A squared
[[ 0 16]
 [49  9]
 [ 4 49]]

A square-rooted
[[ 0.          2.        ]
 [ 2.64575131  1.73205081]
 [ 1.41421356  2.64575131]]

B square-rooted
[[[ 1.41421356  0.          0.        ]
  [ 2.          2.23606798  2.23606798]
  [ 2.44948974  2.82842712  2.        ]]

 [[ 1.          2.          3.        ]
  [ 2.82842712  1.          1.        ]
  [ 2.64575131  3.          3.        ]]

 [[ 1.73205081  2.44948974  2.64575131]
  [ 1.41421356  0.          1.73205081]
  [ 2.23606798  3.          2.        ]]]
A + C
[[ 4  8]
 [13  9]
 [ 6 11]]
```

```
B + C
[[[ 6  4  4]
  [10 11 11]
  [10 12  8]]

 [[ 5  8 13]
  [14  7  7]
  [11 13 13]]

 [[ 7 10 11]
  [ 8  6  9]
  [ 9 13  8]]]

B
[[[2 0 0]
  [4 5 5]
  [6 8 4]]

 [[1 4 9]
  [8 1 1]
  [7 9 9]]

 [[3 6 7]
  [2 0 3]
  [5 9 4]]]
Sub array B - 20
[[[-18 -20   0]
  [-16 -15   5]
  [  6   8   4]]

 [[-19 -16   9]
  [-12 -19   1]
  [  7   9   9]]

 [[-17 -14   7]
  [-18 -20   3]
  [  5   9   4]]]
```

Linear algebra

```
import numpy as np
from numpy import linalg

A = np.random.randint(0,10, size = (4,4))
print("Matrix A:")
print(A)
print("")
print("inv(A)")
A_1 = linalg.inv(A)
print(A_1)
print("")
print(np.dot(A,A_1))
```

```
Matrix A:
[[3 7 5 5]
 [0 1 5 9]
 [3 0 5 0]
 [1 2 4 2]]

inv(A)
[[ 0.23121387  0.05780347  0.38150289 -0.83815029]
 [ 0.09537572 -0.10115607 -0.16763006  0.21676301]
 [-0.13872832 -0.03468208 -0.02890173  0.50289017]
 [ 0.06647399  0.1416185   0.03468208 -0.30346821]]

[[  1.00000000e+00   0.00000000e+00   2.77555756e-17   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00]
 [ -2.77555756e-17   0.00000000e+00   6.93889390e-17   1.00000000e+00]]
```

```
import numpy as np

a = np.array([1, 2, 3, 1, 2, 3,1, 1 ,1])
A = a.reshape((3,3))
B = np.random.randint(0,10, size = (3,2))

print("A (3x3)")
print(A)
print("")
print("B (3x2)")
print(B)
print("")
print("AxB (3x2)")
print(A.dot(B))
print("")
print("A transposed:")
print(A.T)
```

```
A (3x3)
[[1 2 3]
 [1 2 3]
 [1 1 1]]

B (3x2)
[[4 3]
 [4 4]
 [8 4]]

AxB (3x2)
[[36 23]
 [36 23]
 [16 11]]

A transposed:
[[1 1 1]
 [2 2 1]
 [3 3 1]]
```

https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.13 Manual   index | next | previous

NumPy Reference

Release: 1.13
Date: June 10, 2017

This reference manual details functions, modules, and objects included in NumPy, describing what they are and what they do. For learning how to use NumPy, see also NumPy User Guide.

- Array objects
  - The N-dimensional array (`ndarray`)
  - Scalars
  - Data type objects (`dtype`)
  - Indexing
  - Iterating Over Arrays
  - Standard array subclasses
  - Masked arrays
  - The Array Interface
  - Datetimes and Timedeltas
- Universal functions (`ufunc`)
  - Broadcasting
  - Output type determination
  - Use of internal buffers
  - Error handling
  - Casting Rules
  - Overriding Ufunc behavior
  - `ufunc`
  - Available ufuncs
- Routines
  - Array creation routines
  - Array manipulation routines

Table Of Contents

- NumPy Reference
  - Acknowledgements

Previous topic

Beyond the Basics

Next topic

Array objects

http://sciprolab1.readthedocs.io/en/latest/practical9.html

Exercises

1. Write a function that converts a numpy ndarray of temperatures expressed in Degrees Celsius into Degrees Farenheit. The formula to convert a temperature C in Celsius into F in Farenheit is the following:

   $$F = C * 9/5 + 32$$

   Write then a function that converts a numpy ndarray of temperatures in Farenheit into Celsius.

   Finally:

   1. apply the Celsius to Farenheit conversion on an ndarray containing the following October's minimum and maximum temperatures in Trento: tmin = [12, 11, 11, 8, 9, 10, 3, 8, 4, 5, 10, 9, 8, 9, 8, 7, 6, 4, 5, 6, 9, 9, 3, 3, 5] and tmax =[15, 22, 18, 20, 22, 22, 20, 21, 21, 21, 21, 23, 24, 24, 24, 25, 22, 22, 20, 20, 19, 15, 20, 23, 19];
   2. check that both functions work correctly by converting the values from Celsius to Farenheit and back to Celsius;
   3. plot the minimum and maximum temperatures in celsius on the same graph. Since the temperatures refer to the first 25 days of the month of October, the x coordinate can be a range(1,26);

Show/Hide Solution

2. Solve the following system of linear equations:

$$\begin{cases} 3x - 4y + 2z - 5w = -9 \\ -4x + 4y + 10z + w = 32 \\ -x + 2y + -7z - w = -7 \\ x + y + z + w = 1 \end{cases}$$