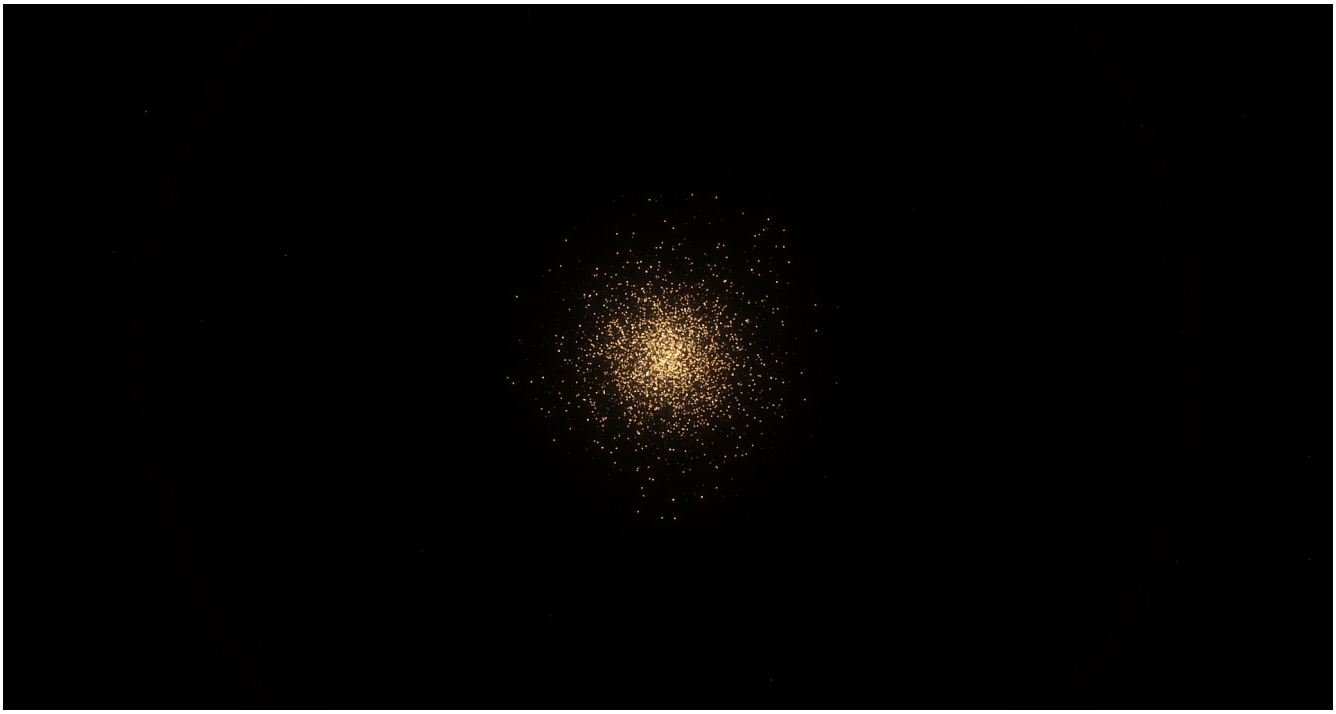
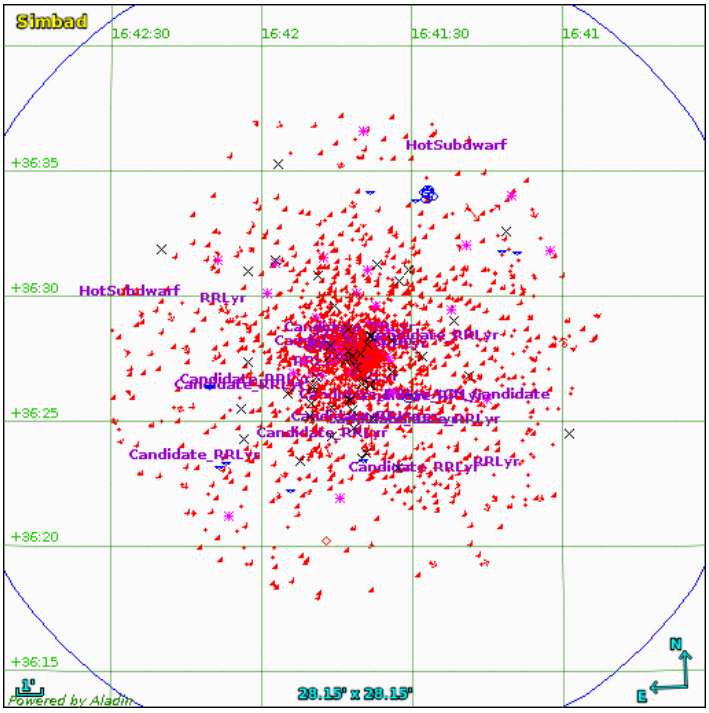
Mapping Star Clusters

2012/03/04

After talking with Dr. Seab yesterday, he thought it would be neat to make a 3d star map, where I feed some coordinates and it gives you a map of the region. I decided to start with the Hercules' Cluster since it's a dense region of space:



Researching, I found this



(http://simbad.u-strasbg.fr/simbad/sim-plot?ident=M+
+13&coo=16+41+41.634%2B36+27+40.75&radius.unit=arcmin&x=57&y=57&radius=10)

This website has an interactive map where you can click on the objects and it brings up the star in question. For example, http://simbad.u-strasbg.fr/simbad/sim-id?Ident=NGC
%20%206205%20%20%20%2045, which has information on it. I am mainly interested in a coordinate system, so I can map the stars relative to each other accurately on a plane, and the parallax, which gives me distance, so I can properly position them correctly in 3d space. It seems clicking on the outer stars gives me something I need, but the inner stars lack a parallax angle. Nonetheless, I think I can work with this.

Exploring the website for a bit, I found that you can collect the data.
http://simbad.u-strasbg.fr/simbad/sim-coo?
Coord=16+41+41.634%2B36+27+40.75&CooFrame=ICRS&CooEqui=2000.0&CooEpoch=J2000&Radius=10.0&Radius.unit=arcmin&submit=get+the+list+of+objects

There are around 9000 stars in this plot. It's way less than 300,000, which is how many are in the Hercules' cluster, but it's a great starting point and might be the best I can get it. It does have a list of distances in arcseconds, and it uses a consistent coordinate system called ICRS, or International Celestial Reference System, which seems to be very to equatorial coordinates used in hobby observational astronomy, which I am familiar with, and Dr. Seab is very familiar with, so he can possible be of great help with this part.

