An introduction to scientific programming with

python™

Session 3:

Scientific Python

Coursework

- A Python program relevant to your research
  - put course material into practice
  - opportunity to become familiar with Python
  - requirement to qualify for credits
- Your program should…
  - be written as an importable module (.py file)
  - do something meaningful: analyse real data or perform a simulation
  - use at least two user functions
  - use at least one of the modules introduced in Sessions 3 – 5
  - produce at least one informative plot
  - comprise >~ 50 lines of actual code (excluding comments and imports)
- Submit as source code (.py file) and pdf/png files of the plot(s)
  - via email (steven.bamford@nottingham.ac.uk)
  - by end of 13 December.

Recap of matplotlib

- User friendly, but powerful, plotting capabilities for python
- http://matplotlib.sourceforge.net/
- Once installed, to use type:

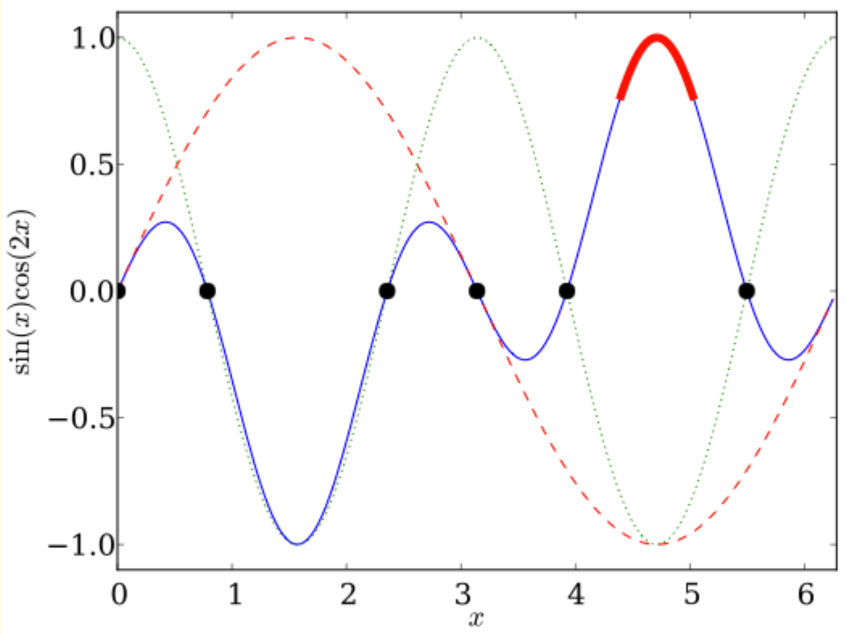
```
>>> import pylab                           # handy for interactive use
>>> from matplotlib import pyplot          # better for in scripts
```

- Settings can be customised by editing ~/.matplotlib/matplotlibrc
    - Set *backend* and default font, colours, layout, etc.
- Helpful website
    - many examples

```
>>> pyplot.ion()   # turn on interactive mode!
```

Plotting – matplotlib

```
>>> from numpy import sin, cos, pi
>>> x = numpy.arange(0, 2*pi, pi/100)
>>> y = sin(x)*cos(2*x)
>>> pyplot.plot(x, y)
>>> pyplot.plot(x, sin(x), '--r')
>>> pyplot.plot(x, cos(2*x),
                linestyle='dotted',
                color='green')
>>> thresh = y > 0.75
>>> pyplot.plot(x[thresh], y[thresh],
...             'r', linewidth=5)
>>> zeros = numpy.abs(y) < pi/200
>>> pyplot.plot(x[zeros], y[zeros],
                'ok', markersize=10)
>>> pyplot.xlabel(r'$x$')
>>> pyplot.ylabel(r'$\sin(x)\cos(2x)$')
>>> pyplot.axis([0, 2*pi, -1.1, 1.1])
>>> pyplot.savefig('wiggles.pdf')
```



