```python
import numpy as np
import matplotlib.pyplot as plt
from astropy.table import Table
from ezpadova import parsec

import os
home = os.getenv('HOME')

# This is the only thing that needs to switch depending on if you're working at the lab
# or on your own computer
onedrive = "C:\\Users\\aka188\\OneDrive - Simon Fraser University (1sfu)"

# This is your private directory.  You should save this python script there too.
working_dir = onedrive + '/PHYS391-Spring-2025-private/Assignment5'

#-----------------------------------------------------------------------------------------
# Get the magnitudes of the stars in M37

# Read the table you just created in photometry.py
masterTable = Table.read('masterTable.fits')

# Get the B-V and V-mag information from the masterTable
magV_M37 = masterTable["mag_PhotV"] # Apparent mag in V
bv_M37 = masterTable["mag_PhotB"] - masterTable["mag_PhotV"] # B-V

# Our image of M37 probably includes a lot of stars that are not part of the cluster.
# Let's limit our selection to within a certain radius of the cluster centre.
# Use DS9 to estimate the cluster's centre in pixels, and also its radius in pixels.
# Then create a mask that is the same length as masterTable, and is True for stars that
# are less within the radius of the cluster.
xcluster,ycluster = 2796 , 2069
rcluster = 800

# The distance of the stars from the cluster centre in pixels (use the xwin_image and
# ywin_image coordinates).
dist = np.sqrt((masterTable['xwin_image'] - xcluster)**2 + (masterTable['ywin_image'] - ycluster)**


# Create the mask that will apply to the mastertable (where dist is less the radius that
# you picked)
maskCluster = (dist < rcluster)

#-----------------------------------------------------------------------------------------
# Download the Padova isochrones (use intervals of 0.1 in logAge)
# No need to modify this section
iso = parsec.get_t_isochrones(6.0, 10.0, 0.1, 0.019)

# Get rid of the post-AGB stars
mask = (iso['label'] != 9)
iso = iso[mask]

# Get the B-V and V mag values from the iso table
bv_isochrone = iso['Bmag']-iso['Vmag']
v_isochrone = iso['Vmag']

# Get the log of the ages from the iso table
logAge_isochrone_unique = np.log10(np.unique(iso['Age'])) # Only the unique values
```

```python
logAge_isochrone = np.log10(iso['Age']) # All the values in the iso table


#-------------------------------------------------------------------------------------------
# Some functions to plot things
def plotCMD():
    # Plot the stars on a colour-magnitude diagram, using the bv_M37 and magV_M37
    # global variables that we defined above.

    # First plot all the stars as "lightgrey" dots. Use the "label" keyword to call
    # these "Non-cluster stars".
    plt.figure()
    plt.plot(bv_M37[~maskCluster], magV_M37[~maskCluster], 'o', color='lightgrey', label='Non-clust
    # Then plot only the cluster stars as blue dots labeled "Cluster stars"
    plt.plot(bv_M37[maskCluster], magV_M37[maskCluster], 'o', color='blue', label='Cluster stars')
    # Label the axes

    plt.xlabel("B-V")
    plt.ylabel("V")

    # Limit the axes to show where most of the stars are
    plt.ylim([16.5, 7.5])
    plt.xlim([0, 2.2])


def plotIsochrone(logAge,DM,EBV):
    # This function plots a single isochone with a given logAge, E(B-V), and distance
    # modulus (modify only 3 lines marked with ...)

    # Find the value in logAge_isochrone_unique that is the closest to logAge:
    idx = np.argmin(np.abs(logAge_isochrone_unique-logAge))
    closestLogAge = logAge_isochrone_unique[idx]

    # Create the mask that selects only those items in logAge_isochrone that correspond
    # to that closestLogAge
    mask = (logAge_isochrone == closestLogAge)
    bv_iso_choose = bv_isochrone[mask]
    v_iso_choose = v_isochrone[mask]

    # Compute what the isochrone (bv_iso_choose and v_iso_choose) would like after the
    # distance modulus and reddening are applied
    Av = 3.1 * EBV # The attenuation in V (refer to Lecture 6.5)
    bv_apparent = bv_iso_choose + EBV # Use bv_iso_choose and EBV
    v_apparent = v_iso_choose + DM + Av # Use v_iso_choose, DM and Av

    # Plot the isochrone as a red curve
    label = f'log Age: {logAge:.2f}\nDistance Modulus: {DM:.2f}\nE(B-V): {EBV:.2f}'
    pltiso, = plt.plot(bv_apparent,v_apparent,c='red',label=label)
    return pltiso

#-------------------------------------------------------------------------------------------
