An introduction to scientific programming with

Session 4:
Python for specialists
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) [Tricky] Write a function to 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.minimize to fit a Gaussian to the histogram created in (4). Plot and compare with scipy.stats.norm.fit.
Session 3 exercise solutions

[link to online notebook]
Python for observers

• Python is great for astronomy
• Support from STSci and other observatories
• Provides friendly and powerful interfaces to standard tools
  • PyFITS – use FITS files
  • PyRAF – access all of IRAF tools
  • PyAST – Starlink WCS library
  • Atpy – Astronomical Tables in Python
• lots of other resources, but not very homogeneous (in past)
Python for observers

astropy
A Community Python Library for Astronomy

- recent and ongoing effort to create a uniform package
  - feature-rich, and rapidly becoming more so
  - strong community support
  - worth supporting and contributing too
    - affiliated packages

Good sites for further information:

http://www.astropy.org
http://www.astrobetter.com
astropy

A Community Python Library for Astronomy

- Astronomical constants, units, times and dates
- Astronomical coordinate systems
- Cosmology calculations
- Virtual Observatory integration
- Astronomy specific additions to numpy/scipy tools:
  - n-dimensional datasets, tables
  - model fitting, convolution, filtering, statistics
- Undergoing rapid development – but mostly stable (v1.x)
- Open source, on GitHub
• Catalogue matching
  • understands astronomical coordinates
  • fast (uses, and stores, KD tree)
  • one-to-one, one-to-many, separations, etc.

```python
from astropy.coordinates import SkyCoord, match_coordinates_sky
catalogcoord = SkyCoord(ra=ra_list, dec=dec_list)
matchcoord = SkyCoord(ra=ra, dec=dec, frame='FK4')

idx, d2d, d3d = match_coordinates_sky(matchcoord, catalogcoord)
```
Matching catalogues

• Don’t use simple nested loops
  • inefficient, don’t handle edge cases, …

• Better to use:
  • astropy libraries
  • searchesorted
  • scipy set library methods
  • do it outside of Python (e.g., using TOPCAT or STILTS)
>>> from astropy.cosmology import WMAP9 as cosmo
>>> cosmo.H(0)
<Quantity 69.32 km / (Mpc s)>
>>> cosmo.comoving_distance([0.5, 1.0, 1.5])
<Quantity [ 1916.0694236 , 3363.07064333, 4451.74756242] Mpc>

>>> from astropy.cosmology import FlatLambdaCDM
>>> cosmo = FlatLambdaCDM(H0=70, Om0=0.3)
>>> cosmo
FlatLambdaCDM(H0=70 km / (Mpc s), Om0=0.3, Tcmb0=2.725 K, Neff=3.04, m_nu=[ 0.  0.  0.] eV)
astropy

A Community Python Library for Astronomy

• Tables
  • Read FITS, ASCII, and more
  • Nice interface, similar to numpy ndarray/recarray
  • Fast, powerful, easy to use, well documented

```python
>>> import astropy.table as tab
>>> Table = tab.Table

>>> data = Table.read('mycatalogue.fits')

>>> print(data)  # print abridged table to screen

>>> data  # even nicer in IPython notebook
```
Astronomical image reduction environment

• Several decades of history and development
• Many, many powerful tasks
• Still a widely used standard tool
• Reduction packages for new instruments are usually written as standalone software or as part of other environments
• If you need it, you will probably know
• A working knowledge is useful for any observational astronomer
STScI Python

- Astronomy software provided by Space Telescope Science Institute
  - **PyRAF** – Python interface to IRAF
  - DrizzlePac
  - pysynphot
  - Numdisplay
  - PyFITS (now in astropy)

- [http://www.stsci.edu/institute/software_hardware/pyraf/stsci_python](http://www.stsci.edu/institute/software_hardware/pyraf/stsci_python)

- STScI also provides STSDAS, TABLES and HST reduction packages for IRAF
Installing IRAF/PyRAF

IRAF
• [http://iraf.noao.edu](http://iraf.noao.edu)
• Fairly straightforward installers for Linux and OS X
• Or ask your sysadmin (Phil Parry in Nottingham) to do it

STScI Python
• Available via pip install stsci.distutils; pip install stscipython

