Galactic Astronomy: Problem Set 4

Jonas Powell, *Wesleyan University*

April 5, 2019

Due: Thursday, April 4 by midnight. Late papers are not accepted after April 4 (midnight). If you cannot complete the assignment, hand in what you have completed before the deadline. Consider the deadline to be like the boarding time for an airplane, or the deadline for a grant submission to NASA or NSF. If you miss the deadline, you do not get on the airplane, no matter how good your excuse is. If you miss an NSF or NASA deadline, you do not get the grant, no matter how good your project is. The best advice is ... finish early. You can submit multiple times, right up to the deadline. Whatever your latest submission is, when the deadline occurs, is what will be graded.

Problem 1 & 2. Make an HR diagram of an open (galactic) cluster based on the Gaia catalogue. You may choose any cluster EXCEPT the Pleiades (since you already did the Pleiades as an exercise last semester in AST 222). Also, please make sure you have made a UNIQUE choice of galactic cluster – do not use the same cluster as any other classmate. (I suggest posting a cluster sign-up board in the student office to be sure that no one duplicates another person's cluster). You can adjust several parameters in selecting members, including the exact position of the center of your search area and the size of your search box. You can use parallaxes and/or kinematic information (proper motions and/or space motions) to help eliminate field stars. Your objective is to get as clean an HR diagram of your cluster as possible – ie. containing enough cluster members to clearly define the cluster HR diagram with as few non-members as possible. Do NOT remove stars arbitrarily from the diagram just because they don't fit your pre-conceived notion of what the HR diagram should look like! (That's a bad practice in science ...).

Once you have a nice cluster HR diagram, overlay on it a ZAMS with the metallicity appropriate to the cluster. You can use the Dartmouth Stellar Evolution Data Base (linked on the course Moodle page) to obtain model results in the Gaia magnitude system. For the galactic cluster, you can use a metallicity of 0 – i.e. solar. If it is a globular cluster, you will need to look up the metallicity of the cluster and choose an appropriate ZAMS. Adjust the horizontal location of the ZAMS to account for the foreground reddening of the cluster (which you will have to look up or figure out yourself). Then adust the vertical location of the ZAMS to match the cluster MS and use the amount of adjustment to calculate the distance to the cluster.

Answer 1 & 2. Presented here are plots for Messier 42, or The Orion Nebula. The code that created these plots is attached at the end of the problem set.

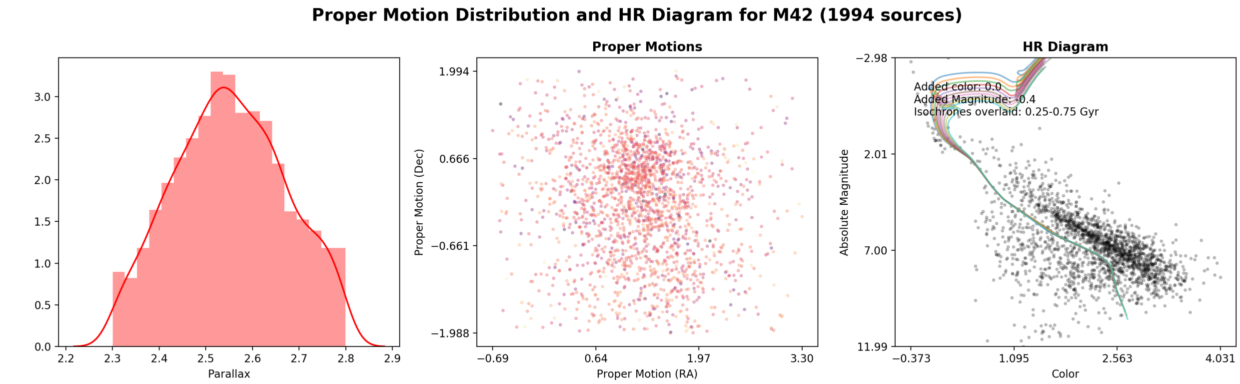
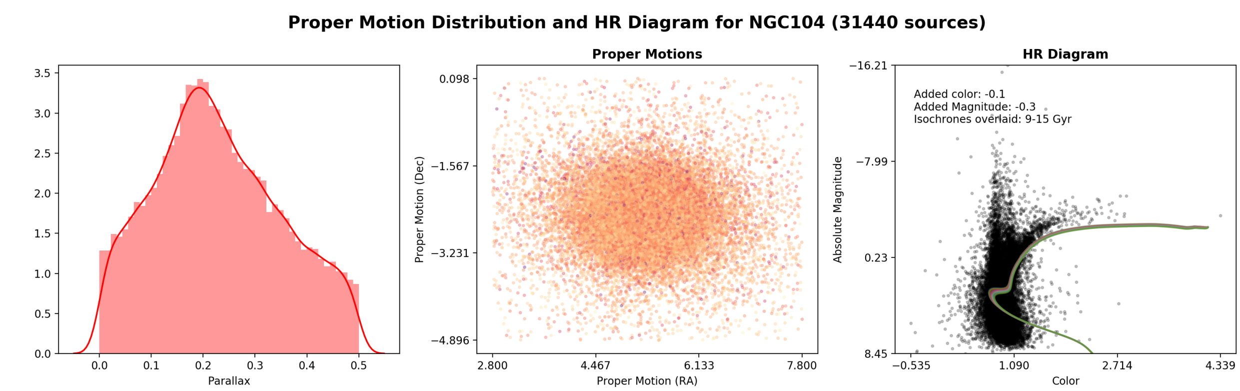


