Python Advance Course via Astronomy street

Sérgio Sousa (CAUP)
ExoEarths Team (http://www.astro.up.pt/exoearths/)

Python Advance Course via Astronomy street

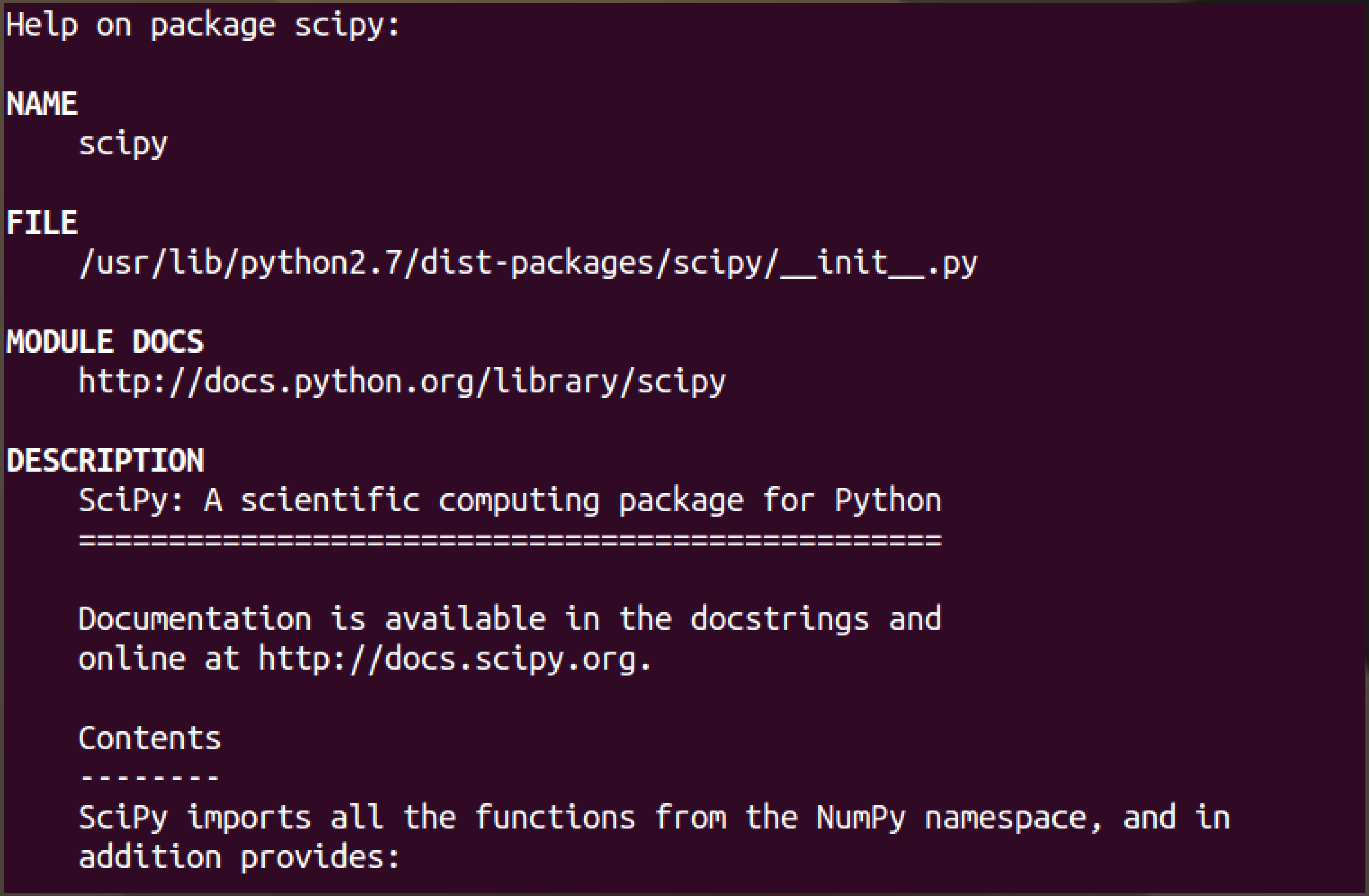
Advance Course Outline:

- Lesson 1: Python basics (T)
- Lesson 2: Python with Numpy and Matplotlib (T)
- Lesson 3: Science modules (Scipy) (T)

SciPy

http://www.scipy.org/

```
>>> import scipy
>>> help(scipy)
```



```
Help on package scipy:

NAME
    scipy

FILE
    /usr/lib/python2.7/dist-packages/scipy/__init__.py

MODULE DOCS
    http://docs.python.org/library/scipy

DESCRIPTION
    SciPy: A scientific computing package for Python
    ================================================

    Documentation is available in the docstrings and
    online at http://docs.scipy.org.

    Contents
    --------
    SciPy imports all the functions from the NumPy namespace, and in
    addition provides:
```

SciPy

http://www.scipy.org/

- Scipy is a big module with several toolboxes for scientific computing;
- It is divided in several specific submodules:
    - interpolation;
    - integration;
    - optimization;
    - image processing;
    - statistics;
- Scipy is comparable with GSL (Gnu Scientific Library) for C/C++ or Matlab's toolboxes
- Strong efficient dependence on Numpy arrays

Advice: Check the content of Scipy module before starting to reinvent the wheel

SciPy

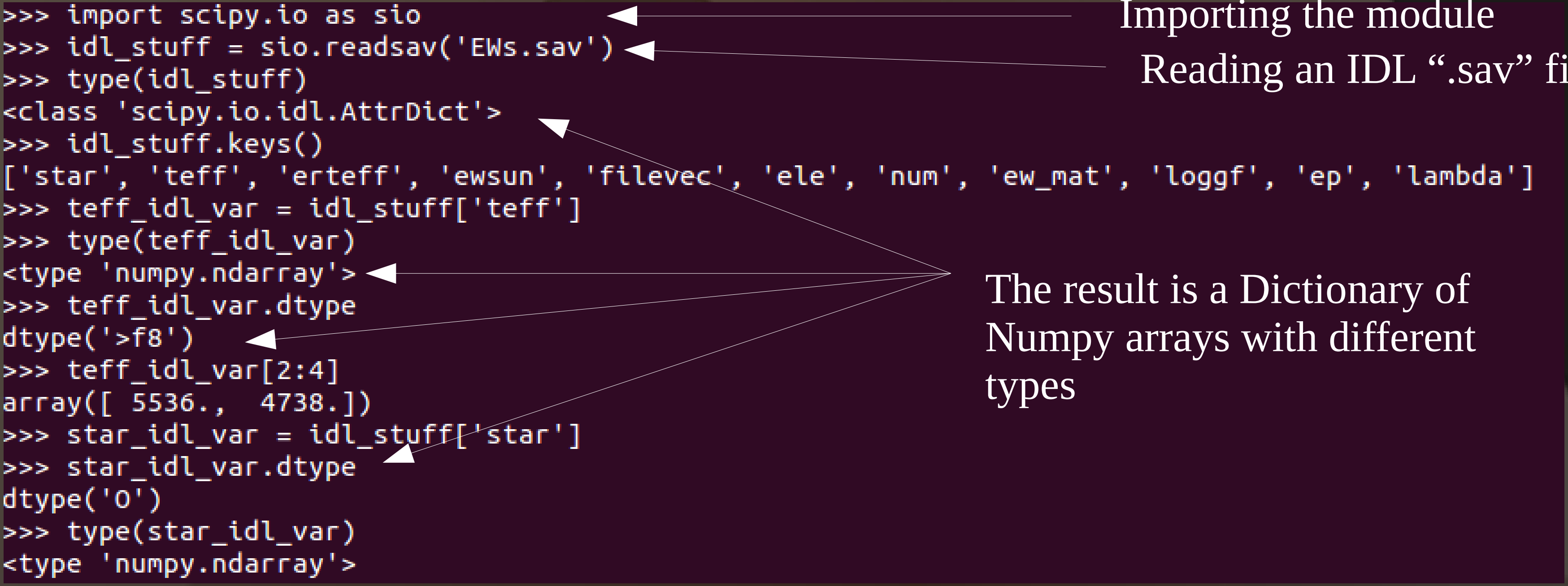
http://www.scipy.org/

File Input-Output : scipy.io

There are couple of functions to read specific files:

- Matlab;
- IDL;
- Fortran unformatted;
- WAV sound;
- other stuff: Matrix Market files, Arff files, Netcdf

Example IDL sav file: scipy.io.readsav(...)



```
>>> import scipy.io as sio
>>> idl_stuff = sio.readsav('EWs.sav')
>>> type(idl_stuff)
<class 'scipy.io.idl.AttrDict'>
>>> idl_stuff.keys()
['star', 'teff', 'erteff', 'ewsun', 'filevec', 'ele', 'num', 'ew_mat', 'loggf', 'ep', 'lambda']
>>> teff_idl_var = idl_stuff['teff']
>>> type(teff_idl_var)
<type 'numpy.ndarray'>
>>> teff_idl_var.dtype
dtype('>f8')
>>> teff_idl_var[2:4]
array([ 5536.,  4738.])
>>> star_idl_var = idl_stuff['star']
>>> star_idl_var.dtype
dtype('O')
>>> type(star_idl_var)
<type 'numpy.ndarray'>
```

Importing the module

Reading an IDL ".sav" file

The result is a Dictionary of Numpy arrays with different types

SciPy.org Sponsored By ENTHOUGHT

SciPy

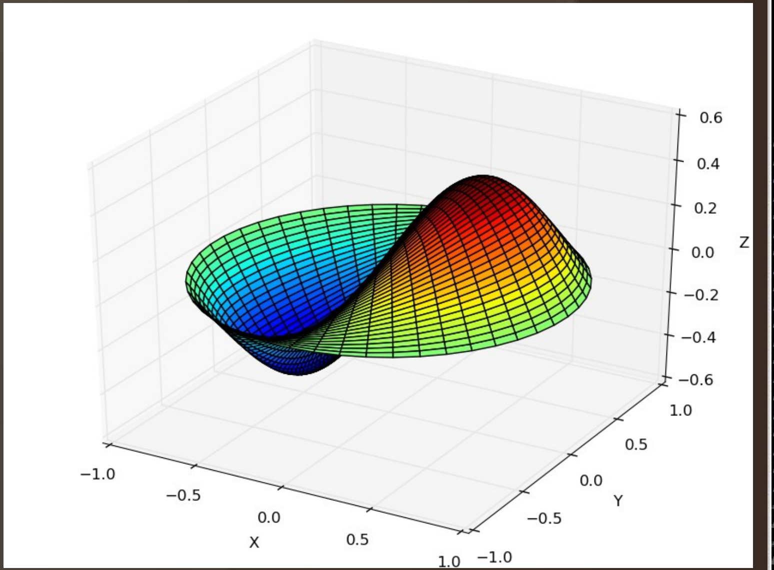
http://www.scipy.org/

Special Functions : scipy.special

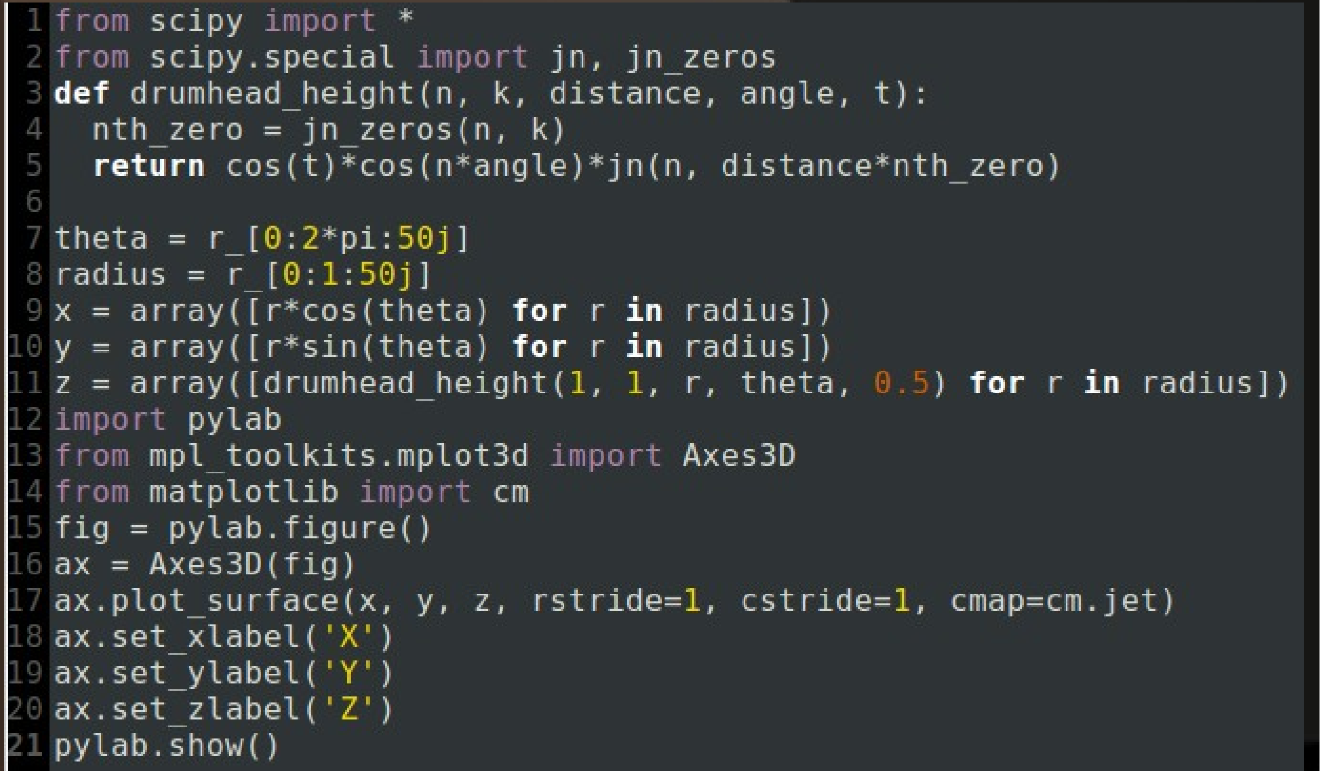
This submodule includes a series of built in special functions:

- Airy functions;
- Elliptic functions and integrals;
- Bessel functions, zeros, integrals, derivatives, spherical;
- Raw statistical Functions;
- Gamma and related functions;
- Legendre Functions
- Orthogonal Polynomials
- Spheroidal Wave functions
- ...

Example: Using the bessel function



http://docs.scipy.org/doc/scipy/reference/tutorial/special.html



```
from scipy import *
from scipy.special import jn, jn_zeros
def drumhead_height(n, k, distance, angle, t):
  nth_zero = jn_zeros(n, k)
  return cos(t)*cos(n*angle)*jn(n, distance*nth_zero)

theta = r_[0:2*pi:50j]
radius = r_[0:1:50j]
x = array([r*cos(theta) for r in radius])
y = array([r*sin(theta) for r in radius])
z = array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = pylab.figure()
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')
pylab.show()
```