Plotting – matplotlib

- Plots can be altered in an object oriented manner

For example,

```
>>> fig = pyplot.figure(1)
>>> ax = fig.axes[0]
>>> ax.xaxis.labelpad = 10
>>> pyplot.draw()
>>> l = ax.lines[2]
>>> l.set_linewidth(3)
>>> pyplot.draw()
>>> ax.xaxis.set_ticks((0, pi/2, pi, 3*pi/2, 2*pi))
>>> pyplot.draw()
>>> ax.xaxis.set_ticklabels(('0', r'$\frac{1}{2}\pi$', r'$\pi$',
                             r'$\frac{3}{2}\pi$', r'$2\pi$'))
>>> pyplot.draw()
>>> pyplot.subplots_adjust(bottom=0.25)
```

```
Shorthand to get current axes
>>> ax = pyplot.gca()
```

Plotting – matplotlib

Some useful functions:

- figure – create a new figure, or get an existing figure object
- plot – add line or points
- hist – create a histogram
- axis – set axis limits
- subplot – set to draw on subset of the canvas
- subplots_adjust – adjust the canvas margins

Some functions update the plot, others don't (for efficiency)

To update the plot display:

- draw() – draw plot and continue
- show() – blocks interpreter until window closed

- close(), close('all') – close figure windows

Exercises 2

1) Create an array x = [-3.00, -2.99, -2.98, …, 2.98, 2.99, 3.00]

2) Create a corresponding array for the Gaussian function $y = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$

3) Check the result is unit normalised: $\sum_i y_i \, \delta x = 1$

4) For convenience, put x and y together in a recarray and save to a file

5) Create a sample of one hundred Gaussian random numbers

6) Plot your Gaussian curve, x versus y, with axis labels

7) Add a histogram of the random numbers to your plot

8) Over-plot another histogram for a different sample and prettify
(try histtype='stepfilled' and 'step', and transparency, e.g., alpha=0.5)

Scientific Python

- So far…
  - **Session 1:**
    - core Python language and libraries
  - **Session 2:**
    - fast, multidimensional arrays
    - plotting tools
- **Session 3:**
  - libraries of fast, reliable scientific functions

Scientific Python (SciPy)

- Suite of numerical and scientific tools for Python
- http://scipy.org/
- http://docs.scipy.org/

Scipy subpackages

- 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-finding
- 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

```
# scipy submodules
# must be explicitly
# imported, e.g.,
import scipy.fftpack
# or
from scipy import stats
```

SciPy

Some simple examples:

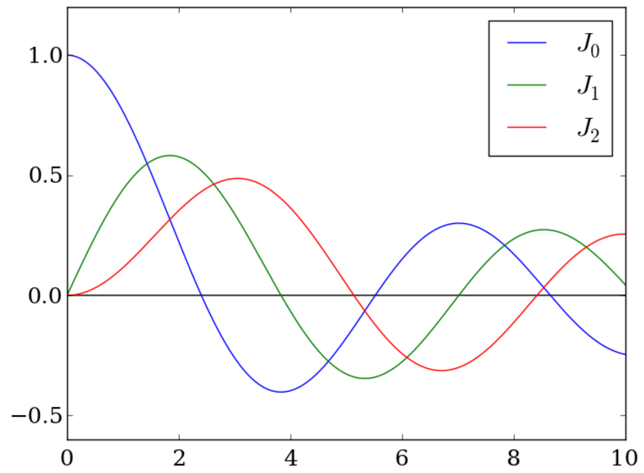
- Special functions (special)
- Root finding (optimize)
- Integration (integrate)
- Statistics (stats)
- Image processing (ndimage)
- Interpolation (interpolate)
- Optimisation (optimize)

Scipy – special functions

- Huge number of functions, including…
  - Bessel functions
  - Gamma functions
  - Fresnel integrals
  - Hypergeometric functions
  - Orthogonal polynomials

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

```
e.g., Bessel functions of order 1, 2, 3
>>> from scipy import special
>>> x = numpy.arange(0, 10.001, 0.01)
>>> for alpha in range(3):
...     y = special.jv(alpha, x)
...     pyplot.plot(x, y,
                 label=r'$J_%i$'%alpha)
>>> pyplot.hlines(0, 0, 10)
>>> pyplot.legend()
```



