# Introductory Scientific Computing with Python

## More on numpy arrays

FOSSEE

Department of Aerospace Engineering
IIT Bombay

SciPy India, 2015
December, 2015

# Outline

# Outline

# Matrices: Introduction

All matrix operations are done using `arrays`

# Matrices: 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]])
```

# Initializing some special matrices

```
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])
```

# 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?

# Slicing

```
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 . . .

```
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

```
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 a matrix

```
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 a matrix

# Elementary image processing

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

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

**imread** returns an array of shape (NX, NY, 4) which represents an image of NX x NY 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 bird
- Resize image to half by dropping alternate pixels

# Solutions

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

In []: imshow(a[100:300, 300:550])
Out[]: <matplotlib.image.AxesImage object at 0xb7

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

15 m

# Transpose of a Matrix

```
In []: a = array([[ 1,  1,  2, -1],
   ...:            [ 2,  5, -1, -9],
   ...:            [ 2,  1, -1,  3],
   ...:            [ 1, -3,  2,  7]])

In []: a.T
Out[]:
array([[ 1,  2,  2,  1],
       [ 1,  5,  1, -3],
       [ 2, -1, -1,  2],
       [-1, -9,  3,  7]])
```

# Matrix Addition

```
In []: b = array([[3,2,-1,5],
                  [2,-2,4,9],
                  [-1,0.5,-1,-7],
                  [9,-5,7,3]])
In []: a + b
Out[]:
array([[  4. ,   3. ,   1. ,   4. ],
       [  4. ,   3. ,   3. ,   0. ],
       [  1. ,   1.5,  -2. ,  -4. ],
       [ 10. ,  -8. ,   9. ,  10. ]])
```

# Elementwise Multiplication

```
In []: a*b
Out[]:
array([[  3. ,   2. ,  -2. ,  -5. ],
       [  4. , -10. ,  -4. , -81. ],
       [ -2. ,   0.5,   1. , -21. ],
       [  9. ,  15. ,  14. ,  21. ]])
```

# Matrix Multiplication

```
In []: dot(a, b)
Out[]:
array([[ -6. ,   6. ,  -6. ,  -3. ],
       [-64. ,  38.5, -44. ,  35. ],
       [ 36. , -13.5,  24. ,  35. ],
       [ 58. , -26. ,  34. , -15. ]])
```

# Inverse of a Matrix

```
In []: inv(a)
Out[]:
array([[-0.5 ,  0.55, -0.15,  0.7 ],
       [ 0.75, -0.5 ,  0.5 , -0.75],
       [ 0.5 , -0.15, -0.05, -0.1 ],
       [ 0.25, -0.25,  0.25, -0.25]])
```

Try this: `I = dot(a, inv(a))`

# Determinant and sum of all elements

```
In []: det(a)
Out[]: 80.0

In []: sum(a)
Out[]: 12
```

# Eigenvalues and Eigen Vectors

```
In []: e = array([[3,2,4],[2,0,2],[4,2,3]])

In []: eig(e)
Out[]:
(array([-1.,  8., -1.]),
 array([[-0.74535599,  0.66666667, -0.1931126 ],
        [ 0.2981424 ,  0.33333333, -0.78664085],
        [ 0.59628479,  0.66666667,  0.58643303]]))