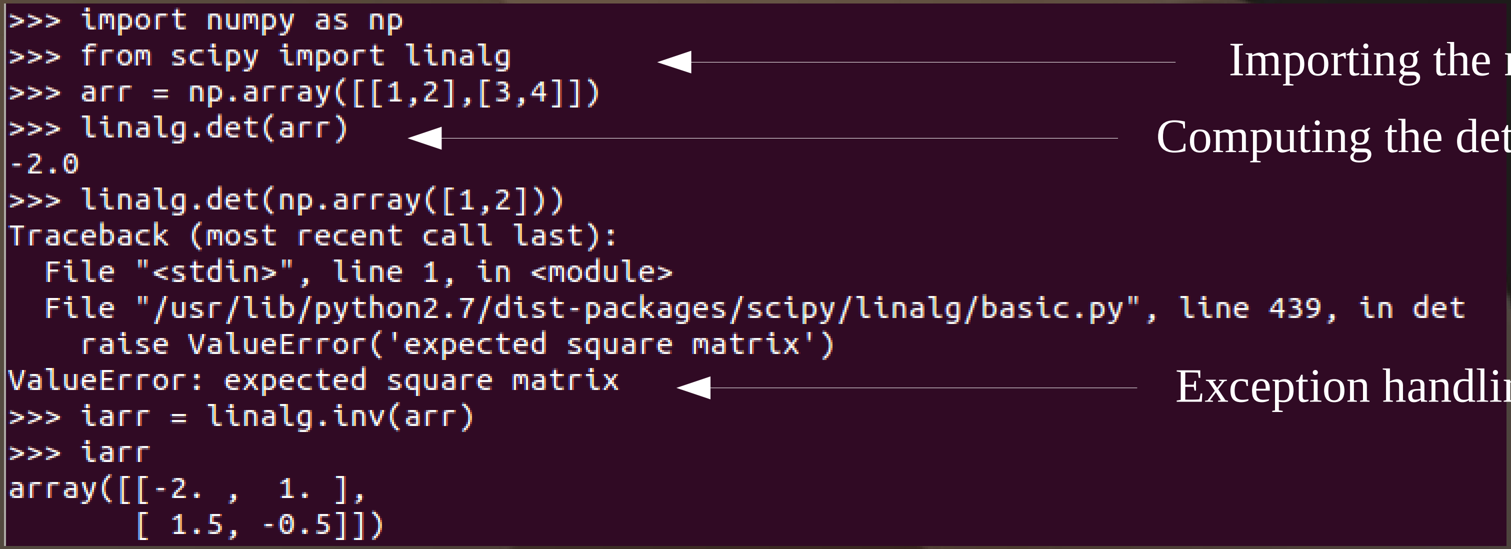
SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Linear Algebra Operations : scipy.linalg

- This submodule includes a series operations of linear algebra.
- Many of these overlap the operations that we can do directly with Numpy;



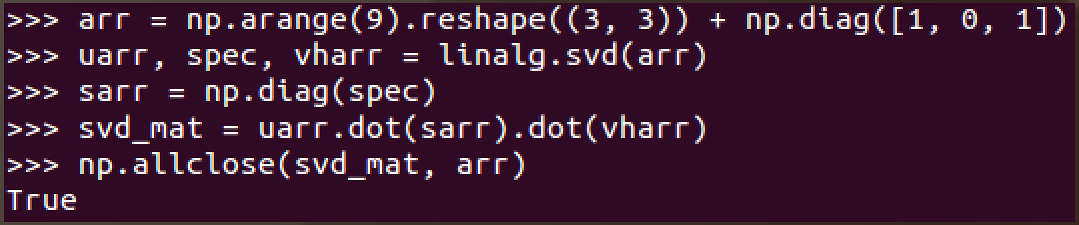
```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[1,2],[3,4]])
>>> linalg.det(arr)
-2.0
>>> linalg.det(np.array([1,2]))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.7/dist-packages/scipy/linalg/basic.py", line 439, in det
    raise ValueError('expected square matrix')
ValueError: expected square matrix
>>> iarr = linalg.inv(arr)
>>> iarr
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Importing the module

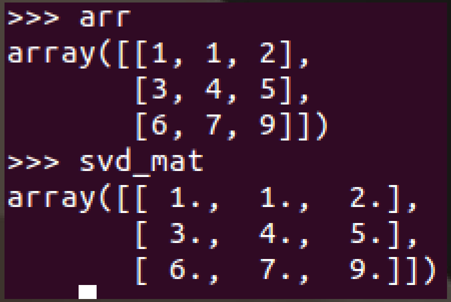
Computing the determinant of the matrix

Exception handling functionalities

- There are however some extra interesting stuff: - help(scipy.linalg) to see the list of funcionalities!
    Example: singular-value decomposition (SVD):



```
>>> arr = np.arange(9).reshape((3, 3)) + np.diag([1, 0, 1])
>>> uarr, spec, vharr = linalg.svd(arr)
>>> sarr = np.diag(spec)
>>> svd_mat = uarr.dot(sarr).dot(vharr)
>>> np.allclose(svd_mat, arr)
True
```



```
>>> arr
array([[1, 1, 2],
       [3, 4, 5],
       [6, 7, 9]])
>>> svd_mat
array([[ 1.,  1.,  2.],
       [ 3.,  4.,  5.],
       [ 6.,  7.,  9.]])
```

allclose() → checks if arrays are equal considering numerical errors in computation

SciPy.org Sponsored By ENTHOUGHT

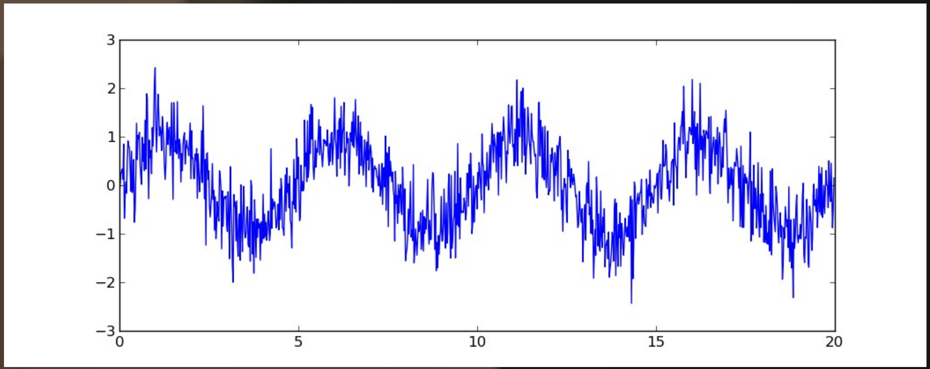
SciPy

http://www.scipy.org/

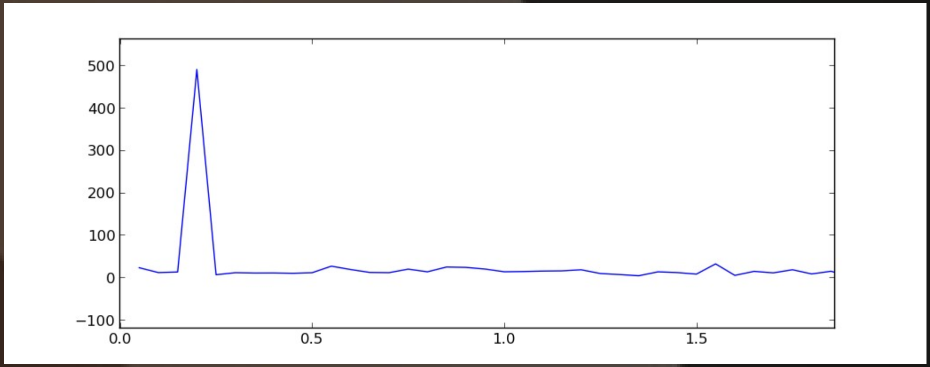
Fast Fourier transforms: scipy.fftpack

- This submodule allows to compute fast Fourier transforms

```
>>> import numpy as np
>>> time_step = 0.02
>>> period = 5.
>>> time_vec = np.arange(0, 20, time_step)
>>> sig = np.sin(2 * np.pi / period * time_vec) + \
...       0.5 * np.random.randn(time_vec.size)
>>> import matplotlib.pyplot as plt
>>> plt.plot(time_vec,sig)
[<matplotlib.lines.Line2D object at 0x379f3d0>]
>>> plt.show()
```



```
>>> from scipy import fftpack
>>> sample_freq = fftpack.fftfreq(sig.size, d=time_step)
>>> sig_fft = fftpack.fft(sig)
>>> pidxs = np.where(sample_freq > 0)
>>> freqs = sample_freq[pidxs]
>>> power = np.abs(sig_fft)[pidxs]
>>> plt.plot(freqs,power)
[<matplotlib.lines.Line2D object at 0x378e210>]
>>> plt.show()
```



Checking the derived frequency:

