AstroQuery

A less tedious way of swimming in data

by *P. Figueira* and *D. Andreasen*

What is Astroquery?

- **astroquery** is a package of **astropy**, a python-based library for astronomy.
- It requires your standard python packages (numpy, sciypy, ...) and can be installed using pip or conda

```
$ pip install astroquery

$ conda install -c astropy astroquery
```

Databases available

- SIMBAD Queries (astroquery.simbad)
- VizieR Queries (astroquery.vizier)
- Besancon Queries (astroquery.besancon)
- NIST Queries (astroquery.nist)
- ESO Queries (astroquery.eso)
- ALMA Queries (astroquery.alma)
- NASA ADS Queries (astroquery.nasa_ads)
- ...

How can I use it?

- It is very easy, and you can manipulate the quantities as python arrays in an almost painless way... but it helps to be familiar with the astropy ***Table*** format.
- A Table is a 2D structure composed of columns, each representing a quantity, and rows, each representing an entrance. The rows share the same properties, but each column can represent different variables, having different names, formating, and units. It's like *starbase* for python (but better).

http://docs.astropy.org/en/stable/table/

Tables

- You can define and manipulate Tables in different ways

```
>>> from astropy.table import Table
>>> a = [1, 4, 5]
>>> b = [2.0, 5.0, 8.2]
>>> c = ['x', 'y', 'z']
>>> t = Table([a, b, c], names=('a', 'b', 'c'), meta={'name': 'first table'})
>>>#an alternative, row oriented version is
>>> data_rows = [(1, 2.0, 'x'),
...              (4, 5.0, 'y'),
...              (5, 8.2, 'z')]
>>> t = Table(rows=data_rows, names=('a', 'b', 'c'), meta={'name': 'first table'},
...           dtype=('i4', 'f8', 'S1'))

>> t['b'].unit = 's'
>>> t
<Table length=3>
  a      b       c
         s
int32 float64 str1
----- ------- ----
    1     2.0    x
    4     5.0    y
    5     8.2    z
```

Tables

- The unit capability is very useful and comes handy often

```
>>> t['b'].quantity
<Quantity [ 2. , 5. , 8.2] s>
>>> t['b'].to('min')
<Quantity [ 0.03333333, 0.08333333, 0.13666667] min>

>>> from astropy.time import Time
>>> from astropy.coordinates import SkyCoord
>>> tm = Time(['2000:002', '2002:345'])
>>> sc = SkyCoord([10, 20], [-45, +40], unit='deg')
>>> t = Table([tm, sc], names=['time', 'skycoord'])
>>> t
<Table length=2>
         time          skycoord
                       deg,deg
        object          object
--------------------- ----------
2000:002:00:00:00.000 10.0,-45.0
2002:345:00:00:00.000  20.0,40.0
```

Tables

- In a python-like way, you can access in a brute-force way the elements

```
>>> t['b'].format = '7.3f'
>>> t['a']       # Column 'a'
<Column name='a' dtype='int32' length=3>
1
4
5

>>> t['a'][1]    # Row 1 of column 'a'
4
>>> t[1]         # Row object for table row index=1
<Row index=1>
  a      b     c
         s
int32 float64 str1
----- ------- ----
    4   5.000    y


>>> t[1]['a']    # Column 'a' of row 1
4
>>> t['a'] = [-1, -2, -3]       # Set all column values
>>> t['a'][2] = 30              # Set row 2 of column 'a'
>>> t[1] = (8, 9.0, "W")        # Set all row values
```

An example using SIMBAD

- You can perform your typical SIMBAD† queries from a python shell

```
>>> from astroquery.simbad import Simbad
>>> result_table = Simbad.query_object("m1")
>>> print(result_table)

MAIN_ID      RA          DEC     RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA COO_ERR_ANGLE COO_QUAL
COO_WAVELENGTH     COO_BIBCODE
------- ---------- ---------- ------- -------- ------------ ------------ ------------- --------
-------------- -------------------
M   1 05 34 31.94 +22 00 52.2       6        6          nan          nan             0        C
       R 2011A&A...533A..10L

>>> result_table = Simbad.query_region("m81")
>>> print(result_table)

            MAIN_ID                 RA           DEC      RA_PREC DEC_PREC ... COO_ERR_ANGLE COO_QUAL
COO_WAVELENGTH     COO_BIBCODE
------------------------- ------------- ------------- ------- -------- ... ------------- --------
-------------- -------------------
[VV2006c] J095534.0+043546 09 55 33.9854 +04 35 46.438       8        8 ...             0        B
          O 2009A&A...505..385A
...
```

