SciPy

Neelofer Banglawala nbanglaw@epcc.ed.ac.uk
Kevin Stratford kevin@epcc.ed.ac.uk

Original course authors:
Andy Turner
Arno Proeme

Reusing this material

www.archer.ac.uk

support@archer.ac.uk

[SciPy] Overview

- 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:
  - Numerical Integration (*integrate*)
  - Interpolation (*interpolate*)
  - Optimisation (*optimize*)
  - Special functions (*special*)
  - Linear Algebra and wrappers to LAPACK & BLAS (*linalg*)
  - Signal processing (*signal*)
  - Image Processing (*ndimage*)
  - Fourier transforms (*fftpack*)
  - Statistical functions (*stats*)
  - Spatial data structures and algorithms (*spatial*)
  - File IO e.g. read MATLAB files (*io*)

[SciPy] Useful links

- Note: no PDE solvers (though other packages exist)
- Documentation:
  - http://docs.scipy.org/doc/scipy/reference/tutorial/
  - http://docs.scipy.org/doc/scipy/reference/
  - http://scipy-cookbook.readthedocs.org

[SciPy] Integration

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

[SciPy] Integration : find π I

Calculate π using the double integral for the area of a circle with radius a:

$$\int_{-a}^{a} \int_{-\sqrt{(a^2-x^2)}}^{\sqrt{(a^2-x^2)}} 1 \, dx \, dy = \pi a^2$$

In [ ]:
```
# numerically solve the integral using dblquad
import numpy as np;
from scipy.integrate import dblquad;

def integrand(y,x): # integrand
    return 1;

# integral for the area of a circle with radius r
def integrate_to_pi(r):
    (area,err) = dblquad(integrand, -1*r,r,
                         lambda x: -1*np.sqrt(r*r-x*x),
                         lambda x: np.sqrt(r*r-x*x)  );
    return area/(r*r) ;
```

[SciPy] Integration : find π II

Calculate the result and compare with *numpy.pi*

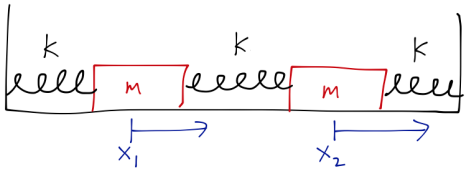
In [ ]:
```
# result!
print integrate_to_pi(50.0);

# compare with numpy pi
np.pi - integrate_to_pi(50.0);

# see timing routine from numpy lecture
# %timeit integrate_to_pi(50.0)
```

In [ ]:
```
# Ex: calculate the double integral of f(x, y) = y * sin (x)
# for x = [0, pi/2], y = [0, 1]
#
# ANSWER: 1/2
```

[SciPy] Integration : coupled masses I

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

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



$$m\,\ddot{x}_1 = -k\,x_1 + k\,(x_2 - x_1)$$
$$m\,\ddot{x}_2 = -k\,(x_2 - x_1) - k\,x_2$$

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

$$v_1 = \dot{x_1}$$
$$\dot{v_1} = -\tfrac{2\,k}{m}\,x_1 + \tfrac{k}{m}(x_2)$$
$$v_2 = \dot{x_2}$$
$$\dot{v_2} = -\tfrac{2\,k}{m}\,x_2 + \tfrac{k}{m}(x_1)$$

[SciPy] Integration : coupled masses II

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

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);
```

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

[SciPy] Integration : coupled masses III

Set up a function to give to the ODE solver

In [ ]:
```
def vectorfield(w, t, p):
    """Defines the differential equations for the coupled
    spring-mass system. Arguments:
        w :  vector of the state variables: w = [x1,y1,x2,y2]
        t :  time
        p :  vector of the parameters: p = [m,k]
    """
    x1, y1, x2, y2 = w;
    m, k = p;

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

[SciPy] Integration : coupled masses IV

Use *odeint* to numerically solve ODEs with initial conditions

In [ ]:
```
# Use ODEINT to solve differential equations
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
y01 = 0.0; y02 = 0.0; # Initial velocities : LEAVE AS ZERO
```

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

In [ ]:
```
# Create time samples for the output of the ODE solver
t = [stoptime * float(i) / (numpoints - 1)
     for i in range(numpoints)];

# contd...
```

[SciPy] Integration : coupled masses V

Use *odeint* to numerically solve ODEs with initial conditions

In [ ]:
```
# ... contd

# Pack up the parameters and initial conditions as lists/arrays:
p =  [m, k];  w0 = [x01, y01, x02, y02];

# 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 VI

Plot exact solutions against saved numerical solutions

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

# get saved values from saved file
t, x1, y1, x2, y2 = np.loadtxt('coupled_masses.dat', unpack=True);

# contd...
```

[SciPy] Integration : coupled masses VII

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] Special functions

- SciPy contains huge set of special functions
  - Bessel functions
  - Legendre functions
  - Gamma functions
  - Bessel functions
  - Airy functions

We will see special functions used in the following sections.

