Cross-Matching

- Neelaksh Sharma

Uses Of Cross Matching

- Cross matching refers to the comparison of different objects or data.

- In astronomy, we use cross-matching to compare data of objects (galaxies, stars), obtained from different telescopes at different wavelengths.

- Positional cross-matching gives us the closest counterpart of the object within a specified radius in the sky.

- We use two catalogs (astronomical databases) here:

    1. The AT20G Bright Source Sample (BSS) catalog, from a radio survey

    2. The SuperCOSMOS all sky galaxy catalog, from an optical survey

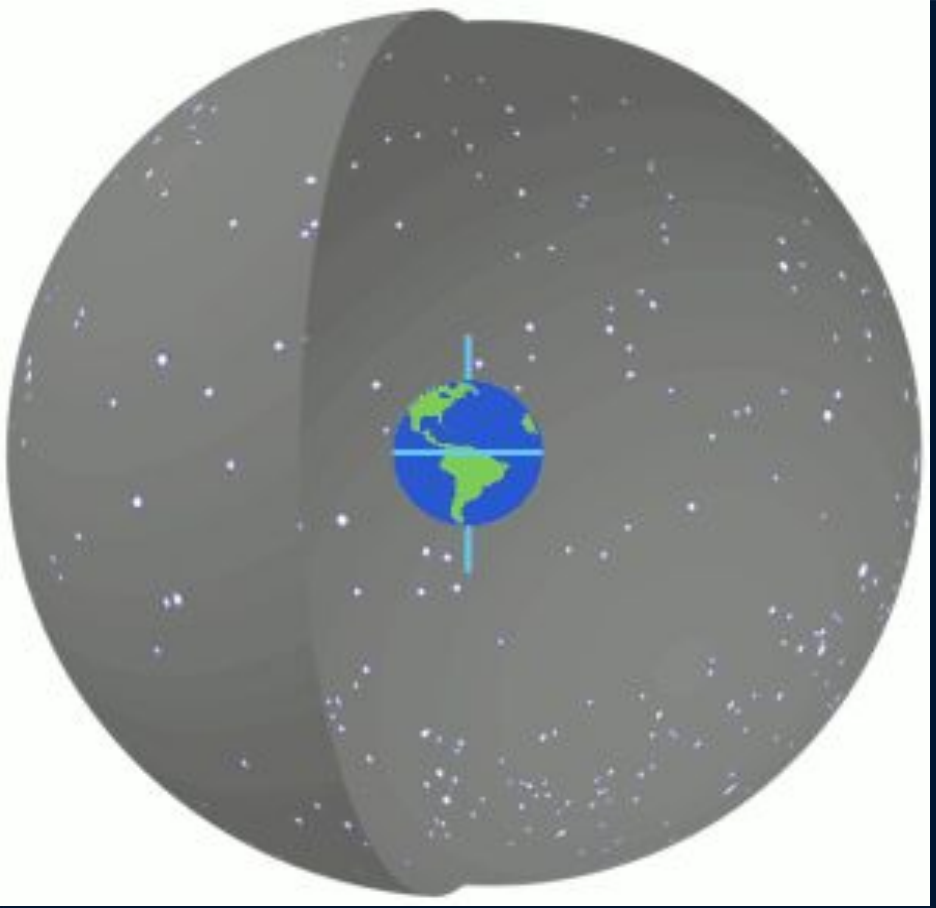
How do we find the position of an Astronomical Object

- Position of an astronomical object is referenced using Equatorial or Galactic coordinates.

- In case of equatorial coordinates, observations are independent of time and place, as they are defined relative to the celestial equator (projection of Earth's equator) and the elliptic (path of the sun throughout the year) which constitute the celestial sphere.

