An introduction to scientific programming with

Session 4:
Python for specialists

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

Python for observers

- recent and ongoing effort to create a uniform package
  - currently a bit feature-light, but well supported
  - includes simple cosmology calculations
  - worth supporting and contributing too

Good sites for further information:

http://www.astropython.org

http://www.astrobetter.com

…

IRAF

To use IRAF you will need it installed

- http://iraf.noao.edu
- Fairly straightforward to install
- Or ask your sysadmin (Phil Parry in Nottingham) to do it

SCISOFT for Mac OS X:

- http://web.mac.com/npirzkal/Scisoft
- Quick way to install many of the most used data reduction packages:
- IRAF, PyRAF, MIDAS, Python and Python extensions and many more…

Ureka:

- http://

STScI Python

- http://www.stsci.edu/resources/software_hardware/stsci_python

- Astronomy software provided by Space Telescope Science Institute
  - PyFITS
  - PyRAF
  - MultiDrizzle
  - Numdisplay
  - pysynphot

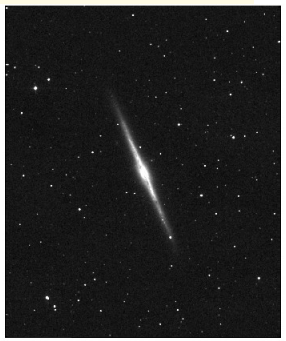
- STScI also provides STSDAS, TABLES and HST reduction packages for IRAF

Handling FITS files – PyFITS

- 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

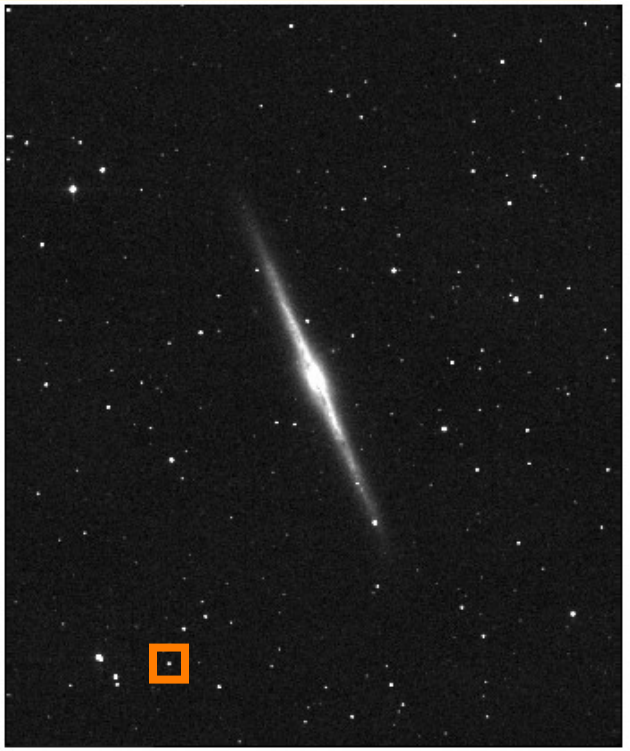
PyFITS – reading FITS images

```
>>> import 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()
-8.661054999999993
>>> img[img > -99].mean()
0.8354629009542302 6
>>> numpy.median(img)
0.07826921343803405 8
```



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  62 110 107  50  13   2   0   0]
 [  2   7  33  91 158 148  68  15   3   0   0]
 [  3   7  27  74 123 115  53  12   2   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

Very useful: `pyfits.info()`

```
>>> tblname = 'data/N891PNdata.fits'
>>> d = pyfits.getdata(tblname)
>>> d.names
('x0', 'y0', 'rah', 'ram', 'ras', 'decd', '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,…
 1838.18994141,  1888.26000977,  1516.2199707 ], dtype=float32)

>>> d.field('x0')   # case-insensitive
array([  928.7199707 ,   532.61999512,   968.14001465,   519.38000488,…
 1838.18994141,  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]
```

