Worming your way around Python

Introduction

The aim of this problem set is to exercise your python know-how and also to get some experience with another useful program, topcat.

Getting started

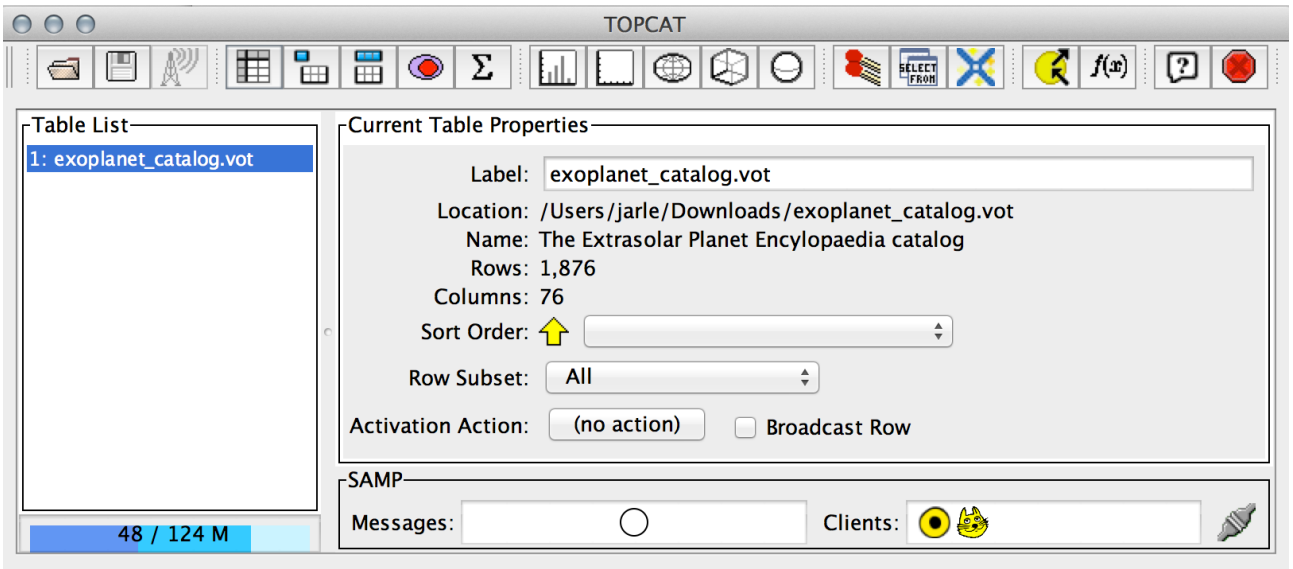
Getting some data
The first task is to get hold of some data to plot - the focus here will be on tabular data for now and we will use the Exoplanet encyclopaedia at exoplanet.eu. Go to that site and download the catalogue in VOTable format.

Getting topcat
The topcat program is a super-useful utility program that everyone working with astronomical data should be aware of.

The first task is to start-up Topcat. It might be installed already but if you need it for your own computer you can find it at:

```
http://www.star.bris.ac.uk/~mbt/topcat/
```

Go there and download the topcat-full.jar file and start this. If this is presenting problems, try the WebStart version a bit further down on the page. If it is installed, just open a terminal window and type topcat. All ok? You should see a display like this, depending on the computer you use

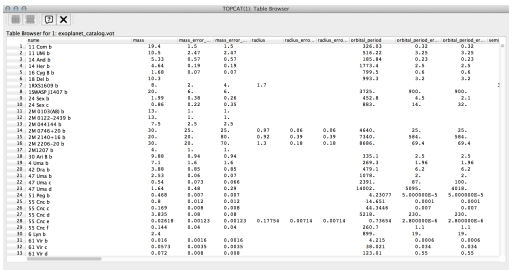


topcat can load most types of tables that astronomers use and it has a useful table viewer and convenient plotting functions. These will be described a bit in a lecture but it is useful to know how to do the basics:

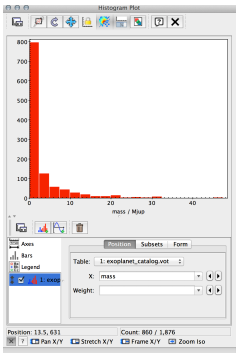
These buttons are used to show the data in table format (left-most), see generic information about the table (second button), see an overview of columns (third button) and to define subsets of the data (right-most).