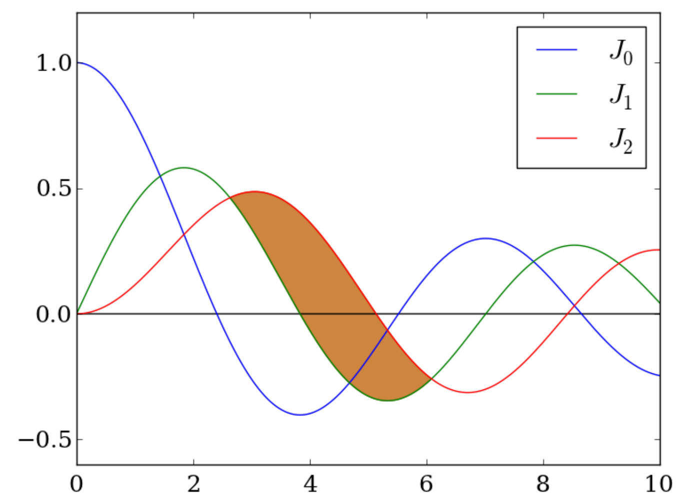
Scipy – root finding

- Accurate automatic root-finding using MINPACK

```
>>> from scipy.optimize import fsolve   # n-dimensional root finder
>>> from scipy.special import jv
```

Define a function to solve
First argument is variable (or array of variables) of interest

```
>>> def f(z, a1, a2):
...     return jv(a1, z) - jv(a2, z)
...

>>> fsolve(f, 2.5, args=(1, 2))
array([ 2.62987411])

>>> fsolve(f, 6, args=(1, 2))
array([ 6.08635978])
```

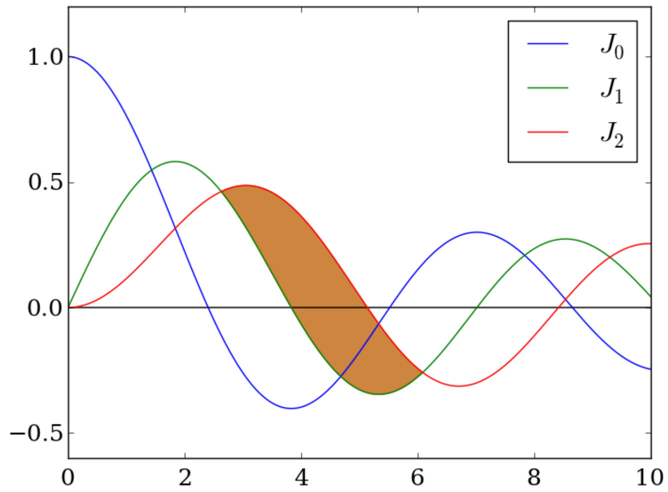



```
>>> pyplot.fill_between(x, special.jv(1, x), special.jv(2, x),
        where=((x > 2.630) & (x < 6.086)), color="peru")
```

Scipy – integration

- Accurate automatic integration using QUADPACK
  - including uncertainty estimate

```
>>> from scipy.integrate import quad   # one-dimensional integration
Using previous function (first argument is variable of interest)
>>> r = fsolve(f, (2.5, 6), args=(1, 2))
>>> print r
[ 2.62987411  6.08635978]
>>> quad(f, r[0], r[1], args=(1, 2))
(-0.98961158607157, 1.09868956829247e-14)
```



- Can specify limits at infinity

  `(-scipy.integrate.Inf, scipy.integrate.Inf)`

```
>>> quad(exp, -integrate.Inf, 0)
(1.0000000000000002, 5.842606742906004e-11)
```

Scipy – integration

- QUADPACK and MINPACK routines provide warning messages
- Extra details returned if parameter full_output=True

```
>>> quad(tan, 0, pi/2.0-0.0001)
(9.210340373641296, 2.051912874185855e-09)

>>> quad(tan, 0, pi/2.0)
Warning: Extremely bad integrand behavior occurs at some points of the
  integration interval.
(38.58895946215512, 8.443496712555953)

>>> quad(tan, 0, pi/2.0+0.0001)
Warning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
(6.896548923283743, 2.1725421039565056)
```