Figure 1: *Left, Center*: Plots of the M42 sample's distribution in parallax and proper motion space, used as checks on the validity of the regional cuts made to isolate the cluster. *Right*: M42's HR diagram, overlaid with some isochrones.

Problem 3. Repeat the process in Problems 1 and 2 for a globular cluster of your choice. Again, please make sure your choice of globular cluster is unique.

Answer 3. Presented here are plots for NGC104, or ξ Tucanae. The code that created these plots is attached at the end of the problem set.



Answer 1 & 2.

Figure 2: *Left, Center*: Plots of the NGC104 sample's distribution in parallax and proper motion space, used as checks on the validity of the regional cuts made to isolate the cluster. *Right*: NGC104's HR diagram, overlaid with some isochrones.

Problem 4. Discuss the HR diagrams above and your procedures for obtaining them. What features can you clearly see on each figure. Identify well known features such as the MS turnoff, the RG, HB, AGB features and the presence or absence of WDs. Can you see a binary sequence? Can you see blue stragglers? Can you see supergiants? Comment on any other aspects of the HR diagram or the procedure for selecting members and non-members that you find noteworthy.

Answer 4. To get these data, I developed a Python `class` object that retrieved Gaia data and the isochrone tracks. The Gaia data were retrieved using `astroquery`'s Gaia querier in SQL. Cuts were made in position, proper motion, and parallax spaces, usually with tolerances (i.e. boxes around the central point) of around half a degree for position, 3 km s^{-1} for proper motions, and 1-3 degrees for parallax.

Very rough isochrone fitting indicated that M42 is in the 0.25-0.75 Gyr range, while NGC104 is between 9-15 Gyr, both of which consistent with the literature values of around 0.3 and 13 Gyr, respectively.

As the annotations on the HR diagrams show, horizontal (color) offsets of 0 and -0.1 were added for M42 and NGC104, respectively, while vertical (magnitude) offsets were -0.4 and -0.3.

Since I chose to put my HR diagram on an absolute-magnitude scale (since we know the parallax of these sources, the conversion becomes quite trivial), the magnitude corrections needed for the isochrones were not significant and generally hovered around zero. However, if I were to put my HR diagram on an apparent magnitude scale, I would be able to calculate the distance to the cluster by using familiar distance modulus and rearranging it for distance:

$$
\begin{aligned}
m - M &= 5 \log_{10}(\frac{d}{10}) \\
&\rightarrow d = 10 \times (100^{(m-M)/5})^2 \text{ parsecs}
\end{aligned}
$$

We can actually do this backwards, using the first of the equations, and find that I would have had to shift the isochrone by about eight magnitudes to compensate for M42's distance of 389 parsecs, while NGC104, at 4kpc, would have required 13 magnitudes of compensation.

