Python for Science and Engg: SciPy

FOSSEE

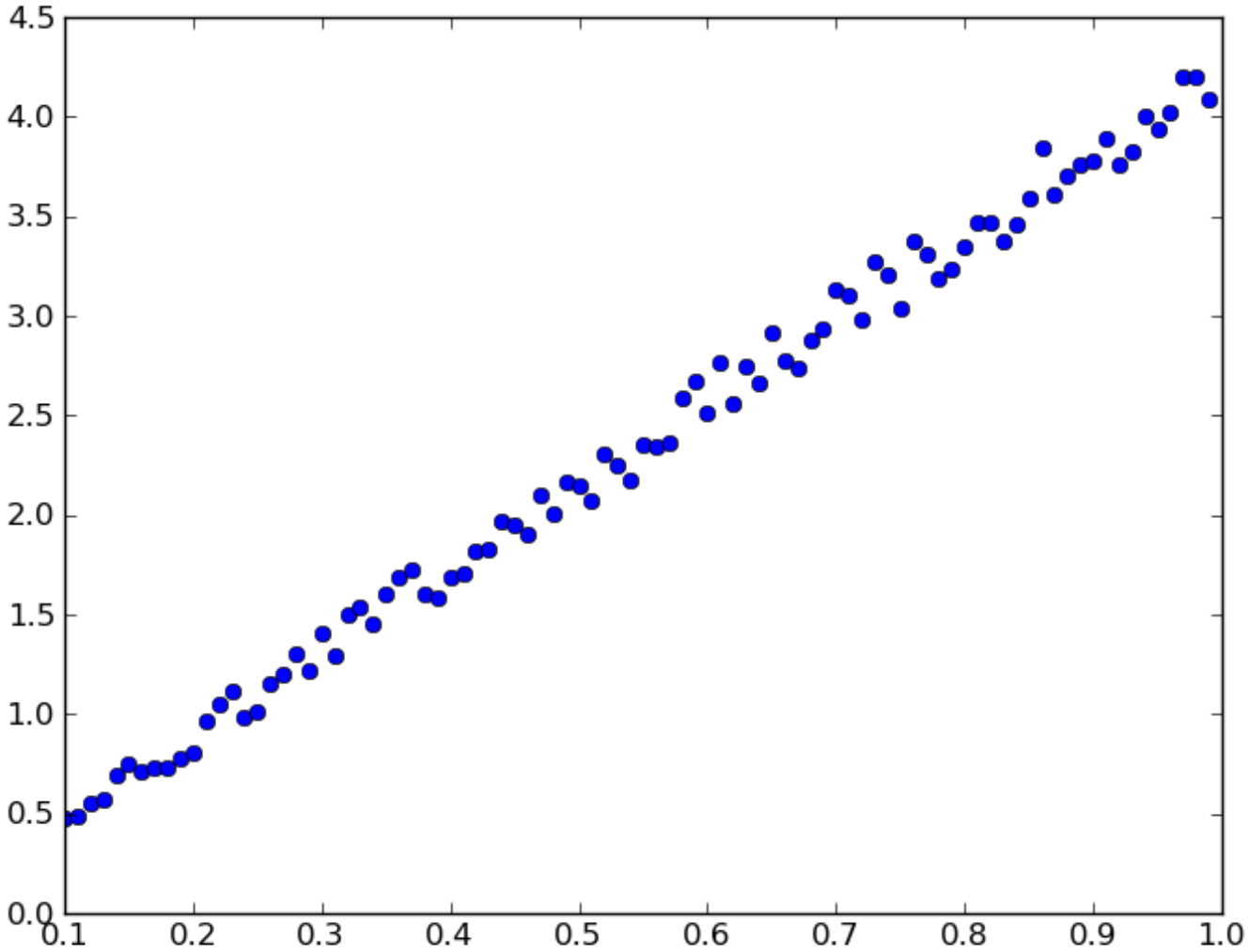
Department of Aerospace Engineering
IIT Bombay