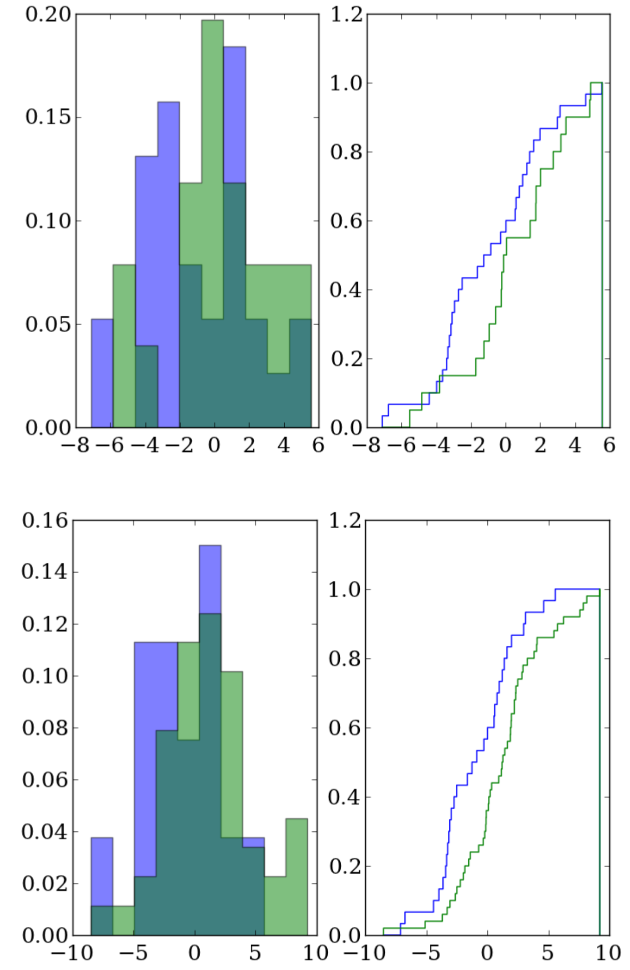
Scipy – statistics

- Probability distributions
  - including: norm, chi2, t, expon, poisson, binom, boltzmann, …
  - methods:
    - rvs – return array of random variates
    - pdf – probability density function
    - cdf – cumulative density function
    - ppf – percent point function
    - … and many more
- Statistical functions
  - including:
    - mean, median, skew, kurtosis, …
    - normaltest, probplot, …
    - pearsonr, spearmanr, wilcoxon, …
    - ttest_1samp, ttest_ind, ttest_rel, …
    - kstest, ks_2samp, …

```
>>> lambda = 10
>>> p = stats.poisson(lambda)

# P(n > 20)
>>> 1 - p.cdf(20)
0.0015882606618580573

# N: P(n < N) = 0.05, 0.95
>>> p.ppf((0.05, 0.95))
array([  5.,  15.])

# true 95% CI bounds on lambda
>>> stats.gamma.ppf((0.025, 0.975),
                    lambda+0.5, 1)
array([  6.14144889,  18.73943795])
```