PyFITS – reading FITS headers

```
>>> 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
COMMENTC= '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
```

PyFITS – writing FITS images

```
>>> 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
```

PyFITS – writing FITS tables

```
>>> import 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)
```

PyFITS – advanced

```
>>> f = pyfits.open(tblname)
>>> f.info()
Filename: data/N891PNdata.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU       4  ()            uint8
1                BinTableHDU     52  223R x 22C    [E, E, E, E, E,
   E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E]

>>> 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

```
>>> 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=float32)
```

- Only works for files up to ~2Gb (due to limit on Python object size)

Matching catalogues

- For small samples can use simple nested loops

  … and hope you don't hit edge cases

- Better to use:
  - astro libraries
  - searchsorted
  - scipy set library methods
  - do it outside of Python (e.g., using TOPCAT or STILTS)

PyRAF – scripting IRAF with Python

- http://www.stsci.edu/resources/software_hardware/pyraf

- 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

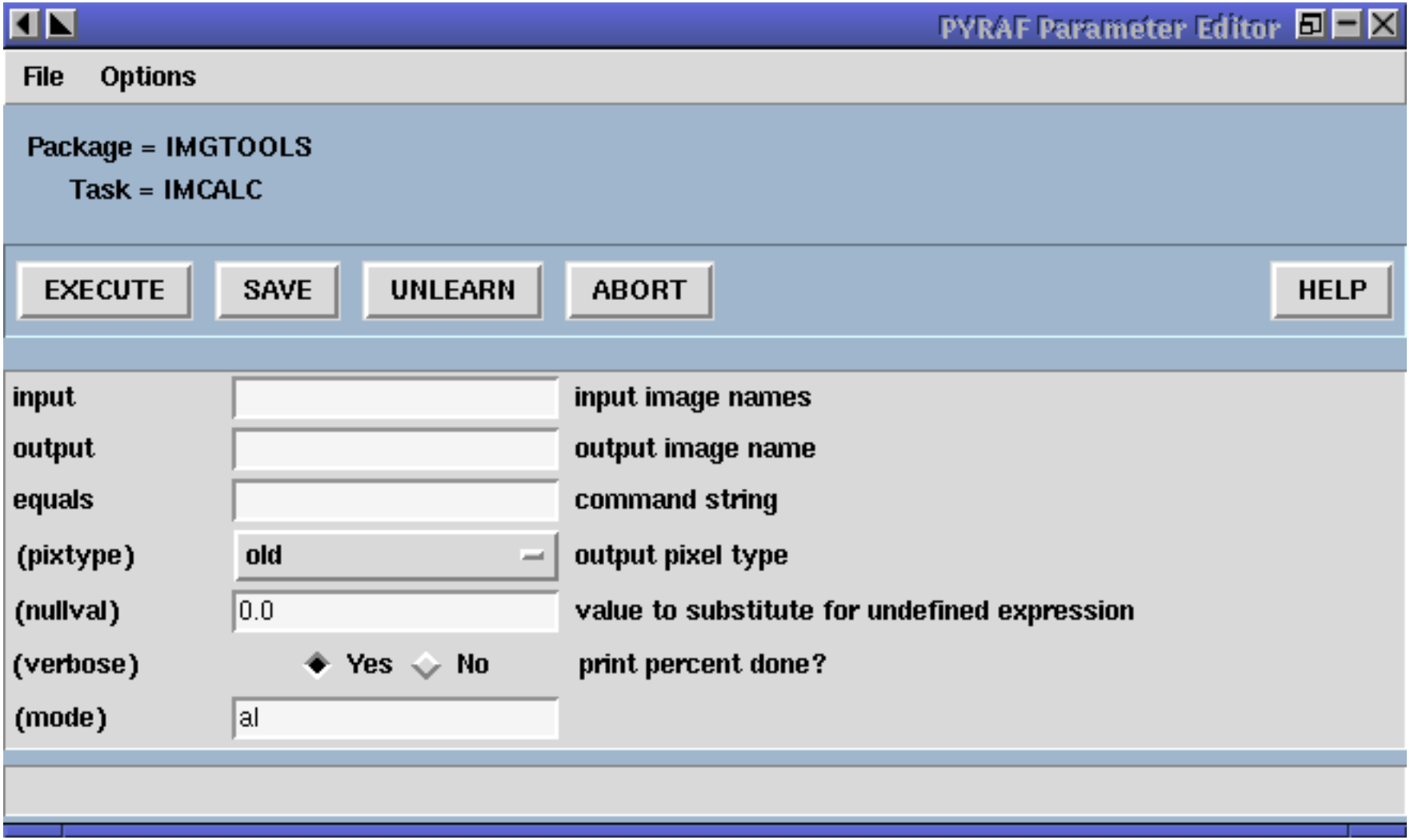
```
--> imstat 2MASS_NGC_0891_K.fits
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
 2MASS_NGC_0891_K.fits   1200000    -8.661     99.44    -1001.     7207.