```
>>> freq = freqs[power.argmax()]
>>> freq
0.20000000000000001
>>> np.allclose(freq, 1./period)
True
```

Numpy also has an implementation of FTT (numpy.ftt). However, in general the scipy version should be prefered, because it uses more efficient underlying implementation.

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

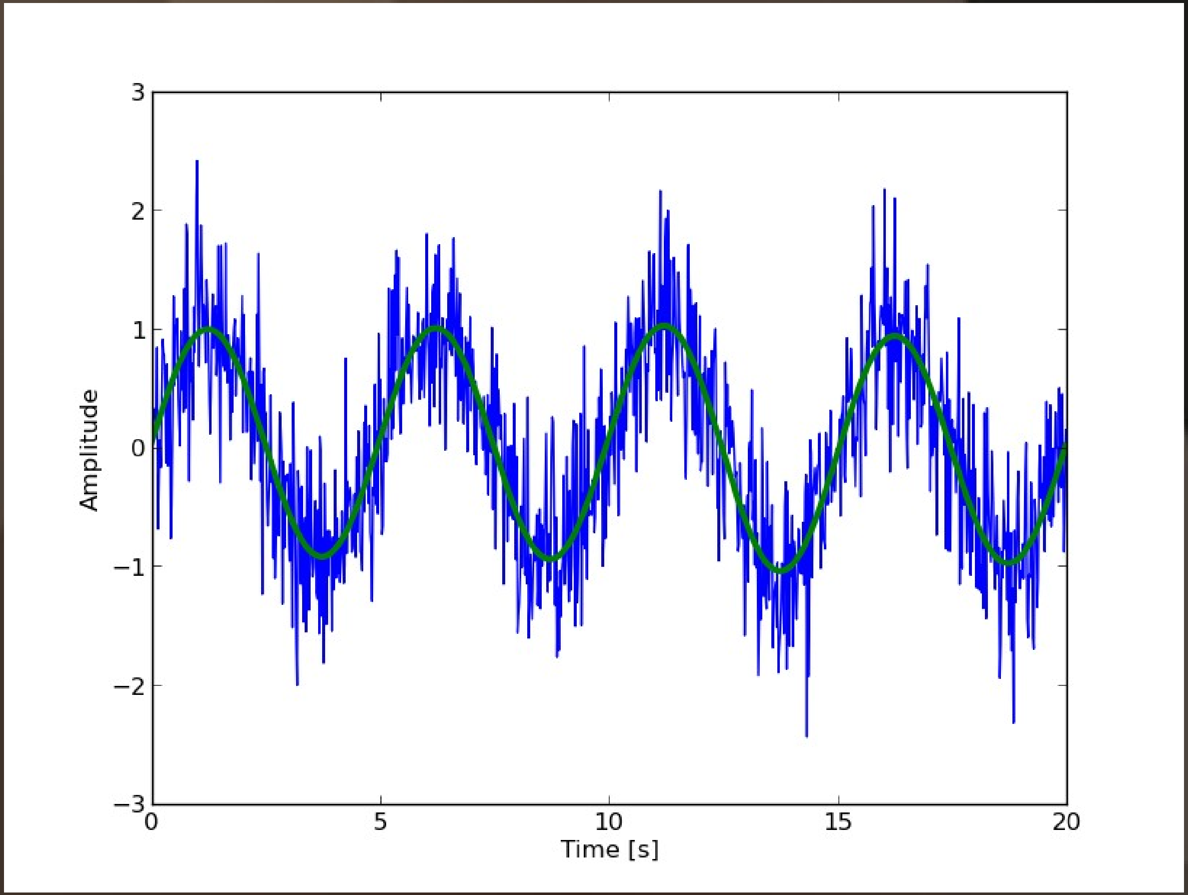
Fast Fourier transforms: scipy.fftpack

Example: Filtering the data

```
>>> sig_fft[np.abs(sample_freq) > freq] = 0
>>> main_sig = fftpack.ifft(sig_fft)
```

Filtering the data

Inverting the fourier transform



SciPy.org Sponsored By ENTHOUGHT

SciPy

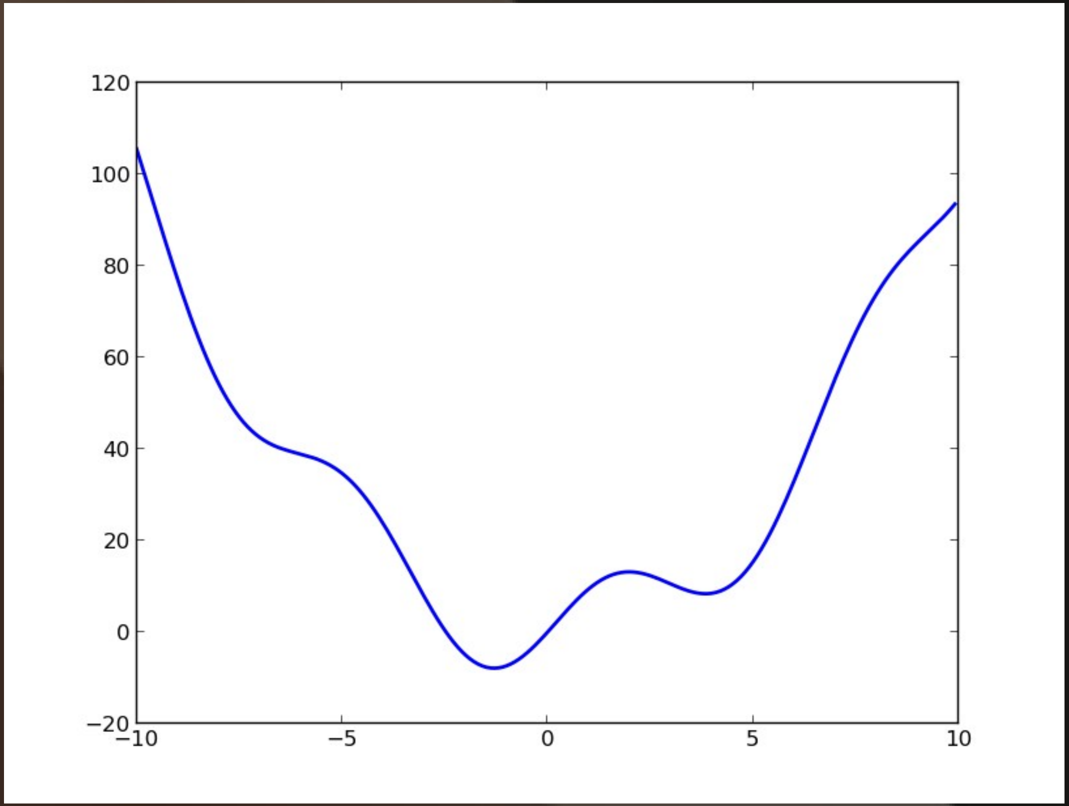
http://www.scipy.org/

Optimization and fitting : scipy.optimize

Optimization is the problem of finding a numerical solution to a minimization or equality.

Finding the minimum of a scalar function:

```
>>> import numpy as np
>>> import scipy.optimize as sop
>>> def f(x):
...   return x**2 + 10*np.sin(x)
...
>>> x = np.arange(-10,10,0.1)
>>> import matplotlib.pyplot as plt
>>> plt.plot(x,f(x))
[<matplotlib.lines.Line2D object at 0x3da4f90>]
>>> plt.show()
```



Using the BFGS algorithm:

