Introductory Scientific Computing with Python

Basic SciPy and IPython notebooks

FOSSEE

Department of Aerospace Engineering
IIT Bombay

SciPy India, 2015
December, 2015

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 should be 0!
We can use **allclose()** to check.

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

5 m

Problem

Solve the set of equations:

$$x + y + 2z - w = 3$$
$$2x + 5y - z - 9w = -3$$
$$2x + y - z + 3w = -11$$
$$x - 3y + 2z + 7w = -5$$

10 m

Solution

Use `solve()`

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

Outline

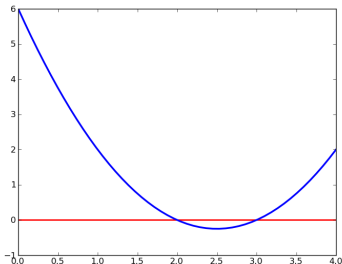# SciPy: `roots`

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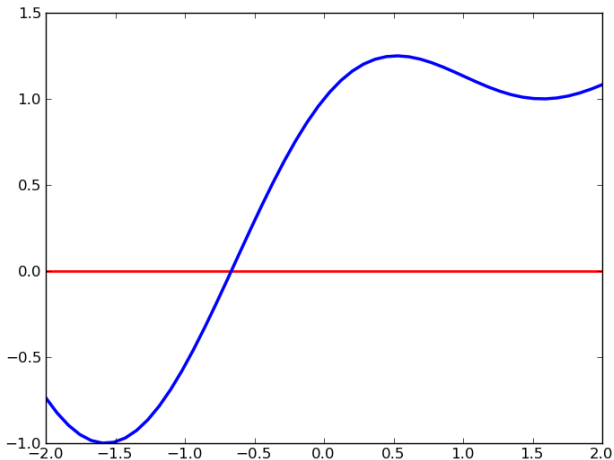- To calculate the roots of $x^2 - 5x + 6$

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

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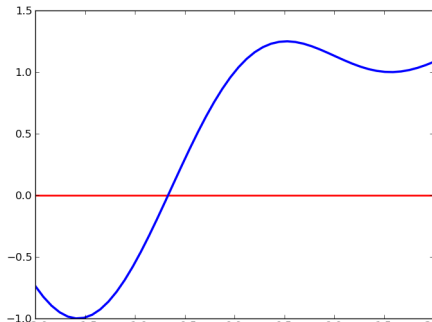
Find the root of $sin(z) + cos^2(z)$ nearest to 0

`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 ...

```
In []: def g(z):
 ....:     return sin(z)+cos(z)*cos(z)
In []: fsolve(g, 0)
Out[]: -0.66623943249251527
```
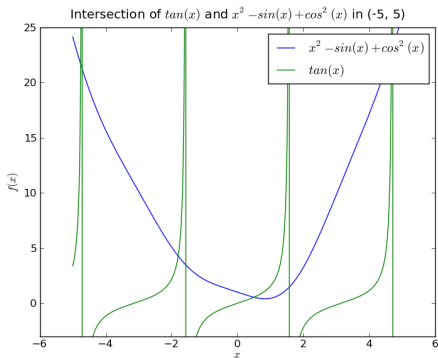
Exercise Problem

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)
```


Intersection of $tan(x)$ and $x^2 - sin(x) + cos^2(x)$ in (-5, 5)

Outline

1. Solving linear systems

2. Finding Roots

3. **ODEs**

4. FFTs

5. IPython notebooks

Solving ODEs using SciPy

- Consider the spread of an epidemic in a population
- y is the number of infected people
- $\frac{dy}{dt} = ky(L - y)$ gives the spread of the disease
- L is the total population.
- Use $L = 2.5 \times 10^5, k = 3 \times 10^{-3}, y(0) = 250$

Solving ODEs using SciPy

- 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)
  ....
```
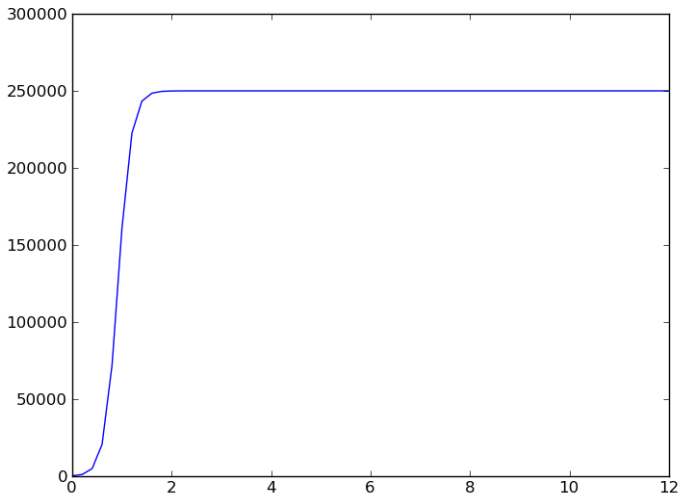
Solving ODEs using SciPy . . .

```
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

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

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

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]
```
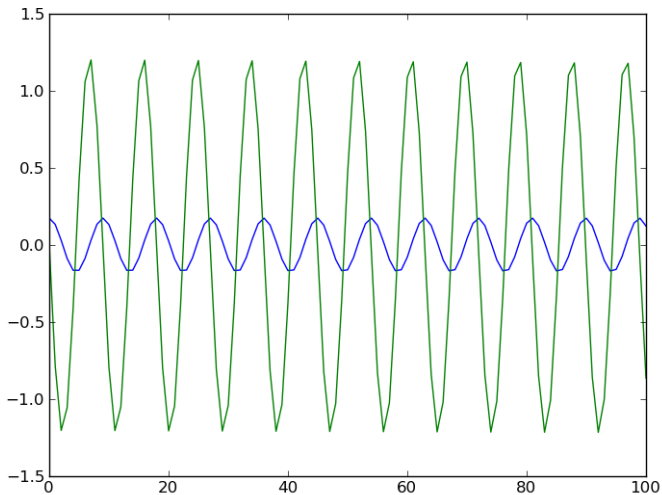
ODEs - Simple Pendulum . . .

```
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 ... 40 m

Outline

Notebooks?

- One document
- HTML (MD) + LaTeX math
- Python code
- Results
- Interactive widgets
- Easy sharing

Starting Notebooks

```
$ ipython notebook
# OR
$ jupyter notebook
```

Notebook demo

Things we have learned

- Solving Linear Equations
- Defining Functions
- Finding Roots
- Solving ODEs
- FFTs and basic signal processing
- Using IPython notebooks

Further reading

- ipython.github.com/ipython-doc
- matplotlib.sf.net/contents.html
- scipy.org/Tentative_NumPy_Tutorial
- docs.scipy.org/doc/scipy/reference/
  tutorial