SciPy.in 2010, Tutorials

Outline

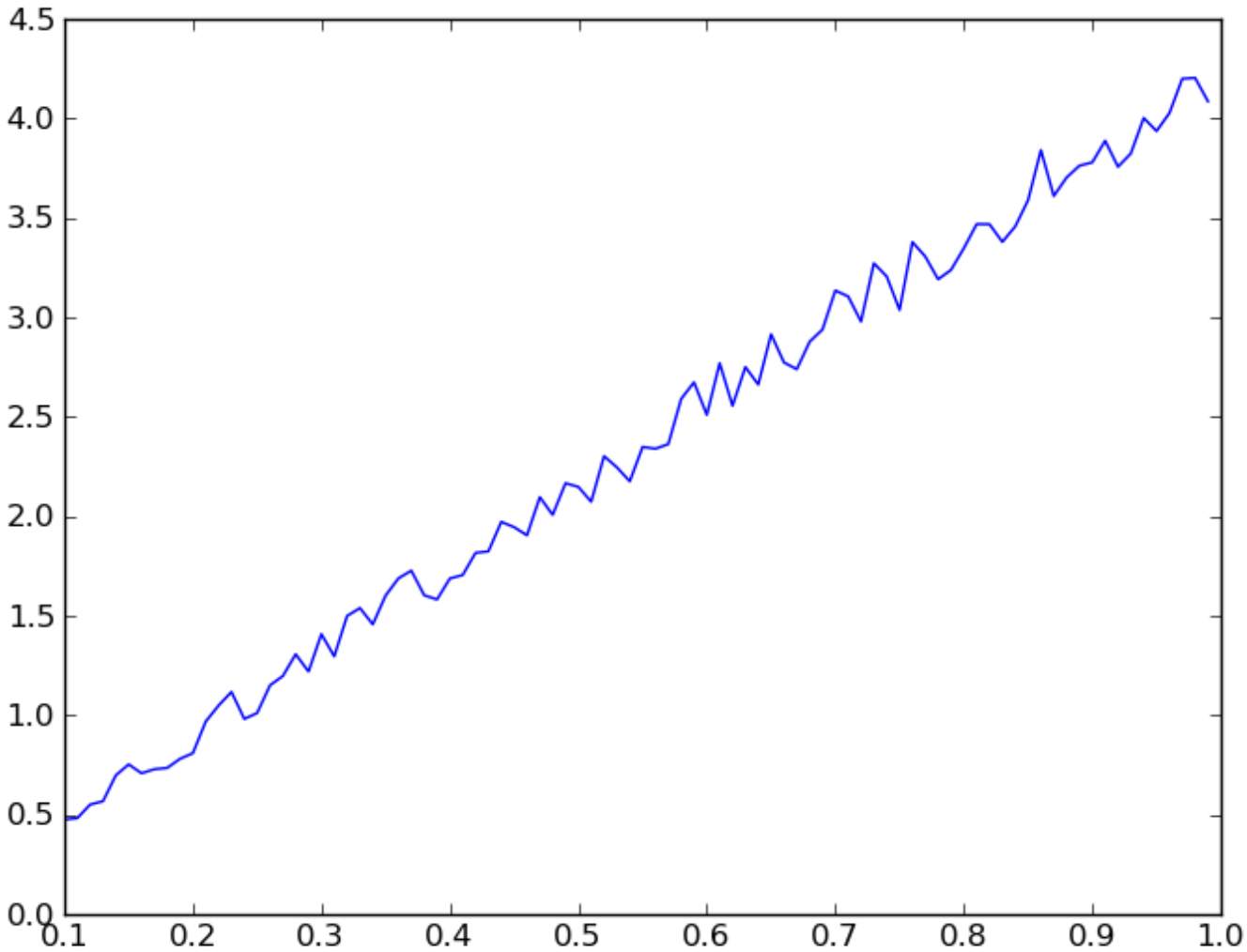
L vs. T^2 - Scatter

Linear trend visible.



