Astronomical data analysis using Python

Lecture 8

Python modules for numerical computing and visualisation

- numpy + scipy
- matplotlib
- gnuplot.py, also sm and PGPLOT have interfaces
- Mathlink - interface to Mathematica
- sympy - Python module for symbolic computation
- MaYaVI - 3D visualisation using VTK
- pymat - Python Matlab interface
- pytave - Python GNU Octave interface
- Rpy - Python R interface
- SageMath: alternative to Magma, Maple, Mathematica, Matlab

SciPy - useful stuff that would bloat numpy http://scipy.org (http://scipy.org)

- constants: physical constants and conversion factors
- cluster: hierarchical clustering, vector quantization, K-means
- fftpack: Discrete Fourier Transform algorithms
- integrate: numerical integration and ODE solvers
- **interpolate**: interpolation and smoothing splines
- io: data input and output
- lib: Python wrappers to external libraries
- linalg: linear algebra routines
- **ndimage**: N dimensional image processing
- odr: Orthogonal distance regression
- **optimize**: optimization algorithms including linear programming
- signal: signal processing tools

SciPy

- sparse: sparse matrix and related algorithms
- spatial: KD-trees, nearest neighbors, distance functions
- special: special functions
- **stats**: statistical distributions and functions

Via RPy, SciPy can interface to the statistical package R.

Curve fitting - scipy.optimize.curve_fit

The algorithm uses the Levenberg-Marquardt algorithm through 'leastsq'. Additional keyword arguments are passed directly to that algorithm. Quick and easy way to fit an arbitary function to some x,y data.

If you quickly want to fit a function to some datapoints, use this function.

Minimization - scipy.optimize.minimize

Many different solvers available - Nelder-Mead, Powell, simulated annealing etc.

Scipy and Numpy documentation

User Guides, Reference Guides and Developer guides are available at:
http://docs.scipy.org (http://docs.scipy.org)

Also check the SciPy Cookbook

https://scipy-cookbook.readthedocs.io/ (https://scipy-cookbook.readthedocs.io/)

This site hosts worked examples of common tasks. Some are introductory in nature, while others are quite advanced.

Some advanced books on Python

- **Numerical Python** by Robert Johansson
- **Programming for Computations - Python** by Linge and Langtangen
- **Fluent Python** by Luciano Ramalho
- **Numerical Python in Astronomy and Astrophyics** by Schmidt and Volschow
- **Python for data analysis** by William McKinney

As always, remember to get the most recent edition of these books.

A Few Years Ago ... a messy situation existed

Several independent developers were developing specific tools for Python...

- PyFITS - enable use of Python to handle FITS files.
- PyWCS - enable World Coordinate System transformations
- atpy / asciitable - handle tables of all kinds
- Some cosmological calculators.
- Some coordinate transformation tools.
- and more...

The problem: Different styles, repeated efforts, no coordination.

The solution: **Astropy** - unite all efforts under one banner!

Astropy www.astropy.org

https://www.astropy.org/astropy-tutorials/ (https://www.astropy.org/astropy-tutorials/) - excellent resources to learn astropy.

Version 5 of astropy was released last week!

Astropy core, coordinated and affiliated packages

- **Core package**
- **Coordinated packages** -- e.g. astropy core, astroquery, photutils, specutils, ccdproc Total: 8 packages.
- **Affiliated packages** - aplpy, synphot, astroML, galpy etc. Depending on your interests, you will need to use one or more of these. Total: 44 packages

The astronomy community across the world has realised the importance of this effort, and many developers are now funded to add functionality to the astropy ecosystem in many countries.