Ureka (all-in-on installer for OS X and Linux)
• Simple, updatable installation of IRAF, Python, STScI Python, etc.
• [http://ssb.stsci.edu/ureka/](http://ssb.stsci.edu/ureka/)
Handling FITS files – PyFITS (astropy.io.fits)

- FITS – file format for storing imaging and table data
  - very common in astronomy, but can be generally used
  - self describing, metadata, efficient, standardised

- PyFITS tutorial-style manual:
  - http://www.stsci.edu/resources/software_hardware/pyfits

- Read, write and manipulate all aspects of FITS files
  - extensions
  - headers
  - images
  - tables

- Low-level interface for details

- High-level functions for quick and easy use
```python
>>> import astropy.io.fits as pyfits
>>> imgname = 'data/2MASS_NGC_0891_K.fits'
>>> img = pyfits.getdata(imgname)
>>> img
array([[  0.        ,   0.        ,   0.        , ..., -999.00860596,
          -999.00860596, -999.00860596],
         [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
          -999.00860596, -999.00860596],
         [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
          -999.00860596, -999.00860596],
         ..., 
         [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
          -999.00860596, -999.00860596],
         [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
          -999.00860596, -999.00860596],
         [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
          -999.00860596, -999.00860596]], dtype=float32)

>>> img.mean()
-8.6610549999999993
>>> img[img > -99].mean()
0.83546290095423026
>>> numpy.median(img)
0.078269213438034058
```
PyFITS – reading FITS images

>>> x = 348; y = 97
>>> delta = 5
>>> print img[y-delta:y+delta+1,
...           x-delta:x+delta+1].astype(numpy.int)

[[  1   1   1   1   1   0   0   0   1   0   -2]
 [  2   2   4   6   7   7   4   3   1   0   -1]
 [  1   4   11  24  40  40  21   7   2   0   0]
 [  1   6  23  62 107  50  13   2   0   0   0]
 [  2   7  33  91 158  46  15   3   0   0   0]
 [  3   7  27  74 115  53  12   2   0   0   0]
 [  2   4  12  32  54  51  24   5   1   0   0]
 [  1   1   2   7  12  12   5   0   0   0   0]
 [  0   0   0   1   2   2   1   0   0   1   0]
 [  0   0   0   1   0   0   0   0   0   0   0]
 [ -1   0   1   0   0   0   0   0   0   0   0]]

• row = y = first index
• column = x = second index
• numbering runs as normal (e.g. in ds9) BUT zero indexed!
PyFITS – reading FITS tables

```python
>>> tblname = 'data/N891PNdata.fits'
>>> d = pyfits.getdata(tblname)
>>> d.names
('x0', 'y0', 'rah', 'ram', 'ras', 'decl', 'decm', 'decs', 'wvl', 'vel',
 'vhel', 'dvel', 'dvel2', 'xL', 'yL', 'xR', 'yR', 'ID', 'radeg', 'decdeg',
 'x', 'y')

>>> d.x0
array([ 928.7199707 ,  532.61999512,  968.14001465,  519.38000488,
        1383.18994141,  1888.26000977,  1516.2199707 ], dtype=float32)

>>> d.field('x0')  # case-insensitive
array([ 928.7199707 ,  532.61999512,  968.14001465,  519.38000488,
        1383.18994141,  1888.26000977,  1516.2199707 ], dtype=float32)

>>> select = d.x0 < 200
>>> dsel = d[select]  # can select rows all together
>>> print dsel.x0
[ 183.05000305 165.55000305 138.47999573 158.02999878 140.96000671
 192.58000183 157.02999878 160.1499939 161.1000061 136.58999634
 175.19000244]
```