I noticed that there's a "Query" command being "coord 16 41 41.634+36 27 40.75 (ICRS, J2000, 2000.0), radius: 10.0 arcmin", which to me seem pretty intutitive. You choose a spot in the sky and the radius around it and it gives you your stars. Since, there seems to be no way to just download the current data. I wonder if I can pull it from their website in Python, so I will begin searching for a way to do so.

A beautiful thing about python is that there is almost always a library to do exactly what you want. In this case it's astroquery. This page tells me how to query SIMBAD, which is where I want to pull the data from.

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

To install it, I did it as you would any normal python package. I opened a terminal (Linux) and ran:

```
# pip install astroquery
```

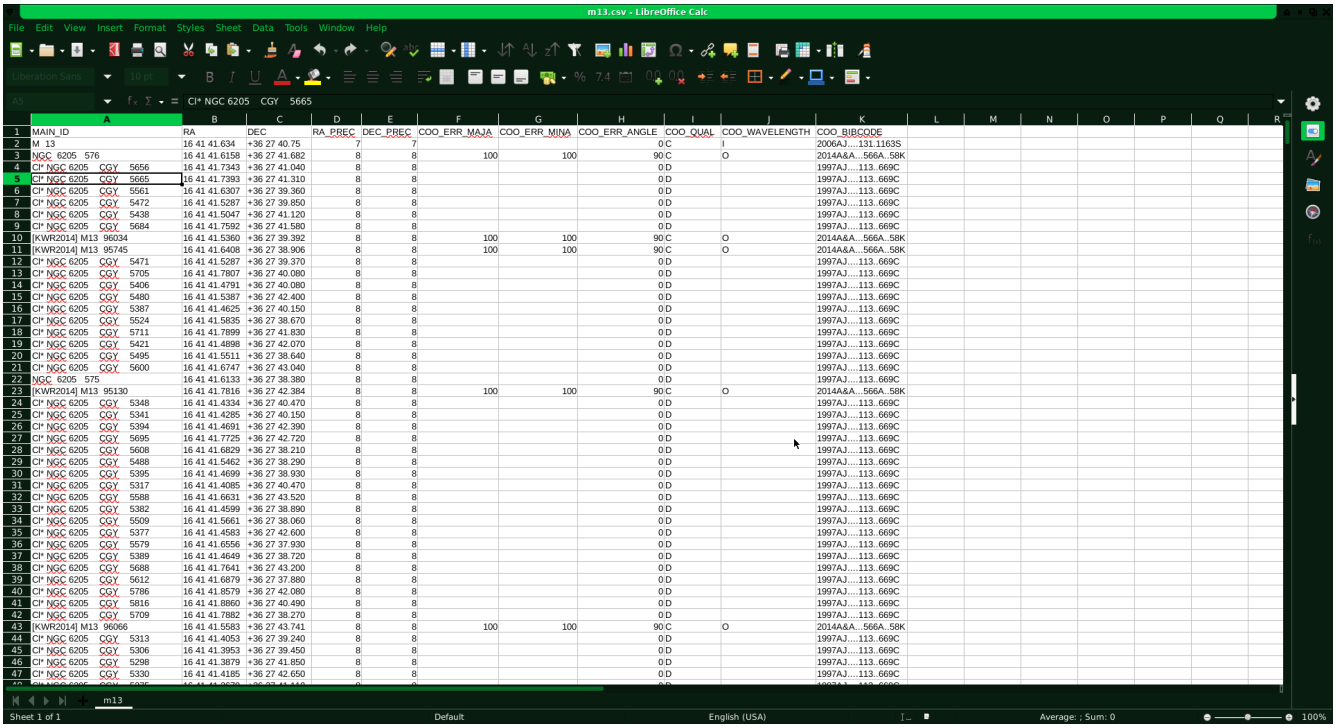
Then I ran a python3 environment and did the following commands

```
>>> from astroquery.simbad import Simbad
>>> result_table = Simbad.query_region("m13",radius='0d10m0s')
>>> print(result_table)
         MAIN_ID                RA      ...     COO_BIBCODE
                              "h:m:s"    ...
---------------------------- ------------ ... -------------------
                    M  13  16 41 41.634 ... 2006AJ....131.1163S
             NGC  6205   576 16 41 41.6158 ... 2014A&A...566A..58K
  Cl* NGC 6205    CGY    5656 16 41 41.7343 ... 1997AJ....113..669C
  Cl* NGC 6205    CGY    5665 16 41 41.7393 ... 1997AJ....113..669C
  Cl* NGC 6205    CGY    5561 16 41 41.6307 ... 1997AJ....113..669C
  Cl* NGC 6205    CGY    5472 16 41 41.5287 ... 1997AJ....113..669C
                     ...          ... ...              ...
    SDSS J164148.72+361755.2 16 41 48.7256 ... 2018yCat.1345....0G
  Cl* NGC 6205     CM       1 16 40 53.1433 ... 2018yCat.1345....0G
[SLB2011b] 250.61825+36.41084  16 42 28.381 ... 2009yCat.2294....0A
    SDSS J164212.27+361954.6 16 42 12.2725 ... 2018yCat.1345....0G
  Cl* NGC 6205    KAD     610 16 42 01.0633 ... 2018yCat.1345....0G
  Cl* NGC 6205    KAD     686 16 42 29.3988 ... 2018yCat.1345....0G
    SDSS J164228.01+362408.2 16 42 28.0162 ... 2018yCat.1345....0G
Length = 8239 rows
```

Which seems to be the same exact data I was looking for. I was able to write the data into a .csv file using Astropy

```
>>> from astropy.io import ascii
>>> ascii.write(result_table, 'm13.csv', format='csv', fast_writer=False)
```

The file can be read in excel for convenience, or in my case, since I don’t have access to excel because I use Linux, LibreOffice Calc can be used as well.