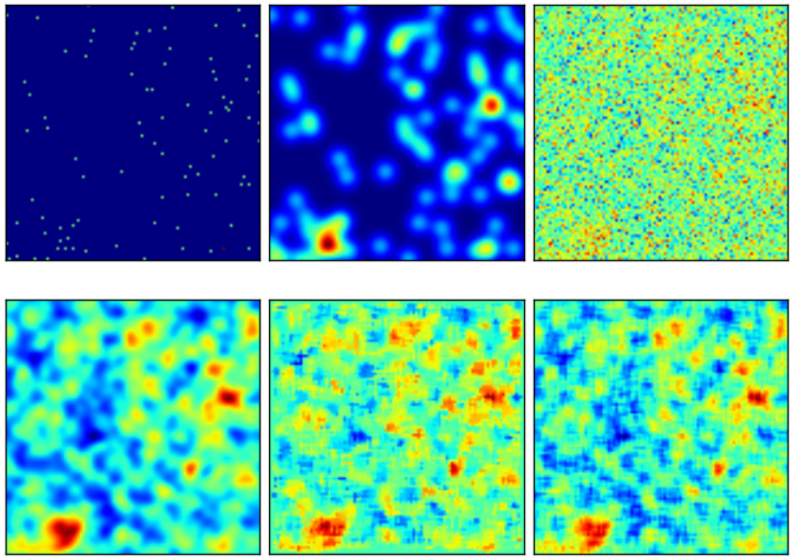
Scipy – statistics

```
>>> x = stats.norm.rvs(-1, 3, size=30)  # specify pdf parameters
>>> n = stats.norm(1, 3)  # create 'frozen' pdf
>>> y = n.rvs(20)
>>> z = n.rvs(50)
>>> p = pyplot.subplot(121)
>>> h = pyplot.hist((x, y), normed=True,
        histtype='stepfilled', alpha=0.5)
>>> p = pyplot.subplot(122)
>>> h = pyplot.hist((x, y), histtype='step',
        cumulative=True, normed=True, bins=1000)

>>> stats.ks_2samp(x, y)
(0.29999999999999999, 0.18992875018013033)
>>> stats.ttest_ind(x, y)
(-1.4888787966012809, 0.14306062943339182)

>>> stats.ks_2samp(x, z)
(0.31333333333333335, 0.039166429989206733)
>>> stats.ttest_ind(x, z)
(-2.7969511393118509, 0.0064942129302196124)

>>> stats.kstest(x, stats.norm(1, 3).cdf)
(0.31388990356819 28, 0.0039905619713858087)
```



Scipy – image processing

```
>>> x = stats.poisson.rvs(0.01, size=(100,100)).astype(numpy.float)
>>> y = ndimage.gaussian_filter(x, 3, mode='constant')
>>> z = y + stats.norm.rvs(0.0, 0.01, size=(100,100))
>>> zg = ndimage.gaussian_filter(z, 2, mode='constant')
>>> zm = ndimage.median_filter(z, 5, mode='constant')
>>> zu = ndimage.uniform_filter(z, 5, mode='constant')
>>> for i, img in enumerate([x, y, z, zg, zm, zu]):
...     ax = pyplot.subplot(2, 3, i)
...     pyplot.imshow(img, interpolation=None)
...     ax.yaxis.set_ticks([])
...     ax.yaxis.set_ticks([])
...
```



- Additional routines for filtering in 'signal' submodule

Aside – mgrid, ogrid

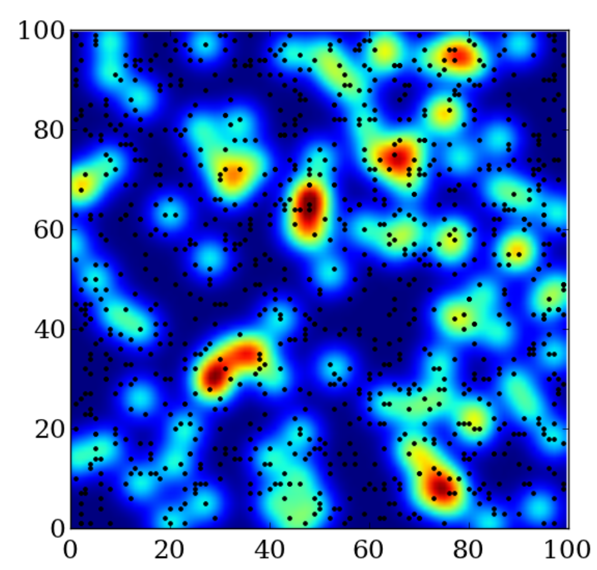
```
>>> numpy.mgrid[0:5,0:5]
array([[[0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1],
        [2, 2, 2, 2, 2],
        [3, 3, 3, 3, 3],
        [4, 4, 4, 4, 4]],

       [[0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4]]])


>>> numpy.mgrid[0:5:2,0:5:2]
array([[[0, 0, 0],
        [2, 2, 2],
        [4, 4, 4]],

       [[0, 2, 4],
        [0, 2, 4],
        [0, 2, 4]]])
```

```
>>> numpy.mgrid[0:2:5j,0:2:5j]
array([[[ 0. ,  0. ,  0. ,  0. ,  0. ],
        [ 0.5,  0.5,  0.5,  0.5,  0.5],
        [ 1. ,  1. ,  1. ,  1. ,  1. ],
        [ 1.5,  1.5,  1.5,  1.5,  1.5],
        [ 2. ,  2. ,  2. ,  2. ,  2. ]],

       [[ 0. ,  0.5,  1. ,  1.5,  2. ],
        [ 0. ,  0.5,  1. ,  1.5,  2. ],
        [ 0. ,  0.5,  1. ,  1.5,  2. ],
        [ 0. ,  0.5,  1. ,  1.5,  2. ],
        [ 0. ,  0.5,  1. ,  1.5,  2. ]]])


>>> numpy.ogrid[0:2:5j,0:2:5j]
[array([[ 0. ],
       [ 0.5],
       [ 1. ],
       [ 1.5],
       [ 2. ]]),
 array([[ 0. ,  0.5,  1. ,  1.5,  2. ]])]
```

