CMD_project_template

December 31, 2023

1 Colour-Magnitude Diagrams

In this data project, you will work with catalogs of observed properties of stars in the local neighbourhood. You will work with observed parallaxes to determine distances to stars, and convert from apparent to absolute magnitudes. You will plot colour-magnitude diagrams, and compare the observed data with theoretical curves that predict the locations of stars in the CMD based on the age of the population.

First, you will investigate stellar properties from the Hipparcos catalog.

Next, you will use data from the Gaia telescope to identify stellar cluster members using a combination of observed parallax and proper motion.

```
[160]: # First, import the usual libraries to help you in your calculations and plots:
import numpy as np
import astropy.constants as c
import astropy.units as u
import matplotlib.pyplot as plt
```

2 Section 1: the Hipparcos CMD for the local stellar neighbourhood

The Hipparcos data that you will be using are in the file 'hipparcos_parallax.dat'. Open the file in your favourite text editor to take a look at the columns.

There are two header rows you will need to skip when reading in the data. The columns (listed in the header rows) are: * p: parallax in milli-arcseconds * sig_p: parallax uncertainty (same units) * v: apparent V magnitude (magnitudes) * bv: B-V color (magnitudes) * sig_bv: B-V uncertainty (magnitudes)

Recall that we use different filters to observe stars and estimate their colour (which is related to their effective temperatures). With the Hipparchos data, you are looking at the stellar brightness in the V (visual) and B (blue) filters.

```
[161]: # Read in the data using np.loadtxt. See previous assignments for syntax if␣
↪needed.
# You will want separate arrays for each column!
p, sig_p, v, bv, sig_bv = np.loadtxt('hipparcos_parallax.
↪dat',usecols=(0,1,2,3,4), skiprows=2, unpack=True)
```

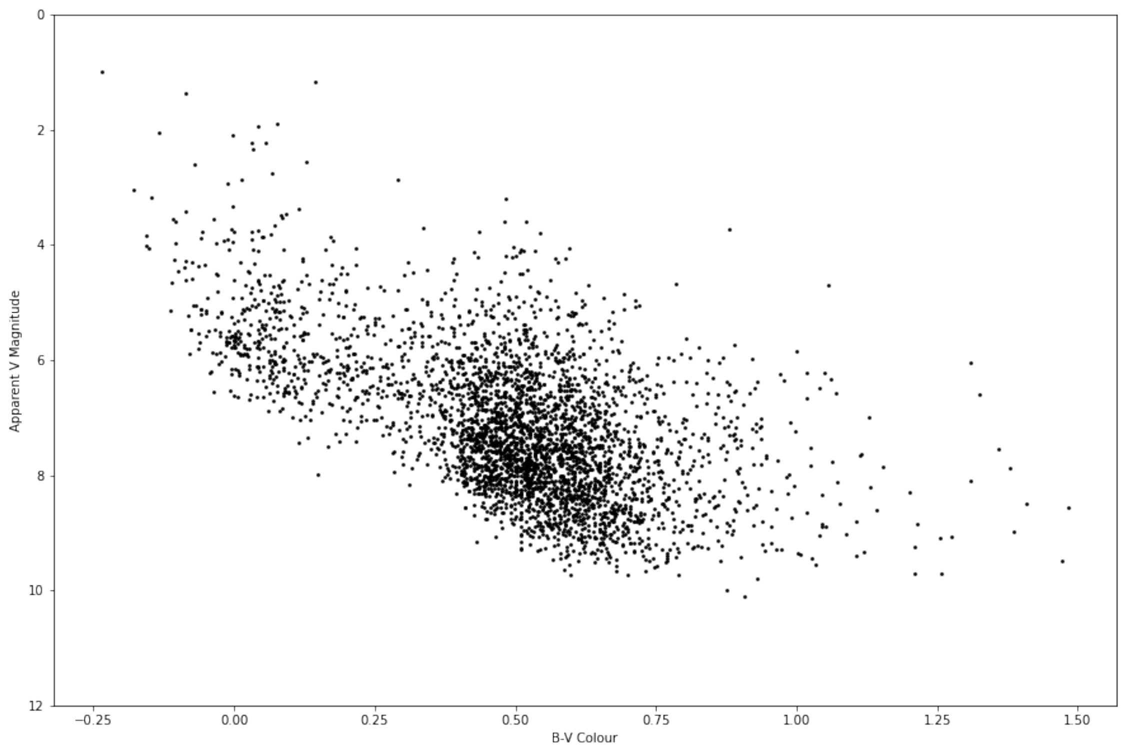
```
print(v)
```

```
[7.83 8.69 7.5  … 8.41 6.27 6.46]
```

Recall that a colour magnitude diagram (CMD) usually plots the absolute visual magnitude (V) against the colour. Right now you only have apparent magnitudes. This isn't a problem for colour; the B-V colour will remain the same if you calculate it using apparent or absolute magnitudes! As a first look at a CMD, we can plot the apparent visual vs. the B-V colour. The B-V colour is simply the difference in B and V magnitudes.

Save this plot and include it in your report.

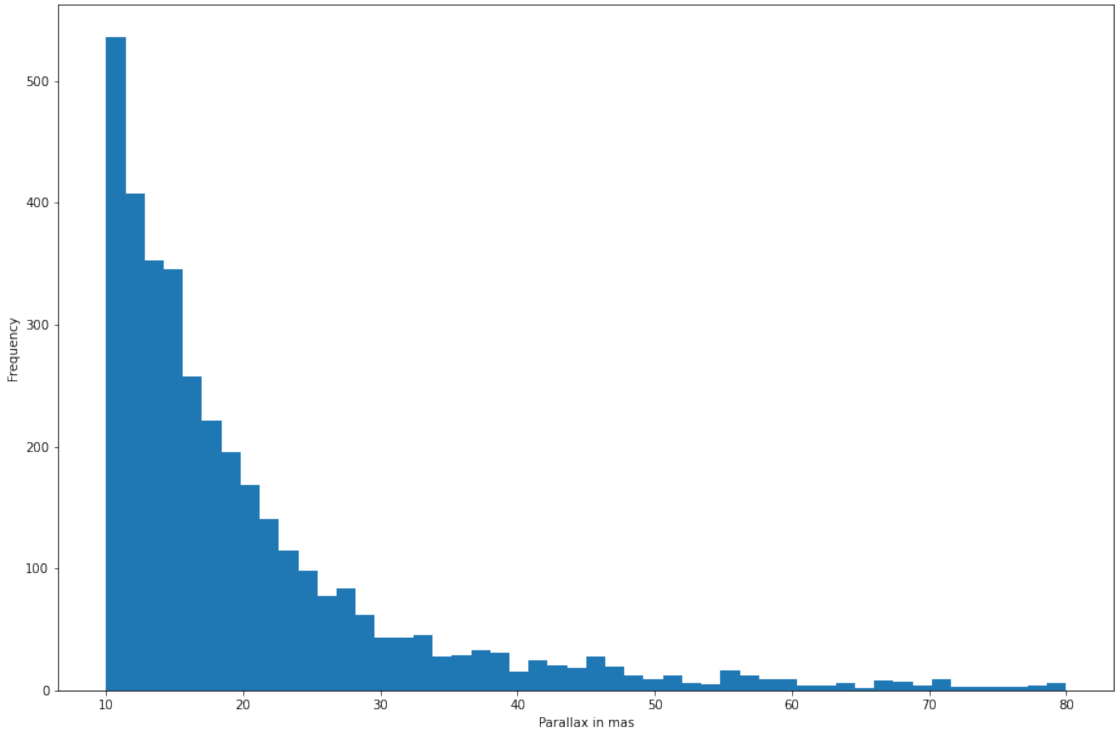
```
[162]: # Using a scatter plot, plot the CMD of the Hipparchos data as apparent V␣
         ↪magnitude vs. B-V colour.
       # Because there are so many stars, make the size of your plotted points smaller
       # (i.e., set s=5 or so in the plt.scatter command)
       # Add axis labels. Remember that lower magnitudes are brighter - set your plot␣
         ↪ranges accordingly!
       fig=plt.figure(figsize=(15,10))
       plt.scatter(bv, v, s=3, color='black')
       plt.ylim(12,0)
       plt.xlabel('B-V Colour')
       plt.ylabel('Apparent V Magnitude')
       fig.savefig('1_ApparentVMag_vs_B-V.png')
```



Your CMD may not look like you expected - perhaps you don't see a clear trend suggestive of a main sequence? Let's see why that is.

Next we will make a histogram of the parallax values in the dataset.

[163]:
```
# There is a very quick and easy function to plot histograms in matplotlib:
# plt.hist(data,bins='auto',range=(x1,x2))
# where data is the array you want to make a histogram of, and x1 and x2 are␣
 ↪lower and upper limits of the data.
# You can look up the documentation on plt.hist if you want to get more fancy.
# Start without setting a range - see if this is needed
# You might see that many bins are pretty empty, and can limit the range to␣
 ↪better see the distribution.
# Make a histogram of the parallax values:
fig=plt.figure(figsize=(15,10))
plt.hist(p,bins='auto', range=(10,80))
plt.xlabel('Parallax in mas')
plt.ylabel('Frequency')
fig.savefig('1_Parallax.png')
```



We have a wide range of parallax values. What does this imply about the distances? To plot a true CMD, write a small function that will calculate the absolute magnitude given input apparent magnitude and parallax.

Recall that to write a function, you use the syntax:

```
def my_function(input1, input2):     value = some calculation involving input1
and input2     return value
```

And to use your function, you would then use the syntax:

```
calculated_value = my_function(input1,input2)
```

[164]:
```
# Write your function to calculate absolute magnitude given the apparent␣
↪magnitude and parallax:
def absolute_mag(p, v):
    p_in_arc = p*0.001
    d = 1/p_in_arc
    mag = v - 5*np.log10(d/10)
    return mag
```

[165]:
```
# Use your function here to calculate absolute V magnitudes
# You can give lists as inputs to your function, and you will get a list of␣
↪absolute magnitudes in return
absv = absolute_mag(p, v)
print (absv)
```

```
[3.21683953 4.70244159 3.66243933 … 4.46293012 2.93017139 1.49233021]
```

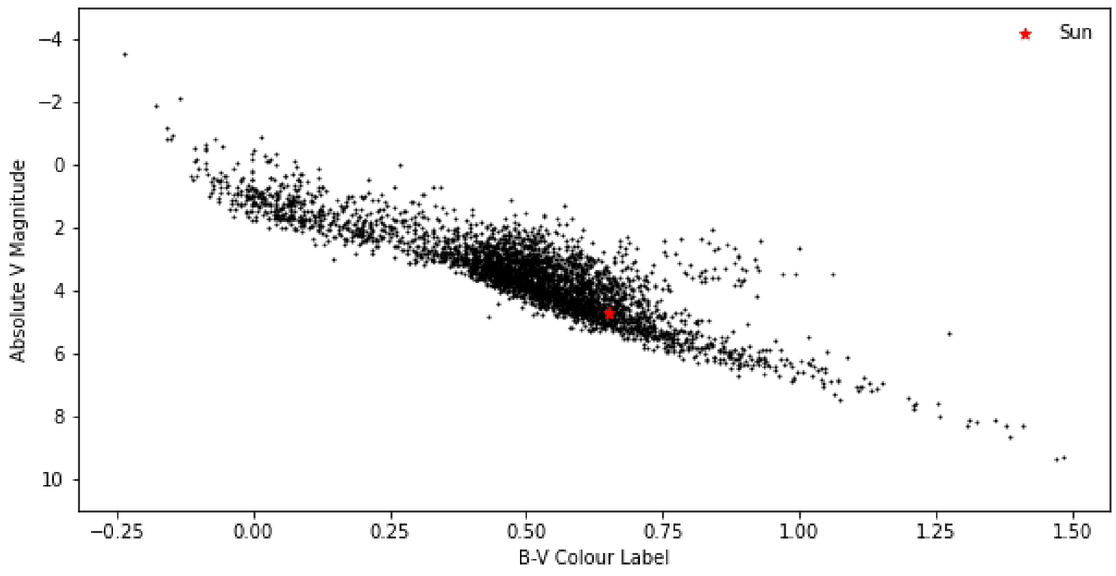
Now redo your CMD, using the absolute V magnitudes and original B-V colour Label and use smaller point size as before.