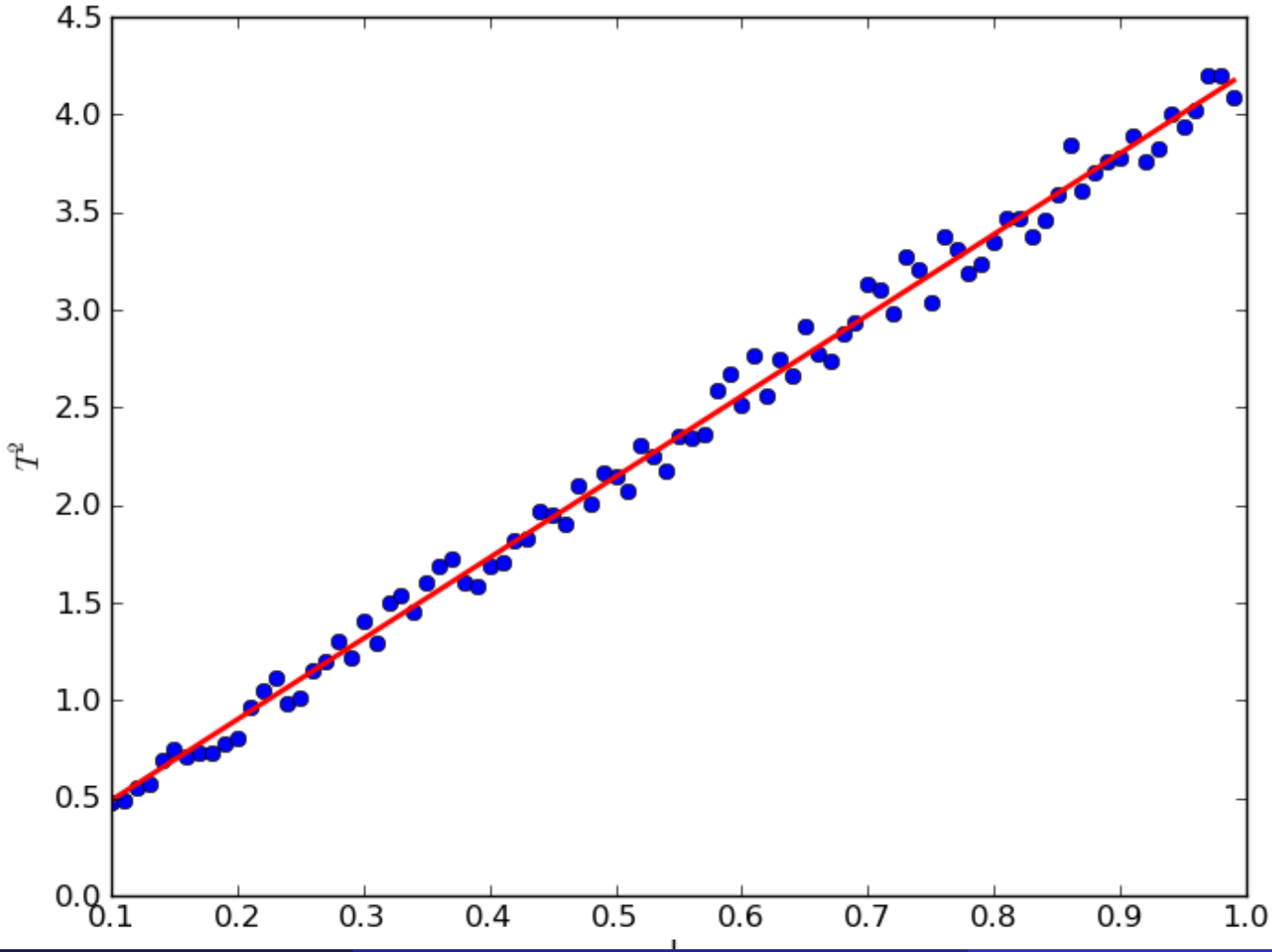
L vs. T^2 - Line

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 $T^2 = m \cdot L + c$
- 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]
```

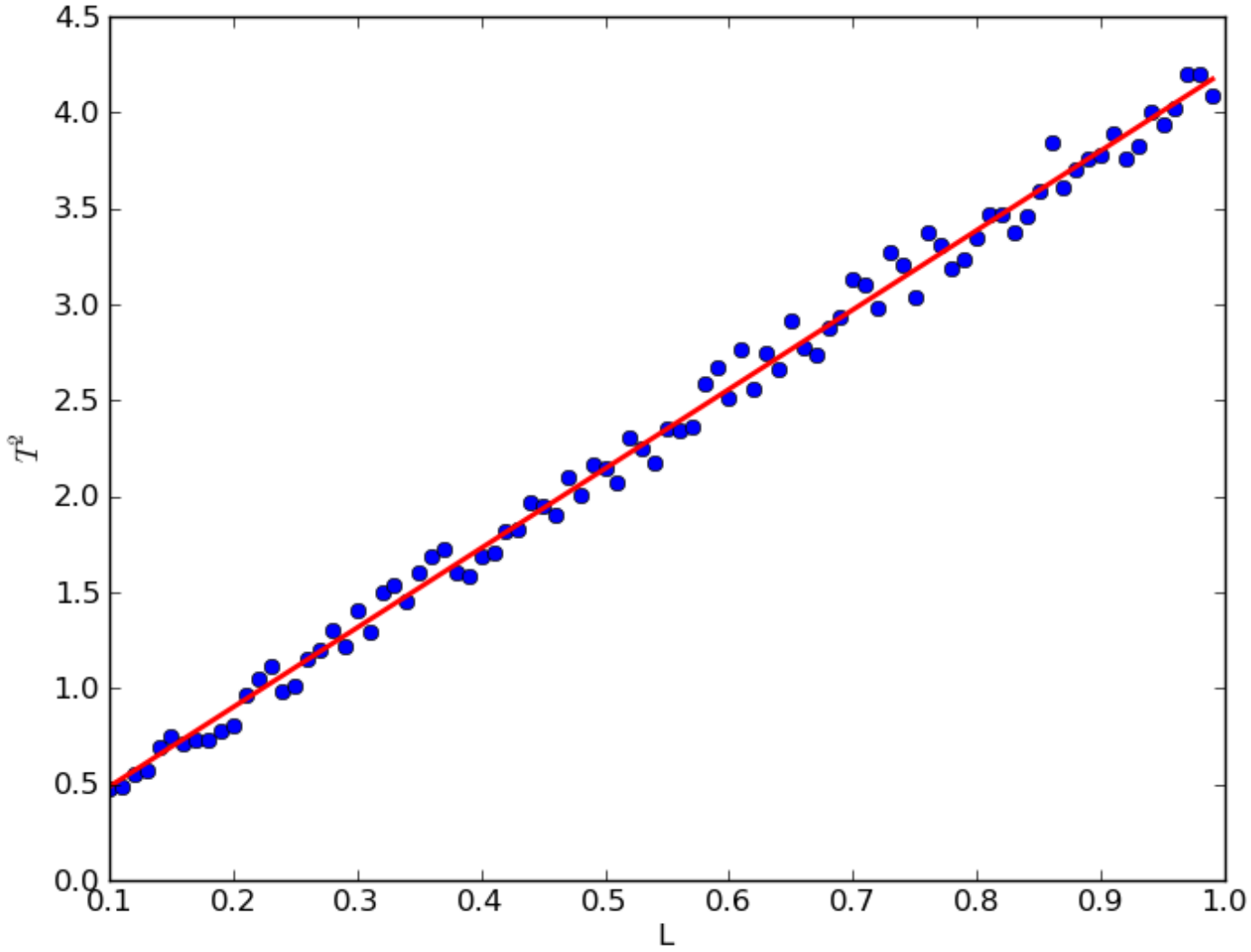
```
In []: Tline.shape
```

- 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

Solution of equations

Consider,

$$3x + 2y - z = 1$$
$$2x - 2y + 4z = -2$$
$$-x + \frac{1}{2}y - z = 0$$

Solution:

$$x = 1$$
$$y = -2$$
$$z = -2$$

Solving using Matrices

Let us now look at how to solve this using **matrices**

```
In []: A = array([[3,2,-1],
                  [2,-2,4],
                  [-1, 0.5, -1]])
In []: b = array([1, -2, 0])
In []: x = solve(A, b)
```

