Python for Science and Engg: Matrices & Least Square Fit

FOSSEE

Department of Aerospace Engineering
IIT Bombay

7 November, 2009
Day 1, Session 4

Outline

Outline

Matrices: Introduction

All matrix operations are done using **`arrays`**

Matrices: Initializing

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

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, 5])
Out[ ]: array([1, 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 = array([[1,1,2],
                  [2,4,1],
                  [-1,3,7]])

In [ ]: C[1][2]
Out[ ]: 1

In [ ]: C[1,2]
Out[ ]: 1

In [ ]: C[1]
Out[ ]: array([2, 4, 1])
```

Changing elements

```
In [ ]: C[1,1] = -2
In [ ]: C
Out[ ]:
array([[ 1,  1,  2],
       [ 2, -2,  1],
       [-1,  3,  7]])

In [ ]: C[1] = [0,0,0]
In [ ]: C
Out[ ]:
array([[ 1,  1,  2],
       [ 0,  0,  0],
       [-1,  3,  7]])
```

How to change one column?

Slicing

```
In [ ]: C[:,1]
Out[ ]: array([1, 0, 3])

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

In [ ]: C[0:2,:]
Out[ ]:
array([[1, 1, 2],
       [0, 0, 0]])

In [ ]: C[1:3,:]
Out[ ]:
array([[ 0,  0,  0],
       [-1,  3,  7]])
```

Slicing . . .

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

In [ ]: C[1:,:]
Out[ ]:
array([[ 0,  0,  0],
       [-1,  3,  7]])

In [ ]: C[1:,:2]
Out[ ]:
array([[ 0,  0],
       [-1,  3]])
```

Striding

```
In [ ]: C[::2,:]
Out[ ]:
array([[ 1,  1,  2],
       [-1,  3,  7]])

In [ ]: C[:,::2]
Out[ ]:
xarray([[ 1,  2],
       [ 0,  0],
       [-1,  7]])

In [ ]: C[::2,::2]
Out[ ]:
array([[ 1,  2],
       [-1,  7]])
```

Slicing & Striding Exercises

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

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

In [ ]: A.shape
Out[ ]: (512, 512, 4)
```

- 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
```