Hopefully it looks more familiar!

Where does the Sun lie on this plot? Appendix A in Carroll & Ostlie gives the Sun's absolute bolometric magnitude M_B, and the apparent blue and visual magnitudes. Add a red star to overlay the Sun's location on your plot above using a second `plt.scatter` command.

You can set the marker style in `plt.scatter` using `marker = '*'`, and change the colour using `c = 'red'`.

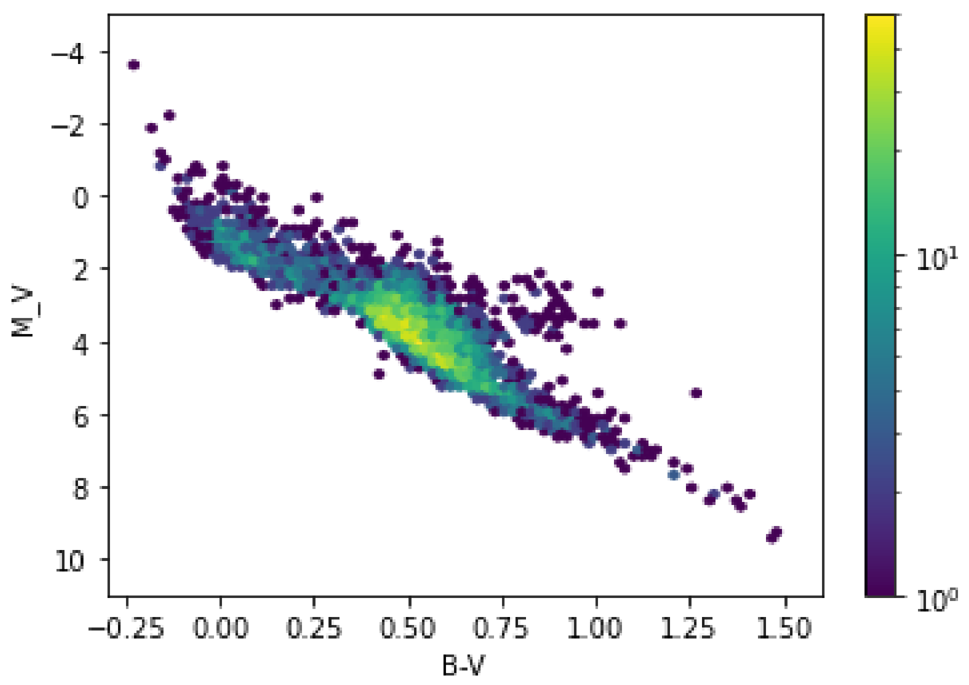
[203]:
```
# CMD plot abs(V) vs. B-V here, add Sun:
fig=plt.figure(figsize=(10,5))
plt.scatter(bv, absv, s=1, color='black')
plt.scatter(0.65,4.74, marker = '*', color = 'red',label='Sun' )
plt.xlabel('B-V Colour Label')
plt.ylabel('Absolute V Magnitude')
plt.ylim(11,-5)
plt.legend(frameon = False)
fig.savefig('1_AbsV_vs_BV.png')
```



```
[167]: # There are still a lot of points here. For fun, here is how you can make a␣
        ↪density plot.
       # You will need to change your x, y for your plot to match your own list names.
       # The colour scale depends on how many points you have in each bin.
       # This is useful to look at trends in densely-packed datasets.

       plt.hexbin(bv, absv, extent=(-0.3, 1.6, 11, -5), bins="log", gridsize=80)
       plt.xlabel("B-V")
       plt.ylabel("M_V")
       plt.xlim(-0.3, 1.6)
       plt.ylim(11, -5)
       plt.colorbar()
```

```
[167]: <matplotlib.colorbar.Colorbar at 0x7f9266e39670>
```



We have discussed the idea of the ‘Zero Age Main Sequence (ZAMS)’ in class. The file ‘zams_02.dat’ is a list of masses, absolute V magnitudes, and B-V colours for a theoretical ZAMS for stars with metallicity $Z = 0.02$.

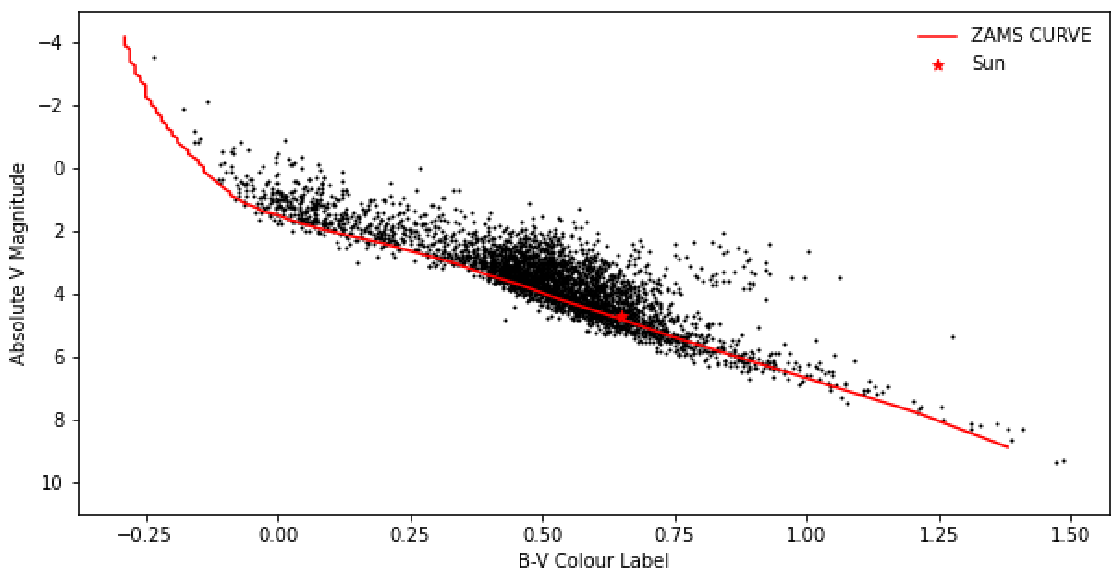
Read in the ZAMS data and overplot V vs. B-V as a red curve (use plt.plot()) on your CMD. Include a label for your ZAMS curve and a legend (see A2 for the syntax on how to do this!).

If your absolute magnitude calculates are correct, you should see your data and the curve line up well!

Again, include the Sun’s data point, and give it a label as well.

Save this plot and include it in your report.

[168]:
```
# Read in ZAMS isochrone data
mass, M_V, B_V = np.loadtxt('zams_02.dat',usecols=(0,1,2), skiprows=5,␣
  ↪unpack=True)
# Plot ZAMS over CMD
fig=plt.figure(figsize=(10,5))
plt.plot(B_V, M_V, color='red', label='ZAMS CURVE')
plt.scatter(bv, absv, s=1, color='black')
plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')
plt.xlabel('B-V Colour Label')
plt.ylabel('Absolute V Magnitude')
plt.ylim(11,-5)
plt.legend(frameon = False)
fig.savefig('1_zams.png')
```



2.1 Answer these questions in Section 1 of your report

What do you notice about the alignment of the data in your CMD and the ZAMS?

Do you see evidence of stars evolving off the main sequence? Why or why not?

3 Section 2: Gaia data for the Hyades cluster

The Hipparchos data includes all stars within a particular distance of the Sun. These stars were born at different times, in different locations, and therefore we see stars in a range of evolutionary stages in the plot. Some massive stars are still on the main sequence! Some less massive stars have evolved into giants.

Next, you will investigate the data for a single stellar cluster, where we can assume that all stars formed at roughly the same time. Hopefully, this means that we can see clearly the main sequence, and the main sequence turn-off where the most massive stars have already exhausted their hydrogen in their cores, and have started evolving onto the giant branch.

We will use data from the Gaia archive. The Gaia mission is measuring the magnitudes, locations, parallaxes and proper motions of stars within 10,000 pc of the Sun.

The Hyades data are in the file 'gaia_hyades_search.csv'. This is the output from a search of the Gaia database https://gea.esac.esa.int/archive/. The Gaia archive contains a LOT of data on a LOT of stars! Here, I've limited the search to an area of 240 arcminutes around the centre of the Hyades cluster. I've also limited the results to contain only stars with parallaxes close to the expected value for the cluster.

Here, you will first use the measured parallax and proper motion values to select the set of stars that are truly members of the Hyades cluster. Then you will convert apparent to absolute magnitudes, plot the CMD, and compare your CMD with several isochrones. Isochrones (same time) are tables that plot the expected CMD for a population of stars with a range of masses at a given age.

3.0.1 Gaia data

The Gaia data file contains the following columns:

- ra: Right ascension
- dec: Declination
- parallax: parallax (in milli-arcseconds)
- pmra: proper motion (milli-arcseconds per year, in right ascension)
- pmdec: proper motion (milli-arcseconds per year, in declination)
- phot_g_mean_mag: apparent G magnitude
- phot_bp_mean_mag: apparent B magnitude
- phot_rp_mean_mag: apparent R magnitude

The Gaia archive outputs the search results as a .csv file, and some stars don't have data for all columns. Unfortunately this messes up `np.loadtxt`. To read these data in, use `np.genfromtxt()` with `delimeter=','`, and `skip_header` rather than `skiprows` (as in `np.loadtxt`).