†http://simbad.u-strasbg.fr/simbad/sim-fid

An example using SIMBAD

- The Table format is very powerful and allows you to use different units and queries.

```
>>> from astroquery.simbad import Simbad
>>> import astropy.units as u
>>> result_table = Simbad.query_region("m81", radius=0.1 * u.deg)
>>> # another way to specify the radius.
>>> result_table = Simbad.query_region("m81", radius='0d6m0s')

>>> #and another way using a center in ICRS coordinates
>>> import astropy.coordinates as coord
>>> result_table = Simbad.query_region("05h35m17.3s -05h23m28s", radius='1d0m0s')
>>> print(result_table)

        MAIN_ID               RA      ...     COO_BIBCODE
----------------------- ------------- ... -------------------
             HD  38875 05 34 59.7297 ... 2007A&A...474..653V
         TYC 9390-799-1 05 33 58.2222 ... 1998A&A...335L..65H
         TYC 9390-654-1  05 35 27.395 ... 2000A&A...355L..27H
         TYC 9390-656-1  05 30 43.665 ... 2000A&A...355L..27H
                    ...           ... ...                 ...
         TYC 9373-779-1  05 11 57.788 ... 2000A&A...355L..27H
         TYC 9377-513-1 05 10 43.0669 ... 1998A&A...335L..65H
```

An example using SIMBAD

- You can do things that range from useful to scary

```
>>> from astroquery.simbad import Simbad
>>> result_table = Simbad.query_bibcode('2013A&A*', wildcard=True)
>>> print(result_table)

References
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
2013A&A...549A...1G  --  ?
Astron. Astrophys., 549A, 1-1 (2013)
GENTILE M., COURBIN F. and MEYLAN G.
Interpolating point spread function anisotropy.
Files: (abstract) (no object)
...

>>> result_table = Simbad.query_objectids("Polaris")
>>> print(result_table)
         ID
--------------------
        ADS  1477 AP
        ** STF   93A
    WDS J02318+8916A
   ...
```

Common traps

- Remember that the query configurations will lead to different results!

```
>>> from astroquery.simbad import Simbad
>>> Simbad.ROW_LIMIT = 15 # now any query fetches at most 15 rows
>>> Simbad.TIMEOUT = 60 # sets the timeout to 60s – standard is 100s!
```

- See all this and much more in:

https://astroquery.readthedocs.org/en/latest/simbad/simbad.html

More Complex...

- Stuff I did to prepare ESPRESSO GTO observations

```
from astroquery.vizier import Vizier
columns_HIP =['HIP', 'HD', 'RAhms', 'DEdms', 'SpType', 'r_SpType', 'B-V', 'HvarType', 'VarFlag', 'Ncomp', 'Vmag', 'Plx']
criteria =  {"Vmag":"<14.5", "DEdms":"<30", "VarFlag":"=|null|1", "HvarType":"=|C|M|R|U", 'Ncomp':"=1"}
HIP_catalog = Vizier(catalog="I/239/hip_main", columns = columns_HIP, column_filters = criteria, row_limit=-1)
catalog_formated = HIP_catalog.query_constraints(Vmag="-5.0..15.5")[0]

columns_LG =['PM', 'HIP', 'CNS3', 'RAJ2000', 'DEJ2000', 'Vmag', 'Jmag', 'Hmag', 'Kmag', 'plx', 'pplx']
criteria =  {"Vmag":"<14.5", "DEJ2000":"<30"}
HIP_catalog = Vizier(catalog="J/AJ/142/138", columns = columns_LG, column_filters = criteria, row_limit=-1)
catalog_formated = HIP_catalog.query_constraints(Vmag="-5.0..15.5")[0]

#adapting columns
catalog_formated['pplx'].name = 'Plx'

if(store):
    catalog_formated.write(firstcat_output, format="ascii", delimiter="\t")
    print("The Hipparcos/first catalog is stored in the file %s and has %d entries." %( firstcat_output,
len(catalog_formated)))
```

More Complex...

- Stuff I did to prepare ESPRESSO GTO observations (not all of these are recommendable)

```
for value in zip(catalog_formated['HIP'], catalog_formated['_RAJ2000'], catalog_formated['_DEJ2000']):
        #does a region query for each of the entries
        result = Vizier(columns = columns_TwoMASS).query_region(coord.SkyCoord(ra=value[1], dec=value[2],\
        unit=(u.deg, u.deg), frame='icrs'), radius=Angle(5, "arcsec"), catalog='II/246')

    columns_UCAC4 =['RAJ2000', 'DEJ2000', 'Vmag', 'Jmag', 'Hmag', 'Kmag', 'H', 'Tycho-2', '2Mkey']
    criteria =  {"Vmag":"<14.5", "DEJ2000":"<30", "db":"0"}
    # note that          db: UCAC4 double star flag
    UCAC4_catalog = Vizier(catalog="I/322A/out", columns = columns_UCAC4, row_limit=-1, timeout=-1)

    print "\nDigging UCAC4 catalog from Vizier..."
    cat1 = UCAC4_catalog.query_region("0d 0d", radius="10d")
    cat2 = UCAC4_catalog.query_region("0d 0d", inner_radius="10d", radius="20d")
    print "... 2/18 regions done."
    cat3 = UCAC4_catalog.query_region("0d 0d", inner_radius="20d", radius="30d")
    cat4 = UCAC4_catalog.query_region("0d 0d", inner_radius="30d", radius="40d")
   ...
```

A python program

- Take a look at this program:

https://github.com/DanielAndreasen/astro_scripts/blob/master/vizier_query_cli.py