	MAIN_ID	RA	DEC	RA_PREC	DEC_PREC	COO_ERR_MAJA	COO_ERR_MINA	COO_ERR_ANGLE	COO_QUAL	COO_WAVELENGTH	COO_BIBCODE
2	M 13	16 41 41.634	+36 27 40.75	7	7			0	C	I	2006AJ....131.1163S
3	NGC 6205 576	16 41 41.6158	+36 27 41.682	8	8	100	100	90	C	O	2014A&A...566A..58K
4	Cl* NGC 6205 CGY 5656	16 41 41.7343	+36 27 41.040	8	8			0	D		1997AJ....113..669C
5	Cl* NGC 6205 CGY 5665	16 41 41.7393	+36 27 41.310	8	8			0	D		1997AJ....113..669C
6	Cl* NGC 6205 CGY 5561	16 41 41.6307	+36 27 39.360	8	8			0	D		1997AJ....113..669C
7	Cl* NGC 6205 CGY 5472	16 41 41.5287	+36 27 39.850	8	8			0	D		1997AJ....113..669C
8	Cl* NGC 6205 CGY 5438	16 41 41.5047	+36 27 41.120	8	8			0	D		1997AJ....113..669C
9	Cl* NGC 6205 CGY 5684	16 41 41.7592	+36 27 41.580	8	8			0	D		1997AJ....113..669C
10	[KWR2014] M13 96034	16 41 41.5360	+36 27 39.392	8	8	100	100	90	C	O	2014A&A...566A..58K
11	[KWR2014] M13 95745	16 41 41.6408	+36 27 38.906	8	8	100	100	90	C	O	2014A&A...566A..58K
12	Cl* NGC 6205 CGY 5471	16 41 41.5287	+36 27 39.370	8	8			0	D		1997AJ....113..669C
13	Cl* NGC 6205 CGY 5705	16 41 41.7807	+36 27 40.080	8	8			0	D		1997AJ....113..669C
14	Cl* NGC 6205 CGY 5406	16 41 41.4791	+36 27 40.080	8	8			0	D		1997AJ....113..669C
15	Cl* NGC 6205 CGY 5480	16 41 41.5387	+36 27 42.400	8	8			0	D		1997AJ....113..669C
16	Cl* NGC 6205 CGY 5387	16 41 41.4625	+36 27 40.150	8	8			0	D		1997AJ....113..669C
17	Cl* NGC 6205 CGY 5524	16 41 41.5835	+36 27 38.670	8	8			0	D		1997AJ....113..669C
18	Cl* NGC 6205 CGY 5711	16 41 41.7899	+36 27 41.830	8	8			0	D		1997AJ....113..669C
19	Cl* NGC 6205 CGY 5421	16 41 41.4898	+36 27 42.070	8	8			0	D		1997AJ....113..669C
20	Cl* NGC 6205 CGY 5495	16 41 41.5511	+36 27 38.640	8	8			0	D		1997AJ....113..669C
21	Cl* NGC 6205 CGY 5600	16 41 41.6747	+36 27 43.040	8	8			0	D		1997AJ....113..669C
22	NGC 6205 575	16 41 41.6133	+36 27 38.380	8	8			0	D		1997AJ....113..669C
23	[KWR2014] M13 95130	16 41 41.7816	+36 27 42.384	8	8	100	100	90	C	O	2014A&A...566A..58K
24	Cl* NGC 6205 CGY 5348	16 41 41.4334	+36 27 40.470	8	8			0	D		1997AJ....113..669C
25	Cl* NGC 6205 CGY 5341	16 41 41.4285	+36 27 40.150	8	8			0	D		1997AJ....113..669C
26	Cl* NGC 6205 CGY 5394	16 41 41.4691	+36 27 42.390	8	8			0	D		1997AJ....113..669C
27	Cl* NGC 6205 CGY 5695	16 41 41.7725	+36 27 42.720	8	8			0	D		1997AJ....113..669C
28	Cl* NGC 6205 CGY 5608	16 41 41.6829	+36 27 38.210	8	8			0	D		1997AJ....113..669C
29	Cl* NGC 6205 CGY 5488	16 41 41.5462	+36 27 38.290	8	8			0	D		1997AJ....113..669C
30	Cl* NGC 6205 CGY 5395	16 41 41.4699	+36 27 38.930	8	8			0	D		1997AJ....113..669C
31	Cl* NGC 6205 CGY 5317	16 41 41.4085	+36 27 40.470	8	8			0	D		1997AJ....113..669C
32	Cl* NGC 6205 CGY 5588	16 41 41.6631	+36 27 43.520	8	8			0	D		1997AJ....113..669C
33	Cl* NGC 6205 CGY 5382	16 41 41.4599	+36 27 38.890	8	8			0	D		1997AJ....113..669C
34	Cl* NGC 6205 CGY 5509	16 41 41.5661	+36 27 38.060	8	8			0	D		1997AJ....113..669C
35	Cl* NGC 6205 CGY 5377	16 41 41.4583	+36 27 42.600	8	8			0	D		1997AJ....113..669C
36	Cl* NGC 6205 CGY 5579	16 41 41.6556	+36 27 37.930	8	8			0	D		1997AJ....113..669C
37	Cl* NGC 6205 CGY 5389	16 41 41.4649	+36 27 38.720	8	8			0	D		1997AJ....113..669C
38	Cl* NGC 6205 CGY 5688	16 41 41.7641	+36 27 43.200	8	8			0	D		1997AJ....113..669C
39	Cl* NGC 6205 CGY 5612	16 41 41.6879	+36 27 37.880	8	8			0	D		1997AJ....113..669C
40	Cl* NGC 6205 CGY 5786	16 41 41.8579	+36 27 42.080	8	8			0	D		1997AJ....113..669C
41	Cl* NGC 6205 CGY 5816	16 41 41.8860	+36 27 40.490	8	8			0	D		1997AJ....113..669C
42	Cl* NGC 6205 CGY 5709	16 41 41.7882	+36 27 38.270	8	8			0	D		1997AJ....113..669C
43	[KWR2014] M13 96066	16 41 41.5583	+36 27 43.741	8	8	100	100	90	C	O	2014A&A...566A..58K
44	Cl* NGC 6205 CGY 5313	16 41 41.4053	+36 27 39.240	8	8			0	D		1997AJ....113..669C
45	Cl* NGC 6205 CGY 5306	16 41 41.3953	+36 27 39.450	8	8			0	D		1997AJ....113..669C
46	Cl* NGC 6205 CGY 5298	16 41 41.3879	+36 27 41.850	8	8			0	D		1997AJ....113..669C
47	Cl* NGC 6205 CGY 5330	16 41 41.4185	+36 27 42.650	8	8			0	D		1997AJ....113..669C

I will try to plot these stars in python so I can see if it indeed resembles M13.