Very useful: `pyfits.info()`
PyFITS – reading FITS headers

```python
>>> h = pyfits.getheader(imgname)
>>> print h
SIMPLE = T
BITPIX = -32
NAXIS = 2
NAXIS1 = 1000
NAXIS2 = 1200
BLOCKED = T / TAPE MAY BE BLOCKED IN MULTIPLES OF 2880
EXTEND = T / TAPE MAY HAVE STANDARD FITS EXTENSIONS
BSCALE = 1.
BZERO = 0.
ORIGIN = '2MASS / 2MASS Survey Camera
CTYPE1 = 'RA---SIN'
CTYPE2 = 'DEC--SIN'
CRPIX1 = 500.5
CRPIX2 = 600.5
CRVAL1 = 35.63922882
CRVAL2 = 42.34915161
CDELT1 = -0.0002777777845
CDELT2 = 0.0002777777845
CROTA2 = 0.
EQUINOX = 2000.
KMAGZP = 20.07760048 / V3 Photometric zero point calibration
COMMENT1= 'CAL updated by T.H. Jarrett, IPAC/Caltech'
SIGMA = 1.059334397 / Background Residual RMS noise (dn)
COMMENT1= '2MASS mosaic image'
COMMENT2= 'created by T.H. Jarrett, IPAC/Caltech'
>>> h['KMAGZP']
20.077600480000001
# Use h.items() to iterate through all header entries
```
```python
>>> newimg = sqrt((sky+img)/gain + rd_noise**2) * gain
>>> newimg[(sky+img) < 0.0] = 1e10

>>> hdr = h.copy()  # copy header from original image
>>> hdr.add_comment('Calculated noise image')

>>> filename = 'sigma.fits'

>>> pyfits.writeto(filename, newimg, hdr)  # create new file
>>> pyfits.append(imgname, newimg, hdr)  # add a new FITS extension
>>> pyfits.update(filename, newimg, hdr, ext)  # update a file

# specifying a header is optional,
# if omitted automatically adds minimum header
```
>>> import astropy.io.fits as pyfits
>>> import numpy as np

>>> # create data
>>> a1 = numpy.array(['NGC1001', 'NGC1002', 'NGC1003'])
>>> a2 = numpy.array([11.1, 12.3, 15.2])

>>> # make list of pyfits Columns
>>> cols = []
>>> cols.append(pyfits.Column(name='target', format='20A', array=a1))
>>> cols.append(pyfits.Column(name='V_mag', format='E', array=a2))

>>> # create HDU and write to file
>>> tbhdu=pyfits.new_table(cols)
>>> tbhdu.writeto('table.fits')

# these examples are for a simple FITS file containing just one
# table or image but with a couple more steps can create a file
# with any combination of extensions (see the PyFITS manual online)
```python
table = f[1]  # data extension number 1 (can also use names)

d = f[1].data  # data, same as returned by pyfits.getdata()

h = f[1].header  # header, same returned by pyfits.getheader()

# make any changes
f.writeto(othertblname)  # writes (with changes) to a new file

f = pyfits.open(tblname, mode='update')  # to change same file

# make any changes
f.flush()  # writes changes back to file
f.close()  # writes changes and closes file
```
PyFITS – memory mapping

• Useful if you only need to access a small region of an image
• Only reads elements from disk as accessed, not whole image

```python
>>> p = pyfits.open('gal.fits')
>>> d = p[0].data  # wait... data now in memory as a numpy array
>>> p = pyfits.open('gal.fits', memmap=True)
>>> d = p[0].data  # data still on disk, not in memory
>>> type(d)
<class 'numpy.core.memmap.memmap'>
>>> x = d[10:12, 10:12]  # only small amount of data in memory
>>> x
memmap([[ 2.92147326,  0.73809952],
         [-16.27580261, -13.62474442]], dtype=\n```

• Only works for files up to ~2Gb (due to limit on Python object size)
PyRAF – scripting IRAF with Python

• Command line to replace cl, allows most normal IRAF commands and Python at same prompt

• Can use IRAF tasks in Python scripts instead of having to create awkward cl scripts (or worse SPP)
PyRAF – command line

- Command and filename completion
- Edit line and access history easily (like ecl or bash)
- Use just as friendlier cl prompt or use Python whenever you want
- Transfer data between IRAF tasks and Python
- Use brackets for tasks when you want it to behave like Python

```python
--> imstat 2MASS_NGC_0891_K.fits
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX

--> fname = "data/2MASS_NGC_0891_K.fits"
--> imstat fname
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
Error reading image fname ...