[169]:
```
# Trying with Gaia data
# Need to use genfromtxt to deal with missing data
# You will want separate arrays for each column!
ra, dec, parallax, parallax_error, pmra, pmdec, phot_g_mean_mag,␣
 ↪phot_bp_mean_mag, phot_rp_mean_mag = np.genfromtxt('gaia_hyades_search.csv',␣
 ↪delimiter=',',usecols=(0,1,2,3,4,5,6,7,8), skip_header=1, unpack=True)
print (phot_rp_mean_mag)
```

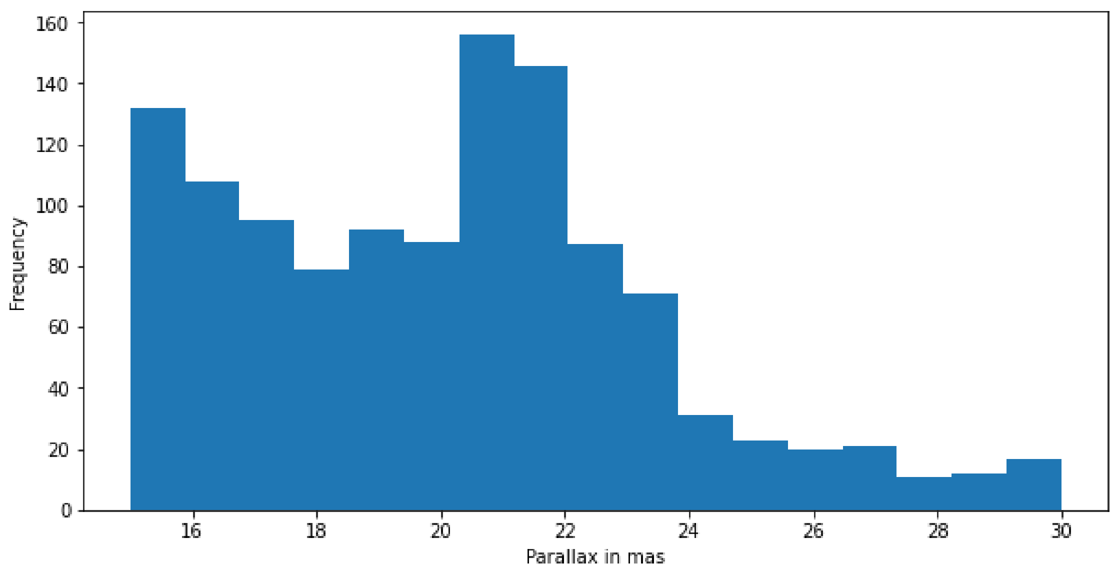
```
[14.995886 13.727656 12.574867 … 10.8709    7.909447 17.967545]
```

The data have already been limited by setting a minimum parallax of 15 milliarcseconds. You should plot a histogram of the parallax values to see whether we need to limit the data further. A cluster of stars should all be at nearly the same parallax.

Use the `plt.hist` function again to plot the parallax values of this dataset. You should see a clear peak in number of stars around a parallax of 21. Again, you may want to set a range to zoom in on the values of interest.

Save this plot and include it in your report.

[170]:
```
# Look at parallax measurements as a histogram
fig=plt.figure(figsize=(10,5))
plt.hist(parallax,bins='auto', range=(15,30))
plt.xlabel('Parallax in mas')
plt.ylabel('Frequency')
fig.savefig('2_Parallax.png')
```



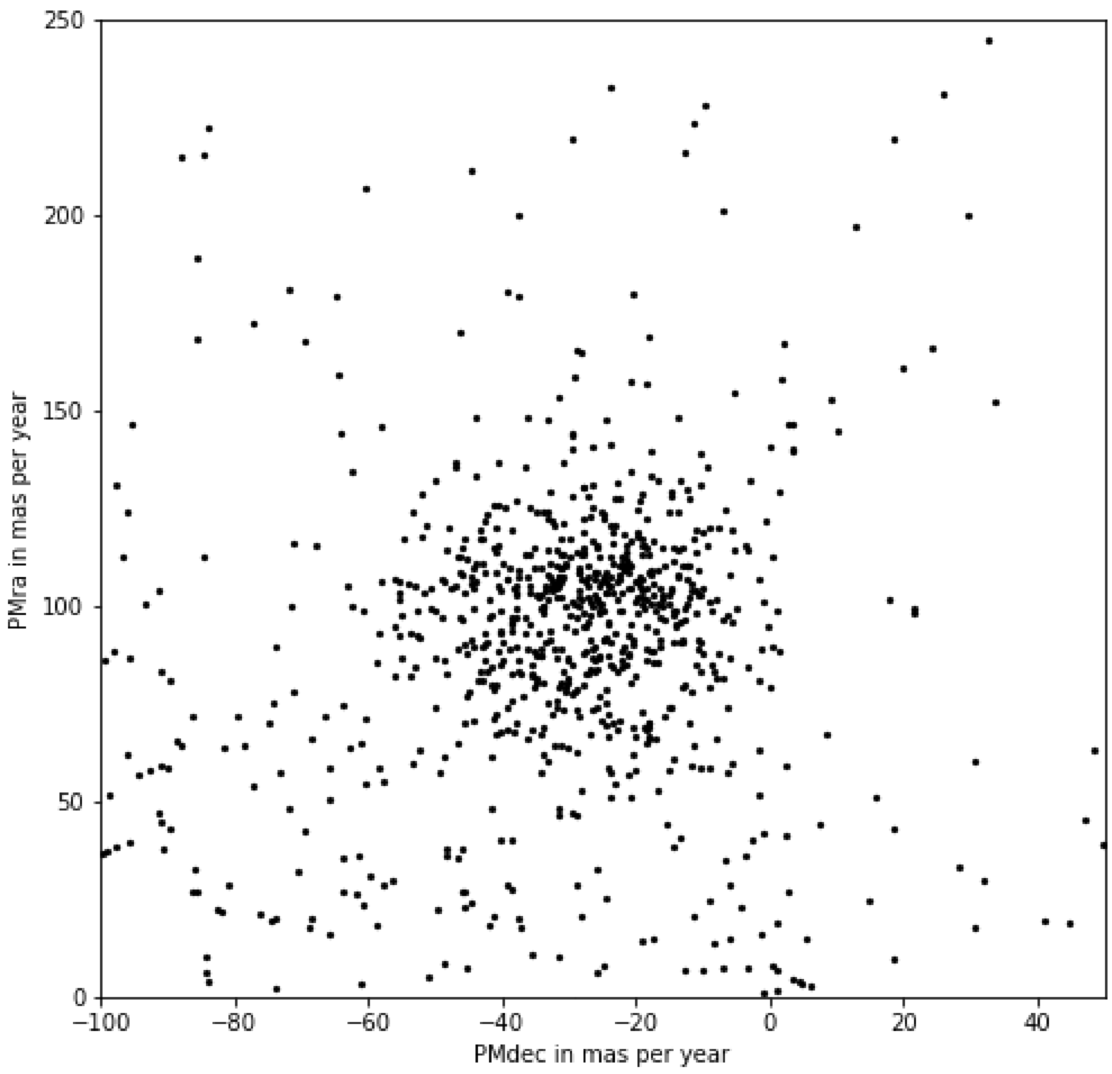
Keep a note of what range of parallax values you think best identifies where the parallax peak contains the most stars.

Next, we will look at the proper motions of the stars. The proper motion is the observed motion of the star on the sky. Because Gaia is observing the sky over many years, with high precision, we are able to measure very small motions of the nearby stars relative to more distant sources.

A cluster of stars, that formed from the same gas cloud, should also be moving at similar speeds and in similar directions. The values won't be exactly the same, as random motions in the original gas cloud result in some randomness in the stellar motions. But if you plot the proper motions of the stars in the catalog, hopefully you will see a clustering of stars at a particular velocity.

Save this plot and include it in your report.

[171]:
```
# First, plot pmra vs pmdec for all the stars as a scatter plot.
# Zoom in if you need to by setting plt.xlim() and plt.ylim().
# You are looking for a tight cluster of points in your x and y ranges.
fig=plt.figure(figsize=(8,8))
plt.scatter(pmdec, pmra, s=5, color='black')
plt.xlabel('PMdec in mas per year')
plt.xlim(-100,50)
plt.ylabel('PMra in mas per year')
plt.ylim(0,250)
fig.savefig('2_PMra_vs_PMdec.png')
```



Next, we will use your limits to select out only those stars that we feel are in the cluster. We do this by identifying the indices of stars whose proper motions and parallaxes fall within the ranges you identified above.

Then we can use these indices to select the values from the parameter lists of only those stars in the cluster for the following analysis.

[172]:
```
# Here, fill in your best estimates for the minimum and maximum values of pmra␣
  ↪and pmdec that best select
# the cluster stars only

pmra_min = 50
pmra_max =140
pmdec_min = -60
pmdec_max = 0
```

```
# Also list your minimum and maximum parallax limits from your histogram:

par_min = 20.5
par_max = 23
```

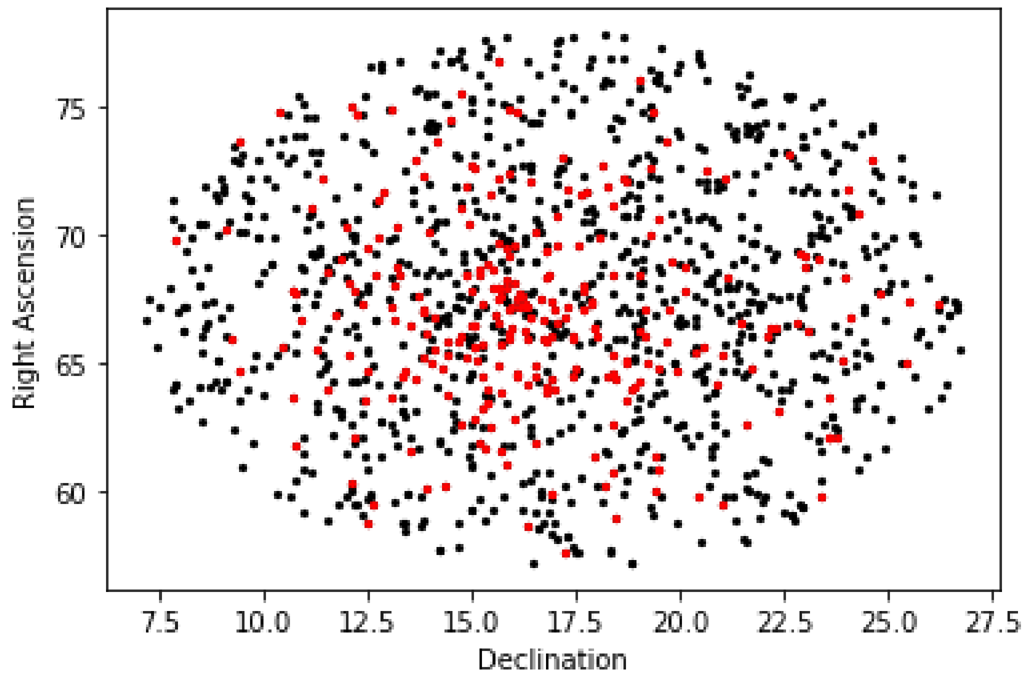
[173]:
```
# Use the np.where() function to get a list of indices. You can stack␣
↪requirements with the &.
cluster_indices = np.where((parallax>par_min) & (parallax<par_max) &␣
↪(pmra>pmra_min) & (pmra<pmra_max) & (pmdec>pmdec_min) & (pmdec<pmdec_max) )
```

Next, plot the star locations in right ascension and declination. First, plot all the stars. Next, plot only the stars that pass your requirements in proper motion and parallax using the syntax `ra[cluster_indices]` and `dec[cluster_indices]`. Use scatter plots, and make the true cluster stars a different colour.

