A Glimpse at Scipy

FOSSEE

June 2010

Abstract

This document shows a glimpse of the features of Scipy that will be explored during this course.

1 Introduction

SciPy is open-source software for mathematics, science, and engineering.

SciPy (pronounced "Sigh Pie") is a collection of mathematical algorithms and convenience functions built on the Numpy extension for Python. It adds significant power to the interactive Python session by exposing the user to high-level commands and classes for the manipulation and visualization of data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling sytems such as *Matlab, IDL, Octave, R-Lab, and Scilab*. [1]

1.1 Sub-packages of Scipy

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the table 1.

1.2 Use of Scipy in this course

Following is a partial list of tasks we shall perform using Scipy, in this course.

1. Plotting [1]
2. Matrix Operations
    - Inverse
    - Determinant
3. Solving Equations
    - System of Linear equations

[1] using `pylab` - see Appendix A

Table 1: Sub-packages available in Scipy

Subpackage	**Description**
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fftpack`	Fast Fourier Transform routines
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`maxentropy`	Maximum entropy methods
`ndimage`	N-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization and root-nding routines
`signal`	Signal processing
`sparse`	Sparse matrices and associated routines
`spatial`	Spatial data structures and algorithms
`special`	Special functions
`stats`	Statistical distributions and functions
`weave`	C/C++ integration

- Polynomials
- Non-linear equations

4. Integration
   - Quadrature
   - ODEs

2 A Glimpse of Scipy functions

This section gives a brief overview of the tasks that are going to be performed using Scipy, in future classes of this course.

2.1 Matrix Operations

Let **A** be the matrix $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$

To input **A** matrix into python, we do the following in `ipython`[2]

```
In []: A = array([[1,3,5],[2,5,1],[2,3,8]])
```

[2] `ipython` must be started with `-pylab` flag

2.1.1 Inverse

The inverse of a matrix $\mathbf{A}$ is the matrix $\mathbf{B}$ such that $\mathbf{AB} = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix consisting of ones down the main diagonal. Usually $\mathbf{B}$ is denoted $\mathbf{B} = \mathbf{A}^{-1}$. In SciPy, the matrix inverse of matrix $\mathbf{A}$ is obtained using `inv(A)`.

```
In []: inv(A)
Out[]:
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
```

2.1.2 Determinant

The determinant of a square matrix $\mathbf{A}$ is denoted $|\mathbf{A}|$. Suppose a_{ij} are the elements of the matrix $\mathbf{A}$ and let $\mathbf{M}_{ij} = |\mathbf{A}_{ij}|$ be the determinant of the matrix left by removing the i^{th} row and j^{th} column from $\mathbf{A}$. Then for any row i

$$|\mathbf{A}| = \sum_j (-1)^{i+j} a_{ij}\mathbf{M}_{ij}$$

This is a recursive way to define the determinant where the base case is defined by accepting that the determinant of a 1×1 matrix is the only matrix element. In SciPy the determinant can be calculated with *det* . For example, the determinant of

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$$

is

$$\begin{aligned} |\mathbf{A}| &= 1\begin{vmatrix} 5 & 1 \\ 3 & 8 \end{vmatrix} - 3\begin{vmatrix} 2 & 1 \\ 2 & 8 \end{vmatrix} + 5\begin{vmatrix} 2 & 5 \\ 2 & 3 \end{vmatrix} \\ &= 1(5 \cdot 8 - 3 \cdot 1) - 3(2 \cdot 8 - 2 \cdot 1) + 5(2 \cdot 3 - 2 \cdot 5) = -25 \end{aligned}$$

In SciPy, this is computed as shown below

```
In []: A = array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
In []: det(A)
Out[]: -25.0
```

2.2 Solving Equations

2.2.1 Linear Equations

Solving linear systems of equations is straightforward using the scipy command `solve`. This command expects an input matrix and a right-hand-side vector.

The solution vector is then computed. An option for entering a symmetrix matrix is offered which can speed up the processing when applicable. As an example, suppose it is desired to solve the following simultaneous equations:

$$\begin{aligned} x + 3y + 5z &= 10 \qquad (1) \\ 2x + 5y + z &= 8 \qquad (2) \\ 2x + 3y + 8z &= 3 \qquad (3) \end{aligned}$$

We could find the solution vector using a matrix inverse:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} -232 \\ 129 \\ 19 \end{bmatrix} = \begin{bmatrix} -9.28 \\ 5.16 \\ 0.76 \end{bmatrix}$$

However, it is better to use the solve command which can be faster and more numerically stable. In this case it however gives the same answer.

```
In []: A = array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
In []: b = array([[10], [8], [3]])
In []: dot(inv(A), b)
Out[]:
array([[-9.28],
       [ 5.16],
       [ 0.76]])

In []: solve(A,b)
Out[]:
array([[-9.28],
       [ 5.16],
       [ 0.76]])
```

