```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  6 16:15:04 2020

@author: jimmy
"""
import numpy as np
import matplotlib.pyplot as plt
import time

# Class to generate intensity-weighted stellar profile
class LimbDarkening():

    def __init__(self,star_temp,gridsize):
        # Inputs:
        #   star_temp: surface temperature of star
        #   gridsize: length and width of grid of points

        self.star_temp=star_temp
        self.gridsize = gridsize

        #   a: first parameter in quadratic limb darkening equation
        #   b: second parameter in quadratic limb darkening equation
        if self.star_temp == 5500:
            self.u1 = 633.27/1000
            self.u2 = 159.56/1000
        elif self.star_temp == 10000:
            self.u1 = .2481
            self.u2 = .2739
        elif self.star_temp == 3600:
            self.u1 = .626
            self.u2 = .226

    # Function to calculate quadratic limb darkening profile
    def QuadIntensity(self,x,y):
        # Inputs:
        #   x: x-coordinate to calculate intensity at
        #   y: y-coordinate to calculate intensity at
        # Returns:
        #   Intensity at that location

        # Set intensity at center of star
        R_star = 1.0

        # Calculate distance from center
        r = np.sqrt(x**2+y**2)

        # Calculate mu and terms that use mu
        mu = np.sqrt(1-abs(r**2/R_star**2))

        first_term = self.u1*(1.0-mu)
        second_term = self.u2*(1.0-mu)**2

        # Calculate intensity at r
        intensity = 1.0*(1-first_term-second_term)

        return(intensity)

    # Function to plot intensity of points on stellar disc
    def Star(self,plot=True):
        # Inputs:
        #   plot: boolean to choose to plot star or not
        # Returns:
        #   intensity colormap of star (if plot=True)
        #   grid of x and y coordinates & intensities at each coordinate
```

```python
        # Set lower bounds and size of grid
        x0, y0 = -1.0,-1.0

        # Generate list of x and y coordinates from near center to 1 R*
        x_list = np.linspace(x0,1.0,self.gridsize)
        y_list = np.linspace(y0,1.0,self.gridsize)
        x,y = np.meshgrid(x_list,y_list)

        # Calculate intensity at each x,y pair
        intensities = self.QuadIntensity(x,y)

        # Plot color grid of intensities at each location
        if plot == True:
            plt.pcolor(x,y,intensities,cmap='hot',shading='nearest')
            cbar = plt.colorbar()
            cbar.set_label('Surface Brightness',fontsize=14)
            plt.xlabel(r'x ($R_{star}$)',fontsize=14)
            plt.ylabel(r'y ($R_{star}$)',fontsize=14)
            plt.title('Surface Brightness of T={0}K Star \n (at '.format(self.star_temp)+r'$5000
            plt.xlim(-1.2,1.2)
            plt.ylim(-1.1,1.1)
            plt.tight_layout()

        return(x,y,intensities)

    # Place star at particular point in
    def Transit(self,rad_planet,b,plot=False):
        # Inputs:
        #   rad_planet: fractional size of planet in terms of stellar radius
        #   b: impact parameter of transit (ranges from 0 to 1)
        #   plot: boolean to decide if visualiz. of transit is shown
        # Returns:

        # Figure out time code started to be used
        start_time = time.time()

        # Generate intensity-weighted coordinate grid
        x_grid,y_grid,intensities_star = self.Star(plot=False)

        # Calculate total intensity of surface elements with no transit
        original_total = np.nansum(intensities_star)

        # Make empty list of relative intensities
        light_curve = []

        # Set loop counter
        i=0

        # Establish array to add data to
        data = np.zeros([len(x_grid[0]), 5])

        # Calculate intensity from visible star throughout transit
        for x in x_grid[0]:

            # Identify location of planet center
            planet_center = [-x,b]