I was surprised to find that my cuts did not yield particularly clean data, as is shown in the plots attached. While the general stellar tracks can more or less be seen, it is strange to me that there are still so many field stars. Still, the plots certainly have sufficient detail to locate familiar features: in M42's HR diagram, we can see that, if we anchor the isochrone at the Main Sequence turnoff, then the majority of what appears to be the Main Sequence actually reveals itself to be PMS stars that have not yet made it to the MS. In M42, as expected, there are no branch features, as the stars are far too young. Conversely, ξ Tucanae's diagram tells a dramatically different story, with loads of branch stars and no main sequence presence (this time because they're all too old). The juxtaposition of extremely young and extremely old clusters in these two plots is fascinating.

```python

import warnings
warnings.filterwarnings("ignore")
# warnings.filterwarnings("ignore", module='astropy.io.votable.tree')

import yaml
import numpy as np
import pandas as pd
import seaborn as sns
import datashader as ds
import matplotlib.pyplot as plt
from astropy.coordinates import SkyCoord

from numba import jit
from datashader import transfer_functions as tf
from datashader.colors import inferno, viridis
from astroquery.gaia import Gaia


class Cluster:
    """A cluster."""

    def __init__(self, cluster_name='m43', n_sources=2000):

        print("Reminder: RA/Dec should be in degrees. To convert hms/dms -> degrees,")
        print("use: pos_in_degs = SkyCoord('02h31m49.09s', '+89d15m50.8s')\n\n")

        f = yaml.load(open('cluster_params.yaml', 'r'))[cluster_name.lower()]

        self.name = cluster_name
        self.isochrone_path = f['isochrone_path']
        self.n_sources = n_sources
        self.cmap = 'viridis'

        self.isochrone_start = f['isochrone_start']
        self.isochrone_stop  = f['isochrone_stop']

        # If you know the parallax a priori, add it to the field in the .yaml file.
        # Otherwise, convert from distance (mas)
        self.plx = 1000/f['distance'] if f['parallax'] is None else f['parallax']
        coords = SkyCoord(f['ra'], f['dec'], frame='icrs', unit='deg')
        self.ra, self.dec = [float(i) for i in coords.to_string().split()]


        self.ra_min, self.ra_max         = self.ra - f['radec_radius'], self.ra + f['radec_radius']
        self.dec_min, self.dec_max       = self.dec - f['radec_radius'], self.dec + f['radec_radius']
        self.pm_ra_min, self.pm_ra_max   = f['pm_ra'] - f['pm_radius'], f['
```

```
pm_ra'] + f['pm_radius']
        self.pm_dec_min, self.pm_dec_max = f['pm_dec'] - f['pm_radius'], f['pm_dec'] + f['pm_radius']
        self.plx_min, self.plx_max       = self.plx - f['parallax_radius'], self.plx + f['parallax_radius']


        # This operates in place. It should create its own attributes.
        self.get_gaia_data()
        self.read_isochrones()

    # Compile the GAIA dataframe
    def get_gaia_data(self):
        """
        Build a data structure for the Gaia data.

        Paper on bands, colors, and equivalencies for GAIA:
        arxiv.org/pdf/1008.0815.pdf'

        Basically, since GAIA uses longer wavelength light, their colors don't map
        to UBVRI bandpasses (p. 4, Table 1). This paper refers us to use B-R for
        color, and since we used V magnitude when considering B-V color, I chose
        to use the R band magnitude for my apparent magnitude values.

        Args:
            - ra, dec in degrees
        """
        print("Building GAIA dataframe for {} sources...".format(self.n_sources))

        # AND parallax_error < 2 \

        # try import gaia_results.csv except ImportError:
        gaia_str = "SELECT top {} * FROM gaiadr2.gaia_source \
                    WHERE pmra between {} and {} \
                    AND pmdec between {} and {} \
                    AND ra between {} and {} \
                    AND dec between {} and {} \
                    AND parallax between {} and {} \
                    ".format(self.n_sources,
                             self.pm_ra_min, self.pm_ra_max,
                             self.pm_dec_min, self.pm_dec_max,
                             self.ra_min, self.ra_max,
                             self.dec_min, self.dec_max,
                             self.plx_min, self.plx_max)

        job = Gaia.launch_job(gaia_str)  # , dump_to_file=True)
        gaia_results_raw = job.get_results()
        # gaia_results_raw['phot_rp_mean_mag'].description