- Hence the point on the celestial sphere is given by two coordinates:
  - **Right ascension (RA)**, the angular distance of an object measured eastward from the Vernal Equinox. Astronomical equivalent of longitude. Given in hours-minutes-seconds (HMS) notation.
  - **Declination**, the angle subtended by an object, north or south of the celestial equator. Equivalent to latitude. Recorded in degrees-minutes-seconds (DMS) notation

***Vernal Equinox: Intersection of celestial equator and the elliptic where the latter rises above the former going east**



```
import numpy as np

def angular_dist(RA1, dec1, RA2, dec2):
  # Convert to radians
  r1 = np.radians(RA1)
  d1 = np.radians(dec1)
  r2 = np.radians(RA2)
  d2 = np.radians(dec2)

  a = np.sin(np.abs(d1 - d2)/2)**2
  b = np.cos(d1)*np.cos(d2)*np.sin(np.abs(r1 -
r2)/2)**2

  angle = 2*np.arcsin(np.sqrt(a + b))

  # Convert back to degrees
  return np.degrees(angle)
```

Angular_dist that calculates the angular distance between any two points on the celestial sphere given their right ascension and declination.

```
>>> ra1, dec1 = 21.07, 0.1
>>> ra2, dec2 = 21.15, 8.2
>>> angular_dist(ra1, dec1, ra2, dec2)
8.1003923181465041
```

```
# Write your import_bss function here.
import numpy as np

def hms2dec(hr, m, s):
  dec = hr + m/60 + s/3600
  return dec*15

def dms2dec(d, m, s):
  sign = d/abs(d)
  dec = abs(d) + m/60 + s/3600
  return sign*dec

def import_bss():
  res = []
  data = np.loadtxt('bss.dat', usecols=range(1, 7))
  for i, row in enumerate(data, 1):
    res.append((i, hms2dec(row[0], row[1], row[2]), dms2dec(row[3], row[4], row[5])))
  return res

def import_super():
  data = np.loadtxt('super.csv', delimiter=',', skiprows=1, usecols=(0, 1))
  res = []
  for i, row in enumerate(data, 1):
    res.append((i, row[0], row[1]))
  return res
```

Your functions should work like this:

```
>>> import_bss()
[(1, 1.1485416666666666, -47.60530555555556), (2,
2.6496666666666666, -30.463416666666667), (3,
2.7552916666666665, -26.209194444444442)]
>>> import_super()
[(1, 1.0583407, -52.916240199999997), (2,
2.6084425000000002, -41.500575300000001), (3,
2.7302499, -27.706955000000001)]
```

This Is A Very Lengthy Process!!!

- This cross matching procedure is very slow as for every object in BSS, we need to calculate the position of every object in SuperCOSMOS.As a result, even our simple task requires 160 x 500 = 80,000 calculations!

- Even if a single calculation takes a few microseconds, as a whole it adds up to seconds or minutes.
  - Seconds might sound good, but with the full version of SuperCOSMOS catalog, having over 126 million objects, you might have to wait quite a while.
  - And if we take a catalog other than AT20G BSS, with a size comparable to SuperCOSMOS, cross matching might take months or even years!

So what Should Be Our Plan Of Action ??

Therefore, we need to modify our algorithm, in order to make our operations efficient. As we are concerned with the efficiency of computations, we can take randomly generated catalogs instead of specific ones taken above, so as to make the algorithm better generally and not only for a specific case.

Also to improve the efficiency of the code further, we can use built in functions in the numpy library instead of using for loop for iterating over a list.

```
# Write your find_closest function here
import numpy as np

def hms2dec(hr, m, s):
  dec = hr + m/60 + s/3600
  return dec*15

def dms2dec(d, m, s):
  sign = d/abs(d)
  dec = abs(d) + m/60 + s/3600
  return sign*dec

def import_bss():
  res = []
  data = np.loadtxt('bss.dat', usecols=range(1, 7))
  for i, row in enumerate(data, 1):
    res.append((i, hms2dec(row[0], row[1], row[2]), dms2dec(row[3], row[4], row[5])))
  return res

def import_super():
  data = np.loadtxt('super.csv', delimiter=',', skiprows=1, usecols=[0, 1])
  res = []
  for i, row in enumerate(data, 1):
    res.append((i, row[0], row[1]))
  return res
```

```
>>> cat = import_bss()
>>> find_closest(cat, 175.3, -32.5)
(156, 3.7670580226469053)
```

And here's another example:

```
>>> cat = import_bss()
>>> find_closest(cat, 32.2, 40.7)
(26, 57.729135775621295)
```

find_closest function that takes a catalogue and the position of a target source (a right ascension and declination) and finds the closest match for the target source in the catalogue.