Naive Cross-Matching

The **BSS catalogue lists the brightest sources from the AT20G radio survey** while the **SuperCOSMOS catalogue lists galaxies observed by visible light surveys**. If we can **find an optical match for our radio source**, we are one step closer to working out what kind of object it is, e.g. a galaxy in the local Universe or a distant quasar.

https://commons.wikimedia.org/wiki/File%3ARa_and_dec_demo_animation_small.gif

The positions of stars, galaxies and other astronomical objects are usually recorded in either **equatorial** or **Galactic** coordinates. Equatorial coordinates are fixed relative to the celestial sphere, so the positions are independent of when or where the observations took place. They are defined relative to the celestial equator (which is in the same plane as the Earth's equator) and the ecliptic (the path the sun traces throughout the year).
A point on the celestial sphere is given by two coordinates:

- **Right ascension**: the angle from the vernal equinox to the point, going east along the celestial equator;
- **Declination**: the angle from the celestial equator to the point, going north (negative values indicate going south).

The vernal equinox is the intersection of the celestial equator and the ecliptic where the ecliptic rises above the celestial equator going further east.

Right Ascension is often given in **Hours-Minutes-Seconds (HMS) notation,** because it is convenient to calculate when a star would appear above the horizon. a full circle in HMS notation is **24h,** 1hr in HMS notation is 15 degrees.

You can convert 23 hours, 12 minutes and 6 seconds (written as `23:12:06` or `23h12m06s`) to degrees like this:

print(15*(23 + 12/60 + 6/(60*60)))
~ 348.025

Declination, on the other hand, is traditionally recorded in **degrees-minutes-seconds** (DMS) notation. A full circle is 360 degrees, each degree has 60 arcminutes and each arcminute has 60 arcseconds.
For example: 73 degrees, 21 arcminutes and 14.4 arcseconds (written `73:21:14.4` or `73° 21' 14.4"` or `73d21m14.4s`) can be converted to decimal degrees like this:

print(73 + 21/60 + 14.4/(60*60))
`~ 73.354`
With negative angles like `-5° 31' 12"` the negation applies to the whole angle, including arcminutes and arcseconds:

print(-1*(5 + 31/60 + 12/(60*60)))
`-5.52`

Angular Distance

To crossmatch two catalogues we need to compare the angular distance between objects on the celestial sphere.
People loosely call this a "distance", but technically its an *angular* distance: the projected angle between objects as seen from Earth.

If we have an object on the celestial sphere with right ascension and declination (α1,δ1), then the angular distance to another object with coordinates (α2,δ2) is:

$$d = 2 \arcsin \sqrt{\sin^2 \frac{|\delta_1 - \delta_2|}{2} + \cos \delta_1 \cos \delta_2 \sin^2 \frac{|\alpha_1 - \alpha_2|}{2}}$$

Angular distances have the same units as angles (degrees). There are **other equations** for calculating the angular distance but this one, called the
haversine formula, is good at avoiding fl oating point errors when the two points are close together.

Segmentation of the formula: (In terms of Numpy)

```
a = np.sin(np.abs(d1 - d2)/2)**2
b = np.cos(d1)*np.cos(d2)*np.sin(np.abs(r1 - r2)/2)**2
d = 2*np.arcsin(np.sqrt(a + b))
```

Note

Trig functions in most languages and libraries (including Python and NumPy) take angle arguments in units of radians, but the databases we're working with use angles of degrees.
Fortunately, NumPy provides convenient conversion functions:

```
a_rad = np.radians(a_deg)
a_deg = np.degrees(a_rad)
```

The variable `a_deg` is in units of degrees and `a_rad` is in radians.

Reading Data

Before we can crossmatch our two catalogues we first have to import the raw data. Every astronomy catalogue tends to have its

own unique format so we'll need to look at how to do this with each one individually.
We'll look at the AT20G bright source sample survey first. The raw data we'll be using is the file `table2.dat` from **this page** in the VizieR archives, but we'll use the filename `bss.dat` from now on.
Every catalogue in VizieR has a detailed README file that gives you the exact format of each table in the catalogue.

.dat files
The catalogue is organised in *fixed-width* columns, with the format of the columns being:

- **1**: Object catalogue ID number (sometimes with an asterisk)
- **2-4**: Right ascension in HMS notation (inclusive range)
- **5-7**: Declination in DMS notation
- **8-**: Other information, including spectral intensities

np.loadtxt('bss.dat',usecols = range(1,7))
~since at index 0 there are row index values which we don't require