Solution:

```
In [ ]: x
Out[ ]: array([ 1., -2., -2.])
```

Let's check!

```
In [ ]: Ax = dot(A, x)
In [ ]: Ax
Out[ ]: array([  1.00000000e+00,  -2.00000000e+00,
-1.11022302e-16])
```

The last term in the matrix is actually 0!
We can use `allclose()` to check.

```
In [ ]: allclose(Ax, b)
Out[ ]: True
```

10 m

Problem

Solve the set of equations:

$$
\begin{aligned}
x + y + 2z - w &= 3 \\
2x + 5y - z - 9w &= -3 \\
2x + y - z + 3w &= -11 \\
x - 3y + 2z + 7w &= -5
\end{aligned}
$$

15 m

Solution

Use **`solve()`**

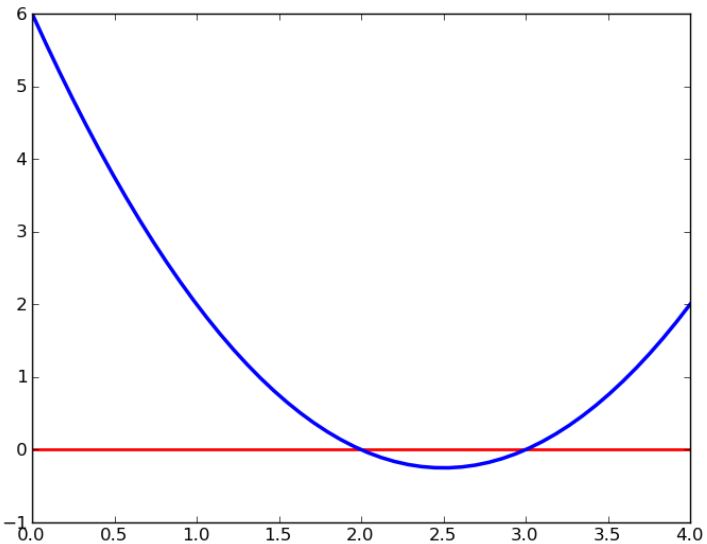
$$x = -5$$
$$y = 2$$
$$z = 3$$
$$w = 0$$

Outline

SciPy: `roots`

- Calculates the roots of polynomials
- To calculate the roots of $x^2 - 5x + 6$