2.2.2 Polynomials

Solving a polynomial is straightforward in scipy using the `roots` command. It expects the coefficients of the polynomial in their decreasing order. For example, let's find the roots of $x^3 - 2x^2 - \frac{1}{2}x + 1$ are 2, $\sqrt{2}$ and $-\sqrt{2}$. This is easy to see.

$$\begin{aligned} x^3 - 2x^2 - \frac{1}{2}x + 1 &= 0 \\ x^2(x-2) - \frac{1}{2}(x-2) &= 0 \\ (x-2)(x^2 - \frac{1}{2}) &= 0 \\ (x-2)(x - \frac{1}{\sqrt{2}})(x + \frac{1}{\sqrt{2}}) &= 0 \end{aligned}$$

We do it in scipy as shown below:

```
In []: coeff = array([1, -2, -2, 4])
In []: roots(coeff)
```

2.2.3 Non-linear Equations

To find a root of a set of non-linear equations, the command `fsolve` is needed. For example, the following example finds the roots of the single-variable transcendental equation

$$x + 2\cos(x) = 0,$$

and the set of non-linear equations

$$x_0 \cos(x_1) = 4, \tag{4}$$
$$x_0 x_1 - x_1 = 5 \tag{5}$$

The results are $x = -1.0299$ and $x_0 = 6.5041$, $x_1 = 0.9084$.

```
In []: def func(x):
   ...:     return x + 2*cos(x)

In []: def func2(x):
  ...:     out = [x[0]*cos(x[1]) - 4]
  ...:     out.append(x[1]*x[0] - x[1] - 5)
  ...:     return out

In []: from scipy.optimize import fsolve
In []: x0 = fsolve(func, 0.3)
In []: print x0
-1.02986652932

In []: x02 = fsolve(func2, [1, 1])
In []: print x02
[ 6.50409711  0.90841421]
```

2.3 Integration

2.3.1 Quadrature

The function `quad` is provided to integrate a function of one variable between two points. The points can be $\pm\infty$ ($\pm$ `inf`) to indicate infinite limits. For example, suppose you wish to integrate the expression $e^{\sin(x)}$ in the interval $[0, 2\pi]$, i.e. $\int_0^{2\pi} e^{\sin(x)} dx$, it could be computed using

```
In []: def func(x):
   ...:     return exp(sin(x))

In []: from scipy.integrate import quad
In []: result = quad(func, 0, 2*pi)
In []: print result
(7.9549265210128457, 4.0521874164521979e-10)
```

2.3.2 ODE

We wish to solve an (a system of) Ordinary Differential Equation. For this purpose, we shall use `odeint`. As an illustration, let us solve the ODE

$$\frac{dy}{dt} = ky(L-y) \tag{6}$$

$$L = 25000,\ k = 0.00003,\ y(0) = 250$$

We solve it in scipy as shown below.

```
In []: from scipy.integrate import odeint
In []: def f(y, t):
  ...:     k, L = 0.00003, 25000
  ...:     return k*y*(L-y)
  ...:
In []: t = linspace(0, 12, 60)
In []: y0 = 250
In []: y = odeint(f, y0, t)
```

Note: To solve a system of ODEs, we need to change the function to return the right hand side of all the equations and the system and the pass the required number of initial conditions to the `odeint` function.

A Plotting using Pylab

The following piece of code, produces the plot in Figure 1 using `pylab`[2] in `ipython`[3][3]

```
In []: x = linspace(0, 2*pi, 50)
In []: plot(x, sin(x))
In []: title('Sine Curve between 0 and $\pi$')
In []: legend(['sin(x)'])
```

[3]start `ipython` with `-pylab` flag

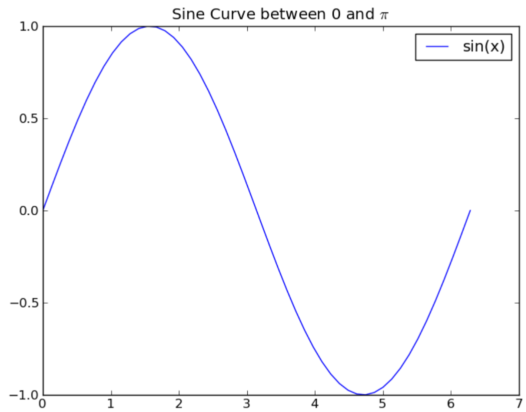

Figure 1: Sine curve

References

[1] Eric Jones and Travis Oliphant and Pearu Peterson and others, *SciPy: Open source scientific tools for Python*, 2001 – , `http://www.scipy.org/`

[2] John D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 90-95, May/June 2007, doi:10.1109/MCSE.2007.55

[3] Fernando Perez, Brian E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53.