Introduction to High Performance Scientific Computing

Autumn, 2017

Python lecture 3

Imperial College
London

Numpy: scientific computing with Python

Numpy overview

- ***Numpy* is an add-on package which introduces arrays into Python**

- **It has to be imported into codes, usually:**

```
import numpy as np
```

Numpy overview

- *Numpy* is an add-on package which introduces arrays into Python

- It has to be imported into codes, usually:

```
import numpy as np
```

- *Numpy* provides Matlab-like linear algebra capabilities

- With *matplotlib*, *numpy* provides Matlab-like visualization

- With *scipy*, *numpy* provides tools for differential equations, optimization, and much more…

Building arrays

Three methods:

1. **Make a list:**

```
In [112]: x = np.array([3., 5, 7, 9, 11, 13])

In [113]: print(x)
[  3.   5.   7.   9.  11.  13.]

In [114]: type(x)
Out[114]: numpy.ndarray
```

Building arrays

Three methods:

1. **Make a list:**

```
In [112]: x = np.array([3., 5, 7, 9, 11, 13])

In [113]: print(x)
[  3.   5.   7.   9.  11.  13.]

In [114]: type(x)
Out[114]: numpy.ndarray
```

2. **Use** np.linspace(start, stop, N) **-- same as matlab:**

```
In [115]: y=np.linspace(3,13,6)

In [116]: print(y)
[  3.   5.   7.   9.  11.  13.]
```

Building arrays

Three methods:

1. **Make a list:**

```
In [112]: x = np.array([3., 5, 7, 9, 11, 13])

In [113]: print(x)
[  3.   5.   7.   9.  11.  13.]

In [114]: type(x)
Out[114]: numpy.ndarray
```

2. **Use** np.linspace(start, stop, N) **-- same as matlab:**

```
In [115]: y=np.linspace(3,13,6)

In [116]: print(y)
[  3.   5.   7.   9.  11.  13.]
```

3. **Use** np.arange(start, stop, step) **– same as range, generates array instead of a list; does not include “stop” value:**

```
In [121]: z=np.arange(3,14,2)

In [122]: print(z)
[ 3  5  7  9 11 13]
```

Math with arrays

- Math works basically as you would expect. A few examples:

```
In [128]: print(x)
[  3.   5.   7.   9.  11.  13.]

In [129]: y=x+3

In [130]: print(y)
[  6.   8.  10.  12.  14.  16.]

In [131]: print(x*3)
[  9.  15.  21.  27.  33.  39.]

In [132]: print(x**2)
[   9.   25.   49.   81.  121.  169.]

In [133]: print(sin(x))
[ 0.14112001 -0.95892427  0.6569866   0.41211849 -0.99999021  0.42016704]
```

(What would happen if x **was a list?)**

Math with arrays

- **But be careful when making a copy:**

```
print(x)
[ 3  5  7  9 11 13]

In [91]: y=x

In [92]: id(x),id(y)
Out[92]: (4504682688, 4504682688)
```

x **and** y **occupy the same location in memory so changing** x **can also change** y**!:**

```
In [109]: x[3]=100

In [110]: print(x)
[  3   5   7 100  11  13]

In [111]: print(y)
[  3   5   7 100  11  13]
```

Instead, use `y=x.copy()`

Analyzing arrays

- **Use <tab> completion to see your options:** x.<tab>
- **A few examples:**

```
In [19]: print(x)
[  3.   5.   7.   9.  11.  13.]

In [20]: x.sum()
Out[20]: 48.0

In [21]: sum(x)
Out[21]: 48.0

In [22]: max(x)
13.0

In [23]: mean(x)
8.0

In [24]: x.var()
11.6666666667

In [25]: cumsum(x)
[  3.   8.  15.  24.  35.  48.]
```

Analyzing arrays

- **Use conditional statements easily:**

```
In [21]: x=linspace(-1,1,11)

In [22]: x
[-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8
1. ]

In [23]: x[x<=0]
[-1.  -0.8 -0.6 -0.4 -0.2  0. ]
```

