Python for Science and Engg: Arrays

FOSSEE

Department of Aerospace Engineering
IIT Bombay

SciPy.in 2010, Tutorials

Outline

Outline

Why arrays?

- Speed!
- Convenience
- Easier to handle multi-dimensional data

Speed

```
In []: a = linspace(0, 100*pi, 1000000)
# array with a million elements
In []: b = []
In []: for each in a:
   ...:     b.append(sin(each))
   ...:
   ...:
In []: sin(a)
```

Convenience

The pendulum problem could've been solved as below::

```
In []: L, T = loadtxt('pendulum.txt',
                       unpack=True)
In []: tsq = T*T
In []: plot (L, tsq, '.')
```

Outline

Initializing

```
In []: c = array([[11,12,13],
                  [21,22,23],
                  [31,32,33]])

In []: c
Out[]:
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

Some special arrays

```
In []: ones((3,5))
Out[]:
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])

In []: ones_like([1, 2, 3, 4])
Out[]: array([1, 1, 1, 1])

In []: identity(2)
Out[]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Also available `zeros, zeros_like, empty, empty_like`

Accessing elements

```
In []: c
Out[]:
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])

In []: c[1][2]
Out[]: 23
In []: c[1,2]
Out[]: 23

In []: c[1]
Out[]: array([21, 22, 23])
```

Similar to `lists` but improved!

Changing elements

```
In []: c[1,1] = -22
In []: c
Out[]:
array([[ 11,  12,  13],
       [ 21, -22,  23],
       [ 31,  32,  33]])

In []: c[1] = 0
In []: c
Out[]:
array([[11, 12, 13],
       [ 0,  0,  0],
       [31, 32, 33]])
```

How do you access one column? – Enter Slicing!

Outline

Slicing: Lists

Define a list

```
In []:  p = [ 2, 3, 5, 7, 11, 13]
```

```
In []: p[1:3]
Out[]: [3, 5]
```

A slice

```
In []: p[0:-1]
Out[]: [2, 3, 5, 7, 11]
In []: p[:]
Out[]: [2, 3, 5, 7, 11, 13]
```

Striding: Lists

Striding over `p`

```
In []: p[::2]
Out[]: [2, 5, 11]
In []: p[1::2]
Out[]: [3, 7, 13]
In []: p[1:-1:2]
Out[]: [3, 7]
In []: p[::3]
Out[]: [2, 7]
```

`list[initial:final:step]`

Slicing & Striding: Lists

What is the output of the following?

```
In []: p[1::4]

In []: p[1:-1:3]
```

Slicing: arrays

```
In []: c[:,1]
Out[]: array([12,  0, 32])

In []: c[1,:]
Out[]: array([0, 0, 0])

In []: c[0:2,:]
Out[]:
array([[11, 12, 13],
       [ 0,  0,  0]])

In []: c[1:3,:]
Out[]:
array([[ 0,  0,  0],
       [31, 32, 33]])
```

Slicing: `arrays` ...

```
In []: c[:2,:]
Out[]:
array([[11, 12, 13],
       [ 0,  0,  0]])

In []: c[1:,:]
Out[]:
array([[ 0,  0,  0],
       [31, 32, 33]])

In []: c[1:,:2]
Out[]:
array([[ 0,  0],
       [31, 32]])
```

Striding: arrays

```
In []: c[::2,:]
Out[]:
array([[11, 12, 13],
       [31, 32, 33]])

In []: c[:,::2]
Out[]:
array([[11, 13],
       [ 0,  0],
       [31, 33]])

In []: c[::2,::2]
Out[]:
array([[11, 13],
       [31, 33]])
```

Shape of an `array`

```
In []: c
Out[]:
array([[11, 12, 13],
       [ 0,  0,  0],
       [31, 32, 33]])

In []: c.shape
Out[]: (3, 3)
```

Shape specifies shape or dimensions of an array

Elementary image processing

```
In []: a = imread('lena.png')

In []: imshow(a)
Out[]: <matplotlib.image.AxesImage object at 0xa0
```

imread returns an array of shape (512, 512, 4) which represents an image of 512x512 pixels and 4 shades.

imshow renders the array as an image.

Slicing & Striding Exercises

- Crop the image to get the top-left quarter
- Crop the image to get only the face
- Resize image to half by dropping alternate pixels

Solutions

```
In []: imshow(a[:256,:256])
Out[]: <matplotlib.image.AxesImage object at 0xb6

In []: imshow(a[200:400,200:400])
Out[]: <matplotlib.image.AxesImage object at 0xb7

In []: imshow(a[::2,::2])
Out[]: <matplotlib.image.AxesImage object at 0xb7
```

Outline

Operations: Addition

Operations on arrays, as already mentioned, are element-wise

```
In []: a = array([[-3,2.5],
                  [2.5,2]])

In []: b = array([[3,2],
                  [2,-2]])

In []: a + b
Out[]:
array([[ 0. ,  4.5],
       [ 4.5,  0. ]])
```

Elementwise Multiplication

```
In []: a*b
Out[]:
array([[-9.,  5.],
       [ 5., -4.]])
```

Matrix Operations using `arrays`

We can perform various matrix operations on `arrays`
A few are listed below.

Operation	How?	Example
Transpose	`.T`	`A.T`
Product	`dot`	`dot(A, B)`
Inverse	`inv`	`inv(A)`
Determinant	`det`	`det(A)`
Sum of all elements	`sum`	`sum(A)`
Eigenvalues	`eigvals`	`eigvals(A)`
Eigenvalues & Eigenvectors	`eig`	`eig(A)`
Norms	`norm`	`norm(A)`
SVD	`svd`	`svd(A)`

Outline

What did we learn?

- Arrays
  - Initializing
  - Accessing elements
  - Slicing & Striding
  - Element-wise Operations
  - Matrix Operations