```
In []: coeffs = [1, -5, 6]
In []: roots(coeffs)
Out[]: array([3., 2.])
```



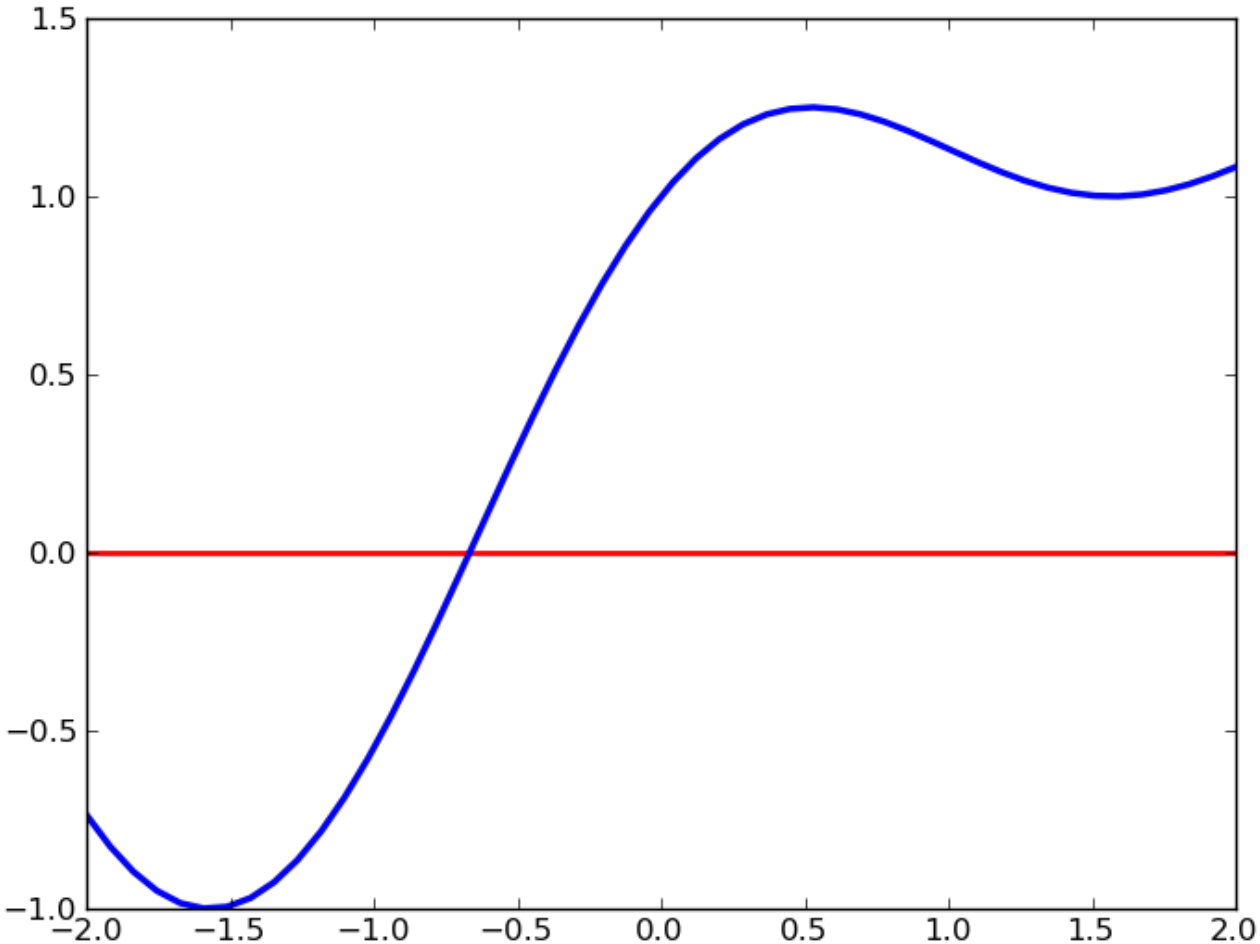
SciPy: `fsolve`