END OF 2021/03/04

Week 2

3/8/2021

Dr. Seab recommended that I use an open cluster rather than a globular cluster, so I will try everything on m37 for now on, and he gave me more websites to check out. “That may be sufficient, but look at NAVO and CDS as well, and MAST of course.”

NAVO has instructions on how to pull from their catalog in Python:
https://github.com/NASA-NAVO/navo-workshop/blob/master/CS_Catalog_Queries.ipynb

Going through it, it seems rather complicated when I want to just plot a group of static stars, so I would only need location and various magnitudes in filters. SIMBAD gives me an easy way to do this.

CDS points me to SIMBAD so they might be related

MAST seems even easier to pull from. I just searched up “m37” and got all the cataloged stars in it
https://mast.stsci.edu/portal/Mashup/Clients/Mast/Portal.html

I got a CSV file from them. I will send them (and from SIMBAD) to Dr. Seab and see what he might think be better for creating a map.

03/09/2021

I will begin writing the code to plot these stars.

I can simply read the data like so

from astropy.io import ascii #importing datasets
data = ascii.read("SIMBAD_m37.csv")

for a star, the Right Ascension and Declination coordinates can be read like data[i][1] and data[i][2] where “i” is just any number.

They are not in the most friendly format: 05 52 18 and +32 33.2, for example, so I would need to convert to a usable number to plot.

Nevermind, it’s not tricky, astropy has an object called a “skycoord” and they can accept inputs like that. So to generate a skycoord for a star I do

```
from astropy import units as u
from astropy.coordinates import SkyCoord
 c = SkyCoord(data[0][1] + " " + data[0][2], unit=(u.hourangle, u.deg))
print(c)
```

and for my first object specifically, I get

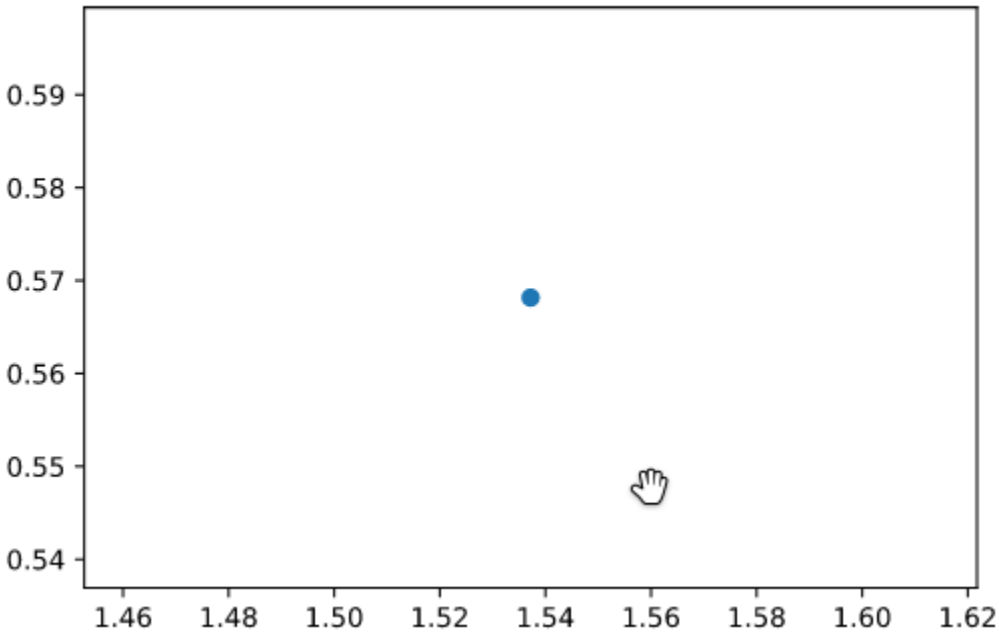
<SkyCoord (ICRS): (ra, dec) in deg (88.075, 32.55333333)>

Which seems easily workable with. Thanks astropy!

I ended up plotting it like so:

```
data = ascii.read("SIMBAD_m37.csv")

ra = data["RA"]
dec = data["DEC"]

star_coords = SkyCoord(ra, dec, unit=(u.hourangle, u.deg), frame='icrs')
ra_rad = c.ra.wrap_at(180*u.deg).radian
dec_rad = c.dec.radian

plt.plot(ra_rad, dec_rad, 'o')

#now we plot!
print(star_coords)
```

I got

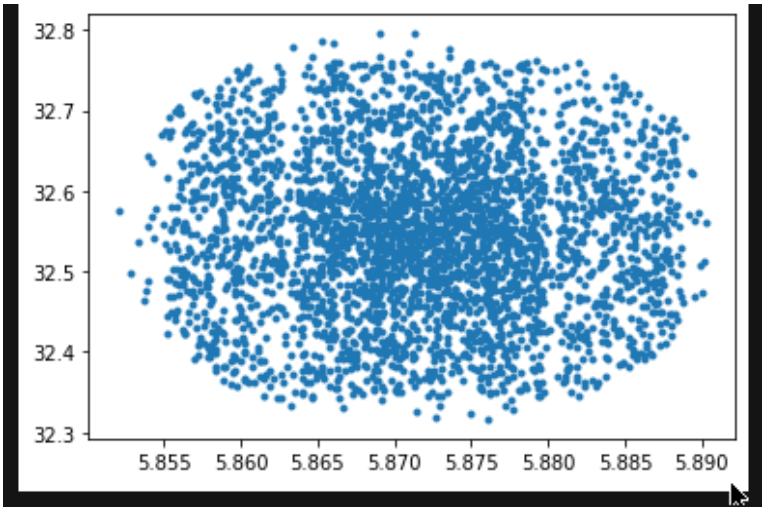


Which is obviously wrong. I've been at this for about an hour and a half so I think I will stop here

END OF 3/8/2021

3/9/2021

I will get this plot right, I will just try plotting the coordinates directly.