            # Calculate x,y, and total distances of all points from planet center
            xdist = x_grid - planet_center[0]
            ydist = y_grid - planet_center[1]
            dist = np.sqrt(xdist**2+ydist**2)

            # Find pixels within planet radius
            planet_ids = np.where(dist < rad_planet)

            # Set intensity to 0 wherever planet blocks the star
            non_transit_intensities = intensities_star[planet_ids]
```

```
        intensities_star[planet_ids] = 0

        # Remove NaNs from intensity list
        real_intensities_transit = [z for z in intensities_star.flatten() if ~np.isnan(z)]

        # Calculate total observed intensity at this point in transit
        transit_total = np.nansum(intensities_star)

        # Calculate relative intensity to non-transit
        light_fraction = transit_total/original_total
        light_curve.append(light_fraction)

        # Print status of loop
        #print("Now completing Loop {0} out of {1}: Rel. Intens. = {2:.5f}".format(i,len(x_g

        # Plot star with planet in front if user desires
        if plot==True:
            # Initialize figure and axis object for plotting
            fig, ax = plt.subplots()

            # Plot star
            ax.pcolor(x_grid,y_grid,intensities_star,cmap='hot',shading='nearest')
            ax.set_xlim(-1.2,1.2)
            ax.set_ylim(-1.1,1.1)
            ax.set_facecolor('black')
            #cbar = plt.colorbar(ax)
            #cbar.set_label('Surface Brightness',fontsize=14)
            ax.set_xlabel(r'x ($R_{star}$)',fontsize=14)
            ax.set_ylabel(r'y ($R_{star}$)',fontsize=14)
            ax.set_title('Surface Brightness of T={0}K Star \n (at '.format(self.star_temp)+
            #fig.savefig("C:/Users/Jimmy/Downloads/Test/test_{0}.png".format(i),)
            plt.pause(0.05)

            plt.tight_layout()
            #fig.canvas.draw()
        plt.show()

        # Save important data
        data[i] = self.star_temp, rad_planet, b, -x, light_fraction

        # Reset intensities to original
        intensities_star[planet_ids] = non_transit_intensities
        #print(np.nansum(non_transit_intensities) - np.nansum(intensities_star[planet_ids]))
        i += 1

        # Determine how long program has been running
        #looptime = time.time() - start_time
        #print("Time elapsed: {0:.3f}".format(looptime))

    """# Save important data to text file (only has to be run once)
    fileout = 'C:/Users/Jimmy/ASTR5490/HW3/TransitData/Transit_{0}Rstar_b={1}_{2}K.dat'.form
    np.savetxt(fileout, data, fmt = "%11.2f %11.2f %11.2f %11.9f %11.9f",comments='#',
           header="{:^10s}{:^11s}{:^11s}{:^11s}{:^11s}"\
                  .format('star_temp(K)','rad_planet(R*)', 'b', 'x_pos', 'rel_intens'))"""

    # Plot transit light curve
    plt.scatter(x_grid[0],light_curve)
    plt.xlabel(r'Horizontal Distance from Star Center ($R_{star}$)',fontsize=14)
    plt.ylabel('Relative Intensity',fontsize=14)
    plt.title('Transit of {0}'.format(rad_planet)+r'$R_{star}$ Planet'\
              +'\n'+r'($T_{star}$ = '+'{0}K, b = {1})'.format(self.star_temp,b),fontsize=18)
    #plt.savefig('Transit_{0}Rstar_b={1}_{2}K.png'.format(rad_planet,self.b,self.star_temp))

    # Determine how long it took the program to run
    runtime = time.time() - start_time
    print("My program took {0:.2f} minutes to run".format(runtime/60.0))
```

```
    # Function to calculate rotational velocity at point in star
    def RV(self,x,y,vel_eq=10.0):
        # Inputs:
        #   x: array of x-positions (sourced from 'Star' function)
        #   y: array of y-positions (sourced from 'Star' function)
        #   vel_eq: equatorial velocity of rotating star
        # Returns:
        #   vel_rad: array of radial velocities for all pixels in star

        # Set equatorial velocity as global variable
        self.vel_eq = vel_eq