These buttons are used to plot the data. The first creates histograms and the second plots scatter plots (x versus y).

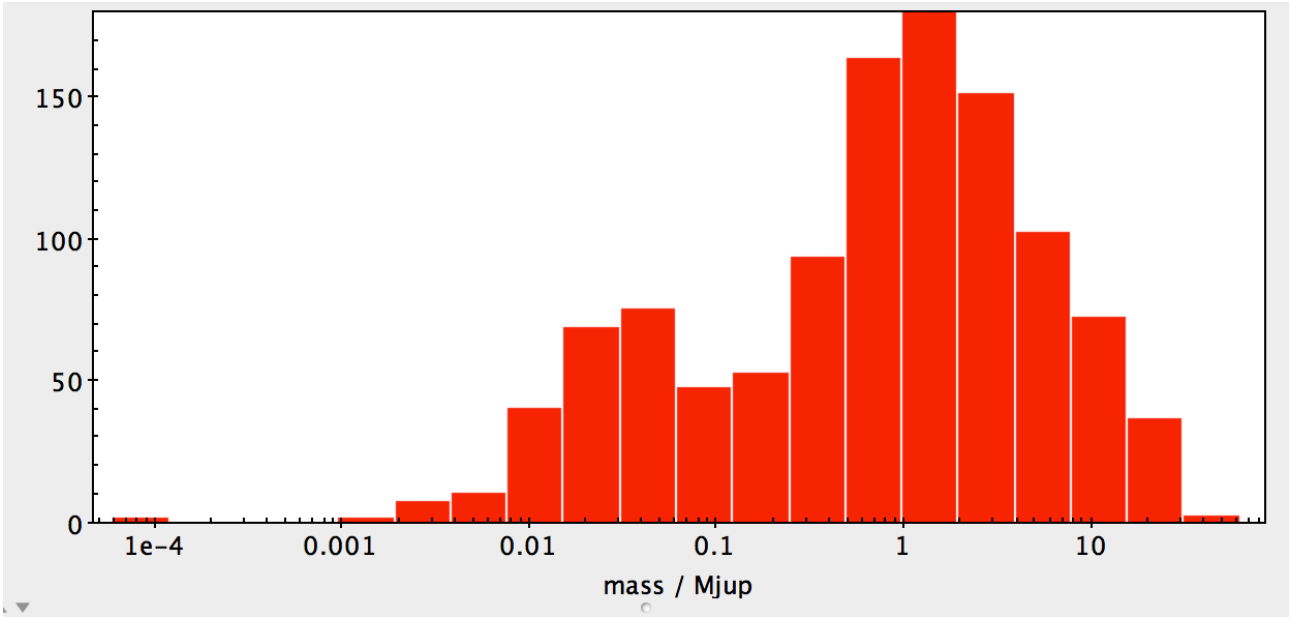
Click on the table button to bring up a display that looks like this:



you can use this interface to browse the table and when you have plotted columns in this table you can click on a table row to highlight that object in the plot, and vice versa. But for now, click the histogram button to create a histogram of the mass of the exoplanets:



That actually looks a bit sub-optimal. You can see that there are a lot of low-mass exo-planets but they all end up in a single bin on the left. In cases like this it is often useful to use a logarithmic x-axis, so click on the ‘Axes’ option in the left panel and tick the option for X log in the panel. This should give you a result looking like this:



Moving to python

Topcat is very versatile and useful but a key aspect of doing science is reproducibility - that means that when you present a result, you have to provide enough information for the interested reader to be able to repeat your investigation. When you use an interactive program like topcat this is difficult to ensure, thus we often use topcat to get a feel for the data and then move to python to visualise and analyse the data properly.

The first step is therefore to read the data into python. It is possible to read a VOTable directly into python and feel free to figure that out, but in the introduction lecture you saw how to read it in in either FITS or CSV format. To make use of this you first need to use topcat to save the table in one of those formats.

pyfits.getdata(file, 1)

Done? (you can of course also download the table in CSV format from the exoplanet.eu web site!) In my examples below I have read the data from a FITS table into a single variable, t.