```
In []: from scipy.optimize import fsolve
```

- Finds the roots of a system of non-linear equations
- Input arguments - Function and initial estimate
- Returns the solution

fsolve

Find the root of $sin(z) + cos^2(z)$ nearest to 0



fsolve

Root of $sin(z) + cos^2(z)$ nearest to 0

```
In []: fsolve(sin(z)+cos(z)*cos(z), 0)
NameError: name 'z' is not defined
```

fsolve

```
In []: z = linspace(-pi, pi)
In []: fsolve(sin(z)+cos(z)*cos(z), 0)
```

```
TypeError:
'numpy.ndarray'object is not callable
```

Functions - Definition

We have been using them all along. Now let's see how to define them.

```
In []: def g(z):
  ....:     return sin(z)+cos(z)*cos(z)
```

- `def` – keyword
- name: `g`
- arguments: `z`
- `return` – keyword

Functions - Calling them

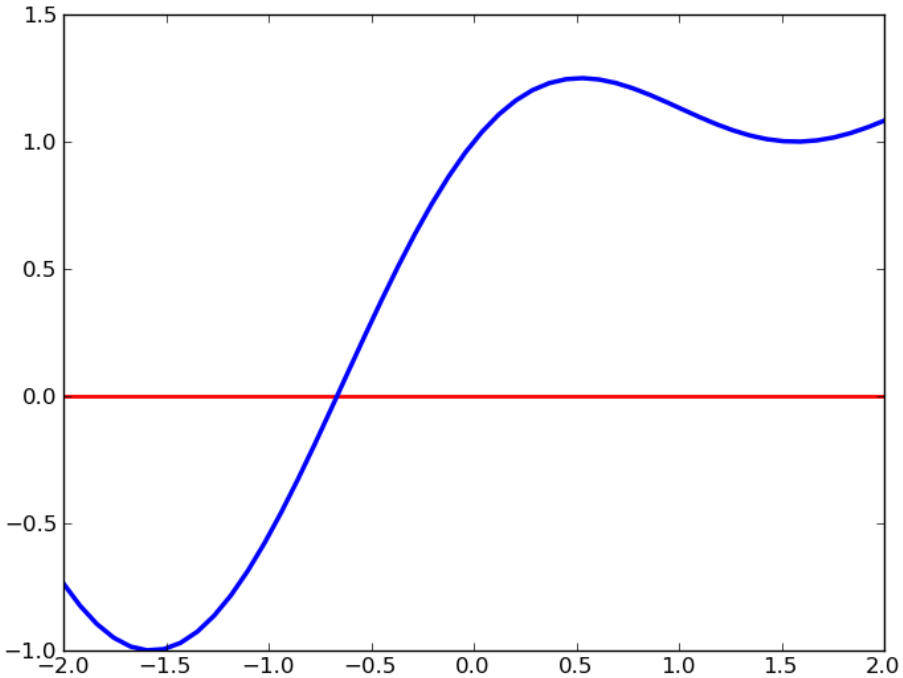
```
In [ ]: g()
---------------------------------------

TypeError:g() takes exactly 1 argument
(0 given)

In [ ]: g(0)
Out[ ]: 1.0
In [ ]: g(1)
Out[ ]: 1.1333975665343254
```

More on Functions later . . .

fsolve ...

Find the root of $sin(z) + cos^2(z)$ nearest to 0