What `astropy` contains

Data structures and transformations

- Constants (`astropy.constants`)
- Units and Quantities (`astropy.units`)
- N-Dimensional Datasets (`astropy.nddata`)
- Data Tables (`astropy.table`)
- Time and Dates (`astropy.time`)
- Time Series (`astropy.timeseries`)
- Astronomical Coordinate Systems (`astropy.coordinates`)
- World Coordinate System (`astropy.wcs`)
- Models and Fitting (`astropy.modeling`)
- Uncertainties and Distributions (`astropy.uncertainty`)

What `astropy` contains

Files, I/O, and Communication

- Unified File Read/Write Interface
- FITS File Handling (`astropy.io.fits`)
- ASCII Tables (`astropy.io.ascii` `astropy.table`)
- VOTable XML Handling (`astropy.io.votable`)
- Miscellaneous: HDF5, YAML, ASDF, Parquet, pickle (`astropy.io.misc`)
- SAMP (Simple Application Messaging Protocol) (`astropy.samp`)

What `astropy` contains

Computations and utilities

- Cosmological Calculations (`astropy.cosmology`)
- Convolution and Filtering (`astropy.convolution`)
- Data Visualization (`astropy.visualization`)
- Astrostatistics Tools (`astropy.stats`)

What will we cover in astropy?

- Table management features.
- Handling FITS files, including datacubes.
- WCS operations.
- Cosmological Calculations.

Table Management in Python

is done via the "table" sub-module inside Astropy.

astropy.io.ascii vs. astropy.table

- astropy.io.ascii is meant purely for reading and writing tables.
- Is a collection of "extensible" classes which can be extended to support newer formats.

astropy.table

- builds upon io.ascii using its functionality for reading / writing tables
- and adding its own powerful table operations.

You won't need to read much about io.ascii unless your tables have some special outstanding features.

In Brief - The ”Class” Concept

We have discussed the concept of an "object" earlier.

- Objects have well defined behavior.
- They have methods which help you perform supported operations on them.
- Where are all these rules defined?

A "class" is crudely put, a definition which allows one to create objects.

To create table objects, we will need a Table class.

Let's Start

In [2]:
```
# First we need the Table class to create table objects.
# The warning that will be flashed has so far not affected
# any functional features of Table class
from astropy.table import Table
```

In [3]:
```
# Next we need to create the Table object using a file.
demo_table = Table.read("demo.txt", format = "ascii")
```

What if the table does not load?

If you get errors when using read() method, it means that your file is formatted in a way that the standard parser is unable to understand the structure of your file.

What to do? Understand the io.ascii.read() method in detail and supply the various options to Table.read().

eg. `header_start = ";"` or `delimiter="|"` , etc.

Displaying Tables.

In [4]:
```
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [5]:
```
demo_table.pprint() # Does exactly the same thing.
# but you can supply options such as
# max_lines, max_width, show_unit, show_name
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [6]:

```
# In this example, we are suppressing column names from appearing.
demo_table.pprint(show_name=False)
```

```
 M31 2012-01-02 17.0 17.5
 M31 2012-01-02 17.1 17.4
M101 2012-01-02 15.1 13.5
 M82 2012-02-14 16.2 14.5
 M31 2012-02-14 16.9 17.3
 M82 2012-02-14 15.2 15.5
M101 2012-02-14 15.0 13.6
 M82 2012-03-26 15.7 16.5
M101 2012-03-26 15.1 13.5
M101 2012-03-26 14.8 14.3
```

More Ways to Print Tables.

Using an interactive table scrolling tool.

```
demo_table.more()
```

Or display it as a formatted table in a browser.

```
demo_table.show_in_browser()
```

Quickly Check Basic Properties of Loaded Table

In [7]:
```
print (len(demo_table)) # Number of rows.
```

```
10
```

In [8]:
```
print (demo_table.colnames) # The names of the columns.
```

```
['name', 'obs_date', 'mag_b', 'mag_v']
```

You can also print any meta information, if available.

```
demo_table.meta
```

Accessing Columns of the Table

In [9]:
```
print (demo_table["name"]) # one column
```

```
name
----
 M31
 M31
M101
 M82
 M31
 M82
M101
 M82
M101
M101
```

In [10]:
```
print (demo_table["name", "mag_b"]) # more than one column
```

```
name mag_b
---- -----
 M31  17.0
 M31  17.1
M101  15.1
 M82  16.2
 M31  16.9
 M82  15.2
M101  15.0
 M82  15.7
M101  15.1
M101  14.8
```

Accessing Rows in a Table

In [11]:
```
print (demo_table[0])
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
```