[SciPy] Special functions : drumhead I

- Let's plot the height of a drumhead (Bessel functions)

In [ ]:
```
# plot drumhead
from scipy import special;

def drumhead_height(n, k, distance, angle, t):
    kth_zero = special.jn_zeros(n, k)[-1];
    return (np.cos(t) * np.cos(n*angle) *
            special.jn(n, distance*kth_zero));
```

In [ ]:
```
theta = np.r_[0:2*np.pi:50j];
radius = np.r_[0:1:50j];
x = np.array([r * np.cos(theta) for r in radius]);
y = np.array([r * np.sin(theta) for r in radius]);
z = np.array([drumhead_height(1, 1, r, theta, 0.5) ;
              for r in radius]);
# contd...
```

[SciPy] Special functions : drumhead II

- Let's plot the height of a drumhead subject to normal modes

In [ ]:
```
# ...contd
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D;
from matplotlib import cm;

fig = plt.figure(figsize=(10, 6));
ax = Axes3D(fig);
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet);
ax.set_xlabel('X');
ax.set_ylabel('Y');
ax.set_zlabel('Z');
```

[SciPy] Optimisation

- Several classical optimisation algorithms
  - Quasi-Newton type optimisations
  - Least squares fitting
  - Simulated annealing
  - General purpose root finding

[SciPy] Optimisation : maxima I

Let's locate a few maxima of a Bessel function.

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
from scipy import optimize, special;
x = np.arange(0,10,0.01);

for k in np.arange(0.5,5.5):
     y = special.jv(k,x); # Bessel function
     plt.plot(x,y);
     # flip Bessel function about x-axis, turn maxima into minima
     f = lambda x: -1*special.jv(k,x);
     # find minimum between 4 and 10
     x_max = optimize.fminbound(f,0,6);
    plt.plot([x_max], [special.jv(k,x_max)],'ro') ;
     plt.title('Different Bessel functions & their local maxima');
```

[SciPy] Optimisation : maxima II

Time for an exercise! Find maxima of the Airy function, **Ai(x)**

In [ ]:
```
# Ex: find ** all ** maxima of Airy function in range x = [-8, 2]
x = np.arange(-8,2,0.01); # x range
ai =  special.airy(x)[0]; # Airy function, see online docs
```

In [ ]:
```
# What does the Airy function look like?
plt.figure(figsize=(8,4)); plt.plot(x,ai);
```

[SciPy] Optimisation : maxima III

In [ ]:
```
# find each maximum (given you can plot the function)
f = lambda x: -special.airy(x)[0]; # use negated of airy function

plt.plot(x,ai);
x_max = optimize.fminbound(f,-8,-6);
plt.plot([x_max], [special.airy(x_max)[0]],'ro');
x_max = optimize.fminbound(f,-6,-3);
plt.plot([x_max], [special.airy(x_max)[0]],'ro');
x_max =  optimize.fminbound(f,-3,2);
plt.plot([x_max], [special.airy(x_max)[0]],'ro');
plt.title('Different Airy functions and their local maxima');
```

[SciPy] Optimisation : maxima IV

In [ ]:
```
# ... contd
# But what if you don't know what function looks like?
# Could use a fixed 'window' within which to search
# BUT this could pick up a false maximum
minw = -8; maxw = -6; # min, max values for fixed "window"
plt.plot(x,ai);
for w in np.arange(4):
    x_max = optimize.fminbound(f,minw,maxw); # find minima / maxima
    plt.plot([x_max], [special.airy(x_max)[0]],'ro');
    minw+=2; maxw +=2;

plt.title('Different Airy functions and their local maxima');
```

In [ ]:
```
# Ex ++: how could you check you have found a genuine
# maximum or minimum?
```

[SciPy] Optimisation : least-squares fit I

Let's use *leastsq* to fit some measured data, $\{x_i,\ y_i\}$, to a function :

$$yi\ =\ A\ \sin(2\pi k x_i\ +\ \theta)$$

where the parameters A, k, and θ are unknown. The residual vector, that will be squared and summed by `leastsq` to fit the data, is:

$$e_i\ =||\ y_i\ -\ A\sin(2\pi k x_i + \theta)\ ||$$

By defining a function to compute the residuals, e_i, and, selecting appropriate starting values, `leastsq` can be used to find the best-fit parameters $\hat{A}$, $\hat{k}$, $\hat{\theta}$.

[SciPy] Optimisation : least-squares fit II

Create a sample of the true function and the "measured" (noisy) data. Define the residual function and initial values.

In [ ]:
```
# set up true function and "measured" data
x = np.arange(0, 6e-2, 6e-2 / 30);
A, k, theta = 10, 1.0 / 3e-2, np.pi / 6;
y_true = A * np.sin(2 * np.pi * k * x + theta);
y_meas = y_true + 2*np.random.randn(len(x));
```

In [ ]:
```
# residual function, e_i
def residuals(p, y, x):
    A, k, theta = p;
    err = y - A * np.sin(2 * np.pi * k * x + theta);
    return err;
```

[SciPy] Optimisation : least-squares fit III

In [ ]:
```
# for easy evaluation of model function parameters [A, K, theta]
def peval(x, p):
    return p[0] * np.sin(2 * np.pi * p[1] * x + p[2]);