Analyzing arrays

- **Use conditional statements easily:**

```
In [21]: x=linspace(-1,1,11)

In [22]: x
[-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8
1. ]

In [23]: x[x<=0]
[-1.  -0.8 -0.6 -0.4 -0.2  0. ]
```

- **Example: generate step function**

```
In [24]: y=x.copy()

In [25]: y[x>0]=1.0

In [26]: y[x<=0]=0.0

In [27]: print(y)
[ 0.  0.  0.  0.  0.  0.  1.  1.  1.  1.  1.]
```

Building matrices

First approach -- array of arrays:

```
In [45]: row1 = np.array([1, 3, 5])

In [46]: row2 = np.array([2, 4, 6])

In [47]: row3 = np.array([3, 5, 7])

In [48]: M = np.array([row1,row2,row3])
```

Building matrices

First approach -- array of arrays:

```
In [45]: row1 = np.array([1, 3, 5])

In [46]: row2 = np.array([2, 4, 6])

In [47]: row3 = np.array([3, 5, 7])

In [48]: M = np.array([row1,row2,row3])
```

```
In [57]: print M
[[1 3 5]
 [2 4 6]
 [3 5 7]]

In [58]: shape(M)
Out[58]: (3, 3)
```

Building matrices

Second approach: *meshgrid*

```
In [90]: x=np.linspace(0,1,11)

In [91]: y=np.linspace(0,2,6)

In [92]: [xg,yg]=np.meshgrid(x,y)

In [93]: print xg[0:3,0:3]
[[ 0.   0.1  0.2]
 [ 0.   0.1  0.2]
 [ 0.   0.1  0.2]]

In [94]: print yg[0:3,0:3]
[[ 0.   0.   0. ]
 [ 0.4  0.4  0.4]
 [ 0.8  0.8  0.8]]
```

See also *mgrid and ogrid*

Building matrices

A few more useful commands are *zeros*, *ones*, and *eye*:

```
In [108]: np.zeros((3,2))
Out[108]:
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])

In [109]: np.ones((3,2))
Out[109]:
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])

In [110]: np.eye(3)
Out[110]:
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Linear algebra

Use *dot* for dot products and matrix multiplications.

- **dot product:**

```
x
Out[116]: array([ 0.,  1.,  2.,  3.,  4.])

In [117]: y=x*2

In [118]: print(y)
[ 0.  2.  4.  6.  8.]

In [119]: z=dot(x,y)

In [120]: print(z)
60.0
```

(what will x * y **give?)**

Linear algebra

Use *dot* for dot products and matrix multiplications.

- **matrix-vector:**

```
In [129]: print(M)
[[1 3 5]
 [2 4 6]
 [3 5 7]]

In [130]: v=array([1, 2, 1])

In [131]: dot(M,v)
Out[131]: array([12, 16, 20])
```

Linear algebra

Use *dot* for dot products and matrix multiplications.

- **matrix-matrix:**

```
In [148]: print(M1)
[[1 2]
 [3 4]]

In [149]: print(M2)
[[1 1]
 [2 2]]

In [150]: print( dot(M1,M2) )
[[ 5  5]
 [11 11]]

In [151]: print( dot(M2,M1) )
[[ 4  6]
 [ 8 12]]
```

Linear algebra

For more standard operations, np.linalg. <tab>

- **Examples for determinant, inverse, and eigenvalues/eigenvectors:**

```
In [20]: det(M1)
Out[20]: -2.0000000000000004

In [21]: inv(M1)
Out[21]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

In [22]: [l,v]=eig(M1)

In [23]: print(l)
[-0.37228132  5.37228132]

In [24]: print(v)
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Linear algebra

Solve Ax=b **using…** solve:

```
In [63]: A
Out[63]:
array([[1, 2],
       [3, 4]])

In [64]: b
Out[64]: array([1, 2])

In [65]: x=solve(A,b)

In [66]: print(x)
[ 0.   0.5]

In [67]: print( dot(A,x)-b )
[ 0.  0.]
```