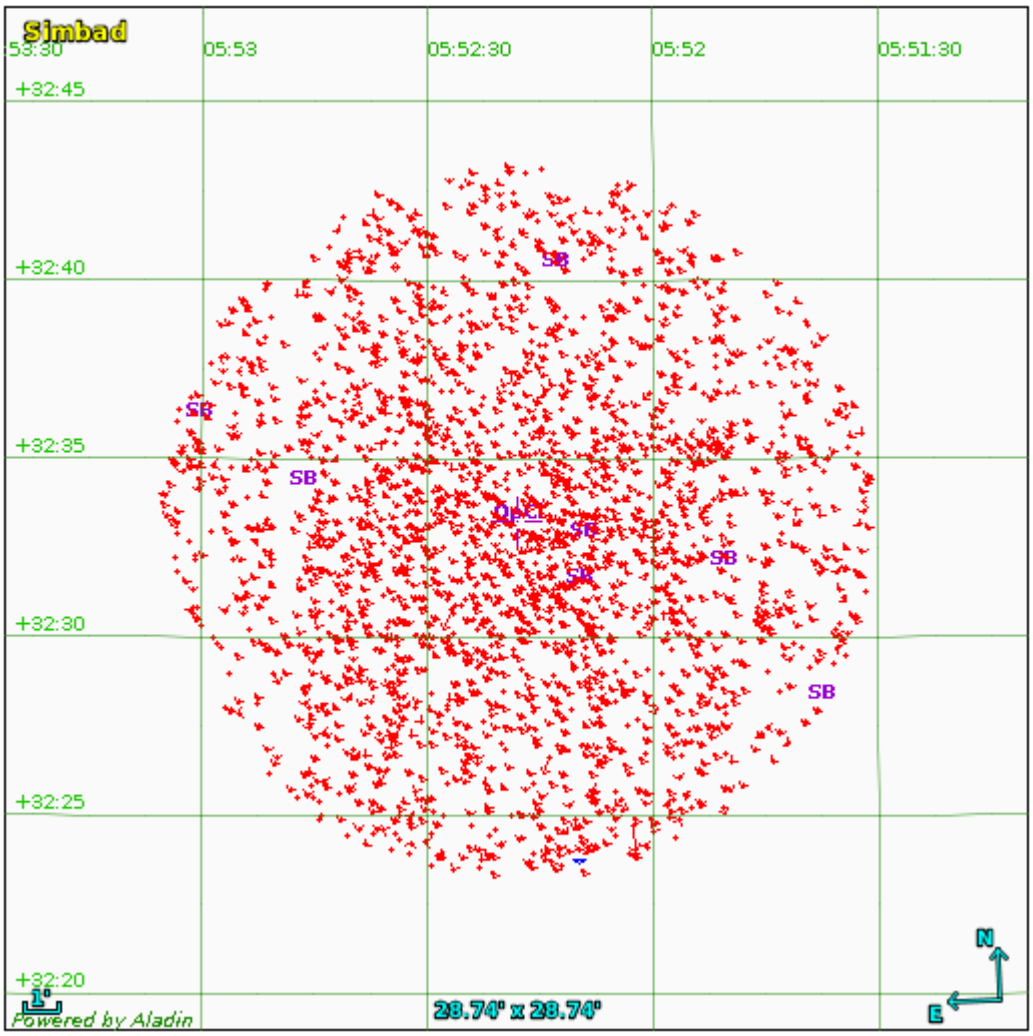
I just did `plt.plot(star_coords.ra, star_coords.dec, '.')`

It's something, I will aim to make it look more like a star chart.

Compared to:

I can’t see and similarities. I imagine my plot is not suitable for a star map. There’s probably a projection I need to apply or something.

From SIMBAD:



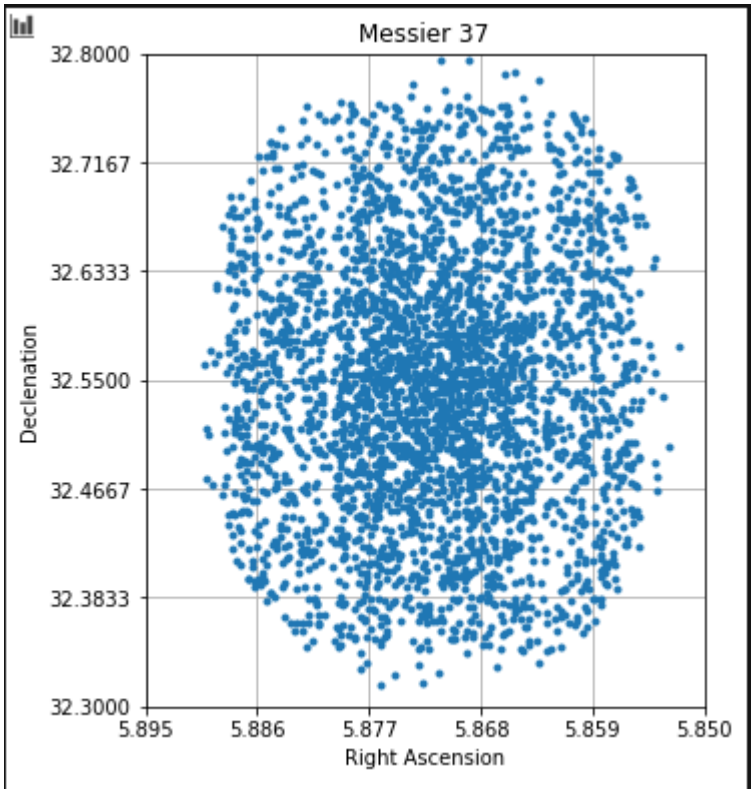
My y-axis is similar in ticking, and the x-axis is similar too but flipped like a typical star map. One thing I should do is make the plot a square.

This is done with `plt.figure(figsize=(6, 6))`
before the plt.plot().

Plus, I need to flip the x-axis, and add a grid.

After tinkering I was able to look like this:

*



It bares some resemblance to the SIMBAD plot. Although getting an accurate star map plot is not the biggest concern to me, since I will eventually go 3d, and I will be converting these coordinates to 3D Cartesian coordinates and I won't have to worry about scaling as much.

This was my generation code

```
from astropy.io import ascii #importing datasets
from astropy import units as u
import astropy.coordinates as coord
import numpy as np
import matplotlib.pyplot as plt


data = ascii.read("SIMBAD_m37.csv")

ra = data["RA"]
dec = data["DEC"]


star_coords = coord.SkyCoord(ra, dec, unit=(u.deg, u.deg), frame='icrs')

plt.figure(figsize=(5, 6))

plt.grid()

plt.xlim(5.895,5.85)
```

```
plt.ylim(32.3, 32.8)

plt.xticks(np.linspace(5.895,5.85,6))
plt.yticks(np.linspace(32.3,32.8,7))

plt.title("Messier 37")
plt.xlabel("Right Ascension")
plt.ylabel("Declenation")
plt.plot(star_coords.ra, star_coords.dec, '.')
```

Now I know that the data isn't messed, up I can start working towards determining what the z coordinate would be so I can map this cluster in 3d.