# starting values of A, k and theta
p0 = [8, 1 / 2.3e-2, np.pi / 3];
print(np.array(p0));
```

[SciPy] Optimisation : least-squares fit IV

Do least squares fitting and plot results

In [ ]:
```
# do least squares fitting
from scipy.optimize import leastsq

plsq = leastsq(residuals, p0, args=(y_meas, x));
print(plsq[0]); print(np.array([A, k, theta]));
```

In [ ]:
```
# and plot the true function, measured (noisy) data
# and the model function with fitted parameters
plt.plot(x, peval(x, plsq[0]),x,y_meas,'o',x,y_true);

plt.title('Least-squares fit to noisy data');
plt.legend(['Fit', 'Noisy', 'True']);
```

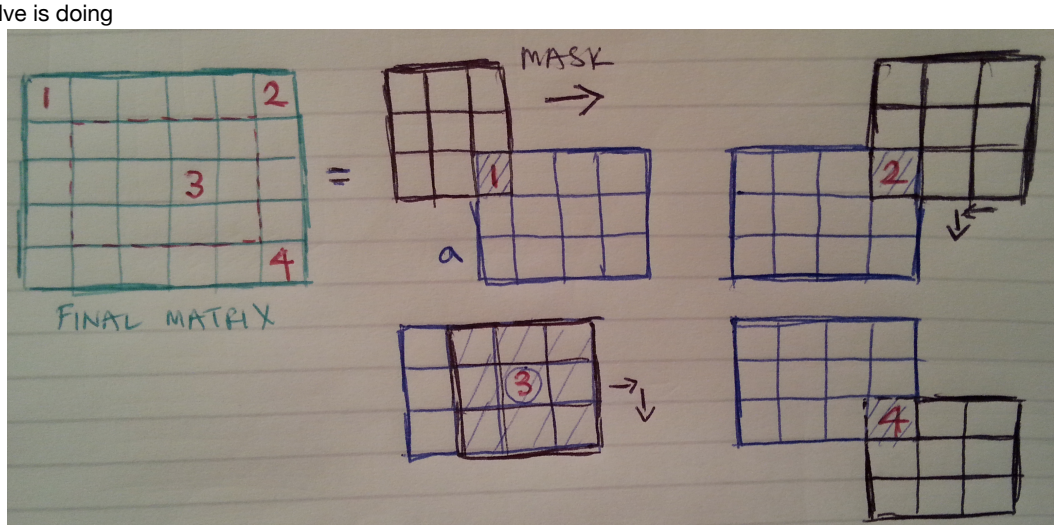
[SciPy] Convolutions functions I

In image and signal processing, the result of applying a filter (or mask) to an image (or signal) can be determined by a convolution between the two.

In [ ]:
```
# for example
import numpy as np
from scipy import signal
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],[9, 10, 11, 12]])
mask = np.array([[1,1,1],[1,0,1],[1,1,1]])
cS = signal.convolve(a, mask)
print a; print; print mask; print; print cS
```

[SciPy] Convolutions functions II

What signal.convolve is doing



[SciPy] Convolutions functions III

SciPy has another *convolve* function

In [ ]:
```
# Ex: Scipy has another convolve function, ndimage.convolve
# What does this produce, when used in 'constant' mode (below)?
from scipy import ndimage
cN = ndimage.convolve(a, mask, mode='constant')

# The default mode for ndimage.convolve is 'reflect'. Investigate.
```

In [ ]:
```
# ANSWER
#
# ndimage.convolve only produces the inner dotted matrix as shown
# in the green matrix in the figure i.e. convolution that fits
# the size of array a (not a true (commutative) convolution, like
# signal.convolve. With 'reflect' it replaces 'missing' values
# with adjacent values, e.g.
# cN[0,0]= 23= 5+6+2 (as constant mode)
#       +5+1 (from the left)  +1+1+2 (from the top)
# cN[0,3]= 41= 3+7+8 (same as constant mode)
#       +8+4 (from the right) +3+4+4 (from the top)
cN = ndimage.convolve(a, mask, mode='reflect')
print cN
```

[SciPy] Linear algebra

- Wider set of linear algebra operations than in Numpy
  - various decompositions (eigen, singular value)
  - matrix exponentials, trigonometric functions
  - particular matrix equations and special matrics
  - low-level LAPACK and BLAS routines
- 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 matrix
from scipy import linalg
A = np.array([[1,3,5],[2,5,1],[2,3,8]])

invA = linalg.inv(A) # the inveser
print invA; print;

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

# This really is the identity if we clean up the errors
np.abs(np.around(identity,0)) # (round up small errors)
```

[SciPy] Other packages

- Pandas
  - Offers R-like statistical analysis of numerical tables and time series
- SymPy
  - Python library for symbolic computing
- scikit-image
  - Advanced image processing
- scikit-learn
  - Package for machine learning
- Sage
  - Open source replacement for Mathematica / Maple / Matlab (built using Python)