Intermezzo 1: Working with ipython and friends

An efficient working environment is always important - most of us do not prepare dinner in our bathroom, and so it is with python as well. There are many ways to interact with python and integrated coding environments abound. It is not the aim of this course to guide you through these - you need to explore on your own, but it is probably a good idea to look at ipython.

This is a replacement for the normal python command line - if you type python on the command line you get a display more or less like this:

```
leukermeer [33] > python
Python 2.7.8 (default, Nov 10 2014, 08:19:18)
[GCC 4.9.2 20141101 (Red Hat 4.9.2-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

you can then proceed to type in commands, and it will work fine. However it is quite limited in what it provides and ipython (ipython.org) is a powerful alternative, although it takes a bit longer to start up. If you start ipython you get:

```
leukermeer [34] > ipython
Python 2.7.8 (default, Nov 10 2014, 08:19:18)
Type "copyright", "credits" or "license" for more information.

IPython 1.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

which already tells you there are is more help to be had. One particularly nice aspect is that it helps you find out what an object can do. As an example let us see what the functions are for a normal numpy variable:

```
In [1]: import numpy as np
```

```
In [2]: a = np.zeros(5)

In [3]: a.<PRESS TAB HERE!>
a.T             a.clip          a.dot           a.item          a.prod          a.setasflat     a.swapaxes
a.all           a.compress      a.dtype         a.itemset       a.ptp           a.setfield      a.take
a.any           a.conj          a.dump          a.itemsize      a.put           a.setflags      a.tofile
a.argmax        a.conjugate     a.dumps         a.max           a.ravel         a.shape         a.tolist
a.argmin        a.copy          a.fill          a.mean          a.real          a.size          a.tostring
a.argsort       a.ctypes        a.flags         a.min           a.repeat        a.sort          a.trace
a.astype        a.cumprod       a.flat          a.nbytes        a.reshape       a.squeeze       a.transpose
a.base          a.cumsum        a.flatten       a.ndim          a.resize        a.std           a.var
a.byteswap      a.data          a.getfield      a.newbyteorder  a.round         a.strides       a.view
a.choose        a.diagonal      a.imag          a.nonzero       a.searchsorted  a.sum
```

so as you can see - if you press <TAB> you get a list of what functions you have. So you can now find out what the sum of this variable is:

```
In [3]: a.sum()
Out[3]: 0.0
```

You can also write your code in some text editor and then paste it into ipython - to do that you select the text and then type %paste in ipython to glue it in.

Finally, another very powerful aspect of ipython is its notebook facility. Type ipython notebook on the command prompt and if all goes well your web browser will display a notebook interface. This works like a python prompt but it also supports interspersed text and comments - try it out! You do not need to use this for this course but you might find that it is a time-saver and a good way to remember what you did!

How can I do this if it was not in the lectures?

Some of the problems here (and when you do your projects) will require python skills that were not taught in the introduction lecture.

In such cases, Google (or any other search engine) is your friend - often the answers come from stackoverflow.com. Another crucial source is the python documentation (docs.python.org and docs.scipy.org) - you should keep links to these handy!

Task 1: How many of the planets in your list were found by Kepler?

Assume that any exoplanet with a name that starts with 'Kepler' was detected by Kepler and use Python to count the occurrences.

I found a total of 937 - by the time you do this there might be more.

Task 2: What is the mean mass of the planets in the exoplanet.eu catalogue?

Do this in two ways:
1.Find out what the appropriate numpy function is called and use that.
2.Write a function in python to do the same calculation.

Intermezzo: modularity of code

It is almost always best to split your code into smaller units - at the very least using functions. As an example, here is a **bad** way to calculate the sum of the N first numbers from 0 to 5:

```
x = np.arange(1, 1+1)
print “Up to {0} the sum is {1}”.format(1, x.sum())

x = np.arange(1, 2+1)
print “Up to {0} the sum is {1}”.format(2, x.sum())

x = np.arange(1, 3+1)
print “Up to {0} the sum is {1}”.format(3, x.sum())

x = np.arange(1, 4+1)
print “Up to {0} the sum is {1}”.format(4, x.sum())
```

It works, but it is easy to make mistakes and later changes to the code are harder and for anything but the simplest examples this is hard to maintain and understand.

