Numpy Tutorial

Nicolas Pécheux
Alexandre Allauzen

LIMSI/CNRS — Université Paris Sud

January 11, 2017

Why Numpy ?

- Numpy is a Python module for scientific computing
- Written in C
- Fast vector & matrix operations

Import module using an alias

```
1 import numpy as np
2 import numpy.random as npr
```

Arrays

- In numpy we work with arrays

```
>>> lst = [3, 2, 4, 1]
>>> a = np.array(lst) # convert list to numpy array
>>> a
array([3, 2, 4, 1])
```

- All element have same type

```
>>> a.dtype.name
'int64'
```

Vectors

- Vectors are just 1d arrays

```
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])

>>> np.ones(4)
array([ 1.,  1.,  1.,  1.])

>>> npr.randn(2)  # normal distribution
array([ 0.26141913, -0.5324435 ])

>>> np.linspace(0, 1, 5)  # 5 uniform values in [0, 1]
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

Matrices

- Matrices are just 2d arrays

```
>>> np.zeros((2,5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])

>>> npr.randn(3, 3)
array([[-0.64043229,  0.68655366,  1.09269517],
       [ 1.23037221, -0.79673819,  0.01486236],
       [-1.15268157, -1.36926975, -0.64407652]])
```

Array shape

- Array shape can be read and modified

```
>>> z = np.zeros(6)

>>> z.shape
(6,)

>>> z.reshape(3, 2)
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])

>>> np.arange(9).reshape(3, 3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Mathematical operations

- Arithmetic operators apply *elementwise*

```
>>> a = np.arange(6).reshape(2, 3)
>>> b = np.ones((2, 3))

>>> a + b
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])

>>> a * b
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])

>>> a < 3
array([[ True,  True,  True],
       [False, False, False]], dtype=bool)
```

Mathematical operations

- Very usefull !

```
>>> x = np.arange(4).reshape(2, 2)

>>> np.sin(x) * np.exp(-x ** 2 / 2)
array([[ 0.        ,  0.30955988],
       [ 0.12306002,  0.00095086]])
```

Methods

- By default apply to the array as though it were a list of number

```
>>> a = np.array([[0, 9, 2],
                  [3, 4, 1]])

>>> a.min(), a.argmax(), a.sum()
(0, 1, 19)
```

- But one can apply along a specified axis only

```
>>> a.sum(axis=1)
array([11,  8])

>>> a.sort(axis=0)  # Modify a !
>>> a
array([[0, 4, 1],
       [3, 9, 2]])
```

Matrix operations

```
>>> A = np.arange(6).reshape(2, 3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> x = 2 * np.ones(3)

>>> A.dot(x)  # Ax
array([  6.,  24.])

>>> A.dot(A.T)  # AA'
array([[ 5, 14],
       [14, 50]])

>>> # Warning !
>>> A * x  # Broadcast + elementwise
array([[  0.,   2.,   4.],
       [  6.,   8.,  10.]])
```

Indexing and slicing

```
>>> A = array([[0, 1, 2],
               [3, 4, 5]])

>>> A[1, 2]
5

>>> A[1, :]  # : means all elements of that axis
array([3, 4, 5])

>>> A[:, [0, 2]]
array([[0, 2],
       [3, 5]])

>>> mask = A.min(axis=0) > 1  # Very useful !
>>> A[:, mask]
array([[2],
       [5]])
```

Statistics

```
>>> A = array([[0, 1, 2],
               [3, 4, 5]])

>>> A.mean()
2.5

>>> A.std(axis=0)
array([ 1.5,  1.5,  1.5])

>>> np.median(A, axis=1)
array([ 1.,  4.])
```

Random

```
>>> import numpy.random as npr

>>> npr.seed(42)  # Seed the random generators

>>> npr.rand(3, 2)  # random in [0, 1)
  array([[ 0.37454012,  0.95071431],
         [ 0.73199394,  0.59865848],
         [ 0.15601864,  0.15599452]])

>>> letters = np.array(['a', 'b', 'c', 'd'])
>>> npr.shuffle(letters)
>>> letters
array(['a', 'b', 'd', 'c'], dtype='|S1')

>>> npr.choice(letters, 2, replace=True) # numpy 1.7
array(['a', 'c'], dtype='|S1')
```