Hopefully you will see that the cluster stars are more centred in your plot, as we might expect. If you don't see this, check your limits used in parallax and proper motion.

Save this plot and include it in your report.

[174]:
```
# Plot the star locations in RA and Dec
fig=plt.figure()
plt.scatter(dec, ra, s=5, color='black')
plt.xlabel('Declination')
plt.ylabel('Right Ascension')
plt.scatter(dec[cluster_indices], ra[cluster_indices], s=5, color='red')
fig.savefig('2_Ra_vs_Dec.png')
```



If you are satisfied with your limits, make new lists containing only the properties of the cluster members here.

Use the syntax `new_list = old_list[cluster_indices]`

Going forward, use the new lists for your analysis

[175]:
```
# Creating new lists
new_ra = ra[cluster_indices]
new_dec = dec[cluster_indices]
new_parallax = parallax[cluster_indices]
new_parallax_error = parallax_error[cluster_indices]
new_pmra = pmra[cluster_indices]
new_pmdec = pmdec[cluster_indices]
new_phot_g_mean_mag = phot_g_mean_mag[cluster_indices]
new_phot_bp_mean_mag = phot_bp_mean_mag[cluster_indices]
new_phot_rp_mean_mag = phot_rp_mean_mag[cluster_indices]
```

Calculate the mean and standard deviation of your parallaxes.

Use these values to calculate the distance to the Hyades cluster. What is your uncertainty in the result?

Check your results online with a reputable source. Do they agree?

[176]:
```
# Calculate mean and standard deviation of the parallax values.
# Is the standard deviation greater than or less than the parallax measurement␣
 ↪uncertainties?
####### Which one will contribute more to the uncertainty in your distance␣
 ↪measurement?
mean_parallax = np.mean(new_parallax)
std_parallax = np.std(new_parallax)
mean_parallax_error = np.mean(new_parallax_error)
print(std_parallax>mean_parallax_error)
print("Standard deviation: ", std_parallax, "mas")
```

```
True
Standard deviation:  0.6549392560463028 mas
```

[177]:
```
# Calculate the mean cluster distance based on your mean parallax value.
# Remember that these are in units of milli-arcseconds.
# Use partial derivatives to calculate your uncertainty in the distance␣
 ↪measurement, given the
# standard deviation in the parallax values.
parallax_in_arc = mean_parallax*0.001
mean_distance = 1/parallax_in_arc
upper_limit = 1/(parallax_in_arc - std_parallax*0.001)
```

```
lower_limit = 1/(parallax_in_arc + std_parallax*0.001)
print("Mean parallax: ", mean_parallax, "as")
print("Mean distance: ", mean_distance, "pc")
print("Upper limit of distance: ", upper_limit, "pc")
print("Lower limit of distance: ", lower_limit, "pc")
```

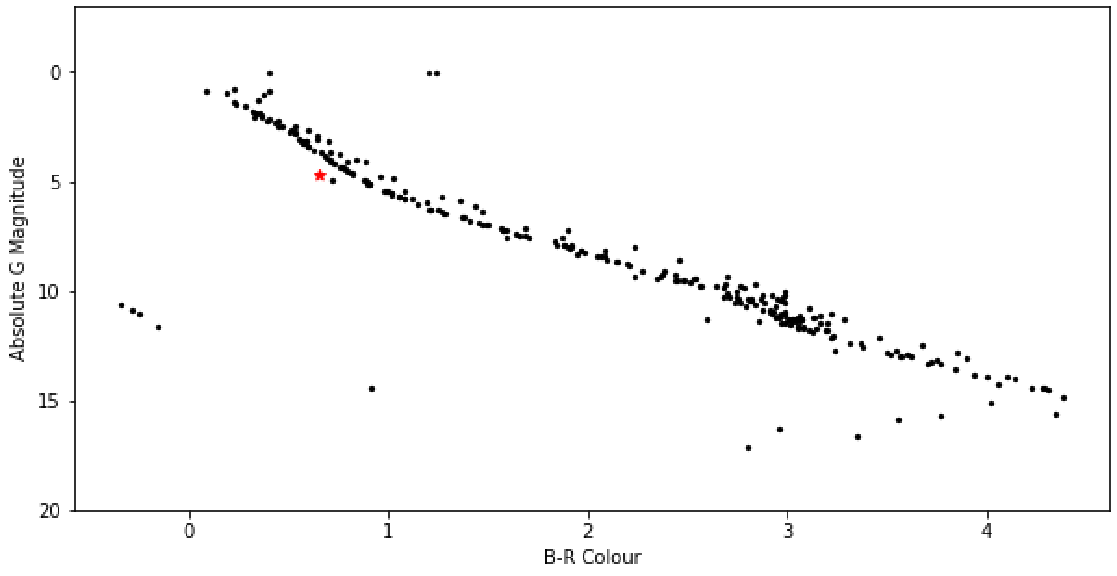
```
Mean parallax:  21.563422680284738 as
Mean distance:  46.37482717037736 pc
Upper limit of distance:  47.82747651801167 pc
Lower limit of distance:  45.00781843709275 pc
```

Since each star has a parallax measurement, we won't use the mean value to convert from apparent magnitudes to absolute magnitudes. Use your function above to calculate the absolute G magnitudes for the Hyades stars.

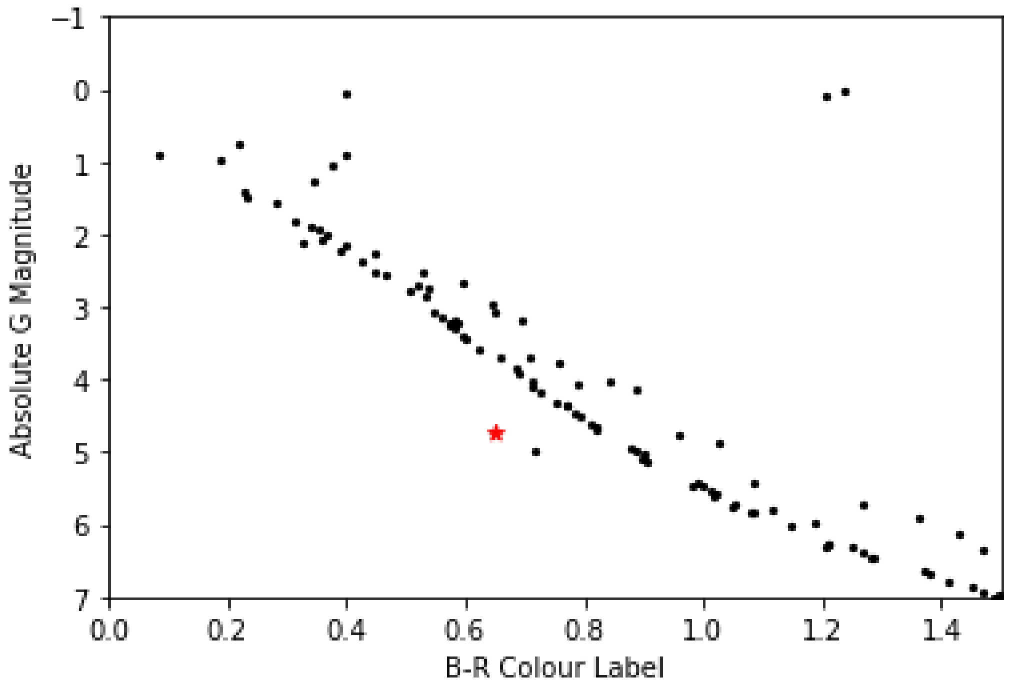
[178]:
```
# Calculate absolute magnitudes:
absg = absolute_mag(new_parallax, new_phot_g_mean_mag)
```

Next plot the Hyades CMD. You will need to calculate the B-R colour for the x axis (rather than B-V). Label your axes.

[179]:
```
# Calculate B-R colour for the Hyades:
br = new_phot_bp_mean_mag - new_phot_rp_mean_mag
# Plot the Hyades CMD
fig=plt.figure(figsize=(10,5))
plt.scatter(br, absg, s=5, color='black')
plt.xlabel('B-R Colour')
plt.ylabel('Absolute G Magnitude')
plt.ylim(20,-3)
plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')
fig.savefig('2_AbsG_vs_BR.png')
```



[180]:
```
# Calculate B-R colour for the Hyades:
br = new_phot_bp_mean_mag - new_phot_rp_mean_mag
# Plot the Hyades CMD
fig=plt.figure()
plt.scatter(br, absg, s=5, color='black')
plt.xlabel('B-R Colour Label')
plt.xlim(0,1.5)
plt.ylabel('Absolute G Magnitude')
plt.ylim(7,-1)
plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')
fig.savefig('AbsG_BR_H.png')
```



Hopefully your plot shows a clear main sequence, and maybe a hint of the turn off that shows at what mass stars are evolving off the main sequence. Estimate the absolute magnitude of a star at the main sequence turn off.

The absolute magnitude of the Sun in Gaia's G colour is $M_G = 4.67$.

Estimate the luminosity of a star at the turn off you see in your plot. It may help to draw a straight horizontal line on your plot above, or make an additional plot to zoom in on the turn off, to estimate a reasonable value. You can always add an extra code box where needed to do this.

Next, given your estimated L, and assuming $L \propto M^4$, estimate the mass of a star at the turn off you see in your plot. Does your result make sense given the Sun's location on the plot?

