L04_SciPy

December 15, 2015

1 SciPy

Kevin Stratford kevin@epcc.ed.ac.uk Emmanouil Farsarakis farsarakis@epcc.ed.ac.uk
Other course authors: Neelofer Banglawala Andy Turner Arno Proeme

www.archer.ac.uk support@archer.ac.uk

```
In [ ]: %matplotlib inline
        from scipy import optimize, special
        import numpy as np
        import matplotlib.pylab as plt
```

[SciPy] Overview I

- NumPy provides arrays, basic linear algebra, random number generation, and Fourier transforms
- SciPy builds on NumPy (e.g. by using arrays) and expands this with (additional) routines for:
- Linear Algebra and wrappers to LAPACK & BLAS (linalg)
- Numerical Integration (integrate)
- Interpolation (interpolate)
- Optimisation (optimize)
- Special functions (special)

[SciPy] Overview II

- Signal processing (signal)
- image processing (ndimage)
- Fourier transforms (fftpack)
- Statistical functions (stats)
- File IO e.g. read MATLAB files (io)
- Useful links
- http://docs.scipy.org/doc/scipy/reference/
- http://scipy-cookbook.readthedocs.org

[SciPy] Linear algebra I

- scipy.linalg is a superset of numpy functionality for linear algebra
- various factorisations, decompositions
- matrix exponentials, trigonometric functions
- particular matrix equations and special matrices
- low-level LAPACK and BLAS routines
- may be faster than numpy (will use BLAS, LAPACK)

[SciPy] Linear algebra II

- Routines also for sparse matrices
- storage formats
- iterative algorithms

[SciPy] Linear algebra : inverse matrix I

Let's find inverse of matrix A

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

which is

$$A^{-1} = \frac{1}{25} \begin{bmatrix} -37 & 9 & 22 \\ 14 & 2 & -9 \\ 4 & -3 & 1 \end{bmatrix} = \begin{bmatrix} -1.48 & 0.36 & 0.88 \\ -0.56 & 0.08 & -0.36 \\ 0.16 & -0.12 & 0.04 \end{bmatrix}$$

Note that $I = AA^{-1}$, where I is the identity matrix $I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

[SciPy] Linear algebra : inverse matrix II

```
In [ ]: # find inverse of a matrix
        import numpy as np
        from scipy import linalg
        A = np.array( [[1,3,5], [2,5,1], [2,3,8]] )

        invA = linalg.inv(A)
        print invA;

        # check inverse gives identity
        identity = A.dot(linalg.inv(A))

        # We round each element to zero decimal places
        np.abs(np.around(identity, 0))
```

[SciPy] Integration

- Can solve Ordinary Differential Equations (ODEs) with initial conditions
- Routines for numerical integration – single, double and triple integrals

[SciPy] Integration : coupled masses I

Solve Ordinary Differential Equations (ODEs) with initial conditions, for example normal mode motion of spring-coupled masses.

Two masses, **m**, are coupled with springs, each with spring constant **k**. Displacement $\mathbf{x_i}$ of each mass is measured from its position of rest.

[SciPy] Integration : coupled masses II

Assuming small displacements, we can write a system of second-order differential equations to describe the motion of these masses according to Newton's 2nd law of motion, $\mathbf{F} = \mathbf{m}\,\mathbf{a}$:

$m\,\ddot{x}_1 = -k\,x_1 + k\,(x_2 - x_1)$

$m\,\ddot{x}_2 = -k\,(x_2 - x_1) - k\,x_2$

[SciPy] Integration : coupled masses III

To use odeint, we rewrite these as 4 first-order differential equations in terms of the masses velocities, $\mathbf{v_i}$:

$v_1 = \dot{x_1}, \quad \dot{v_1} = -\frac{2\,k}{m}\,x_1 + \frac{k}{m}(x_2)$

$v_2 = \dot{x_2}, \quad \dot{v_2} = -\frac{2\,k}{m}\,x_2 + \frac{k}{m}(x_1)$

[SciPy] Integration : coupled masses : IV

Exact time-dependent solution for displacement of masses (assuming zero initial velocities)

Notes * explain what np.vectorize does (doesn't necessarily make code run faster, just convenient way to vectorize your function) * only need `%matplotlib inline` command for notebooks

In [ ]:
```
%matplotlib inline
import numpy as np;
```

In [ ]:
```
def x1_t(t,x1,x2,k,m): # exact solution for mass 1 at time t
    w=np.sqrt(k/m);    # initial velocity assumed zero
    a1=(x1+x2)/2.0;
    a2=(x1-x2)/2.0;
    return a1*np.cos(w*t) + a2*np.cos(np.sqrt(3)*w*t);

def x2_t(t,x1,x2,k,m): # exact solution for mass 2 at time t
    w=np.sqrt(k/m);    # initial velocity assumed zero
    a1=(x1+x2)/2.0;
    a2=(x1-x2)/2.0;
    return a1*np.cos(w*t) - a2*np.cos(np.sqrt(3)*w*t);
```