```
>>> sop.fmin_bfgs(f,0)
Optimization terminated successfully.
         Current function value: -7.945823
         Iterations: 5
         Function evaluations: 24
         Gradient evaluations: 8
array([-1.30644003])
```

SciPy.org Sponsored By ENTHOUGHT

SciPy

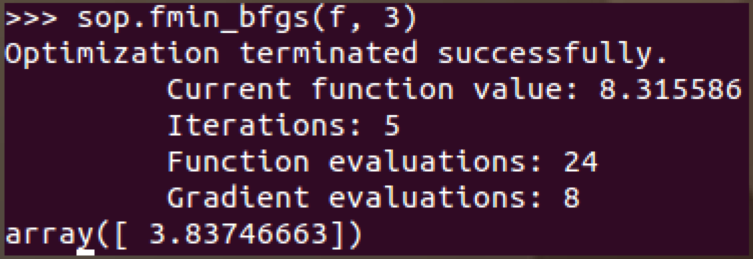
http://www.scipy.org/

Optimization and fitting : scipy.optimize

Optimization is the problem of finding a numerical solution to a minimization or equality.

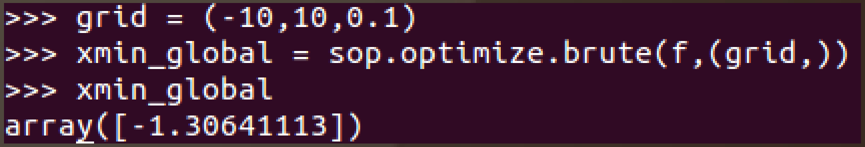
Finding the minimum of a scalar function:

Careful with local minimum
(changing initial guess):



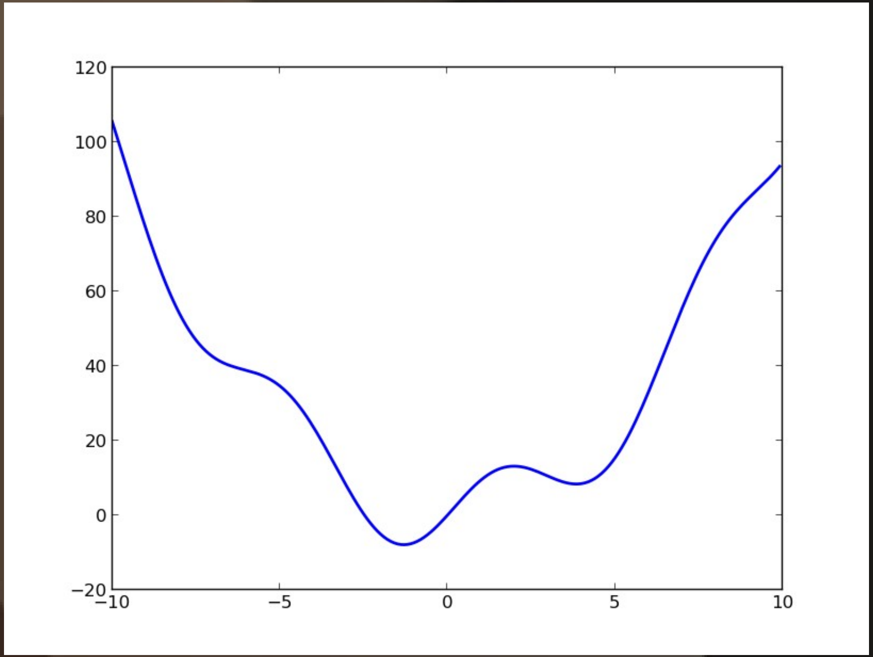
```
>>> sop.fmin_bfgs(f, 3)
Optimization terminated successfully.
         Current function value: 8.315586
         Iterations: 5
         Function evaluations: 24
         Gradient evaluations: 8
array([ 3.83746663])
```

When you don't know the function, you
can simply use brute force:



```
>>> grid = (-10,10,0.1)
>>> xmin_global = sop.optimize.brute(f,(grid,))
>>> xmin_global
array([-1.30641113])
```

Tip: You can always use *help(module.function)*
to know how to use it



Brute force can be quite heavy, in the case where you
need large grids.
Alternatively you can use *scipy.optimize.anneal()*
which uses the simulating annealing minimization
algorithm.

Other modules for Minimization:
OpenOpt, IPOPT, PyGMO and PyEvolve

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Optimization and fitting : scipy.optimize

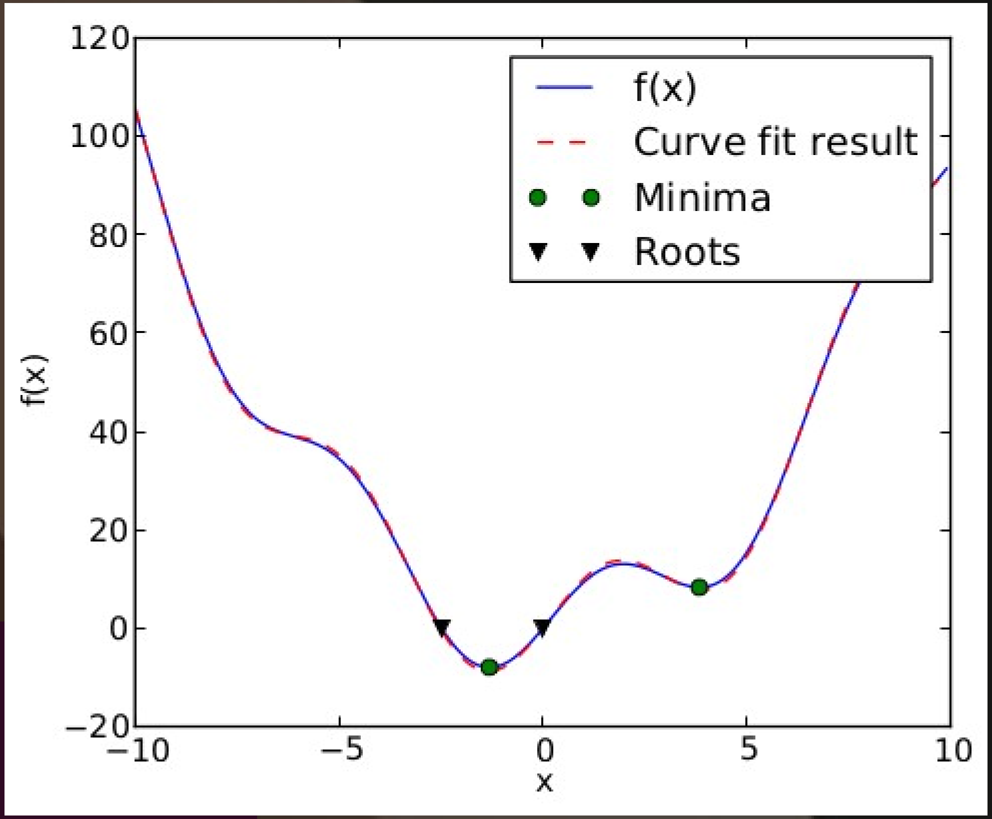
Optimization is the problem of finding a numerical solution to a minimization or equality.

Finding roots of a scalar function:

```
>>> sop.fsolve(f,1)
array([ 0.])
>>> sop.newton(f,1)
-1.0117953699300726e-20
```

Using fsolve()

Using Newton-Raphson or secant method



Curve Fitting:

```
>>> xdata = np.linspace(-10, 10, num=20)
>>> ydata = f(xdata) + np.random.randn(xdata.size)
>>> def f2(x, a, b):
...    return a*x**2 + b*np.sin(x)
...
>>> guess = [2, 2]
>>> params, params_covariance = sop.curve_fit(f2, xdata, ydata, guess)
>>> params
array([  1.0075586 ,  10.05212948])
```

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Statistics and Random Numbers : scipy.stats

This module contains a large number of probability distributions as well as a growing library of statistical functions.

This module is divided into:

- Continuous Distributions;
- Discrete Distributions;
- Statistical functions;
- Contingency table functions;
- General linear model;
- Plot-tests;
- Univariate and multivariate kernel density estimation

The detailled information of the module containts can be found using: help(scipy.stats)

SciPy.org Sponsored By ENTHOUGHT

SciPy

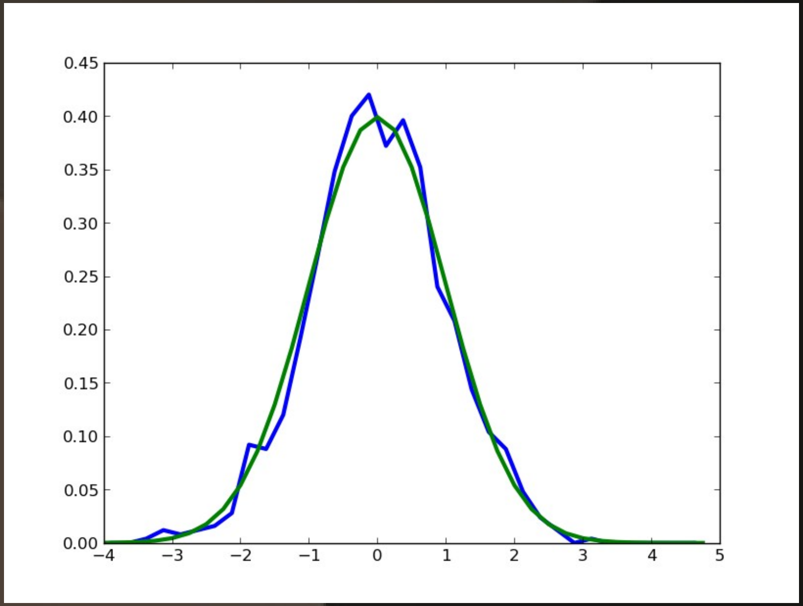
http://www.scipy.org/

Statistics and Random Numbers : scipy.stats

Examples: Using the Normal function:

```
>>> import numpy as np
>>> import scipy.stats as sst
>>> import matplotlib.pyplot as plt
>>> a = np.random.normal(size=1000)
>>> bins = np.arange(-4,5,0.25)
>>> bin_plot = 0.5*(bins[1:] + bins[:-1])
>>> histogram = np.histogram(a, bins=bins, normed=True)[0]
>>> b = sst.norm.pdf(bins)
>>> plt.plot(bin_plot,histogram)
[<matplotlib.lines.Line2D object at 0x45c1910>]
>>> plt.plot(bins,b)
```

Creating random data



Creating an histogram to plot

Geting the values of a Normal Function

Statistical Tests:

```
>>> loc, std = sst.norm.fit(a)
>>> loc,std
(0.032302008648340083, 1.0041030968829048)
```

Using Maximum Likelihood Estimate to extract the mean and the std...

```
>>> a = np.random.normal(0,1,size=100)
>>> b = np.random.normal(1,1,size=10)
>>> sst.ttest_ind(a,b)
(array(-3.3025291483817876), 0.0012997729185973265)
>>> sst.ks_2samp(a,b)
(0.47000000000000003, 0.023495863518120642)
>>> b = np.random.normal(0,1,size=10)
>>> sst.ks_2samp(a,b)
(0.20000000000000007, 0.81585145499712663)
```

Using a T Test on two samples

Using a Kolmogorov-Smirnov Test

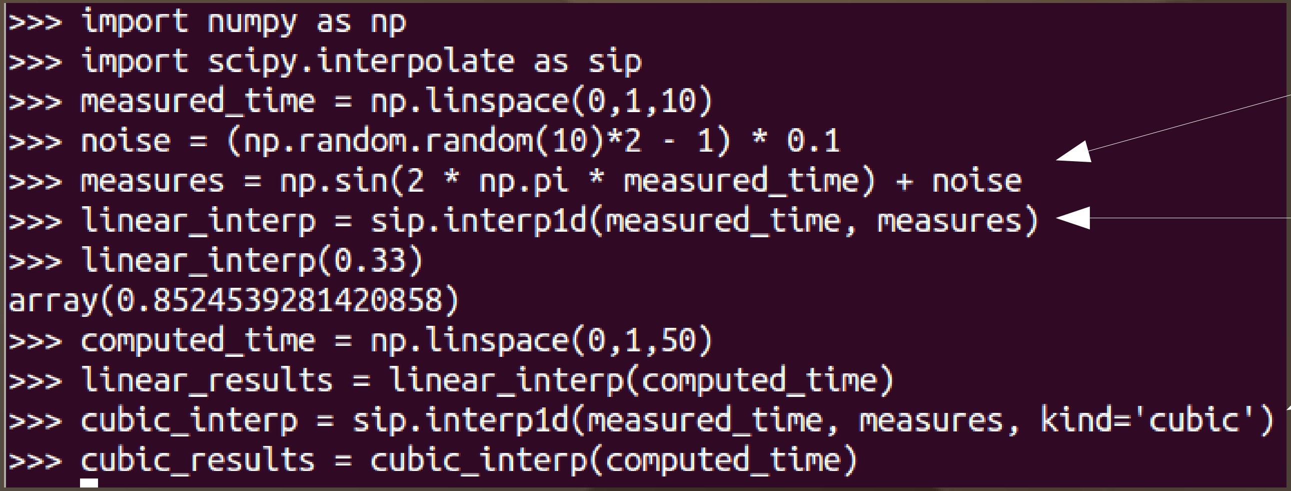
SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Interpolation: scipy.interpolate

The scipy.interpolate is useful for fitting a function from experimental data and thus evaluating points where no measure exists. (The module is based on the FITPACK Fortran subroutines from the netlib project)



```
>>> import numpy as np
>>> import scipy.interpolate as sip
>>> measured_time = np.linspace(0,1,10)
>>> noise = (np.random.random(10)*2 - 1) * 0.1
>>> measures = np.sin(2 * np.pi * measured_time) + noise
>>> linear_interp = sip.interp1d(measured_time, measures)
>>> linear_interp(0.33)
array(0.8524539281420858)
>>> computed_time = np.linspace(0,1,50)
>>> linear_results = linear_interp(computed_time)
>>> cubic_interp = sip.interp1d(measured_time, measures, kind='cubic')
>>> cubic_results = cubic_interp(computed_time)
```

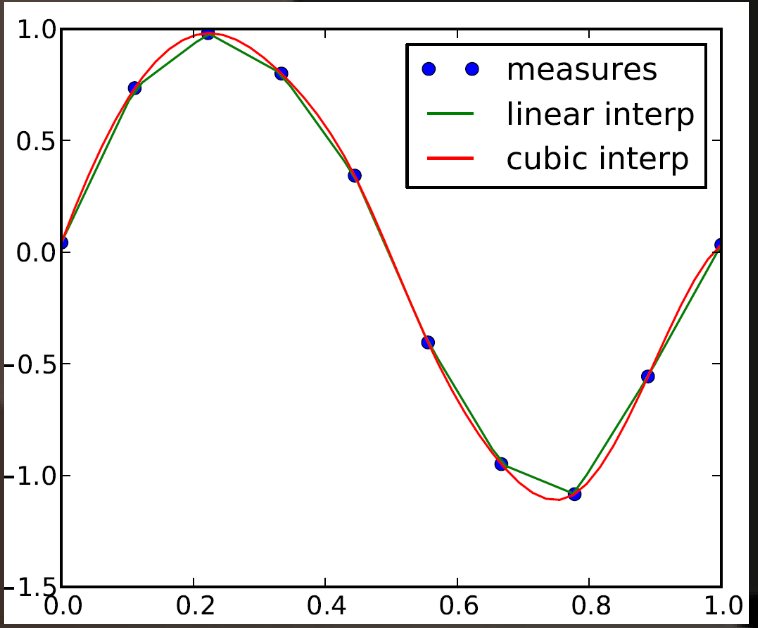
Creating the data sampled with some noise

Linear Interpolation

Cubic Interpolation

When you use linear_interp the result is function (object) that can be used directly like any other defined function.

This function then returns the interpoled values for the input data values.



Note. There are several interpolation functions that can be used in this module. Check: help(scipy.interpolate)

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Numerical Integration : scipy.integrate

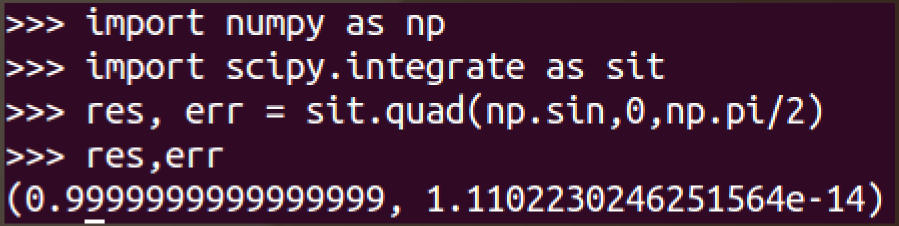
This module contains a several functions to integrate functions / data:

This module is divided into:

- Integrating functions, given function object (ex. quad, romberg);
- Integrating functions, given fixed samples (ex. trapezoidal rule, simpson)
- Integrators of Ordinary Differential equiations (ODE) systems;

The detailled information of the module containts can be found using: help(scipy.integrate)

Most generic integration routine is: scipy.integrate.quad() which uses a technique from the Fortran library QUADPACK:



```
>>> import numpy as np
>>> import scipy.integrate as sit
>>> res, err = sit.quad(np.sin,0,np.pi/2)
>>> res,err
(0.9999999999999999, 1.1102230246251564e-14)
```

Integrating the sin function between 0 and Pi/2.

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/

Signal Processing : scipy.signal

This module contains a large collection of functionalities:

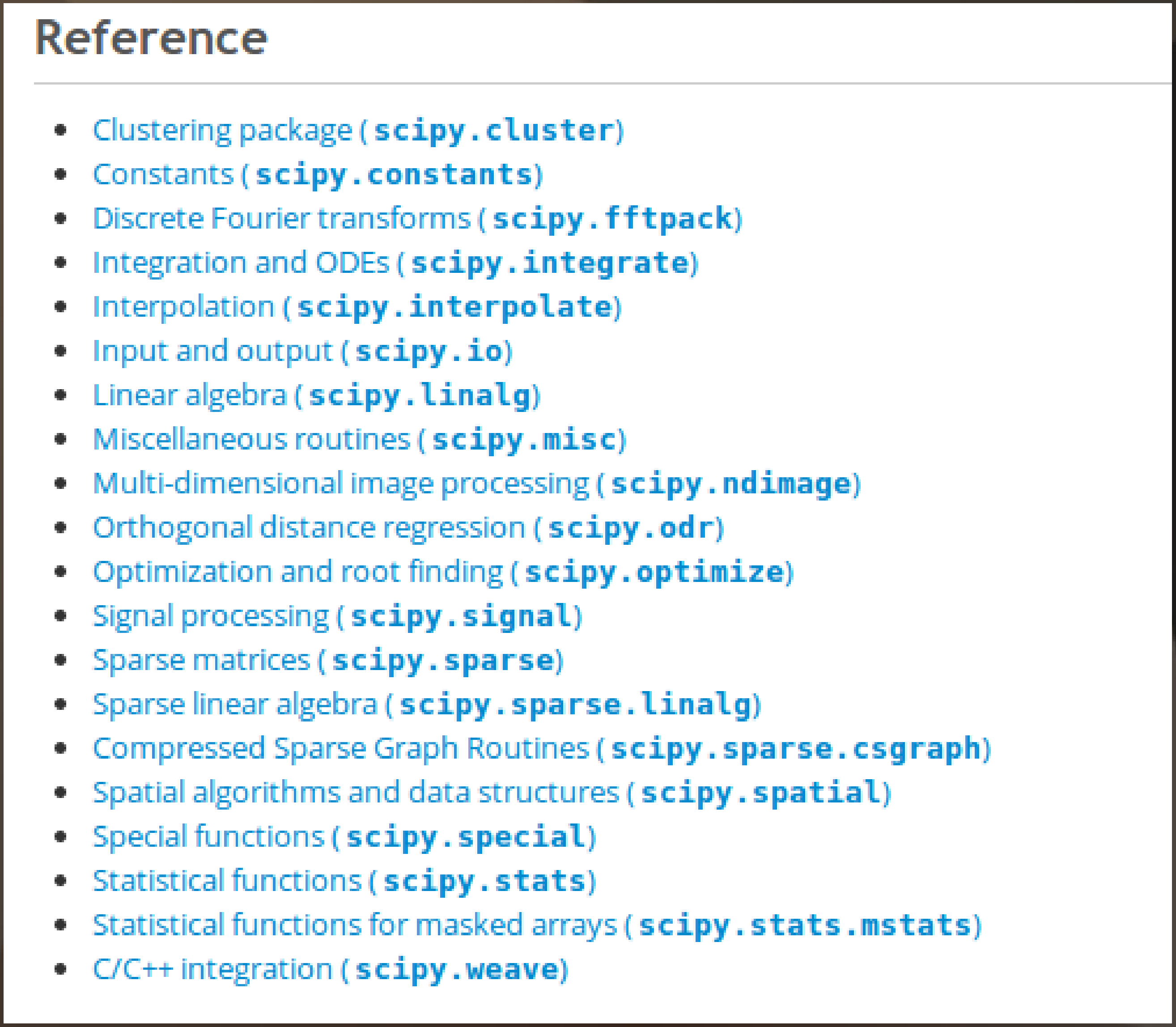
- Convolution;
- B-splines;
- Filtering;
- Filter Design (include some MatLab like functions);
- Continuous (and Discrete)-Time Linear Systems
- Waveforms (ex. Gaussian modulated sinusoid)
- Window Functions (ex. Boxcar window)
- Wavelets
- Peak Finding
- Spectral Analysis (periodogram, lombscarle)

For more information: help(scipy.signal)

SciPy.org Sponsored By ENTHOUGHT

SciPy

http://www.scipy.org/



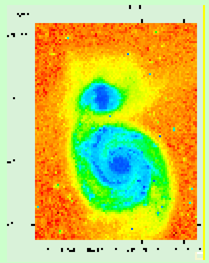
Reference

- Clustering package (**scipy.cluster**)
- Constants (**scipy.constants**)
- Discrete Fourier transforms (**scipy.fftpack**)
- Integration and ODEs (**scipy.integrate**)
- Interpolation (**scipy.interpolate**)
- Input and output (**scipy.io**)
- Linear algebra (**scipy.linalg**)
- Miscellaneous routines (**scipy.misc**)
- Multi-dimensional image processing (**scipy.ndimage**)
- Orthogonal distance regression (**scipy.odr**)
- Optimization and root finding (**scipy.optimize**)
- Signal processing (**scipy.signal**)
- Sparse matrices (**scipy.sparse**)
- Sparse linear algebra (**scipy.sparse.linalg**)
- Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
- Spatial algorithms and data structures (**scipy.spatial**)
- Special functions (**scipy.special**)
- Statistical functions (**scipy.stats**)
- Statistical functions for masked arrays (**scipy.stats.mstats**)
- C/C++ integration (**scipy.weave**)

Python Advance Course via Astronomy street

Lesson 3: Python with Matplotlib, Scipy, Pyfits, Pyraf

- Plotting with Matplotlib
- Using Scipy
- Pyfits – Information
- Pyraf – Easy install



FITS The Astronomical Image and Table Format

Flexible Image Transport System

http://fits.gsfc.nasa.gov/

What is FITS?

- The standard data format used in astronomy
- Stands for 'Flexible Image Transport System'
- Endorsed by NASA and the International Astronomical Union
- Much more than just another image format (such as JPEG or GIF)
- Used for the transport, analysis, and archival storage of scientific data sets