In [12]:
```
lines = demo_table.pformat() # a list of strings, each string a row, includes h
eader.
print (lines[2])
```

```
 M31 2012-01-02  17.0  17.5
```

Individual Element Access

In [13]:
```
demo_table["name"][0]
```

Out[13]:
```
'M31'
```

In [14]:
```
demo_table[0]["name"] # also works the same as above.
```

Out[14]:
```
'M31'
```

Sub-sectioning Tables

In [15]:
```
subsection_col = demo_table["name","mag_b"] # by column.
```

In [16]:
```
subsection_row = demo_table[2:5] # by rows.
```

In [17]:
```
subsection_row2 = demo_table[ [1,5,3] ]
```

In [18]:
```
subsection_both = demo_table["name", "mag_b"] [1:5]
```

Changing elements inside a Table

- You know how to access columns, rows and individual elements.
- Using = sign, you can assign the selected col, row or element another value.

So,

```
demo_table["name"] = ... list of 10 names
demo_table["name"] = "SingleName"
```

will both work.

In [19]:
```
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [20]:

```
demo_table["name"] = "X"
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

Adding New Columns

In [21]:
```
# Method 1
demo_table["NewColumn"] = range(len(demo_table))
print (demo_table)
```

```
name  obs_date  mag_b mag_v NewColumn
---- ---------- ----- ----- ---------
   X 2012-01-02  17.0  17.5         0
   X 2012-01-02  17.1  17.4         1
   X 2012-01-02  15.1  13.5         2
   X 2012-02-14  16.2  14.5         3
   X 2012-02-14  16.9  17.3         4
   X 2012-02-14  15.2  15.5         5
   X 2012-02-14  15.0  13.6         6
   X 2012-03-26  15.7  16.5         7
   X 2012-03-26  15.1  13.5         8
   X 2012-03-26  14.8  14.3         9
```

In [22]:

```
# Method 2, using Column Object
from astropy.table import Column
newcol = Column( data = range(len(demo_table)), name = "2ndNewColN")
demo_table.add_column(newcol, index = 0)
print (demo_table)
```

```
2ndNewColN name  obs_date  mag_b mag_v NewColumn
---------- ---- ---------- ----- ----- ---------
         0    X 2012-01-02  17.0  17.5         0
         1    X 2012-01-02  17.1  17.4         1
         2    X 2012-01-02  15.1  13.5         2
         3    X 2012-02-14  16.2  14.5         3
         4    X 2012-02-14  16.9  17.3         4
         5    X 2012-02-14  15.2  15.5         5
         6    X 2012-02-14  15.0  13.6         6
         7    X 2012-03-26  15.7  16.5         7
         8    X 2012-03-26  15.1  13.5         8
         9    X 2012-03-26  14.8  14.3         9
```

Removing Columns

In [23]:
```
demo_table.remove_columns(["NewColumn","2ndNewColN"])
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

For Rows

Similar functions exist. Please read documentation for details. Or explore using iPython.

```
demo_table.remove_row(5)
demo_table.remove_rows( [5,6])
demo_table.remove_rows( slice(3,6) )
```

Table Sorting

In [24]:
```
demo_table = Table.read("demo.txt", format="ascii")
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [25]:
```
demo_table.sort(["name", "mag_b"]) # sort by name, then magb
```

In [26]:
```
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
M101 2012-03-26  14.8  14.3
M101 2012-02-14  15.0  13.6
M101 2012-01-02  15.1  13.5
M101 2012-03-26  15.1  13.5
 M31 2012-02-14  16.9  17.3
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
 M82 2012-02-14  15.2  15.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  16.2  14.5
```

In [27]:
```
demo_table.reverse() # Reverse existing table. Descending order!
print (demo_table)
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M82 2012-02-14  16.2  14.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  15.2  15.5
 M31 2012-01-02  17.1  17.4
 M31 2012-01-02  17.0  17.5
 M31 2012-02-14  16.9  17.3
M101 2012-03-26  15.1  13.5
M101 2012-01-02  15.1  13.5
M101 2012-02-14  15.0  13.6
M101 2012-03-26  14.8  14.3
```