[SciPy] Integration : coupled masses : V

Set up a function to give to the ODE solver (LSODA explicit solver from FORTRAN library odepack)

In [ ]:
```
# "vectorize" solutions to act on an array of times
x1_sol = np.vectorize(x1_t);
x2_sol = np.vectorize(x2_t);
```

In [ ]:
```
def vectorfield(w, t, p):
    """Defines differential equations for the coupled masses
        w :  vector of the state variables: w = [x1,v1,x2,v2]
        t :  time
        p :  vector of the parameters: p = [m,k] """
    x1, v1, x2, v2 = w;
    m, k = p;

    # Create f = (x1',y1',x2',y2'):
    f = [v1, (-k * x1 + k * (x2 - x1)) / m,
         v2, (-k * x2 - k * (x2 - x1)) / m];
    return f;
```

[SciPy] Integration : coupled masses VI

Use odeint to numerically solve ODEs with initial conditions

In [ ]:
```
# Use ODEINT to solve ODES defined by vectorfield
from scipy.integrate import odeint;
```

```
In [ ]: # Parameters and initial values
        m = 1.0; k = 1.0;      # mass m, spring constant k
        x01 = 0.5; x02 = 0.0; # Initial displacements
        v01 = 0.0; v02 = 0.0; # Initial velocities : LEAVE AS ZERO

In [ ]: # ODE solver parameters
        abserr = 1.0e-8; relerr = 1.0e-6;
        stoptime = 10.0; numpoints = 250;
```

[SciPy] Integration : coupled masses VI
Use odeint to numerically solve ODEs with initial conditions

```
In [ ]: # Create time samples for the output of the ODE solver
        t = np.linspace(0, stoptime, numpoints);

In [ ]: # Pack up the parameters and initial conditions as lists/arrays:
        p =  [m, k];  w0 = [x01, v01, x02, v02];

        # Call the ODE solver. Note: args is a tuple
        wsol = odeint(vectorfield, w0, t, args=(p,), atol=abserr,
                      rtol=relerr);

In [ ]: # Print and save the solution
        with open('coupled_masses.dat', 'w') as f:
            for t1, w1 in zip(t, wsol):
                print >> f, t1, w1[0], w1[1], w1[2], w1[3]
```

[SciPy] Integration : coupled masses VII
Plot exact solutions against saved numerical solutions

```
In [ ]: import numpy as np
        a=np.array([[0,1,2],[3,4,5]]); b=np.array([[-5,-6,-7],[-8,-9,-10]])
        print np.dstack((a,b)).shape
        print a.shape

In [ ]: # import modules for plotting
        import matplotlib.pyplot as plt;
        from matplotlib.font_manager import FontProperties;

        # get saved values from saved file
        t, x1, v1, x2, v2 = np.loadtxt('coupled_masses.dat', unpack=True);

        # contd...
```

[SciPy] Integration : coupled masses VIII
Plot exact solutions against saved numerical solutions

```
In [ ]: # figure proporties
        plt.figure(1, figsize=(10, 3.5)); plt.xlabel('t');
        plt.ylabel('x'); plt.grid(True); plt.hold(True);

        # plot exact solutions
        time=np.linspace(0,stoptime,50);

        plt.plot(time, x1_sol(time,x01,x02,k,m), 'r*', linewidth=1);
        plt.plot(time, x2_sol(time,x01,x02,k,m), 'mo', linewidth=1);
```

```
    # plot numerical solutions
    plt.plot(t, x1, 'b-', linewidth=1); plt.plot(t, x2, 'g-', linewidth=1);

    plt.legend(('$x_{1,sol}$', '$x_{2,sol}$', '$x_{1,num}$',
            '$x_{2,num}$'),prop=FontProperties(size=12));
    plt.title('Mass Displacements for the\nCoupled Spring-Mass System');
    plt.savefig('coupled_masses.png', dpi=100); # save figure
```

[SciPy] Other packages

- Pandas : R-like statistical analysis of numerical tables and time series
- SymPy : library for symbolic computing
- scikit-image : advanced image processing
- Sage : open source replacement for Mathematica / Maple / Matlab (built using Python)

[SciPy] Summary

- Only just skimmed the surface of what SciPy has to offer
- Close integration with NumPy and the use of Matplotlib makes it easy to do scientific computation
- Next session: Interfacing with Fortran/C