Transpose of a Matrix

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

Sum of all elements

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

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]])
```

Determinant

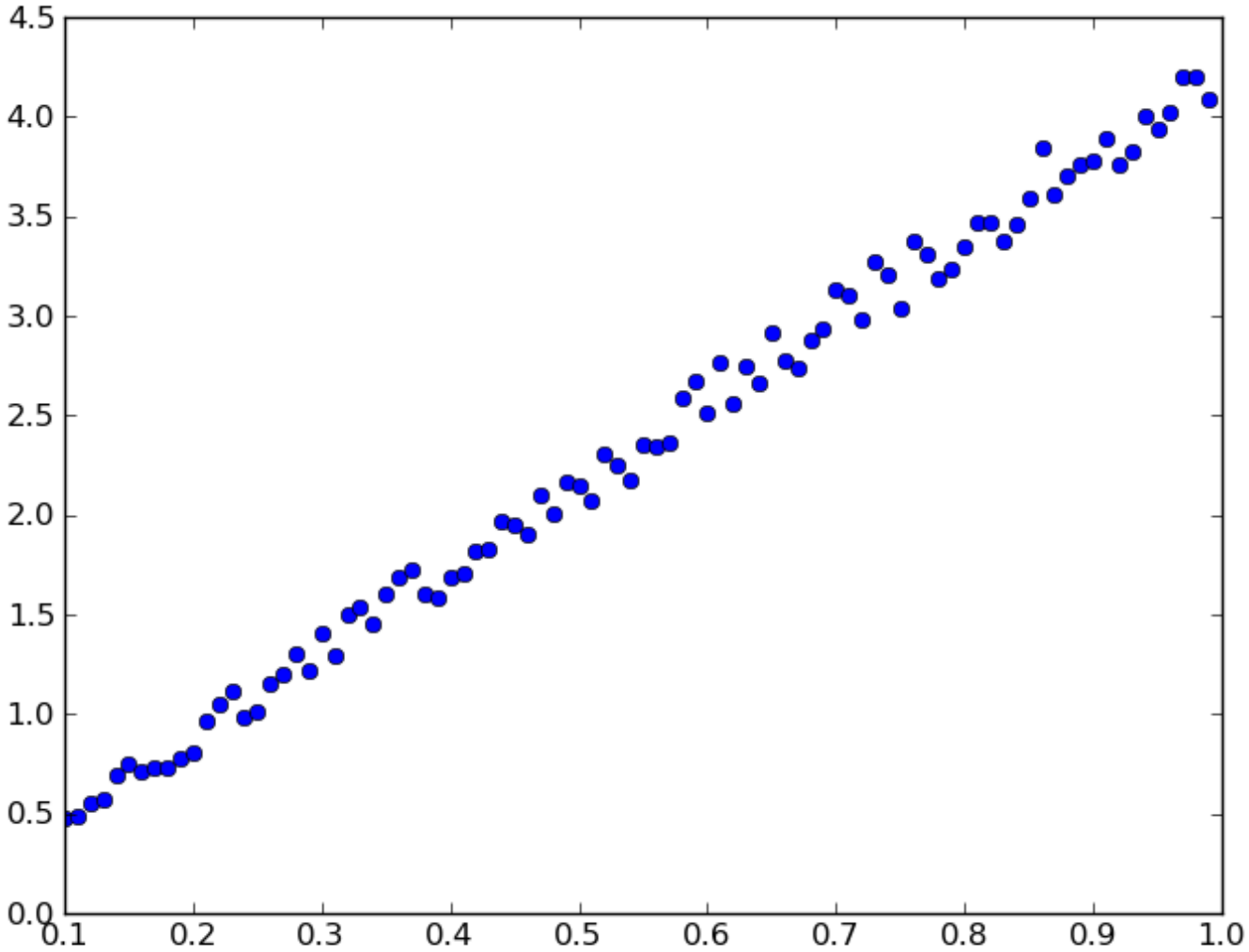
```
In [ ]: det(A)
Out[ ]: 80.0
```

Eigenvalues and Eigen Vectors

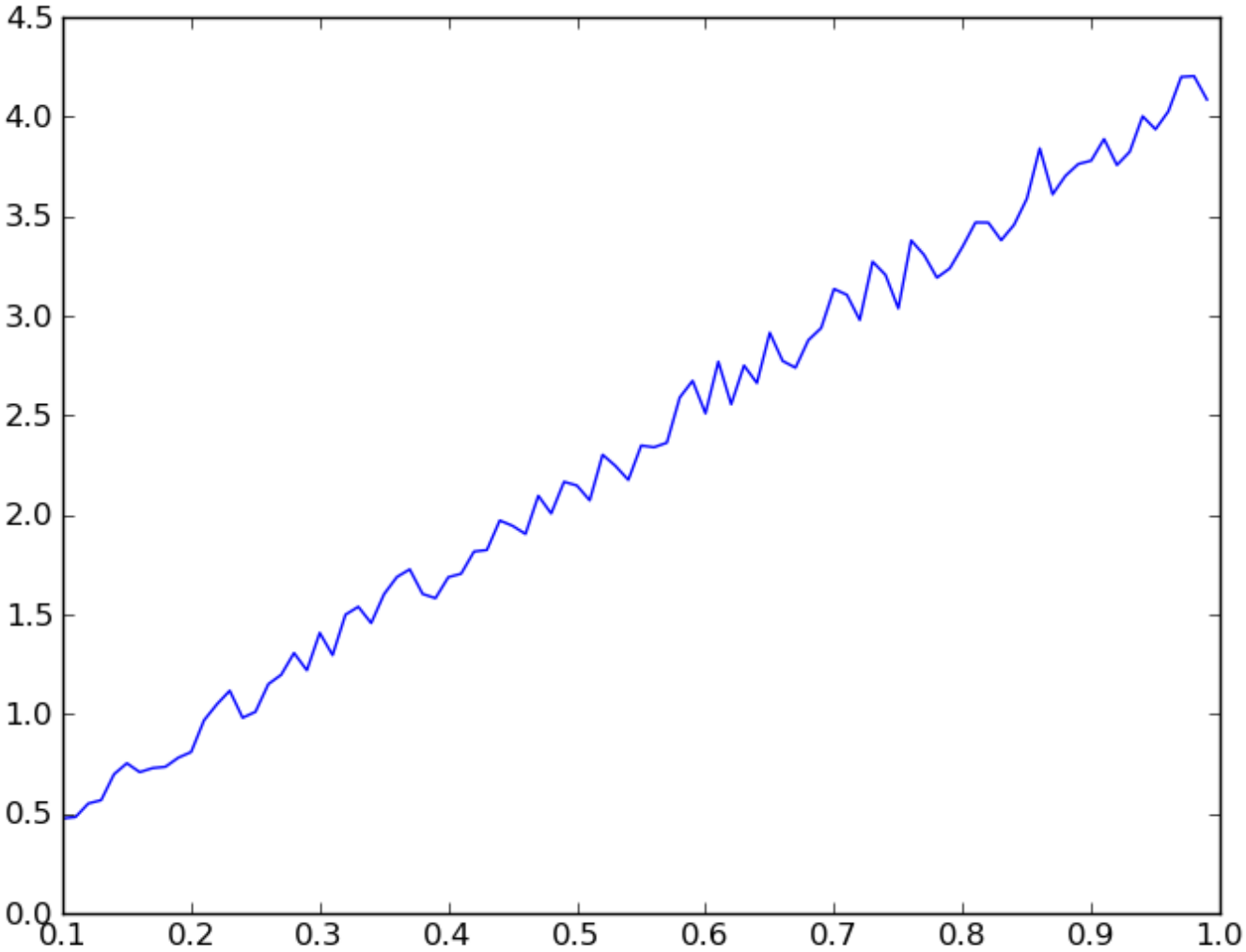
```
In [ ]: E = array([[3,2,4],[2,0,2],[4,2,3]])