[181]:
```
# Calculations can go here:
turn_off_g_abs_mag = 1.5
sun_g_abs_mag = 4.67
turn_off_luminosity = 100**((sun_g_abs_mag-turn_off_g_abs_mag)/5)
turn_off_mass = (turn_off_luminosity)**0.25
print("Turn-off Luminosity:",turn_off_luminosity, "Lsun")
print("Turn-off Mass:",turn_off_mass, "Msun")
```

```
Turn-off Luminosity: 18.535316234148116 Lsun
Turn-off Mass: 2.07491351745491 Msun
```

Write your answers in markdown here (replace X, Y, and Z):

Absolute magnitude of a star at the main sequence turn off: 1.5mags

$L_{TO} = 18.5 L_\odot$

$M_{TO} = 2.1 M_\odot$

What is the nuclear timescale for a star with this mass? Show your calculation and answer below:

[182]:
```
#Nuclear timescale calculation
mass_sun = 2*(10**30)
luminosity_sun = 3.8*(10**26)
c = 3*(10**8)
t_sec = (0.1*0.007*turn_off_mass*mass_sun*(c**2))/
  ↪(turn_off_luminosity*luminosity_sun)
sec_in_year = 31536000
t_year = t_sec/31536000

print("Nuclear Timescale:", t_year, "years")
```

```
Nuclear Timescale: 1177010511.936743 years
```

Next you will compare your CMD for the Hyades with a set of isochrones. These are tables of the predicted absolute magnitude and colour of a cluster of stars at a particular age. If your distance calculations and conversion to absoluate magnitudes are correct, these isochrones should align with your observed main sequence for the Hyades.

The files give you their ages in the filename. For the Hyades, you will want to use the files that include 'z02' in their filename - these are made for stars with solar metallicity (same fraction of metals to hydrogen as the Sun).

An example filename is 'isochrone_z02_316Myr.dat'.

The columns are:

- Mass: stellar mass in units of $M_\odot$
- Gmag: absolute G magnitude
- G_BPmag: absolute B magnitude (in the Gaia system)
- G_RPmag: absolute R magnitude (in the Gaia system)

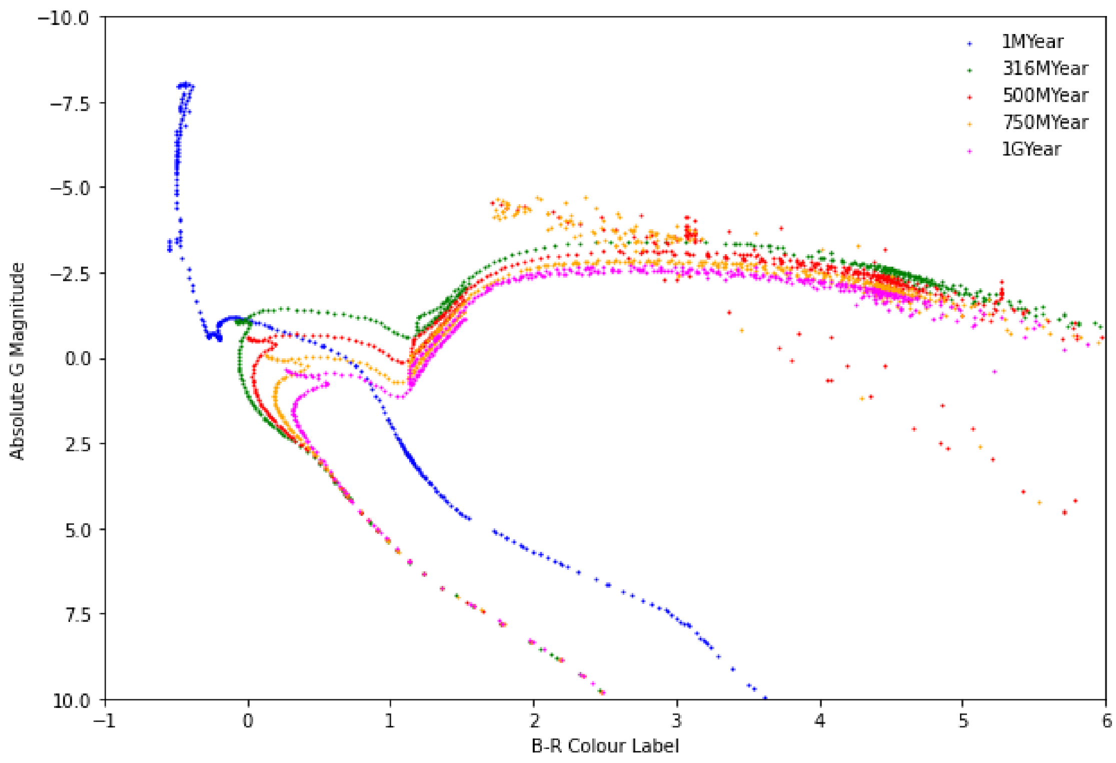
Read in the data. You can use `np.loadtxt` here, or `np.genfromtxt` - either should work fine as long as you set `delimiter=','`.

```
[183]: # Isochrones
mass_1M, gmag_1M, g_bpmag_1M, g_rpmag_1M = np.genfromtxt('isochrone_z02_1Myr.
  ↪dat', delimiter=',', unpack=True)
mass_316M, gmag_316M, g_bpmag_316M, g_rpmag_316M = np.
  ↪genfromtxt('isochrone_z02_316Myr.dat', delimiter=',', unpack=True)
mass_500M, gmag_500M, g_bpmag_500M, g_rpmag_500M = np.
  ↪genfromtxt('isochrone_z02_500Myr.dat', delimiter=',', unpack=True)
mass_750M, gmag_750M, g_bpmag_750M, g_rpmag_750M = np.
  ↪genfromtxt('isochrone_z02_750Myr.dat', delimiter=',', unpack=True)
mass_1G, gmag_1G, g_bpmag_1G, g_rpmag_1G = np.genfromtxt('isochrone_z02_1Gyr.
  ↪dat', delimiter=',', unpack=True)
```

First, plot the G magnitude vs. B-R colour for several of the isochrones against each other. You will likely still want to use `plt.scatter()` rather than `plt.plot`, as at later ages the stars fall in different locations in the diagram and having lines connecting them may make your plot confusing.

What differences do you see in the isochrone shapes? Consider the locations of: * the main sequence * the main sequence turn off * the giant branch * any other features you notice

```
[184]: # Plot several isochrones here. Play around with your x and y limits to show␣
  ↪the most important features.
fig=plt.figure(figsize=(10,7))
br_1M = g_bpmag_1M - g_rpmag_1M
br_316M = g_bpmag_316M - g_rpmag_316M
br_500M = g_bpmag_500M - g_rpmag_500M
br_750M = g_bpmag_750M - g_rpmag_750M
br_1G = g_bpmag_1G - g_rpmag_1G
plt.scatter(br_1M, gmag_1M, s=1, color='blue', label='1MYear')
plt.scatter(br_316M, gmag_316M, s=1, color='green', label='316MYear')
plt.scatter(br_500M, gmag_500M, s=1, color='red', label='500MYear')
plt.scatter(br_750M, gmag_750M, s=1, color='orange', label='750MYear')
plt.scatter(br_1G, gmag_1G, s=1, color='magenta', label='1GYear')
plt.legend(frameon = False)
plt.ylim(10,-10)
plt.xlim(-1,6)
plt.xlabel('B-R Colour Label')
plt.ylabel('Absolute G Magnitude')
fig.savefig('2_AbsG_vs_BR_Iso.png')
```



Plot your CMD again, and overlay the different isochrones in different colours. Include labels and a legend. You don't need to overlay ALL the isochrones, but show at least two. Find one that matches best with your data. Focus on matching the main sequence turn off.

For older isochrones in particular, you may want to limit the x and y axes to focus on where the isochrone lines up with your Hyades data.

Overlay the Sun again.

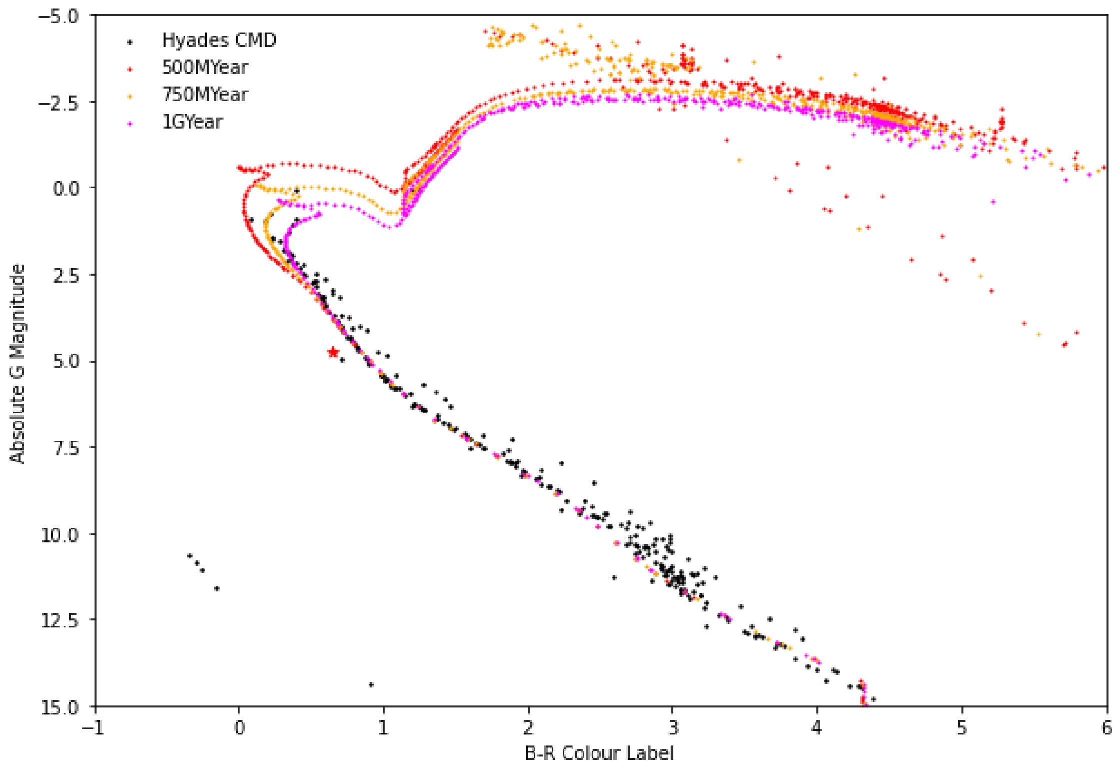
Save this plot and include it in your report.

[185]:
```
# Plot the cluster CMD with your chosen isochrones
fig=plt.figure(figsize=(10,7))
br = new_phot_bp_mean_mag - new_phot_rp_mean_mag
plt.scatter(br, absg, s=2, color='black', label='Hyades CMD')

br_500M = g_bpmag_500M - g_rpmag_500M
br_750M = g_bpmag_750M - g_rpmag_750M
br_1G = g_bpmag_1G - g_rpmag_1G
plt.scatter(br_500M, gmag_500M, s=1, color='red', label='500MYear')
plt.scatter(br_750M, gmag_750M, s=1, color='orange', label='750MYear')
plt.scatter(br_1G, gmag_1G, s=1, color='magenta', label='1GYear')

plt.legend(frameon = False)
plt.xlabel('B-R Colour Label')
plt.ylabel('Absolute G Magnitude')
plt.ylim(15,-5)
plt.xlim(-1,6)
plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')

fig.savefig('2_AbsG_vs_BR_H_Iso.png')
```



3.1 Answer these questions in Section 2 of your report

Do any of the isochrones line up well with your data? What age do you estimate for the Hyades cluster? Does this agree with your nuclear timescale calculation? Why or why not?