For that reason it is best to split off tasks like this into functions. The task above can be written:

```
def sumN (N):
    x = np.arange(1, N+1)
    print "Up to {0} the sum is {1}".format(N, x.sum())

for i in range(1, 5):
      sumN(i)
```

This is now much easier to generalise and modifications are easy as well because everything is in one particular place.

The bottom line: You almost always want to use functions! And cut-and-paste of code is a bad habit to get into!

Task: Exoplanets in the Hertzsprung-Russel diagram

The Hertzsprung-Russel (HR) diagram is a convenient way to visualise stellar populations. A typical plot of this shows the effective temperature of a star on the x-axis and the luminosity of the star in some band on the y-axis - both typically on logarithmic scales.

a) Plot a HR diagram for the host stars of the exoplanets. It is perfectly fine if the luminosity axis is in arbitrary units - indeed it is perhaps easiest to convert mag_v to absolute magnitude.

   There are two points off at around 30,000K - what kind of stars are those?

   Check: Did you add axis labels?

Reversing axes
The convention in astronomy is that the x axis has high temperatures on the left and low on the right - the opposite of the default.

It is easy to do this matplotlib:

```
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.gca().invert_yaxis()
```

b) *More challenging:* That is a fairly low information plot - now let us add information to the plot by colouring each point according to the logarithm of the mass of the exo-planet.

c) *More challenging:* Next create the plot but now scale the symbol size according to the radius of the star. Does your diagram look anything like one of the HR diagrams you find on the net?

The two last points are a bit harder because you need to search a bit around and also making the plot look good is more work. They are not intrinsically hard though.

d) If that was all easy, try finding all stars with >2 planets identified and create a plot illustrating all these extra-solar systems and compare to our own solar system.

Final aside: Comments

It is very important to add comments and instructions to your code. Try to make it a habit that you always add a comment that says what a function does when you write one. Add comments everywhere you write an equation that is anything but extremely trivial.

Comments are also very useful for yourself later, for others using your code and for anyone who you ask to help you with your code!

In python you add comments by placing # before and you add help text to a function using a structure of this kind:

```
def my_func(x):
   """This function will carry out a fancy function.
x – The argument to this function
   """

   return x*x
```

In essence it should be a string literal as the first argument after the declaration of the function. This then becomes the help text so if you later do help(my_func) you will get:

```
In [3]: help(my_func)
Help on function my_func in module __main__:

my_func(x)
    This function will carry out a fancy function
    x -- the argument to this function
```

And this is very useful, particularly for other people and for complex functions. If you want to learn more about this see https://www.python.org/dev/peps/pep-0257/.

If you got this far you could turn the page and have a look at a simple function to make the Hertzsprung-Russel diagram.

It is also useful to know that these particular tasks are easier to do in topcat than in Python - but now you at least have a record of what you have done and can modify it easily in the future.

An example solution of the HR plot

```
def showHR(t):
    """
     Show a Hertzsprung-Russel diagram from the data in the
     table dictionary read from a FITS table given as argument t.
    """

    # First get an absolute magnitude
    Vabs = t['mag_v'] - 5*np.log10(t['star_distance']/10.0)
    Teff = t['star_teff']

    # Not all points are acceptable - select only the good ones but
    # report what fraction was bad.

    good,  = np.where((Teff > 1500) & (Vabs < 30) & (Vabs > -5))
    n_good = len(good)
    n_all = len(Teff)
    n_bad = n_all - n_good
    frac_bad = n_bad/np.float(n_all)

    print "I will plot {0:d} points, that is {1:.1f}% of the
total".format(n_good, 100*frac_bad)

    # Subset the variables to only use the good ones.
    Teff = Teff[good]
    Vabs = Vabs[good]


    # Set up the plot area. We use red dots for the stars
    plt.plot(Teff, Vabs, 'r.')

    # Add an x and y label to the plot.
    plt.xlabel(r'$T_{eff} [K]$')
    plt.ylabel(r'$M_{V}$')

    # Most stars have tempreature <40000K and >1500 so I put these as limits
    plt.xlim(1500, 40000)

    # And use a logarithmic x-axis.
    plt.xscale('log')

    # Invert the x and y-axes to follow astronomical convention
    plt.gca().invert_xaxis()
    plt.gca().invert_yaxis()

    # Finally, show the plot.
    plt.show()
```