Scipy – interpolation

- Several methods
  - most useful are: interp1d, griddata, splrep/splev, bisplrep/bisplev

```
Create samples
>>> p, q = numpy.random.random_integers(99, size=(2, 1000))
>>> sy = y[p,q]
>>> sz = z[p,g]
>>> grid_x, grid_y = numpy.mgrid[0:100, 0:100]
```




```
Plot
>>> vmin = stats.scoreatpercentile(y.ravel(), 0.01)
>>> vmax = stats.scoreatpercentile(y.ravel(), 99.9)
>>> pyplot.imshow(y, interpolation=None, vmin=vmin, vmax=vmax)
>>> pyplot.plot(p, q, '.k')
>>> pyplot.axis([0, 100, 0, 100])
```

Scipy – interpolation

Exact interpolation

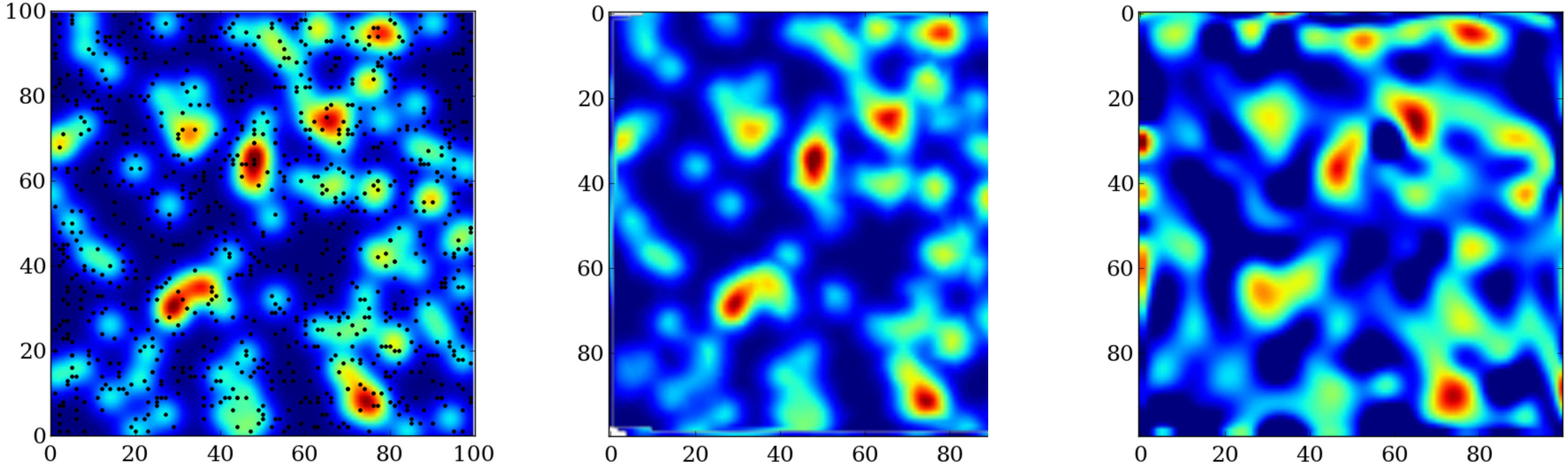
```
>>> gy = interpolate.griddata((p, q), sy, (grid_x, grid_y), method='cubic')
```