```

```python
        gaia_results = gaia_results_raw.to_pandas()
        # gaia_cols = sorted(gaia_results.columns)

        print("Acquired data; now building dataframe...")
        gaia_df = pd.DataFrame()
        gaia_df['RA'] = gaia_results['ra']
        gaia_df['Dec'] = gaia_results['dec']
        gaia_df['Distance'] = (gaia_results['parallax'] * 1e-3)**(-1)
        gaia_df['Proper Motion (RA)'] = gaia_results['pmra']
        gaia_df['Proper Motion (Dec)'] = gaia_results['pmdec']
        gaia_df['mag'] = gaia_results['phot_rp_mean_mag']
        gaia_df['Color'] = gaia_results['bp_rp']
        gaia_df['Absolute Magnitude'] = gaia_df['mag'] - \
            5 * (np.log10(gaia_df['Distance']) - 1)
        # gaia_df['T Effective'] = gaia_results['teff_val']
        gaia_df['Parallax'] = gaia_results['parallax']
        # gaia_df['Plx. Error'] = gaia_results['parallax_error']
        # gaia_df['Confidence'] = 1 - gaia_results['parallax_error']/max(gaia_results['parallax_error'])

        self.df = gaia_df.dropna()
        self.mean_distance = round(np.mean(gaia_df['Distance']), 1)
        self.n_stars = len(gaia_df['Distance'])
        self.complete_sources_df = gaia_results
        self.raw_gaia_results = gaia_results_raw

        print('Finished getting data. Found {} sources (some may have been dropped due to NAs from Gaia or there were insufficient stars in the field).'.format(len(self.df)))
        return None


    def read_isochrones(self):

        f1 = open(self.isochrone_path + '_2', 'r').read().split('#')
        f2 = open(self.isochrone_path, 'r').read().split('#')

        header_info = [f1[:8], f2[:8]]

        data_strings = f1[8:] + f2[8:]


        # TODO: Remove trailing Nones from each df
        i, data, ages = 0, [], []
        while i < len(data_strings):
            age = data_strings[i]
            block = data_strings[i + 1]

            # Parse the age
            age = float(age.split('0 ')[0].split('=')[1])

            # Parse the data
            ser = pd.Series(block.split('\n'))
```

```
            ser[0] = ' ' + ser[0]
            df = ser.str.split('\s+', expand=True)
            df.columns = df.iloc[0]
            df.drop('', axis=1, inplace=True)
            df.drop(0, axis=0, inplace=True)
            df = df.astype('float64')
            df['color_bp_rp'] = df['Gaia_BP'] - df['Gaia_RP']

            data.append(df)
            ages.append(age)
            i += 2

        self.isochrone_data = data
        self.isochrone_header_info = header_info
        self.isochrone_ages = ages

        return None


    def make_plots(self, save=False, d_color=0, d_mag=0, iso_min=None, iso_max=None):
        """Make the necessary plots for each dataset.

        Note that coloring is done basically by the inverse of error, given by:
        confidence = 1 - (sigma/sigma_max)
        """
        plt.close()
        fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(16, 5))

        sns.distplot(self.df['Parallax'], ax=axes[0], color='red')


        self.df.plot.scatter('Proper Motion (RA)', 'Proper Motion (Dec)',
                             cmap='magma', c='Absolute Magnitude',
                             ax=axes[1], marker='.', alpha=0.3,
                             colorbar=False)

        self.df.plot.scatter('Color', 'Absolute Magnitude',