--> imstat(fname)
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
```
PyRAF – command line

• Many IRAF tasks create output images or tables on disk, or just print to screen, so don't need to pass information back (see later for this)

```plaintext
stsdas  # note can't unload packages
improject(sky_file_2D, sky_file_1D, projaxis=2, average='no')
imcalc(sky_file_1D, sky_file_1D, 'im1%f'apwidthratio)
# calculate effective gain and ron due to combined images
gain = 1.91; ron = 5.41
gain_eff = gain * ncombine
ron_eff = ron * sqrt(ncombine)
# sig = sci + sky
imcalc('%s,%s%(sci_file, sky_file_1D), sig_file, 'im1+im2')
# sig = sqrt(sig * gain + ron**2) / gain
equation = 'sqrt(max(im1,0.0)/%(g)8.5f + %(r2)8.5f/%(g2)8.5f)'
equation = equation%{'g': gain_eff, 'r2': ron_eff**2,
                     'g2': gain_eff**2}
imcalc(sig_file, sig_file, equation)
```
PyRAF – scripting

- Use IRAF tasks in Python scripts
- Just import iraf object from pyraf module

```python
from pyraf import iraf
from glob import glob

images = glob('*sci.fits')

for img in images:
    iraf.imstat(img)
    newimg = img.replace('sci', 'sig')
    iraf.imcalc(img, newimg, 'sqrt(im1)')
```
• Can specify 'default' task parameters in neat fashion, instead of having to include on every call of a task

```python
iraf.imstat.nclip = 3
iraf.imstat.lsigma = 5
iraf.imstat.usigma = 5

# now every time I use imstat it uses sigma clipping
iraf.imstat(im1)
iraf.imstat(im2)
iraf.imstat(im3)

# can revert to task defaults by unlearning
iraf.unlearn('imstat')  # note task name is a string
```
PyRAF – scripting

• Useful to make shortcuts

```python
# shortcut for a long task name
crrej = iraf.stsdas.hst_calib.wfpc.crrej
crrej.mask = 'mymask.fits'
crrej.sigma = 5

crrej(in1, out1)
crrej(in2, out2)
crrej(in3, out3)
```
PyRAF – input and output from tasks

- IRAF outputs lots of useful data to screen - how to get at it?
- Some tasks need user input - would be nice to automate these
- PyRAF defines extra parameters Stdout and Stdin for all tasks
  - Stdout can be a filename, file object, or 1 (one) to direct to a list
  - Stdin can be a filename, file object, or string

```python
for img in images:
    # get a list with a string for each line of output
    output = iraf.imstat(img, format='no', fields='midpt', Stdout=1)
    # output can then be parsed as necessary to get at info
    med = float(output[0])
    newimg = img.replace('sci', 'sub')
    iraf.imcalc(img, newimg, 'im1 - %f' %med)
```
Python for theorists

- [Link to SageMath website](http://www.sagemath.org/)
- Python-based mathematics software
  - replacement for Maple, Mathematica
  - runs as a web application
  - Private and collaborative workbooks

Examples:

```python
var('z')
f1(z)=-z*I  # recall that i, the sqrt of -1, is denoted by I in Sage
print f1(5-2*I)
f2(z)=conjugate(z)  # this is the reflection w.r.t. the x-axis
print f2(1), " ", f2(-1)
f3(z)=(cos(pi/4)+sin(pi/4)*I)*z # rotation by pi/4
print f3(1), " ", f3(1-I), f3(I-I)
```

```
3*I - 5
-1 
(1/2*I + 1/2)*sqrt(2)  -(I + 2)*sqrt(2)  (I - 2)*sqrt(2)
```
Python for theorists

- SymPy: http://sympy.org/
- Python library for symbolic mathematics
- Comprehensive documentation
  - with built-in live Sympy shell
  - http://docs.sympy.org
- Use online
  - http://live.sympy.org
SymPy – numbers

- Arbitrary precision
- Rationals and symbols for special constants and irrationals

```python
>>> from sympy import *
>>> a = Rational(1,2)  # create a Rational number
>>> a, a*2, a**2
(1/2, 1, 1/4)
>>> sqrt(8)            # propagates surds
2*2**(1/2)
>>> (exp(pi))**2      # special constants
exp(2*pi)
>>> exp(pi).evalf()   # explicitly request float representation
23.1406926327793
>>> oo > 99999        # infinity
True
```

Thanks to Fabian Pedregosa

http://scipy-lectures.github.com/advanced/sympy.html
SymPy – algebra

- Can define variables to be treated as symbols
- Expressions can be manipulated algebraically

```python
>>> x = Symbol('x')
>>> y = Symbol('y')
```  
```python
>>> x+y+x-y
2*x
>>> (x+y)**2
(x + y)**2
```  
```python
>>> expand((x+y)**3)
3*x*y**2 + 3*y*x**2 + x**3 + y**3
```  
```python
>>> simplify((x+x*y)/x)
1 + y
```  
```python
# define multiple symbols
>>> x, y, z = symbols('x,y,z')
```  
```python
# useful shortcut
>>> f = simplify('(x+y)**2')
```  
```python
# latex output!
>>> print latex(exp(x**2/2))
e^\{\frac{1}{2} x^{2}\}
```
• Limits, derivatives, Taylor expansions and integrals

