Advance Course Outline:

- Lesson 1: Python basics (T + P)
- Lesson 2: Python with Numpy (T + P)
- Lesson 3: Matplotlib, Scipy & Astronomy modules (Pyfits, Pyraf) (T + P)
- Lesson 4: Interfacing with C (P)
Lesson 3: Python with Matplotlib, Scipy, Pyfits, Pyraf

- Plotting with Matplotlib
- Using Scipy
- Pyfits – Information
- Pyraf – Easy install
```python
>>> import scipy
>>> help(scipy)
```

**Help on package scipy:**

**NAME**
```python
scipy
```

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

**MODULE DOCS**
```python
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

- 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 weel
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(...)
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

```python
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()
```
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;

```python
>>> 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
>>> larr = linalg.inv(arr)
>>> larr
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 functionalities!
  Example: singular-value decomposition (SVD):

```python
>>> 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
```

allclose() \(\rightarrow\) checks if arrays are equal considering numerical errors in computation
Fast Fourier transforms: scipy.fftpack

- This submodule allows to compute fast Fourier transforms

```python
>>> 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()
```

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

Checking the derived frequency:

```python
>>> freq = freds[power.argmax()]
>>> freq
0.2000000000000001
>>> np.allclose(freq, 1./period)
True
```

Numpy also has an implementation of FTT (numpy.fft). However, in general the scipy version should be preferred, because it uses more efficient underlying implementation.
Example: Filtering the data

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

Filtering the data

Inverting the fourier transform
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:

```python
>>> 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:

```python
>>> 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

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):

```python
>>> 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:

```python
>>> 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

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:

Using fsolve()

Using Newton-Raphson or secant method

Curve Fitting:

```python
>>> 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])
```
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 detailed information of the module contains can be found using: help(scipy.stats)
Examples: Using the Normal function:

```python
>>> 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

Statistical Tests:

```python
>>> loc, std = sst.norm.fit(a)
>>> loc, std
(0.032302008648340083, 1.0041030968829048)
>>> 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)
(0.4700000000000003, 0.023495863518120642)
>>> b = np.random.normal(0,1,size=10)
>>> sst.ks_2samp(a,b)
(0.20000000000000007, 0.81585145499712663)
```

Getting the values of a Normal Function

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

Using a T Test on two samples

Using a Kolmogorov-Smirnov Test
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)

```python
>>> 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)

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 interpolated values for the input data values.

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

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

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

```python
>>> 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.
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)
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.cgraph**)
- 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**)

SciPy

http://www.scipy.org/
Lesson 3: Python with Matplotlib, Scipy, Pyfits, Pyraf

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


For detailed examples of usage, see the `PyFITS User's Manual:`

**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
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/

An example using pyraff will 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 indentation (forced in Python)
Python is also an object

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

```
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!
```