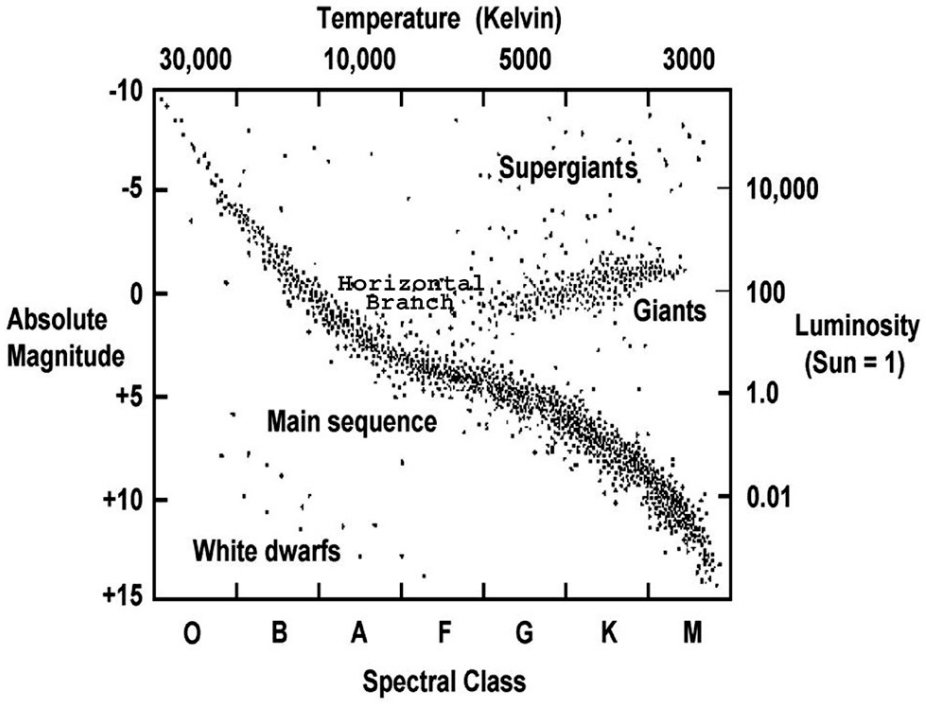
END OF 3/10/2021

3/11/2021

Not much plotting will be going today, but research into how to get the 3rd dimension. The RA-DEC system is pretty much the angular part of spherical coordinates, so in a way, we need to determine what the radial component is. If these catalogs simply had a parallax angle, the physics is done, and it becomes a calculation and computation problem. I will try to explore many ways of calculating distance and pick one best for my purpose.

The first and most common way is parallax angle, which is measuring how much a star drifts in a given time and uses trig to calculate it.

Another way is using the spectra to estimate the luminosity, and you can combine it with the apparent magnitude to get a distance. According to this, https://sci.esa.int/web/education/-/35616-stellar-distances?fbodylongid=1667, it's accurate up the 10kPc, which is roughly a third of the diameter of the milky way, good, for our purposes. There are many ways to find the luminosity, one way is estimating using the HR-diagram.

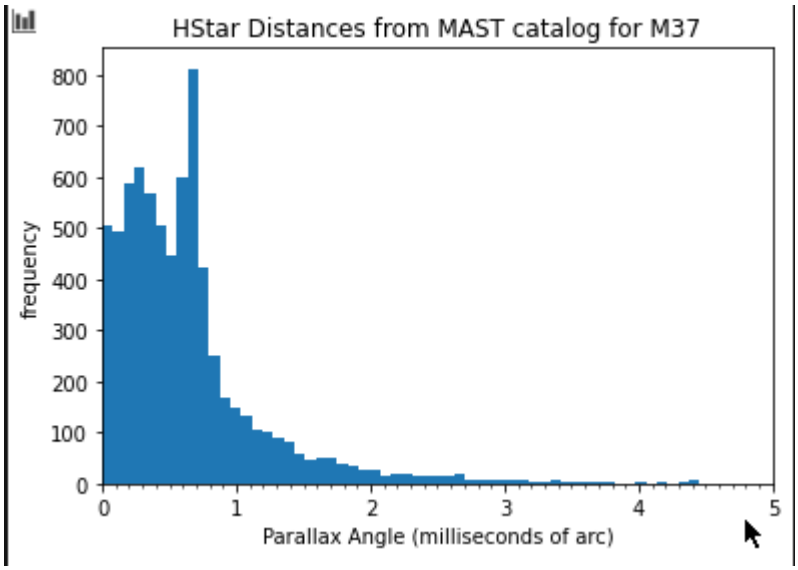


Which, coincidentally, is the assignment I have for astronomy lab. If a cluster is mainly old dead stars or red giants, then this won't be easily done. But a quick look at the wikipedia page for Open Clusters claims that open clusters are populated with hot young blue stars, which is along the horizontal branch.

I think I struck gold. I was exploring the MAST catalog more to see what might be better to use. I dug and found this search query https://mast.stsci.edu/portal/Mashup/Clients/Mast/Portal.html?searchQuery=%7B%22service%22%3A%22GAIADR2%22%2C%22inputText%22%3A%22messier%2037%22%2C%22paramsService%22%3A%22Mast.Catalogs.GaiaDR2.Cone%22%2C%22title%22%3A%22Gaia%20(DR2)%3A%20messier%2037%22%2C%22columns%22%3A%22*%22%7D

If I just did the right query, I was able to find the stars of m37 WITH parallax, which means I don't have to go through the trouble of estimating distance. Some stars have a NaN entry, but most have a number for parallax. No unit is given, but I think it's just arcseconds. M37 is 1400 Parsecs away, so if most are in the ballpark of that, it should be right. I will crutch the numbers on this to see if it checks.

It's milliarcseconds. I made a histogram of the distances.