Check online via a reputable source the age of the Hyades cluster. Do your results agree?

4 Section 3: Gaia data for globular clusters

Next, you will perform a similar analysis for a globular cluster. There are several options for you to choose from with data linked on the project page.

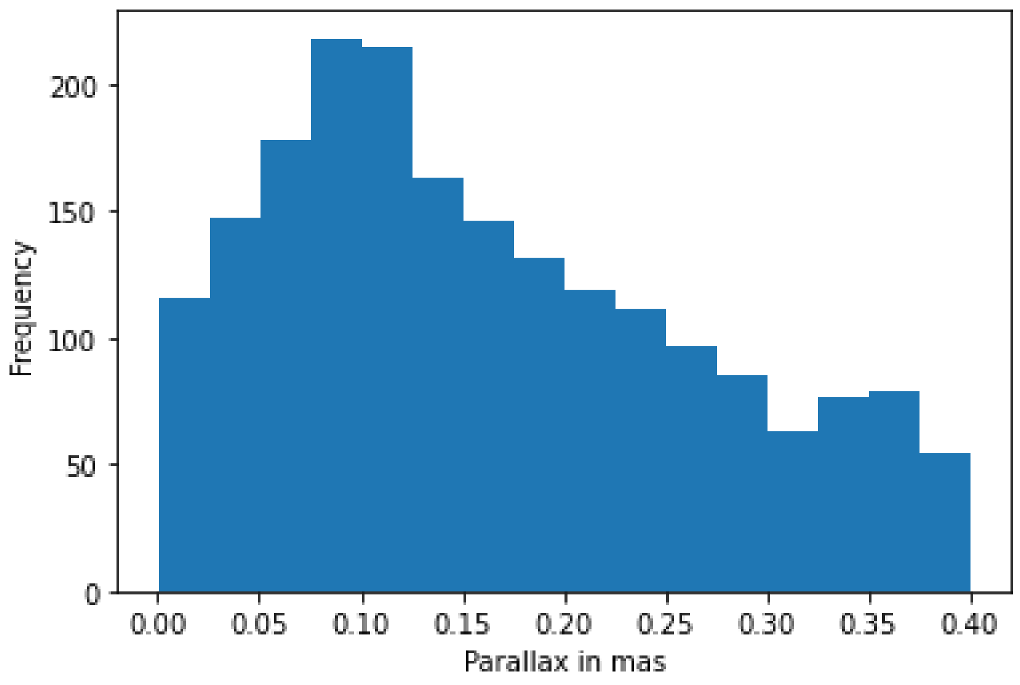
The steps are not given in detail in this case. Follow the same steps as for the Hyades, and save the same plots to include in your report.

For your chosen globular cluster, you should look up the accepted distance, and estimate at what parallax value you should expect to find a peak in the number of stars for that cluster. They are at different distances so the range in parallax values will be different for each.

When comparing with isochrones, use the files with ‘z001’ in the filename. These were made for stars with lower metallicity than the Sun. You should include one additional plot in your report for the globular cluster: a comparison plot between the 1 Gyr isochrones for low and high metallicity (‘z001’ vs. ‘z02’). What differences do you see? Do some research into why and include a brief discussion in your report.

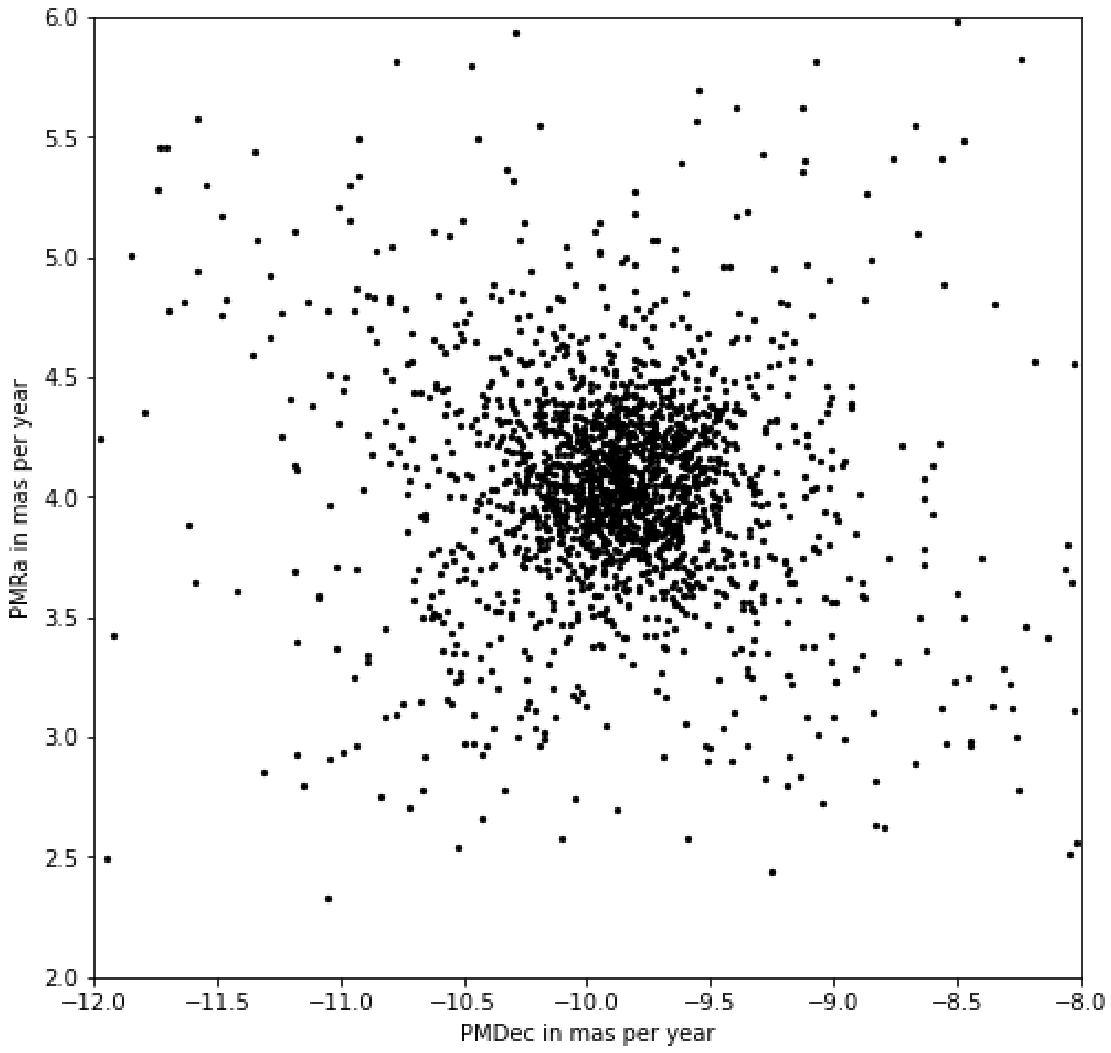
[186]:
```
ra, dec, parallax, parallax_error, pmra, pmdec, phot_g_mean_mag,␣
↪phot_bp_mean_mag, phot_rp_mean_mag = np.genfromtxt('gaia_m5_search.csv',␣
↪delimiter=',',usecols=(0,1,2,3,4,5,6,7,8), skip_header=1, unpack=True)
```

[187]:
```
fig=plt.figure()
plt.hist(parallax,bins='auto')
plt.xlabel('Parallax in mas')
plt.ylabel('Frequency')
fig.savefig('3_Parallax.png')
```



[188]:
```
fig=plt.figure(figsize=(8,8))
plt.scatter(pmdec, pmra, s=5, color='black')
plt.xlabel('PMDec in mas per year')
plt.xlim(-12,-8)
plt.ylabel('PMRa in mas per year')
plt.ylim(2,6)
```

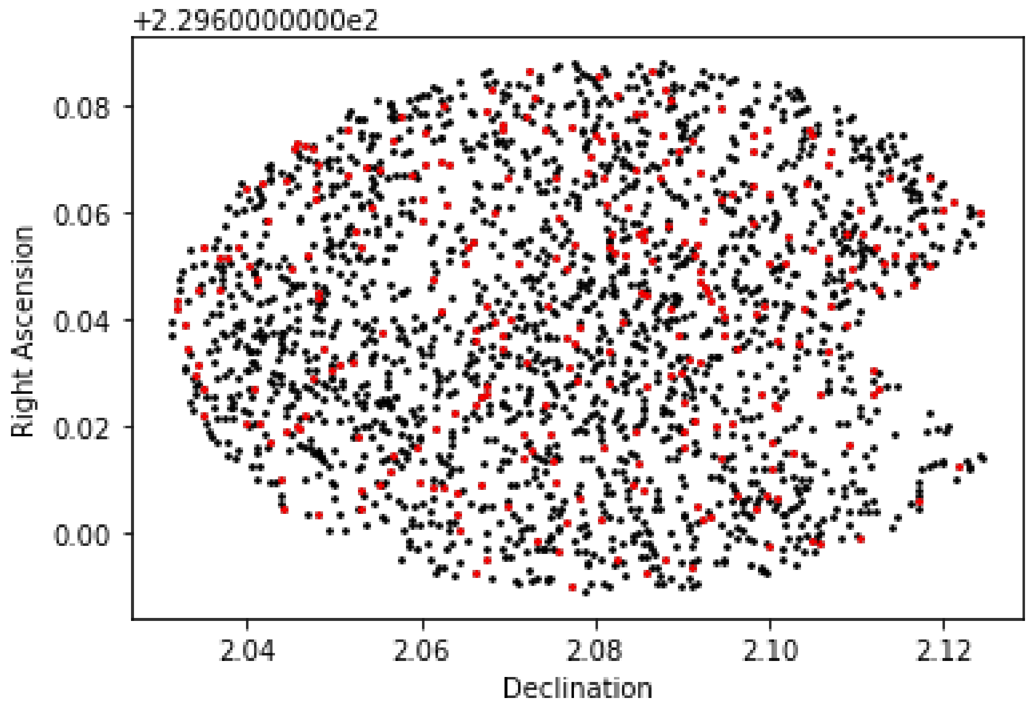
```
fig.savefig('3_PMra_vs_PMdec.png')
```



```
[189]: pmra_min = 3.5
       pmra_max = 4.5
       pmdec_min = -10.4
       pmdec_max = -9.6
       par_min = 0.1
       par_max = 0.155
```

```
[190]: cluster_indices = np.where((parallax>par_min) & (parallax<par_max) &␣
        ↪(pmra>pmra_min) & (pmra<pmra_max) & (pmdec>pmdec_min) & (pmdec<pmdec_max) )
```

```
[191]: fig=plt.figure()
       plt.scatter(dec, ra, s=3, color='black')
       plt.xlabel('Declination')
       plt.ylabel('Right Ascension')
       plt.scatter(dec[cluster_indices], ra[cluster_indices], s=3, color='red')
       fig.savefig('3_Ra_vs_Dec_.png')
```