--> 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
 data/2MASS_NGC_0891_K.fits   1200000    -8.661     99.44    -1001.     7207.
```

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)

```
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 – graphical epar



PyRAF – scripting

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

```
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)')
```

PyRAF – scripting

- Can specify 'default' task parameters in neat fashion, instead of having to include on every call of a task

```
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

```
# 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

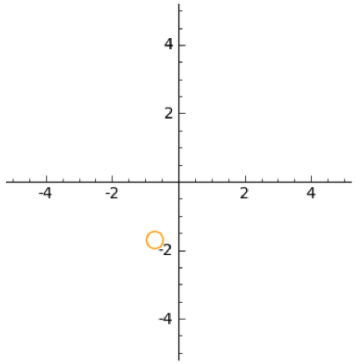
- IRAF outputs lots of useful data to screen - how to get at it?
- Some tasks need user input - would be nice to automate these
- PyRAF defines extra parameters Stdout and Stdin for all tasks
  - Stdout can be a filename, file object, or 1 (one) to direct to a list
  - Stdin can be a filename, file object,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

```
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

- http://www.sagemath.org/
- Python-based mathematics software
  - replacement for Maple, Mathematica
  - runs as a web application
  - Private and collaborative workbooks

```
def r2(zz0):
    zz = complex(zz0) # coerce symbolic expression -> complex number
    return zz.real, zz.imag

def isoshow_act(ff,start,times=7,radius=1/4): #apply ff to start times times
    if times>10:
        times=10
    pts = [start]
    for k in range(times):
        start = ff(start)
        pts.append(start)
    return [circle(r2(pts[i]), radius, hue=i/10)
              for i in range(0,times)]

ll=isoshow_act(compose(-f1,f3),-1,times=4)
a = animate(ll,xmin=-5,ymin=-5,xmax=5,ymax=5,figsize=[4,4])
a.show()
```



Examples:

```
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(I), " ", f2(1)
f3(z)=(cos(pi/4)+sin(pi/4)*I)*z # rotation by pi/4
print f3(1), " ", f3(I-3), f2(f3(I-3))
```

```
3*I - 5
-I   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

```
>>> 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

```
>>> x = Symbol('x')
>>> y = Symbol('y')

>>> x+y+x-y
2*x
>>> (x+y)**2
(x + y)**2

>>> expand((x+y)**3)
3*x*y**2 + 3*y*x**2 + x**3 + y**3

>>> simplify((x+x*y)/x)
1 + y
```

```
# define multiple symbols
>>> x, y, z = symbols('x,y,z')

# useful shortcut
>>> f = simpify('(x+y)**2')

# latex output!
>>> print latex(exp(x**2/2))
e^{\frac{1}{2} x^{2}}
```

SymPy – calculus

- Limits, derivatives, Taylor expansions and integrals

```
>>> 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

```
>>> 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 → simultaneous equations
- Can also factorise polynomials

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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)
```

SymPy – Physics module

- Units

```
>>> 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!