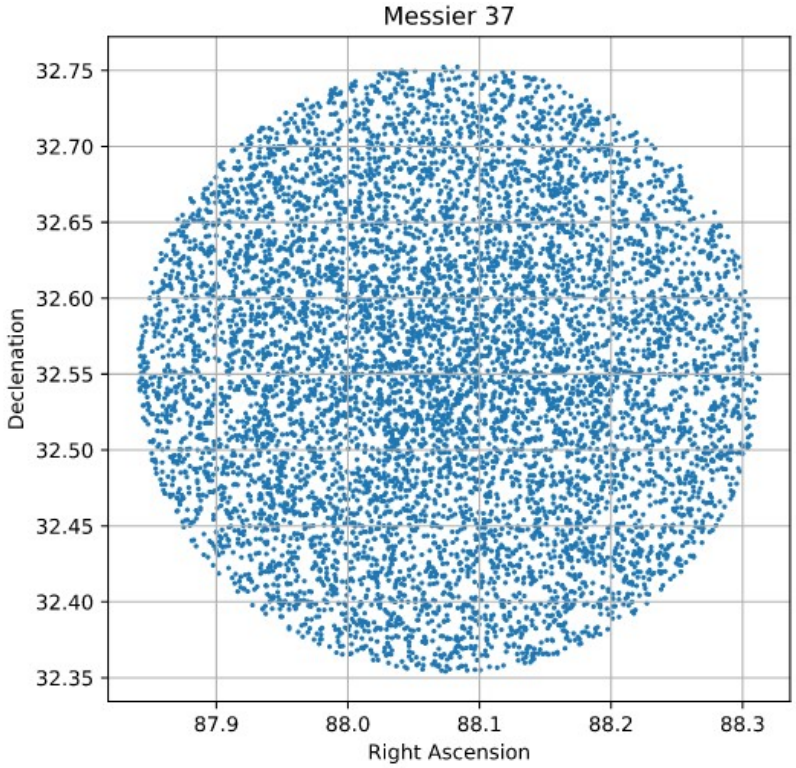
Generated with

```
#number crunching to check if the distances check out.
data = ascii.read("Gaia_DR2_messier_37.csv")


#there are 9000 entries let's hope my computer doesn't die.
p = data["parallax"]

#this line takes a looonng time to run, like 20 seconds. When I optimize the pr
ogram I will rewrite the calculations part in Rust.
#parallax = 1 / np.abs(p) * 1000
```

```
fig, ax = plt.subplots()

ax.hist(np.abs(p), bins=100)
ax.set_xlabel("Parallax Angle (milliseconds of arc)")
ax.set_ylabel("frequency")

ax.set_xlim(0,5)

ax.xaxis.set_major_locator(MultipleLocator(1))
ax.xaxis.set_minor_locator(MultipleLocator(0.1))

plt.title("Star Distances from MAST catalog for M37")
plt.show()
```

Import to mention, with parallax, the smaller number is farther, since parallax is the inverse of distance in parsecs. The peak value, around 0.7 milliarcseconds, is (1 / 0.7) * 1000 ~= 1400 parsecs, which is the distance of M37, so most of the stars are indeed for the cluster, but a lot aren't either. I will need to probably cut those out in the 3d render. (Note: the points at the very left, around 4, is around 250 pc away.)

One more thing for today, I forgot I am using a new data set, so here's the star map of it.



Now that is something! Since I am making good progress and am scared of my hard drive wiping, I will put my code in a github repository.
END OF 3/11/2021

Week 3

3/16/2021

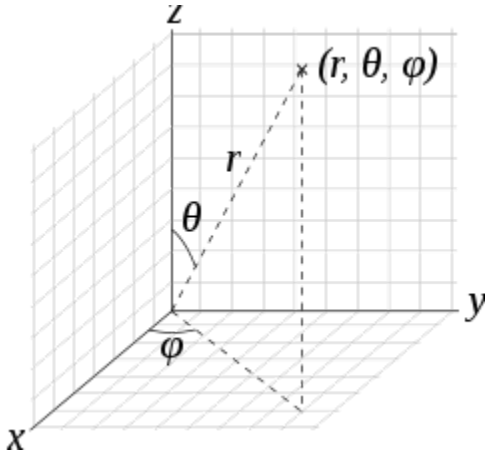
Today begins something I hope will be exciting. I will be plotting the stars in 3d. For this, I play to use Rust for the following reasons:

- It's low level and just as fast as c++
- It's complied so you don't need rust to run it
- It's safe in terms of memory management like python
- Importing modules is just as easy as Python

However, I am not the most comfortable with it at the moment, so I will try out the calculation code in Python then covert it to rust for speed and portability. To show 3d, I will use the vpython module.

pip install vpython

First thing I need to do convert coordinates from spherical to Cartesian. By the physics convention of spherical coordinate representation, r is the inverse of parallax, our distance, θ is our right ascension (RA), and φ is our declination (DEC)



For completeness sake, this is how one converts spherical to Cartesian.

$$\text{cart} \leftrightarrow \text{sph} \qquad \begin{cases} x = \rho \cos\theta \sin\phi, \\ y = \rho \sin\theta \sin\phi, \\ z = \rho \cos\phi, \end{cases} \qquad \begin{cases} \rho = \sqrt{x^2 + y^2 + z^2}, \\ \theta = \arctan \dfrac{y}{x}, \\ \phi = \arctan \dfrac{\sqrt{x^2 + y^2}}{z} \\ \quad = \arccos \dfrac{z}{\sqrt{x^2 + y^2 + z^2}}. \end{cases}$$

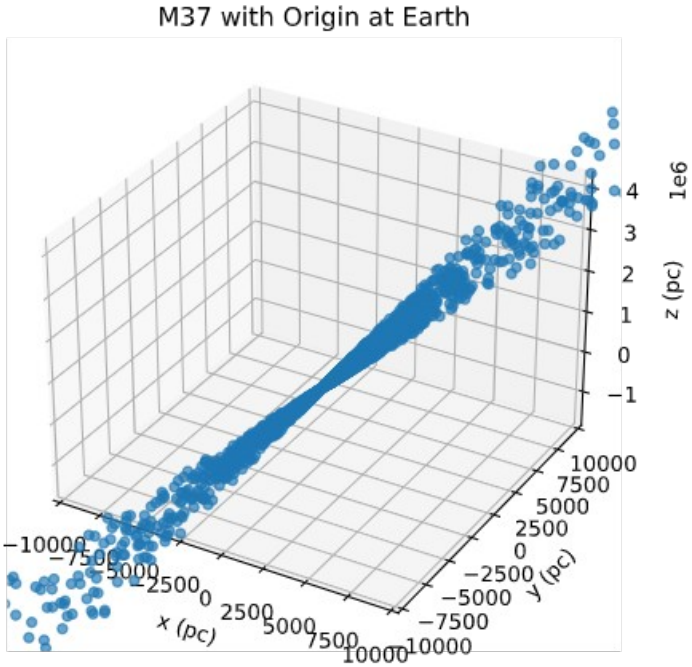
I will do it directly for now, but when it's time to optimize, I might have to work at this bit again because it's quite slow.