[192]:
```
new_ra = ra[cluster_indices]
new_dec = dec[cluster_indices]
new_parallax = parallax[cluster_indices]
new_parallax_error = parallax_error[cluster_indices]
new_pmra = pmra[cluster_indices]
new_pmdec = pmdec[cluster_indices]
new_phot_g_mean_mag = phot_g_mean_mag[cluster_indices]
new_phot_bp_mean_mag = phot_bp_mean_mag[cluster_indices]
new_phot_rp_mean_mag = phot_rp_mean_mag[cluster_indices]
```

[193]:
```
# Which one will contribute more to the uncertainty in your distance␣
 ↪measurement?
mean_parallax = np.mean(new_parallax)
std_parallax = np.std(new_parallax)
mean_parallax_error = np.mean(new_parallax_error)
print(std_parallax>mean_parallax_error)
print("Standard deviation:", std_parallax, "mas")
```

```
False
```

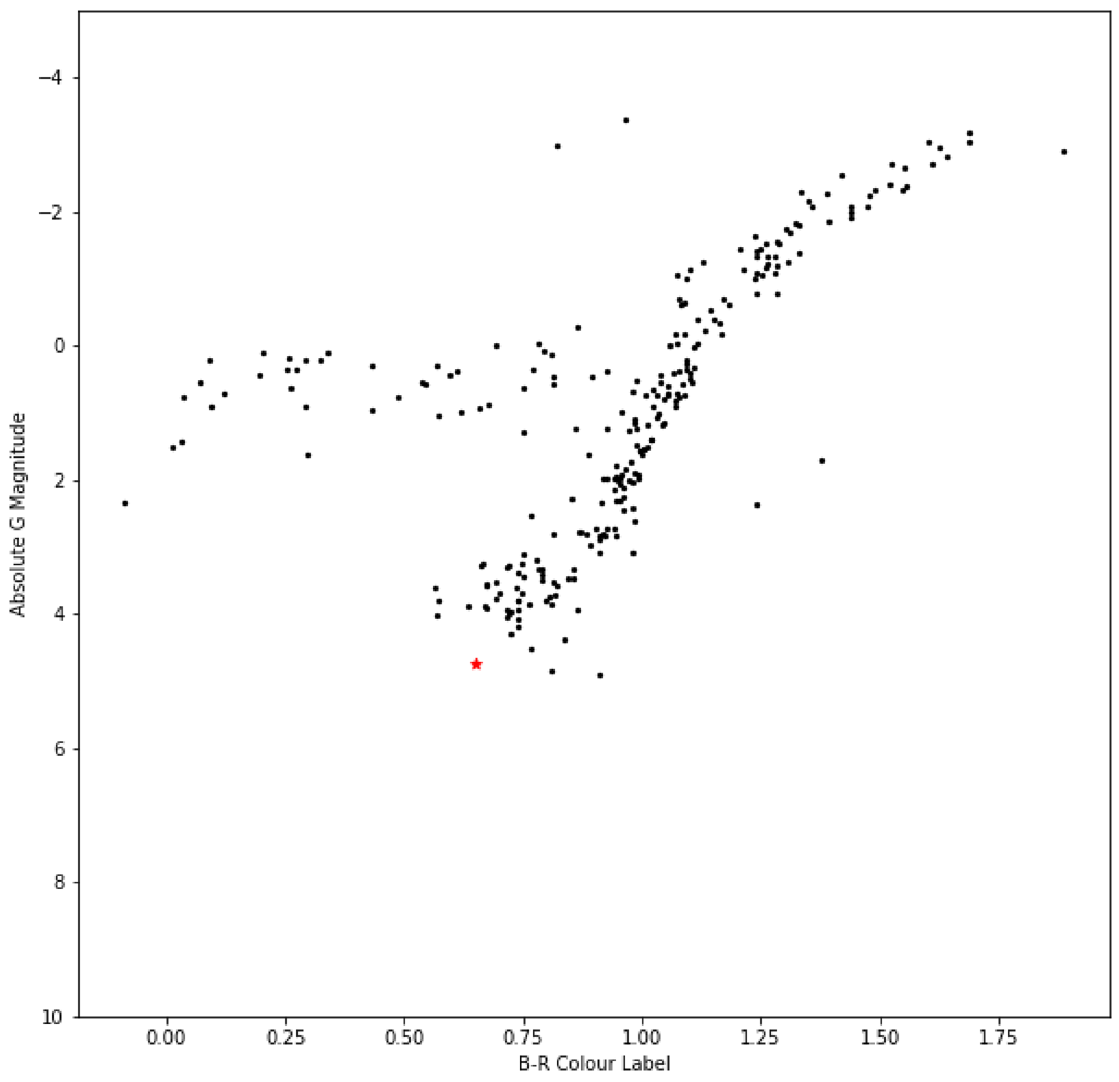
```
Standard deviation: 0.015034560761487816 mas
```

[194]:
```
# Use partial derivatives to calculate your uncertainty in the distance␣
  ↪measurement, given the
# standard deviation in the parallax values.
parallax_in_arc = mean_parallax*0.001
mean_distance = 1/parallax_in_arc
upper_limit = 1/(parallax_in_arc - std_parallax*0.001)
lower_limit = 1/(parallax_in_arc + std_parallax*0.001)
print("Mean parallax: ", mean_parallax, "as")
print("Mean distance: ", mean_distance, "pc")
print("Upper limit of distance: ", upper_limit, "pc")
print("Lower limit of distance: ", lower_limit, "pc")
```

```
Mean parallax:  0.12450316933154386 as
Mean distance:  8031.92404955624 pc
Upper limit of distance:  9135.038921774869 pc
Lower limit of distance:  7166.520476815028 pc
```

[195]:
```
absg = absolute_mag(new_parallax, new_phot_g_mean_mag)
```

[196]:
```
br = new_phot_bp_mean_mag - new_phot_rp_mean_mag
fig=plt.figure(figsize=(10,10))
plt.scatter(br, absg, s=5, color='black')
plt.xlabel('B-R Colour Label')
plt.ylabel('Absolute G Magnitude')
plt.ylim(10,-5)
plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')
fig.savefig('3_AbsG_vs_BR.png')
```



[197]:
```
turn_off_g_abs_mag = 4.6
sun_g_abs_mag = 4.67
turn_off_luminosity = 100**((sun_g_abs_mag-turn_off_g_abs_mag)/5)
turn_off_mass = turn_off_luminosity**4
print("Turn off luminosity: ",turn_off_luminosity, "Lsun")
print("Turn off mass: ", turn_off_mass, "Msun")
```

```
Turn off luminosity:  1.0665961212302582 Lsun
Turn off mass:  1.2941958414499877 Msun
```

[198]:
```
mass_sun = 2*(10**30)
luminosity_sun = 3.8*(10**26)
mass_sun = 2*(10**30)
luminosity_sun = 3.8*(10**26)
```

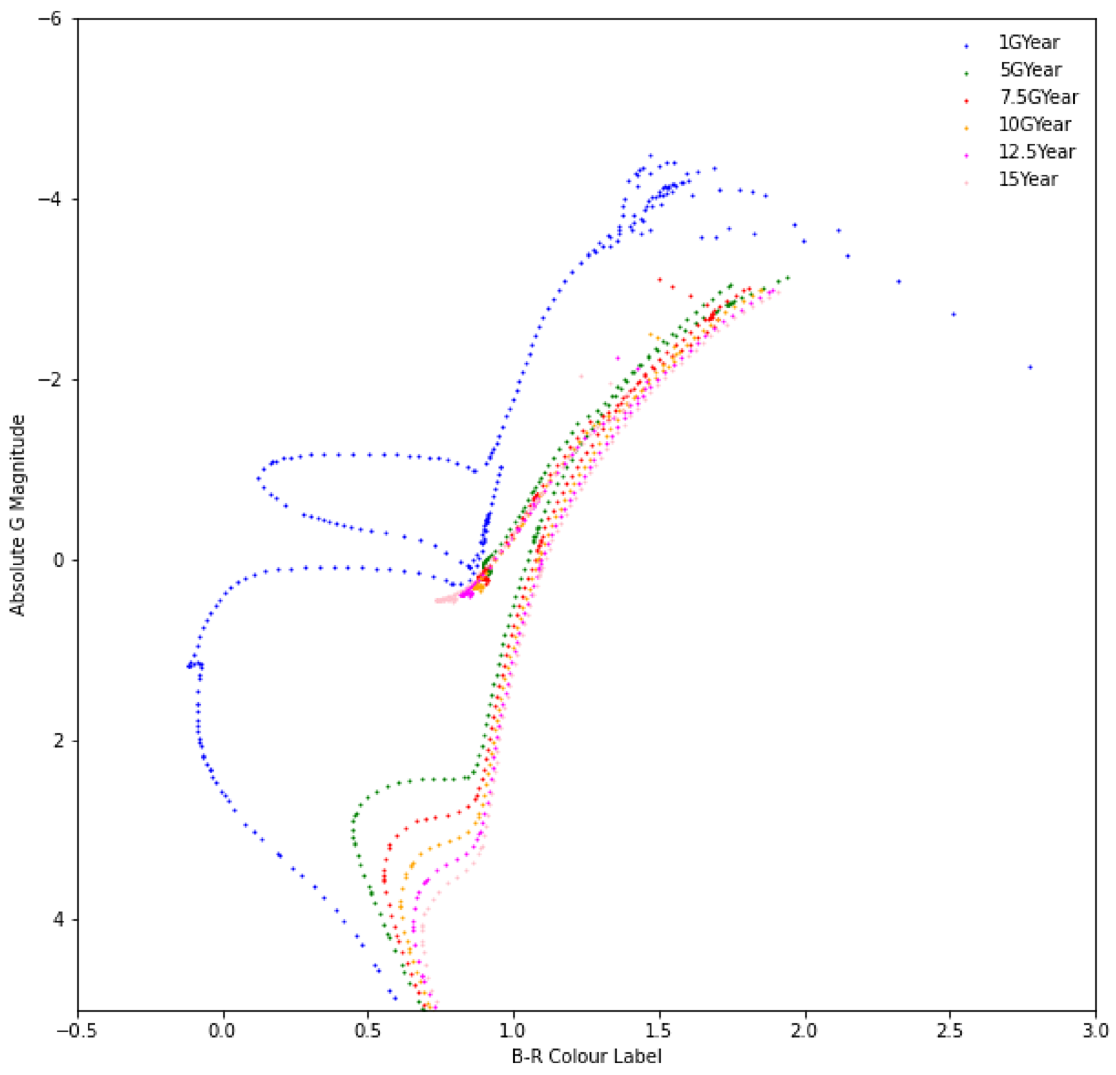
```
c = 3*(10**8)
t_sec = (0.1*0.007*turn_off_mass*mass_sun*(c**2))/
  ↪(turn_off_luminosity*luminosity_sun)
sec_in_year = 31536000
t_year = t_sec/31536000

print(t_year, "years")
```