                             c='black', # cmap='magma',
                             ax=axes[2], marker='.', alpha=0.2)


        # self.df.plot.scatter('RA', 'Dec',
        #                      c='black', # cmap='magma',
        #                      ax=axes[2], marker='.', alpha=0.2)

        # First get prioritize function args over yaml
        i_min = self.isochrone_start if iso_min is None else iso_min
        i_max = self.isochrone_stop if iso_max is None else iso_max

        isochrones = self.isochrone_data[i_min:i_max]
        # Choosing B-R color and R mag to match the decision made in
```

```
get_gaia_data()
        [axes[2].plot(isochrones[i]['color_bp_rp'] + d_color,
                      isochrones[i]['Gaia_RP'] + d_mag, alpha=0.5)
         for i in range(len(isochrones))]


        # Other assorted and processes:
        i_min = 0 if i_min is None else i_min
        i_max = -1 if i_max is None else i_max

        # Get rid of decimals for whole numbers
        age_min = self.isochrone_ages[i_min]
        age_max = self.isochrone_ages[i_max]
        age_min = int(age_min) if round(age_min, 2) == int(age_min) else age_min
        age_max = int(age_max) if round(age_max, 2) == int(age_max) else age_max

        text_str = 'Added color: ' + str(d_color) \
                    + '\nAdded Magnitude: ' + str(d_mag) \
                    + '\nIsochrones overlaid: {}-{} Gyr'.format(age_min, age_max)

        abs_mag, color = self.df['Absolute Magnitude'], self.df['Color']
        text_loc_x = np.nanmin(color) + 0.01 * (np.nanmax(color) - np.nanmin(color))
        text_loc_y = np.nanmax(abs_mag) - 0.8 * (np.nanmax(abs_mag) - np.nanmin(abs_mag))
        text_loc = (text_loc_x, text_loc_y)
        axes[2].annotate(text_str, text_loc) #, weight='bold')

        print('text_x', 'text_y: ', text_loc_x, text_loc_y)



        axes[2].set_ylim(axes[1].get_ylim()[::-1])
        axes[2].set_xticks(np.linspace(np.nanmin(self.df['Color']), np.nanmax(self.df['Color']), 4))
        axes[2].set_yticks(np.linspace(np.nanmin(self.df['Absolute Magnitude']),
                                       np.nanmax(self.df['Absolute Magnitude']), 4))

        axes[1].set_xticks(np.linspace(np.nanmin(self.df['Proper Motion (RA)']),
                                       np.nanmax(self.df['Proper Motion (RA)']), 4))
        axes[1].set_yticks(np.linspace(np.nanmin(self.df['Proper Motion (Dec)']),
                                       np.nanmax(self.df['Proper Motion (Dec)']), 4))

        axes[1].set_title('Proper Motions', weight='bold')
```

```python
        axes[2].set_title('HR Diagram', weight='bold')
        fig.suptitle('Proper Motion Distribution and HR Diagram for {} ({} sources)'.format(self.name.upper(),
                     len(self.df)),
                     weight='bold', fontsize=16)
        fig.tight_layout()
        fig.subplots_adjust(top=0.85, bottom=0.1)
        if save is True:
            outname = './HRD_{}_{}.png'.format(self.name.upper(), self.n_sources)
            plt.savefig(outname, dpi=200)
            print("Saved to ", outname)
        else:
            print("Showing:")
            plt.show(block=False)
        return None


    def plot_HR_isochrones(self, save=False, d_color=0, d_mag=0, iso_min=None, iso_max=None):

        fig, ax = plt.subplots()