NOTE

Fixed-width columns and `loadtxt`

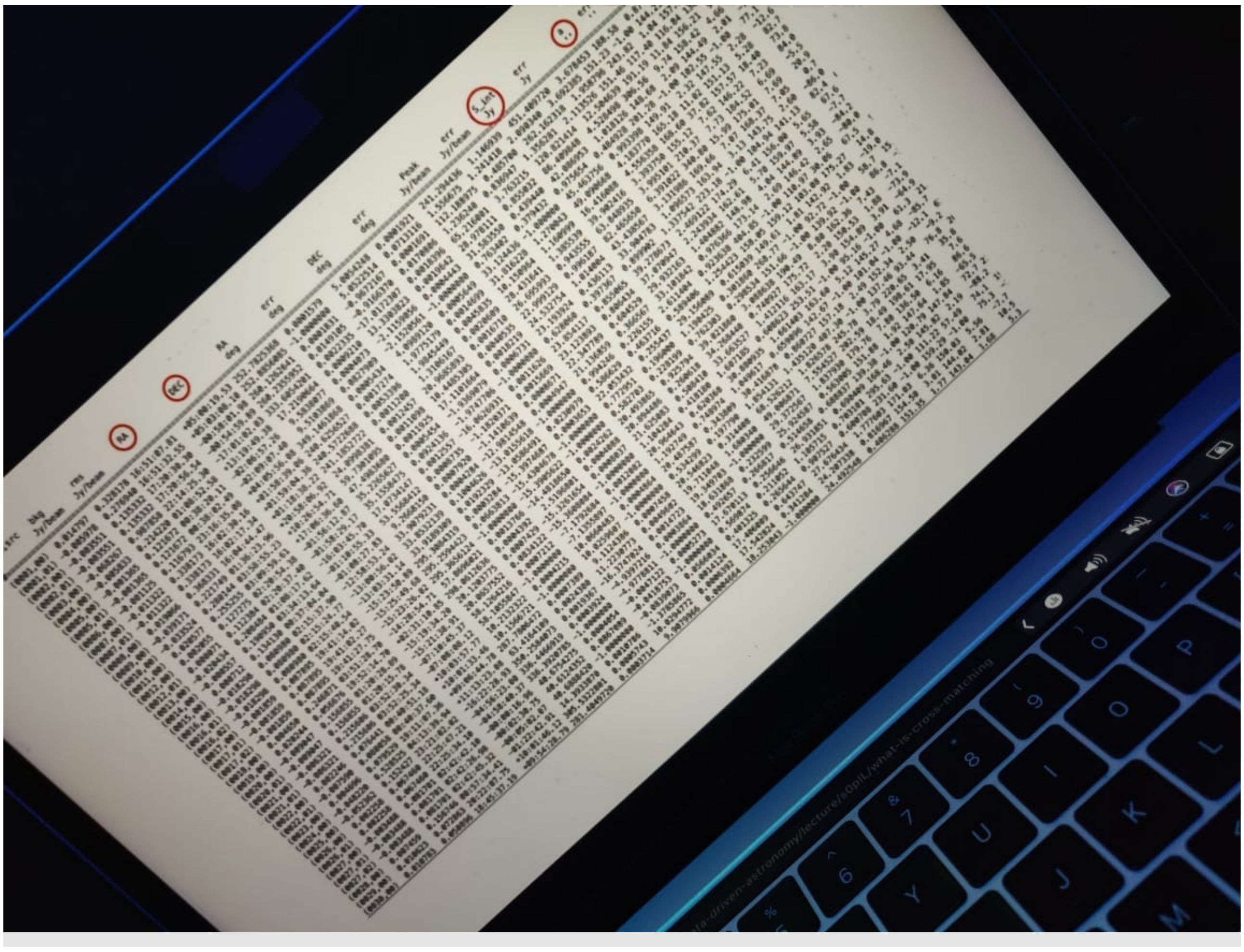
`loadtxt` does not work for fixed-width columns if values are missing. Since there are no missing ID, RA and dec values it is fine for loading the first few columns of the BSS catalogue.

super.csv
The SuperCOSMOS all-sky catalogue is a catalogue of galaxies generated from several visible light surveys.
The original data is available on **this page** in a package called `SCOS_XSC_mCl1_B21.5_R20_noStepWedges.csv.gz`.
Because this catalogue is so large, we've cut it down for these

activities. The cut down version of the file will be named `super.csv`.

cosmoscat =
np.loadtxt('super.csv',delimiter=',',skiprows=1,usecols=[0,1])
row[0] = ascension values
row[1] = declination values

The first few lines of `super.csv` look like this:



Look Around You ..

closest_point = (id_val=None, np.inf)

often basic method to determine the closest point is to find the angular distance between the input **Right Ascension point, Declination** and **all the other RA, DEC of the .dat file** np.inf declared as a reference float64 point, angular distance less than that will our result

Putting it all together :
<u>Cross Match</u>

Crossmatching method is used to match the closest match from one **Electromagnetic survey to the other.**

What I did :

```
def
crossmatch(catalogue1(radio),catalogue2(optical),
max_dist):
```
# max_dist is a preset value ==> for first catalogue its 40/3600, second its 5/3600
```
for ascension, declination in catalogue1:
closest = find_closest(catalogue2, ascension,
declination)
```
# find closest returns, the closest point in the parameterised catalogue and its id as such (id, RA,DEC)
```
if (closest[1] > max_dist) :
nomatchlist.append(datapoint[0])
else:
matchlist.append(datapoint[0],closest[0],closest[1
])
```