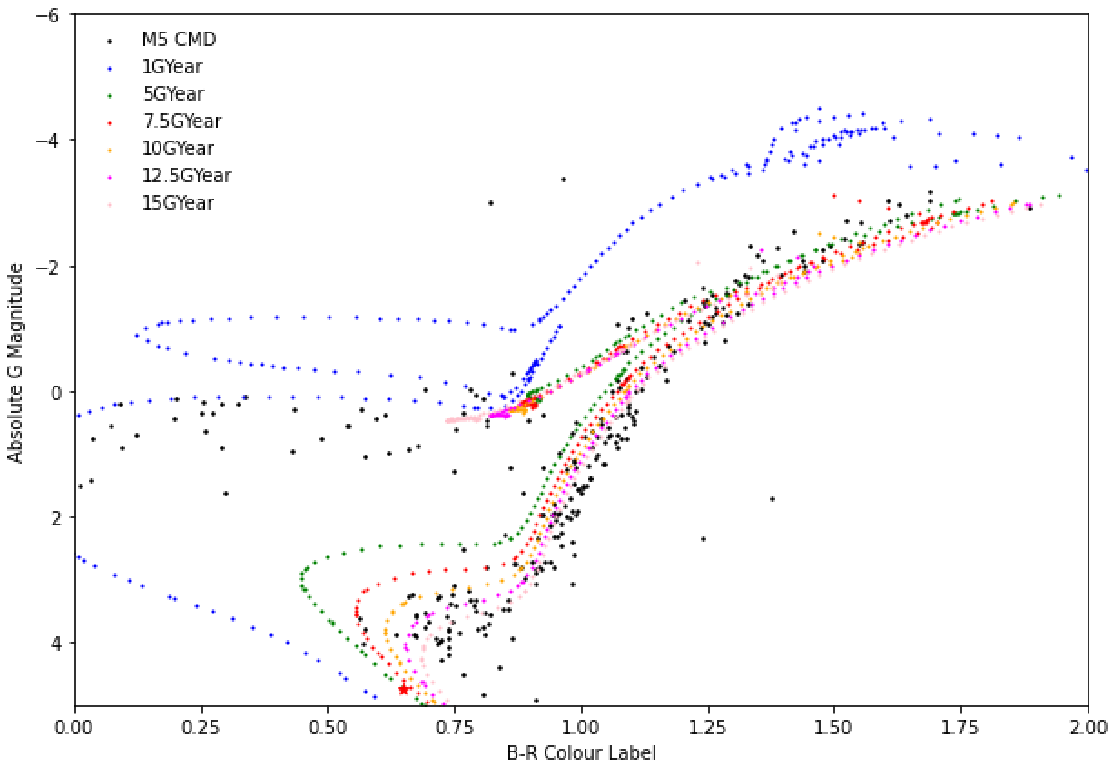
```
12757933719.741125 years
```

[199]:
```
# Isochrones
mass_1G, gmag_1G, g_bpmag_1G, g_rpmag_1G = np.genfromtxt('isochrone_z001_1Gyr.
  ↪dat', delimiter=',', unpack=True)
mass_5G, gmag_5G, g_bpmag_5G, g_rpmag_5G = np.genfromtxt('isochrone_z001_5Gyr.
  ↪dat', delimiter=',', unpack=True)
mass_7G, gmag_7G, g_bpmag_7G, g_rpmag_7G = np.genfromtxt('isochrone_z001_7.5Gyr.
  ↪dat', delimiter=',', unpack=True)
mass_10G, gmag_10G, g_bpmag_10G, g_rpmag_10G = np.
  ↪genfromtxt('isochrone_z001_10Gyr.dat', delimiter=',', unpack=True)
mass_12G, gmag_12G, g_bpmag_12G, g_rpmag_12G = np.genfromtxt('isochrone_z001_12.
  ↪5Gyr.dat', delimiter=',', unpack=True)
mass_15G, gmag_15G, g_bpmag_15G, g_rpmag_15G = np.
  ↪genfromtxt('isochrone_z001_15Gyr.dat', delimiter=',', unpack=True)
```

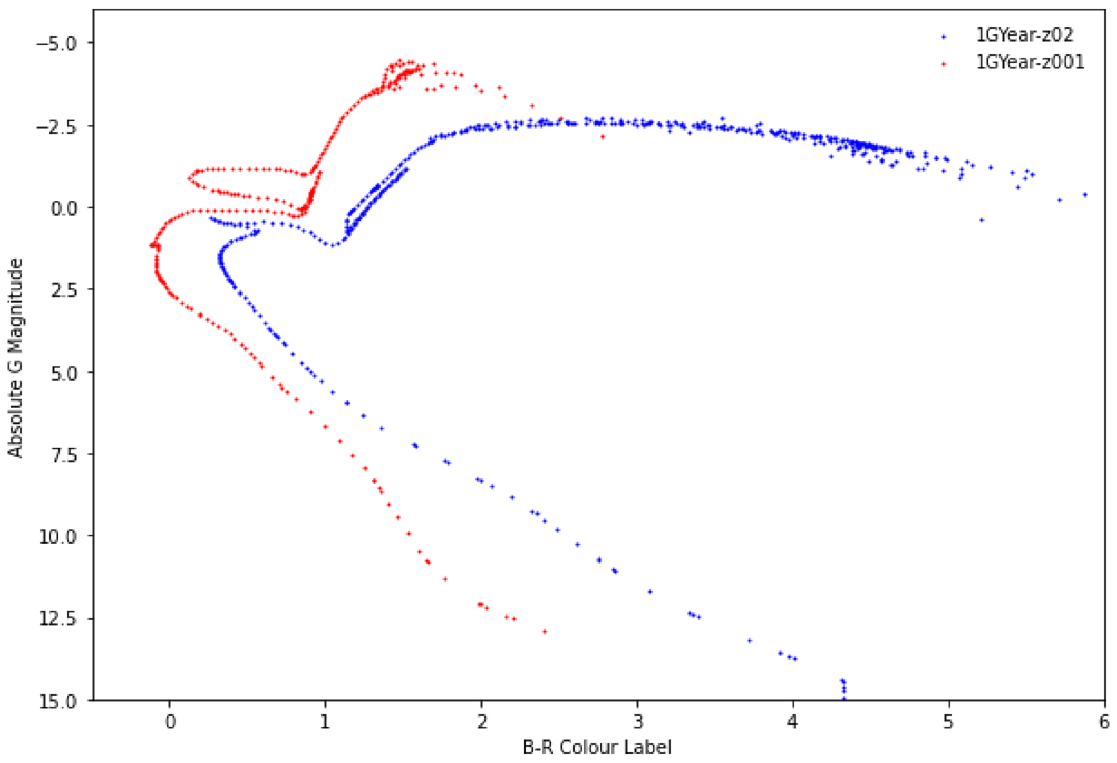
[200]:
```
fig=plt.figure(figsize=(10,10))
br_1G = g_bpmag_1G - g_rpmag_1G
br_5G = g_bpmag_5G - g_rpmag_5G
br_7G = g_bpmag_7G - g_rpmag_7G
br_10G = g_bpmag_10G - g_rpmag_10G
br_12G = g_bpmag_12G - g_rpmag_12G
br_15G = g_bpmag_15G - g_rpmag_15G
plt.scatter(br_1G, gmag_1G, s=1, color='blue', label='1GYear')
plt.scatter(br_5G, gmag_5G, s=1, color='green', label='5GYear')
plt.scatter(br_7G, gmag_7G, s=1, color='red', label='7.5GYear')
plt.scatter(br_10G, gmag_10G, s=1, color='orange', label='10GYear')
plt.scatter(br_12G, gmag_12G, s=1, color='magenta', label='12.5Year')
plt.scatter(br_15G, gmag_15G, s=1, color='pink', label='15Year')
plt.legend(frameon = False)
plt.ylim(5,-6)
plt.xlim(-0.5,3)
plt.xlabel('B-R Colour Label')
plt.ylabel('Absolute G Magnitude')
fig.savefig('3_AbsG_vs_BR_Iso.png')
```



```
[201]: fig=plt.figure(figsize=(10,7))
       br = new_phot_bp_mean_mag - new_phot_rp_mean_mag
       plt.scatter(br, absg, s=2, color='black', label='M5 CMD')
       plt.scatter(br_1G, gmag_1G, s=1, color='blue', label='1GYear')
       plt.scatter(br_5G, gmag_5G, s=1, color='green', label='5GYear')
       plt.scatter(br_7G, gmag_7G, s=1, color='red', label='7.5GYear')
       plt.scatter(br_10G, gmag_10G, s=1, color='orange', label='10GYear')
       plt.scatter(br_12G, gmag_12G, s=1, color='magenta', label='12.5GYear')
       plt.scatter(br_15G, gmag_15G, s=1, color='pink', label='15GYear')
       plt.legend(frameon = False)
       plt.xlabel('B-R Colour Label')
       plt.ylabel('Absolute G Magnitude')
       plt.ylim(5,-6)
       plt.xlim(0,2)
       plt.scatter(0.65,4.74, marker = '*', c = 'red', label='Sun')
       fig.savefig("3_AbsG_vs_BR_M5&Iso")
```



[202]:
```
fig=plt.figure(figsize=(10,7))
mass_z02_1G, gmag_z02_1G, g_bpmag_z02_1G, g_rpmag_z02_1G = np.
  ↪genfromtxt('isochrone_z02_1Gyr.dat', delimiter=',', unpack=True)
mass_1G, gmag_1G, g_bpmag_1G, g_rpmag_1G = np.genfromtxt('isochrone_z001_1Gyr.
  ↪dat', delimiter=',', unpack=True)
br_z02_1G = g_bpmag_z02_1G - g_rpmag_z02_1G
br_1G = g_bpmag_1G - g_rpmag_1G
plt.scatter(br_z02_1G, gmag_z02_1G, s=1, color='blue', label='1GYear-z02')
plt.scatter(br_1G, gmag_1G, s=1, color='red', label='1GYear-z001')
plt.legend(frameon = False)
plt.ylim(15,-6)
plt.xlim(-0.5,6)
plt.xlabel('B-R Colour Label')
plt.ylabel('Absolute G Magnitude')
fig.savefig("Comparing_metallicity")
```


1GYear-z02
1GYear-z001
Absolute G Magnitude
B-R Colour Label
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0
1
2
3
4
5
6