

Engineering programming

Numpy

Thibault Thétier - thibault.thetier@vub.be

Tom Godden - tgodden@vub.be

1ste semester 2021

- ▶ Numerical Python
- ▶ One of the most important packages for numerical computing in Python
- ▶ ndarray: efficient multidimensional array
- ▶ Fast operations on entire arrays of data without loops
- ▶ Linear algebra, random number generation, and Fourier transform capabilities

NUMPY ARRAY

- ▶ N-dimensions
- ▶ All elements of the same type
- ▶ Shape of a tuple
- ▶ Can be used for mathematical operations, Fourier transformations, video and image processing, machine learning, ...
- ▶ Designed for vector operations: whole list editing instead of item by item

PERFORMANCE

- ▶ Consider a NumPy array of one million integers, and the equivalent Python list:

```
1 my_arr = np.arange(1000000)
2 my_list = list(range(1000000))
```

- ▶ Now let's multiply each sequence by 2:

```
1 %time for _ in range(10): my_arr2 = my_arr * 2
2 CPU times: user 20 ms, sys: 50 ms, total: 70 ms
3 Wall time: 72.4 ms
4 %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
5 CPU times: user 760 ms, sys: 290 ms, total: 1.05 s
6 Wall time: 1.05 s
```

- ▶ NumPy-based algorithms are generally 10 to 100 times faster (or more) than their pure Python counterparts and use significantly less memory.

LIBRARIES

There are libraries that are a standard part of Python: math, random, datetime, zlib,

...

You can installing other libraries with package manager

Import in different ways:

```
1 import math
2 math.sin(math.pi)
```

```
1 from math import *
2 sin(pi)
```

```
1 from math import sin, pi
2 sin(pi)
```

The large and growing Python users community provides an increasing number of libraries that already do what you need.

IMPORT CONVENTIONS

The Python community has adopted a number of naming conventions for commonly used modules:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import seaborn as sns
5 import statsmodels as s
```

So you can have for example:

```
1 import math as m
2 m.sin(m.pi)
```

It is considered bad practice to import everything (from math import *) from a large package like NumPy.

CREATE NDARRAYS

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

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

Define data type:

```
a = np.array([[1, 2], [3, 4]], dtype=complex)
```

Empty array:

```
a = np.empty([3,2])
```

- ▶ `np.zeros((3,2))`
- ▶ `np.ones((3,2))`
- ▶ `np.eye(3)`
- ▶ `np.full((3,2),4)`
- ▶ ...

ARITHMETIC

A number of tools exist to manipulate NumPy arrays. The standard operations are supported:

- ▶ `arr = np.array([[1., 2., 3.],
[4., 5., 6.]])`
- ▶ `arr * arr` Or `np.multiply(arr, arr)`
- ▶ `arr - arr` Or `np.subtract(arr, arr)`
- ▶ `1 / arr`
- ▶ `arr ** 0.5`
- ▶ `arr2 = np.array([[0., 4.,
1.], [7., 2., 12.]])`
- ▶ `arr2 > arr`

BASIC INDEXING AND SLICING

- ▶ `a = np.arange(10)`
- ▶ `a[5:8]`
- ▶ `a[5:8] = 12`
- ▶ `a[:] = 64`
- ▶ Data is not copied!
 - ▶ `a_slice = a[5:8]`
 - ▶ `a_slice[1] = 12345`
 - ▶ `a`

- ▶ If you want to copy:
 - ▶ `a[5:8].copy()`
- ▶ `a2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])`
- ▶ `a2d[0][2]`
- ▶ `a2d[0,2]`
- ▶ `a2d[:2, 1:]`
- ▶ `a2d[1, :2]`

FANCY INDEXING

Conditions can be used for indexing:

- ▶ `x = np.random.randint(0, 10, (5, 5))`
- ▶ `x[x > 5]`

You can pass a list or ndarray of integers:

- ▶ `a = np.random.rand(8, 4)`

- ▶ `a[[4, 3, 0, 6]]`

Passing multiple index arrays does something else:

- ▶ `a = np.arange(32).reshape((8, 4))`
- ▶ `a[[1, 5, 7, 2], [0, 3, 1, 2]]`

Here the elements (1, 0), (5, 3), (7, 1), and (2, 2) were selected

Fancy indexing, unlike slicing, always copies the data into a new array.

LINEAR ALGEBRA

Cross product between vectors:

- ▶ `np.cross(a,b)`

Transpose multidimensional arrays:

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

Matrix multiplication:

- ▶ `a.dot(a)`

- ▶ `np.dot(a,a)`

- ▶ `a @ np.ones(3)`

Set of matrix functions:

- ▶ `from numpy.linalg import inv,
det`
- ▶ `mat = a.T.dot(a)`
- ▶ `det(a)`
- ▶ `inv(a)`