    - Multi-dimensional arrays: 1D spectra, 2D images, 3D+ data cubes
    - Tables containing rows and columns of information
    - Header keywords provide descriptive information about the data

PyFits

Pyfits:

Is a module for reading and writing FITS files and manipulating their contents.

Some Useful Links

Space Telescope Science Institute: http://www.stsci.edu/institute/software_hardware/pyfits/

PyFits Tutorial: http://pythonhosted.org/pyfits/users_guide/users_tutorial.html

For detailed examples of usage, see the `PyFITS User's Manual:
http://stsdas.stsci.edu/download/wikidocs/The_PyFITS_Handbook.pdf

Special Note: All of the functionality of PyFITS is now available in Astropy as the astropy.io.fits package, which is now publicly available.

An example will be presented in the practical session

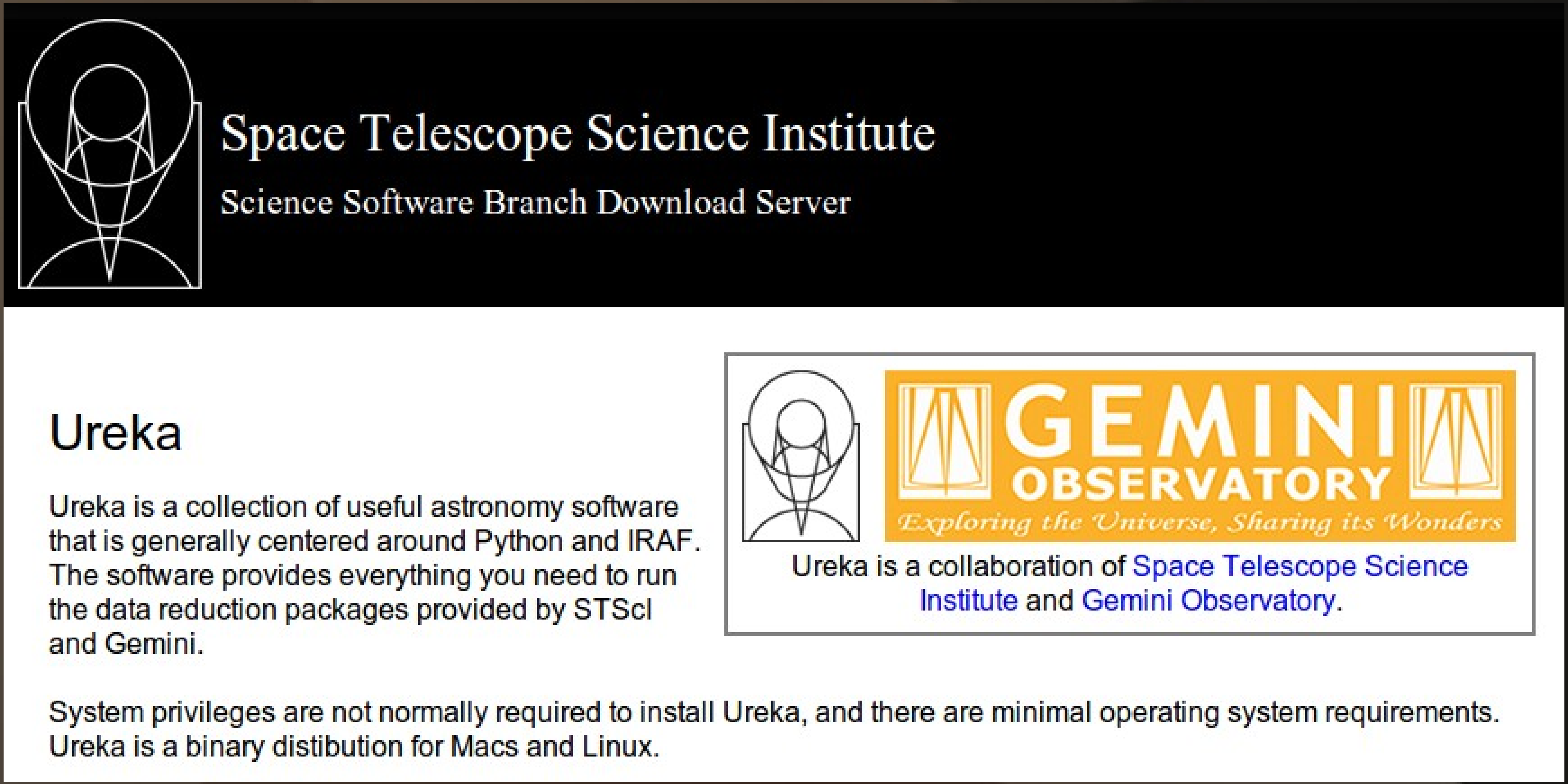
Python Advance Course via Astronomy street

Lesson 3: Python with Matplotlib, Scipy, Pyfits, Pyraf

- Plotting with Matplotlib
- Using Scipy
- Pyfits – Information
- Pyraf – Easy install

PyIRAF by Ureka

http://ssb.stsci.edu/ureka/


Space Telescope Science Institute

Science Software Branch Download Server

Ureka

Ureka is a collection of useful astronomy software that is generally centered around Python and IRAF. The software provides everything you need to run the data reduction packages provided by STScI and Gemini.

Ureka is a collaboration of Space Telescope Science Institute and Gemini Observatory.

System privileges are not normally required to install Ureka, and there are minimal operating system requirements. Ureka is a binary distibution for Macs and Linux.

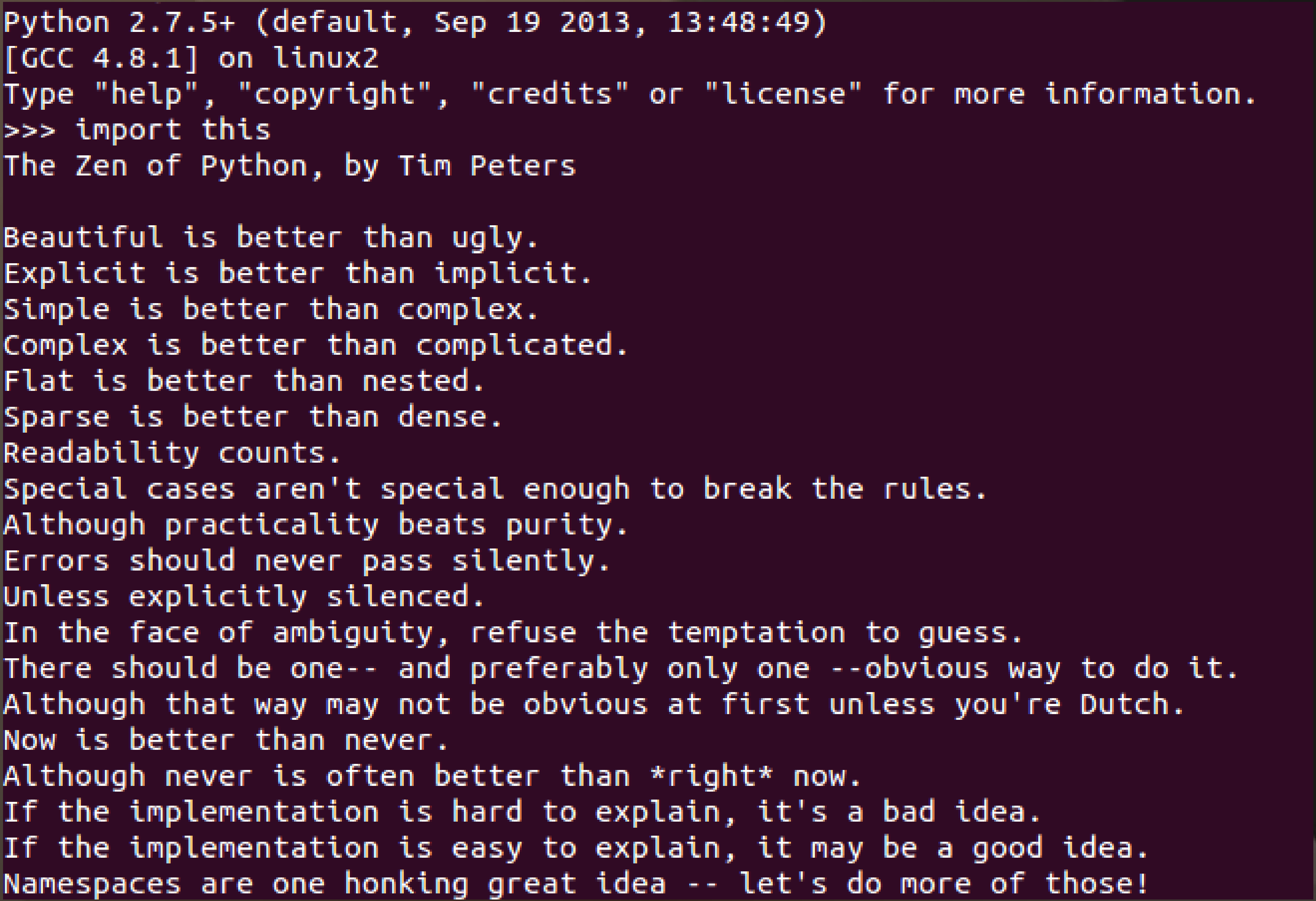
An example using pyrafwill be presented in the practical session

Good Practices in Python

- Explicit variable names (no need of a comment to explain what is in the variable)
- Style: spaces after commas, around =, etc. A certain number of rules for writing “beautiful” code (and, more importantly, using the same conventions as everybody else!) are given in the Style Guide for Python Code and the Docstring Conventions page (to manage help strings).
- Except some rare cases, variable names and comments in English.
- Good identation (forced in Python)

Python is also an object

Sérgio Sousa (CAUP)
ExoEarths Team (http://www.astro.up.pt/exoearths/)



```
Python 2.7.5+ (default, Sep 19 2013, 13:48:49)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```