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.])
```

Outline

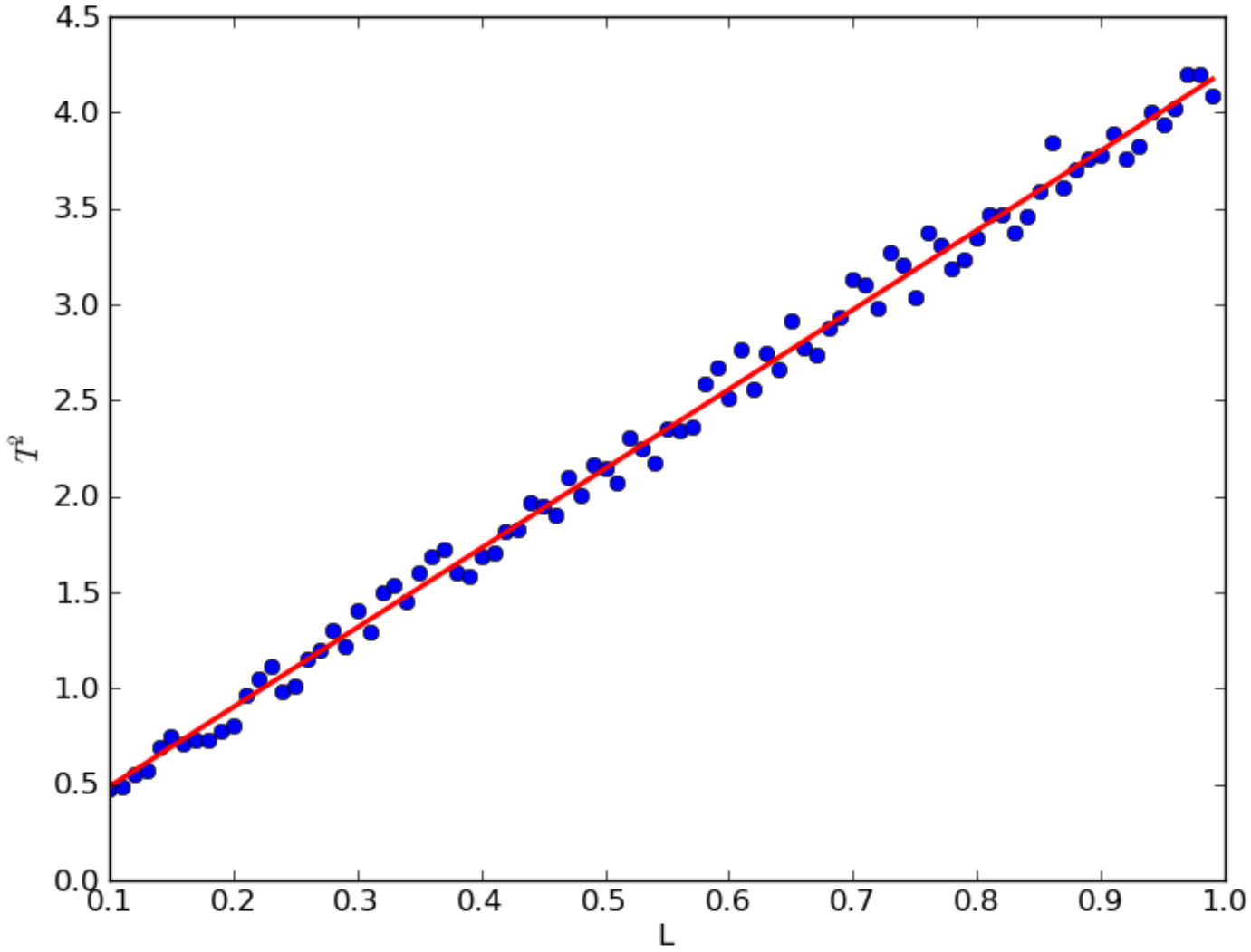
L vs. T^2 - Scatter



L vs. T^2 - Line



L vs. T^2 - Least Square Fit



Least Square Fit Curve

- T^2 and L have a linear relationship
- Hence, Least Square Fit Curve is a line
- we shall use the **`lstsq`** function

lstsq

- We need to fit a line through points for the equation $T^2 = m \cdot L + c$
- In matrix form, the equation can be represented as

$$T^2 = A \cdot p, \text{ where A is } \begin{bmatrix} L_1 & 1 \\ L_2 & 1 \\ \vdots & \vdots \\ L_N & 1 \end{bmatrix} \text{ and p is } \begin{bmatrix} m \\ c \end{bmatrix}$$

- We need to find p to plot the line

Getting L and T^2

If you closed IPython after session 2