        self.df.plot.scatter('Color', 'Absolute Magnitude', ax=ax,
                             marker='.', color='darkblue', alpha=0.3)

        # Prioritize function arguments for which isochrone range to plot, but
        # if none are provided, use the ones from the YAML file (which might also be Nones)
        i_min = self.isochrone_start if iso_min is None else iso_min
        i_max = self.isochrone_stop if iso_max is None else iso_max

        isochrones = self.isochrone_data[i_min:i_max]

        # Choosing B-R color and R mag to match the decision made in get_gaia_data()
        [ax.plot(isochrones[i]['color_bp_rp'] + d_color, isochrones[i]['Gaia_RP'] + d_mag, alpha=0.5)
         for i in range(len(isochrones))]

        ax.set_ylim(ax.get_ylim()[::-1])
        # ax.set_xlim(ax.get_xlim()[::-1])

        if save:
            plt.savefig('{}_HR-isochrones_{}-sources.png'.format(self.name, self.n_sources), dpi=300)
        else:
            plt.show()


    def dsplot(self):
        """Return a Datashader image by collecting `n` trajectory points for
```

```
    the given attractor `fn`"""
        # lab = ("{}, "*(len(vals)-1)+" {}").format(*vals) if label else None
        # df  = self.df[['RA', 'Dec']]
        cvs = ds.Canvas(plot_width = 1000, plot_height = 400)
        agg = cvs.points(self.df, 'RA', 'Dec')
        img = tf.shade(agg, cmap=cmap, name=['RA', 'Dec'])
        return img


m42 = Cluster('m42', n_sources=70000)

m42.make_plots(save=True, d_color=0., d_mag=-0.4, iso_min=0, iso_max=10)









ngc104 = Cluster('ngc104', n_sources=70000)
ngc104.make_plots(save=True, d_color=-0.1, d_mag=-0.3, iso_min=40, iso_max=
    None)






# The End
```

```
"""
Galactic Astronomy: Problem Set 4
Jonas Powell
April 4, 2019


Cool thing: dir(object) returns all that object's attributes and methods.
"""

import warnings
warnings.filterwarnings("ignore")
# warnings.filterwarnings("ignore", module='astropy.io.votable.tree')
import yaml
import numpy as np
import pandas as pd
import seaborn as sns
import datashader as ds
import matplotlib.pyplot as plt
from astropy.coordinates import SkyCoord

from numba import jit
```

```python
from importlib import reload
from datashader import transfer_functions as tf
from datashader.colors import inferno, viridis
from astroquery.gaia import Gaia

from get_cluster import Cluster







m5 = Cluster('m5', n_sources=7000)

m5.make_plots(save=True, d_color=0.2, d_mag=-0.15, iso_min=35, iso_max=None)
# m5.plot_HR_isochrones(d_color=0.3, d_mag=0., iso_min=30, iso_max=40)



m43 = Cluster('m43', n_sources=7000)
m43.make_plots(save=True, d_color=0.5, d_mag=0.5, iso_min=0, iso_max=10)
# m43.plot_HR_isochrones(d_color=0., d_mag=-1.6, iso_min=0, iso_max=10)




m42 = Cluster('m42', n_sources=7000)
m42.make_plots(save=True, d_color=0., d_mag=-1.1, iso_min=0, iso_max=10)
# m42.plot_HR_isochrones(d_color=0., d_mag=-1.6, iso_min=0, iso_max=5)




ngc104 = Cluster('ngc104', n_sources=7000)
ngc104.make_plots(save=True, d_color=0.2, d_mag=0., iso_min=35, iso_max=None)
# ngc104.plot_HR_isochrones(d_color=1., d_mag=3, iso_min=None, iso_max=None)




m67 = Cluster('m67', n_sources=7000)
m67.make_plots(save=True, d_color=0.9, d_mag=1.8, iso_min=0, iso_max=10)





m44 = Cluster('m44', n_sources=7000)
m44.make_plots(save=False, d_color=0.9, d_mag=1.8, iso_min=0, iso_max=2)
```

```
# The End
```