```
In []: fsolve(g, 0)
Out[]: -0.6662394324925153
```

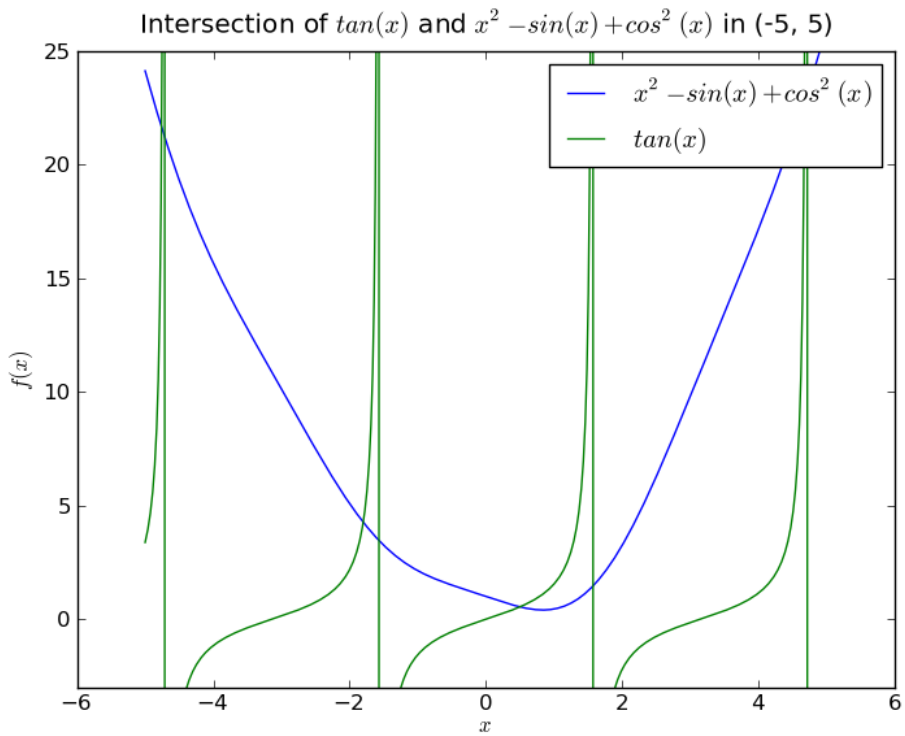


Exercise Problem

Find the root of the equation
$x^2 - sin(x) + cos^2(x) = tan(x)$ nearest to 0

Solution

```
def g(x):
    return x**2 - sin(x) + cos(x)*cos(x) - tan(x)
fsolve(g, 0)
```



Outline

Solving ODEs using SciPy

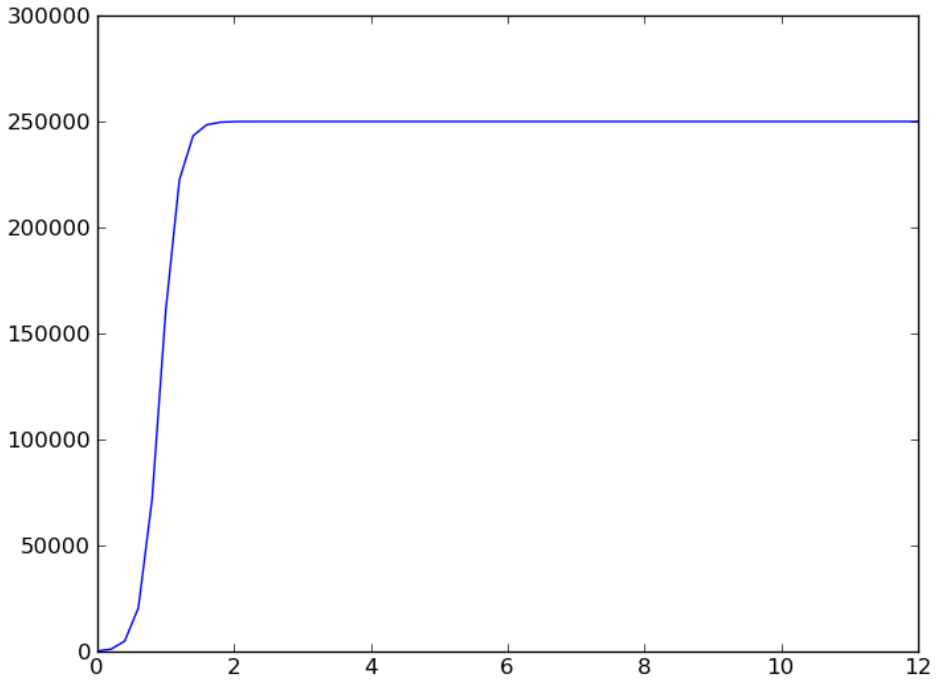
- Consider the spread of an epidemic in a population
- $\frac{dy}{dt} = ky(L - y)$ gives the spread of the disease
- L is the total population.
- Use $L = 2.5E5, k = 3E - 5, y(0) = 250$
- Define a function as below

```
In []: from scipy.integrate import odeint
In []: def epid(y, t):
   ....:     k = 3.0e-5
   ....:     L = 2.5e5
   ....:     return k*y*(L-y)
   ....:
```

Solving ODEs using SciPy . . .