```
In []: l = []
In []: t = []
In []: for line in open('pendulum.txt'):
   ....:     point = line.split()
   ....:     l.append(float(point[0]))
   ....:     t.append(float(point[1]))
   ....:
   ....:
```

Generating *A*

```
In []: A = array([1, ones_like(1)])
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 . . .

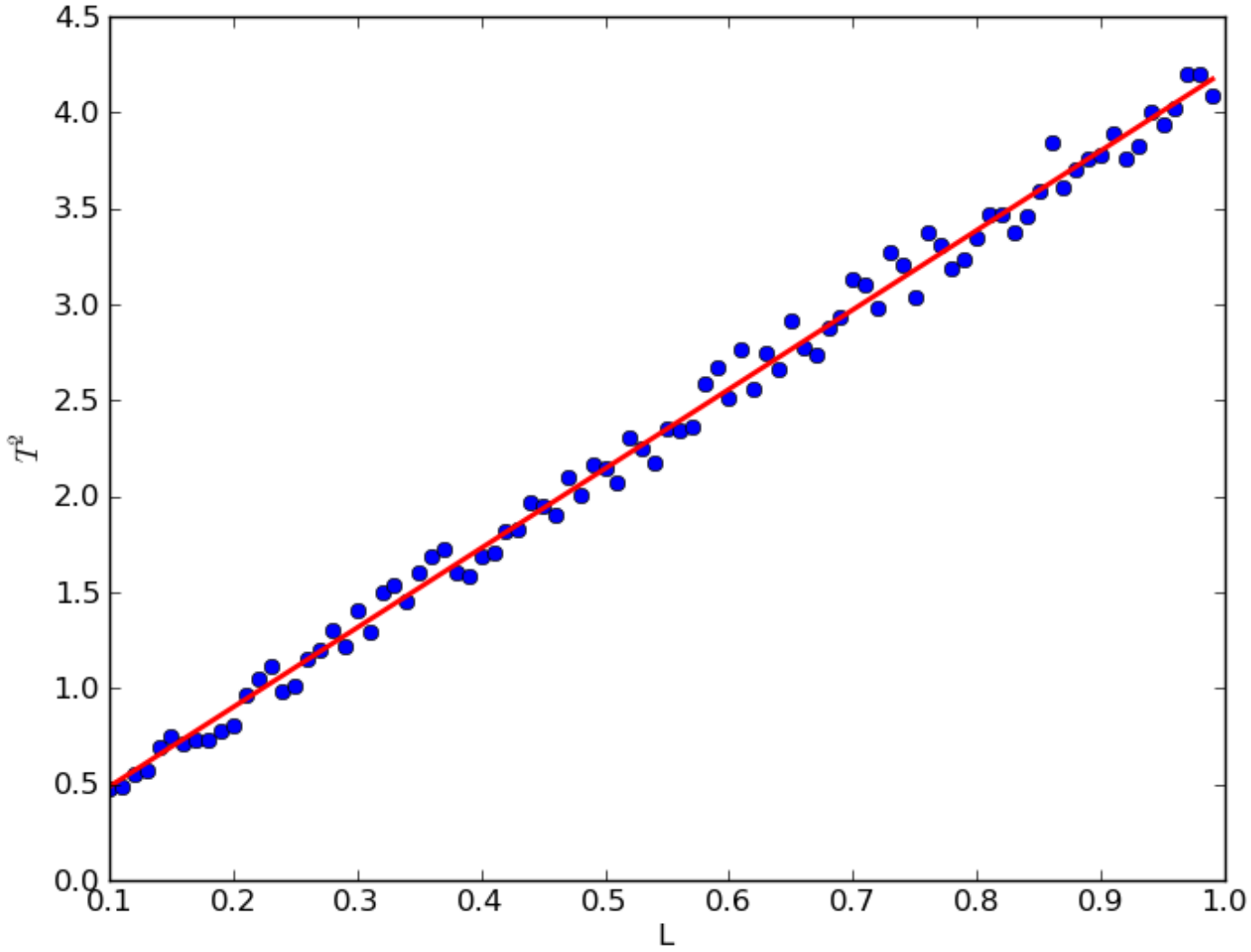
We get the points of the line from **`coef`**

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

- Now plot **`Tline`** vs. **`l`**, to get the Least squares fit line.

```
In []: plot(l, Tline)
```

Least Squares Fit



Outline

What did we learn?

- Matrices
  - Initializing
  - Accessing elements
  - Slicing and Striding
  - Transpose
  - Addition
  - Multiplication
  - Inverse of a matrix
  - Determinant
  - Eigenvalues and Eigen vector
- Least Square Curve fitting