```python
>>> limit(sin(x)/x, x, 0)

>>> diff(tan(x), x)
1 + tan(x)**2

>>> limit((tan(x+y)-tan(x))/y, y, 0)    # check using limit!
1 + tan(x)**2

>>> diff(sin(2*x), x, 3)                # higher order derivatives
-8*cos(2*x)

>>> series(1/cos(x), x, pi/2, 5)       # around x=pi/2 to 5th order
-1/x - x/6 - 7*x**3/360 + O(x**5)
```
SymPy – calculus

- Indefinite and definite integration

```python
>>> integrate(sin(x), x)
-cos(x)

>>> integrate(log(x), x)
-x + x*log(x)

>>> integrate(exp(-x**2)*erf(x), x)  # including special functions
pi**(1/2)*erf(x)**2/4

>>> integrate(sin(x), (x, 0, pi/2))  # definite integral
1

>>> integrate(exp(-x**2), (x, -oo, oo))  # improper integral
pi**(1/2)
```
SymPy – equation solving

- `solve(f, x)` returns the values of x which satisfy f(x) = 0
- f and x can be tuples \(\rightarrow\) simultaneous equations
- Can also factorise polynomials

```python
>>> solve(x**4 - 1, x)
[I, 1, -1, -I]
>>> solve(exp(x) + 1, x)
[pi*I]

>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{y: 1, x: -3}

>>> f = x**4 - 3*x**2 + 1
>>> factor(f)
(1 + x - x**2)*(1 - x - x**2)
```
SymPy – matrices

- Linear algebra

```python
>>> m = Matrix([[1, 1, -1], [1, -1, 1], [-1, 1, 1]])

>>> m.inv()
[1/2, 1/2, 0]
[1/2, 0, 1/2]
[0, 1/2, 1/2]

>>> P, D = m.diagonalize()

>>> D
[1, 0, 0]
[0, 2, 0]
[0, 0, -2]

>>> D == P.inv() * m * P
True
```
SymPy – differential equations

- Can solve some ODEs

```python
>>> g = f(x).diff(x, x) + f(x)

>>> dsolve(g, f(x))
f(x) == C1*cos(x) + C2*sin(x)

# sometimes a hint is helpful:

>>> dsolve(sin(x)*cos(f(x)) + cos(x)*sin(f(x))*f(x).diff(x), f(x),
       hint='separable')
-log(1 - sin(f(x))**2)/2 == C1 + log(1 - sin(x)**2)/2

>>> dsolve(x*f(x).diff(x) + f(x) - f(x)**2, f(x), hint='Bernoulli')
f(x) == 1/(x*(C1 + 1/x))
```
SymPy – Physics module

- Quantum mechanics, classical mechanics, Gaussian optics and more

```python
from sympy import symbols, pi, diff
from sympy.functions import sqrt, sin
from sympy.physics.quantum.state import Wavefunction

x, L = symbols('x,L', positive=True)
n = symbols('n', integer=True)
g = sqrt(2/L)*sin(n*pi*x/L)
f = Wavefunction(g, (x, 0, L))

f.norm
1

f(L-1)
sqrt(2)*sin(pi*n*(L - 1)/L)/sqrt(L)

f(0.85, n=1, L=1)
sqrt(2)*sin(0.85*pi)
```
• Units

```python
>>> from sympy.physics.units import *

>>> 300*kilo*20*percent  # dimensionless units
60000

>>> milli*kilogram     # SI units
kg/1000

>>> gram
kg/1000

>>> joule
kg*m**2/s**2
```

• …also a Differential Geometry module!
MCMC in Python

• Bayesian inference
  • Parameter estimation
  • Model selection

• emcee
• PyMC
• PyMultiNest
• Via RPy…
  • Laplace's Demon
  • NUTS
  • …
emcee

- "The MCMC hammer"
- Affine-invariant sampler
- Parallel tempering
- Easy to use
- Highly effective
- Written and advocated by NYU hipsters

emcee notebook example

[link to online notebook]