In []: eigvals(e)
Out[]: array([-1.,  8., -1.])
```

# Computing Norms

```
In []: norm(e)
Out[]: 8.1240384046359608
```

# Singular Value Decomposition

```
In []: svd(e)
Out[]:
(array(
[[ -6.66666667e-01,  -1.23702565e-16,   7.4535599
 [ -3.33333333e-01,  -8.94427191e-01,  -2.9814239
 [ -6.66666667e-01,   4.47213595e-01,  -5.9628479
 array([ 8.,  1.,  1.]),
 array([[-0.66666667, -0.33333333, -0.66666667],
        [-0.        ,  0.89442719, -0.4472136 ],
        [-0.74535599,  0.2981424 ,  0.59628479]])
```

25 m

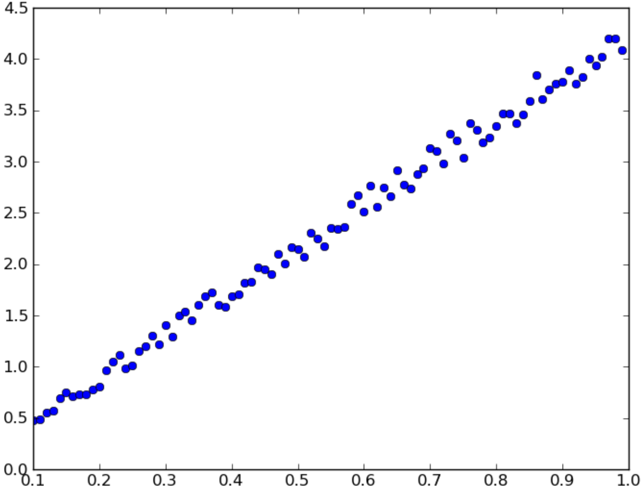
# Outline

# Simple pendulum

- $T \approx 2\pi\sqrt{L/g}$
- $L \propto T^2$

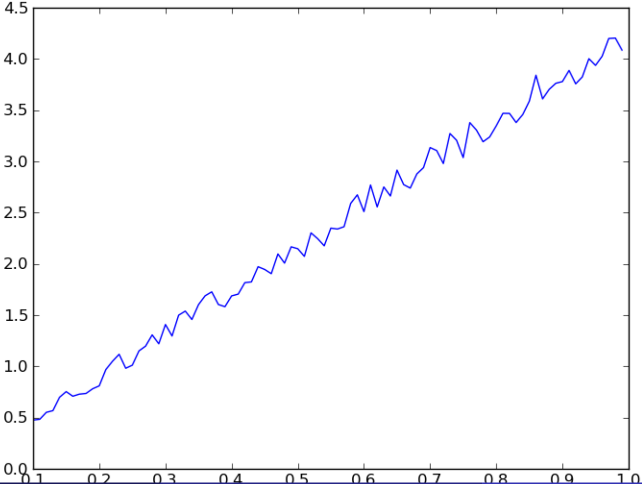
# $L$ vs. $T^2$ - Scatter

Linear trend visible.



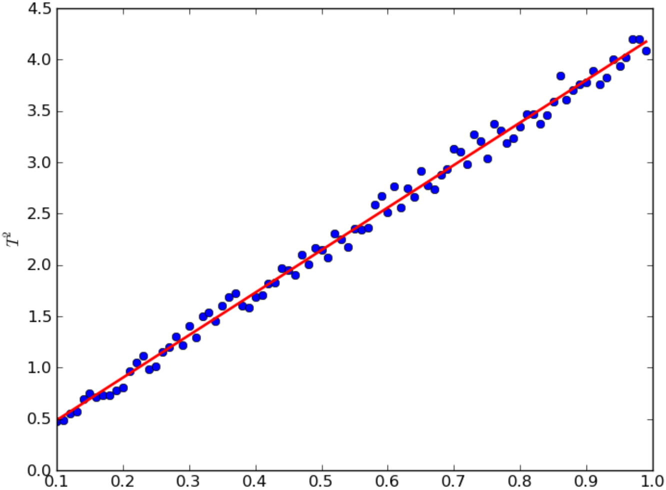
# $L$ vs. $T^2$ - Line

This line does not make any mathematical sense.



# $L$ vs. $T^2$ - Least Square Fit

This is what our intention is.



# Matrix Formulation

- We need to fit a line through points for the equation $T^2 = mL + c$
- We have data in this form:
  $T_1^2 = mL_1 + c$
  $T_2^2 = mL_2 + c$
  $T_3^2 = mL_3 + c$
  $\ldots$
- Find "best" $m$ and $c$

# Matrix Formulation

- In matrix form, the equation can be represented as $T_{sq} = A \cdot p$, where $T_{sq}$ is $\begin{bmatrix} T_1^2 \\ T_2^2 \\ \vdots \\ T_N^2 \end{bmatrix}$ , A is $\begin{bmatrix} L_1 & 1 \\ L_2 & 1 \\ \vdots & \vdots \\ L_N & 1 \end{bmatrix}$ and p is $\begin{bmatrix} m \\ c \end{bmatrix}$
- We need to find $p$ to plot the line

# Getting $L$ and $T^2$

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

# Generating *A*

```
In []: A = array([L, ones_like(L)])
In []: A = A.T
```

# `lstsq`...

- Now use the **`lstsq`** function
- Along with a lot of things, it returns the least squares solution

```
In []: result = lstsq(A,tsq)
In []: coef = result[0]
```

# Least Square Fit Line . . .

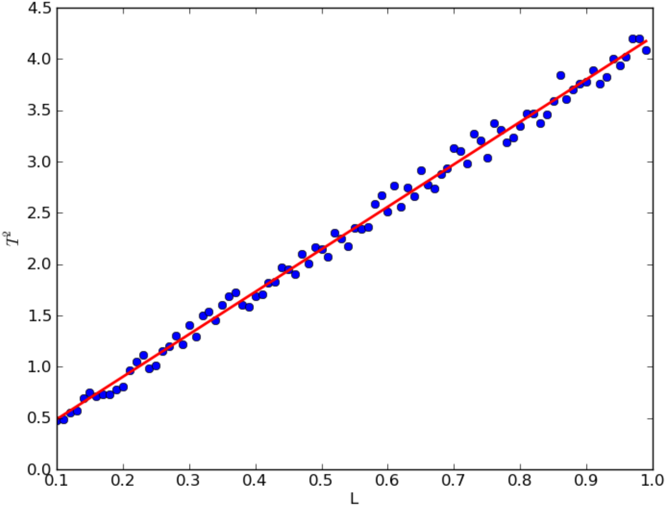
We get the points of the line from `coef`

```
In []: Tline = coef[0]*L + coef[1]

In []: Tline.shape
```

- Now plot `Tline` vs. `L`, to get the least squares fit line.

```
In []: plot(L, Tline, 'r')

In []: plot(L, tsq, 'o')
```

# Least Squares Fit



# Outline

# numpy.random

- Easy random number generation

```
In []: random?
```

Or:

```
In []: import numpy
In []: numpy.random
```

- **random.random**: produces uniform deviates in $[0, 1)$
- **random.normal**: draws random samples from a Gaussian distribution
- Useful to create a random matrix of any shape

# Using the `random` module

```
In []: x = random.random(size=1000)
In []: y = random.random(size=1000)
In []: scatter(x, y) # Scatter plot it.

In []: x,y = random.normal(size=(2,1000))
In []: clf()
In []: scatter(x, y)
```

Note that `size` can be a tuple.

# Using the `random` module

```
In []: img = random.random(
   .                        size=(200, 200))
In []: clf()
In []: imshow(img)
```

40 m

# Outline

# What did we learn?

- Matrices
- Least Squares
- Random numbers