This is what I came up with. I really like the readability of it

```
#spherical_to_cart(rtp) [rtt = r theta phi], converts a tuple containing spherical co
ordinates to a tuple containing x,y,z
#Something really nice about this is all nan radius make the whole coordinate
nan so we can ignore it safely.
def spherical_to_cart(rtp):

   xyz=np.zeros_like(rtp) #empty array same size as coords

   index = 0
   for (r,t,p) in rtp[0]:
      x = r * np.cos(t) * np.sin(p)
      y = r * np.sin(t) * np.cos(p)
      z = r * np.cos(p)

      xyz[0][index] = [x,y,z]
      index+=1

   return xyz

coords = np.dstack((parallax, dec, ra)) #converts [p] [dec] [ra] -> (p, dec, ra)
xyz = spherical_to_cart(coords)

#removes all nan
xyz = xyz[~np.isnan(xyz).any(axis=2)]
```

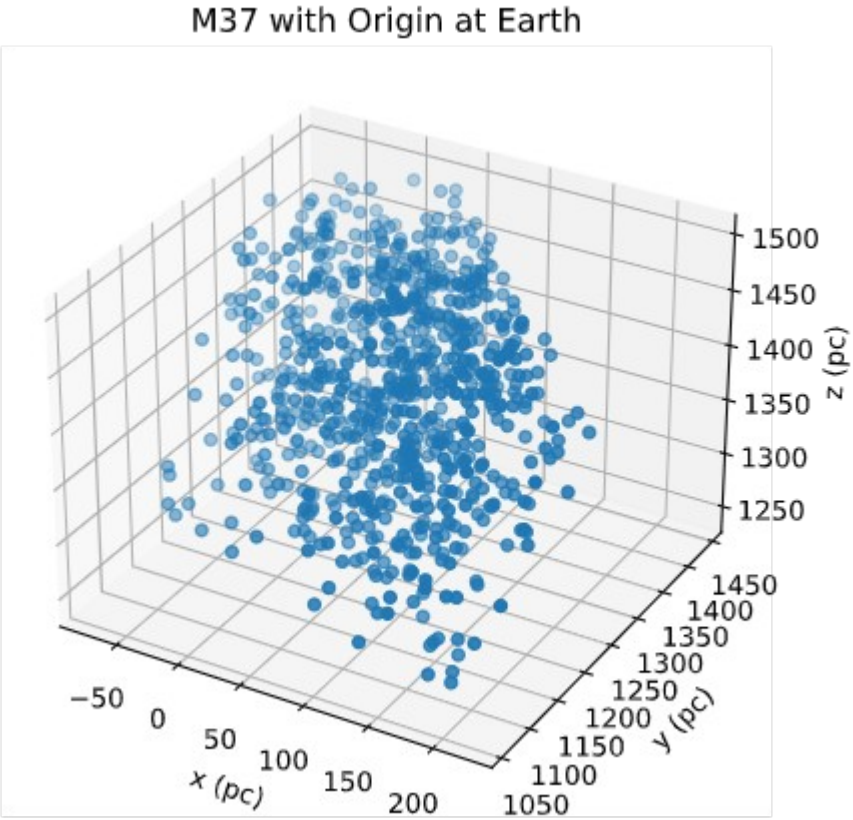
Now I will place this in 3d space! I will make a scatter plot to see if everything looks right.

It doesn't



This pattern reminds me of a gamma ray burst from a

black hole. That certainly isn't what this cluster is. Perhaps I converted wrong. Maybe it's all the stars that don't belong to the cluster making this.



It looks better, but I still feel like it needs adjusting. I will work on this next.

Took a small break, now instead of improving the plot, I will work on converting to rust. I mainly want to do everything computation wise before I plot it in 3d. I will write a rendering part, but that will be after I get this computations working.

For reference, to get from 0 to this in Python it takes 4.60 seconds. That's rather good, but I want to expand it to potentially 2.5 million stars in a reasonable time. (The size of the Hippacros catalog). In python that would take 12 days. Plus I want to be able to spread this program to computers without python. So I will convert the calculations to rust.

This is how you install rust for your machine https://www.rust-lang.org/tools/install

First I did in a terminal

cargo new star-calculator
cd star-calculator

and when I write my program I can compile with either
cargo build //compiles but doesn't run
cargo run //compiles and runs
cargo build –release //compiles slower but the program is overall faster. Used for released software.

First I need some modules to assist me. Mainly a csv reader. In the Cargo.toml file, under [dependencies], I wrote

csv = "1.1"

which downloads and links a csv module to my program for use. You can find theses at http://www.crates.io. I also added

```
serde = { version = "1", features = ["derive"] }
```

which where I'm referencing says it's needed

I wrote

```
use std::error::Error;
use std::io;

use csv;

fn read_from_stdin() -> Result<(), Box<dyn Error>>{

        let mut reader = csv::Reader::from_reader(io::stdin());

        for result in reader.records() {
        let record = result?;

        println!("{:?}", record);
        }

        Ok(())
}

fn main() {

        if let Err(e) = read_from_stdin(){
        eprintln!("{}", e);
        }
}
```

ran cargo build and ran

$ ./target/debug/star-calculator < ../Gaia_DR2_messier_37.csv

which pipes in the catalog and prints it all out.

I trimmed the catalog so I won't have so much useless data because I am about to unserialize, which is a way to read the data and put it into a struct.

```
struct StarData {

        designation: String,

        ra: f64,
        ra_error: f64,
        dec: f64,
        dec_error: f64,
        parallax: f64,
        parallax_error: f64,
        parallax_over_error: f64,
        bp_rp: f64,
        bp_g: f64,
        g_rp: f64,
        l: f64,
        b: f64,
        distance: f64

}
```

and if I change read_from_stdin() to

```
fn read_from_stdin() -> Result<(), Box<dyn Error>>{

        let mut reader = csv::Reader::from_reader(io::stdin());

        for result in reader.deserialize(){
        let record: StarData = result?;

        println!("{:?}", record);
        }

        Ok(())
}
```

This took 259 milliseconds to completely deserialize, in Python it's 274 ms. The number crunching part should be way faster.

After working a bit more, I finished the reader. Tomorrow or the next day is calculations.

END OF 3/16/21