```
In []: t = linspace(0, 12, 61)

In []: y = odeint(epid, 250, t)

In []: plot(t, y)
```

Result



ODEs - Simple Pendulum

We shall use the simple ODE of a simple pendulum.

$$\ddot{\theta} = -\frac{g}{L} sin(\theta)$$

- This equation can be written as a system of two first order ODEs

$$\dot{\theta} = \omega \tag{1}$$

$$\dot{\omega} = -\frac{g}{L} sin(\theta) \tag{2}$$

$$\text{At } t = 0 :$$

$$\theta = \theta_0(10^o) \quad \& \quad \omega = 0 \ (\textit{Initial values})$$

ODEs - Simple Pendulum . . .

- Use **odeint** to do the integration

```
In []: def pend_int(initial, t):
   ....     theta = initial[0]
   ....     omega = initial[1]
   ....     g = 9.81
   ....     L = 0.2
   ....     F=[omega, -(g/L)*sin(theta)]
   ....     return F
   ....
```

ODEs - Simple Pendulum . . .

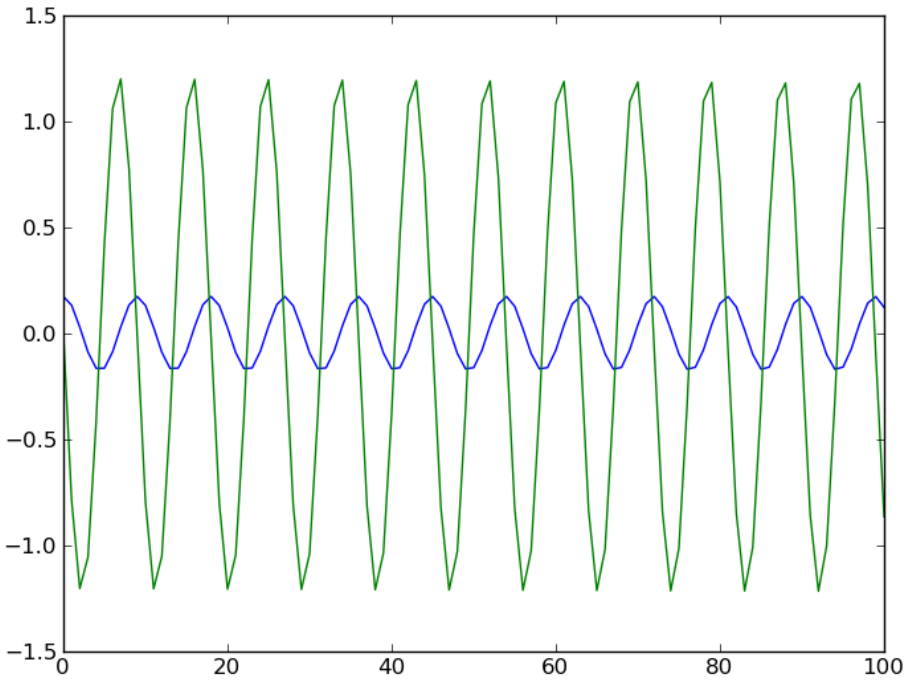
- `t` is the time variable
- `initial` has the initial values

```
In []: t = linspace(0, 20, 101)
In []: initial = [10*2*pi/360, 0]
```

ODEs - Simple Pendulum . . .

```
In [ ]: from scipy.integrate import odeint

In [ ]: pend_sol = odeint(pend_int,
                          initial,t)
```

Result



Outline

The FFT

- We have a simple signal $y(t)$
- Find the FFT and plot it

```
In []: t = linspace(0, 2*pi, 500)
In []: y = sin(4*pi*t)

In []: f = fft(y)
In []: freq = fftfreq(500, t[1] - t[0])

In []: plot(freq[:250], abs(f)[:250])
In []: grid()
```

FFTs cont. . .

```
In []: y1 = ifft(f) # inverse FFT
In []: allclose(y, y1)
Out[]: True
```

FFTs cont. . .

Let us add some noise to the signal

```
In []: yr = y + random(size=500)*0.2
In []: yn = y + normal(size=500)*0.2

In []: plot(t, yr)
In []: figure()
In []: plot(freq[:250],
   ...:        abs(fft(yn))[:250])
```

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

FFTs cont. . .

Filter the noisy signal:

```
In []: from scipy import signal
In []: yc = signal.wiener(yn, 5)
In []: clf()
In []: plot(t, yc)
In []: figure()
In []: plot(freq[:250],
   ...:      abs(fft(yc))[:250])
```

Only scratched the surface here . . .

Things we have learned

- Least Square Fit
- Solving Linear Equations
- Defining Functions
- Finding Roots
- Solving ODEs
- Random number generation
- FFTs and basic signal processing