Spline fit to noisy sample

```
>>> bz = interpolate.bisplrep(p, q, sz, s=0.1, full_output=True)
>>> gz = interpolate.bisplev(numpy.arange(100), numpy.arange(100), bz[0])
```

Plot

```
>>> pyplot.imshow(gy[::-1,:], interpolation=None, vmin=vmin, vmax=vmax)
>>> pyplot.imshow(gz, interpolation=None, vmin=vmin, vmax=vmax)
```

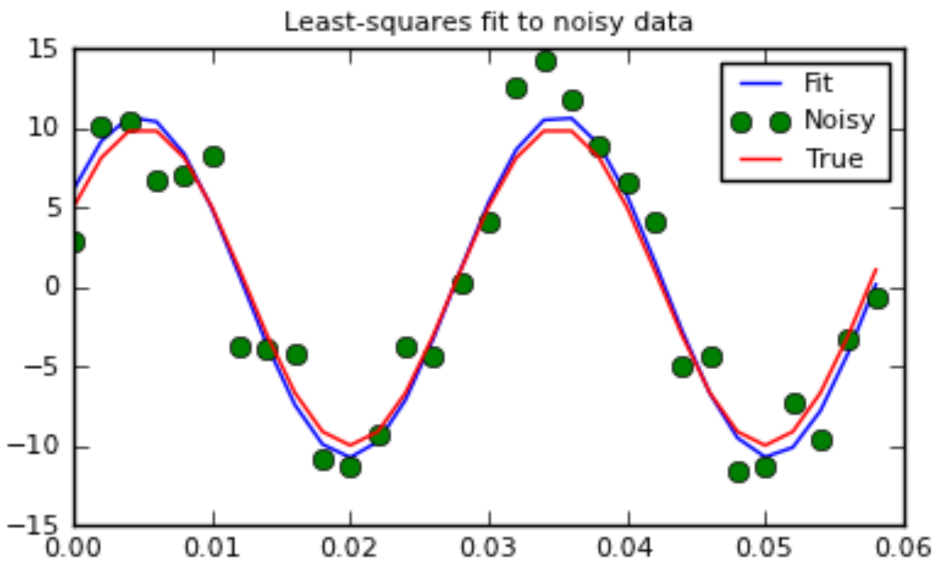


Scipy – optimisation

- Many methods
  - downhill and stochastic minimisation
  - least squares fitting

Details at http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

See the exercises for an example.



PyGSL

- Python wrappers of GNU Scientific Library functions
- PyGSL: http://pygsl.sourceforge.net/
- GSL: http://www.gnu.org/software/gsl/

- Incomplete documentation for Python functions, but almost all of GSL is wrapped, so refer to GSL documentation.
- Most functionality implemented in SciPy
  - Try SciPy first, if you can't find what you need try PyGSL
  - More comprehensive and sometimes more tested, but less 'Pythonic'
  - e.g. Monte Carlo integration

Other tools

- http://rpy.sourceforge.net/
- Wraps R – a statistics analysis language
  - many advanced stats capabilities but quite specialised

- http://www.sagemath.org/
- Python-based mathematics software *(see next session)*
  - replacement for Maple, Mathematica

Exercises 3

1) Plot and use fsolve to find the first root of the zeroth-order Bessel function of the second kind, i.e. x where $Y_0(x) = 0$.

2) Use quad to find the integral of $Y_0(x)$ between x=0 and the first root.

3) Write a function to accurately and efficiently calculate the integral of $Y_0(x)$ up to its *n*th root (remember to ensure fsolve has found a solution). Check for a few n up to n = 100, the integral should be converging to zero.

4) Use scipy.stats.norm.rvs to create 100 samples from a Normal distribution for some mean and sigma. Then use pyplot.hist to create a 10-bin histogram of the samples (see the return values). Convert the bin edges to values at the centre of each bin.

5) Create a function f((m, s), a, x, y) which returns the sum of the squared residuals between the values in y and a Gaussian with mean m, sigma s and amplitude a, evaluated at x.

6) Use function you created in (5) with scipy.optimize.fmin to fit a Gaussian to the histogram created in (4). Plot and compare with scipy.stats.norm.fit.