# Run the next two lines a few times, with different values in plotIsochrone()
# until you get a good fit by eye.
plotCMD() # Then plot the CMD
pltiso = plotIsochrone(logAge=8.75,DM= 9.25,EBV=0.45)
#pltiso = plotIsochrone(logAge=8.85, DM=10.5, EBV=0.32)
# Once you've found a good fit, run these lines:
```

```
plt.title('Colour-Magnitude Diagram of M37')

plt.legend()
plt.savefig(working_dir+'/fitByEye.png')

# plt.close()

# # Now you can continue:
# #------------------------------------------------------------------------------------------
# # Numerical isochrone fitting
# def isochroneDistance(bv,v,lage,dm,ebv):

#     # logAge_isochrone_unique is sorted.  lage is a value that is between two values in
#     # logAge_isochrone_unique.  idx is the index for the value below lage, and idx+1 is
#     #the index of the value above lage.
#     idx = np.searchsorted(logAge_isochrone_unique,lage) - 1

#     # The linear interpolation factor to be used a few lines down
#     factor = (lage - logAge_isochrone_unique[idx])/ \
#              (logAge_isochrone_unique[idx+1]-logAge_isochrone_unique[idx])

#     # What is the attenuation in v?
#     Av = 3.1*ebv

#     # This line creates a 2D array with v and bv values, which are the coordinates of
#     # the stars on the CMD.
#     # Needs to be transposed in order to input to cdist.
#     star_coords = np.array([v,bv]).T

#     # Initialize a 2D array of distances on the CMD
#     twodists = np.empty([v.size,2])
#     for i in range(2): # loops through the two isochrones that are closest in age to the given la
#         mask = (logAge_isochrone == logAge_isochrone_unique[idx+i]) # select the lines in iso tab
#         v_iso = v_isochrone[mask] + dm + Av # Find what the apparent magnitude of the isochrone w
#         bv_iso = bv_isochrone[mask] + ebv # Find what the B-V of the isochrone would be given the

#         iso_coords = np.array([v_iso,bv_iso]).T # These are the coordinates of the isochrone poir

#         # cdist computes the distances of each star to the each isochrone point, creating a big 2
#         # np.nanmin finds the minimum distance for each star (along one axis of the 2D array).
#         twodists[:,i] = np.nanmin(cdist(iso_coords,star_coords),axis=0)

#     dist = (twodists[:,1] - twodists[:,0])*factor + twodists[:,0] # Interpolate between the minim
#     dist_masked = sigma_clip(dist,3.0) # Use sigma_clip to keep only the stars within a distance
#     dist[dist_masked.mask] = 0.0 # Assign the clipped stars a value of 0

#     return dist

# # This is the function that is called by mpfit
# def fitfunction(params,x=None,y=None,err=None,fjac=None):
#     lage,dm,ebv = params
#     bv = x
#     vmag = y

#     dist = isochroneDistance(bv,vmag,lage,dm,ebv)
#     status = 0
```

```
#     return status, dist # dist is the value that will be minimized by mpfit

# # params are the starting values that will be input to mpfit
# params = [8.9,9.5,0.33] # Write the values of three parameters that you found when fitting by eye
# kw = {'y': magV_M37[maskCluster], 'x': bv_M37[maskCluster]}
# parinfo = [{'parname': 'logAge (yr)',      'value': params[0], 'fixed': False, 'limited': [True,Tr
#            {'parname': 'Distance Modulus','value': params[1], 'fixed': False, 'limited': [True,Fa
#            {'parname': 'E(B-V)',          'value': params[2], 'fixed': False, 'limited': [True,Fa
# # (in the above parinfo, 'limited' tells mpfit whether the parameters should be limited to certai
# # and 'limits' gives what those limits are)

# # Call mpfit
# m = mpfit(fitfunction,functkw=kw,parinfo=parinfo)


# #---------------------------------------------------------------------------------------
# # Plot the result
# lage,dm,ebv = m.params

# plotCMD()
# plotIsochrone(lage,dm,ebv)
# plt.legend()
# plt.title('Numerical Fit')
# plt.savefig(working_dir+'/numericalfit.png')
```