        # Calculate polar and azimuthal angles using cartesian pixel coords.
        theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
        phi = np.arctan(y/x)

        # Use angles and vel_eq to find radial velocity of each pixel
        vel_rad = vel_eq*np.sin(np.arctan(theta))*np.cos(phi)

        return(vel_rad)

    # Function to generate rotational velocity profile of rotating star
    def RVProfile(self,bins=100,plot='profile'):
        # Inputs:
        #   bins: number of bins for sorting pixel velocities
        #   plot: string indicating what user want to plot
        # Returns:
        #   rotational velocity profile

        # Generate intensity-weighted coordinate grid
        x_grid,y_grid,intensities_star = self.Star(plot=False)

        # Calculate radial velocity at each position
        velocities = self.RV(x_grid,y_grid)

        # Consider velocities on left half of star to be negative
        velocities[np.where(x_grid<0.0)]*=-1.0

        # Decide what plot to make
        if plot == 'star': # red-blue color coated map of star
            plt.pcolor(x_grid,y_grid,velocities,cmap='bwr',shading='nearest',vmin=-self.vel_eq,v
            plt.xlim(-1.2,1.2)
            plt.ylim(-1.1,1.1)
            cbar = plt.colorbar()
            cbar.set_label(r'Radial Velocity ($\frac{km}{s}$)',fontsize=14)
            plt.xlabel(r'x ($R_{star}$)',fontsize=14)
            plt.ylabel(r'y ($R_{star}$)',fontsize=14)
            plt.title('Velocity of T={0}K Star \n (at '.format(self.star_temp)+r'$5000\AA$)',for

        elif plot == 'profile': # rotational velocity profile (bin pix by vel)
            # Generate array of velocities
            rad_vels = np.linspace(-self.vel_eq,self.vel_eq,bins+1)

            # Establish list of num of pix in each bin
            num_pixels = []
            # Establish list of average velocity value in each bin
            avg_RVs = []

            # Loop over all actual pixel velocities to bin them
            for i in range(bins):
                # Find indices of pixels that fall within bin
                indices = np.where((velocities>rad_vels[i]) & (velocities<rad_vels[i+1]))

                # Count number of pixels in bin and add count to array
                count = np.sum(intensities_star[indices])
                num_pixels.append(count)
```

```
            # Calculate average velocity in bin
            avg_RV = (rad_vels[i]+rad_vels[i+1])/2.0
            avg_RVs.append(avg_RV)

        # Cast num_pixels and avg_RVs as numpy arrays
        num_pixels = np.asarray(num_pixels)*-1
        avg_RVs = np.asarray(avg_RVs)

        # Normalize values of num_pixels array
        num_pixels -= np.min(num_pixels)
        num_pixels /= np.max(num_pixels)

        # Plot normalized line profile ()
        plt.plot(avg_RVs,num_pixels)
        plt.xlabel(r'Radial Velocity ($\frac{km}{s}$)')
        plt.ylabel('Normalized Line Profile')
        plt.title('Line Profile of T={0}K Star \n (No Transit)'.format(self.star_temp))

# Function to generate rotational velocity profile of rotating star
def RVProfileTransit(self,rad_planet=0.05,x_center=.5,b=0.5,bins=100,plot='profile'):
    # Inputs:
    #   rad_planet: fractional size of planet in terms of stellar radius
    #   b: impact parameter of transit (can range from -1 to 1)
    #   bins: number of bins for sorting pixel velocities
    #         MUST BE <= gridsize
    #   plot: string indicating what user want to plot
    # Returns:
    #   rotational velocity profile

    # Generate intensity-weighted coordinate grid
    x_grid,y_grid,intensities_star = self.Star(plot=False)

    # Identify location of planet center
    planet_center = [x_center,b]

    # Calculate x,y, and total distances of all points from planet center
    xdist = x_grid - planet_center[0]
    ydist = y_grid - planet_center[1]
    dist = np.sqrt(xdist**2+ydist**2)

    # Find pixels within planet radius
    planet_ids = np.where(dist < rad_planet)

    # Calculate radial velocity at each position
    velocities = self.RV(x_grid,y_grid)
    velocities_transit = np.copy(velocities)

    # Consider velocities on left half of star to be negative
    velocities[np.where(x_grid<0.0)]*=-1.0
    velocities_transit[np.where(x_grid<0.0)]*=-1.0

    # Set transit velocity to NaN wherever planet blocks the star
    velocities_transit[planet_ids] = np.NaN

    # Decide what plot to make
    if plot == 'star': # red-blue color coated map of star with no transit
        #plt.pcolor(x_grid,y_grid,velocities,cmap='bwr',shading='nearest',vmin=-self.vel_eq,
        plt.pcolor(x_grid,y_grid,velocities_transit,cmap='bwr',shading='nearest',vmin=-self.
        plt.xlim(-1.2,1.2)
        plt.ylim(-1.1,1.1)
        cbar = plt.colorbar()
        cbar.set_label(r'Radial Velocity ($\frac{km}{s}$)',fontsize=14)
        plt.xlabel(r'x ($R_{star}$)',fontsize=14)
        plt.ylabel(r'y ($R_{star}$)',fontsize=14)
        plt.title('Velocity of T={0}K Star \n (at '.format(self.star_temp)+r'$5000\AA$)',for

    elif plot == 'profile': # rotational velocity profile (bin pix by vel)
```

```
            # Generate array of velocities
            rad_vels = np.linspace(-self.vel_eq,self.vel_eq,bins+1)

            # Establish list of num of pix in each bin
            num_pixels = []
            num_pixels_transit = []

            # Establish list of average velocity value in each bin
            avg_RVs = []

            # Loop over all actual pixel velocities to bin them
            for i in range(bins):
                # Find indices of pixels that fall within bin
                indices = np.where((velocities>rad_vels[i]) & (velocities<rad_vels[i+1]))
                indices_transit = np.where((velocities_transit>rad_vels[i]) & (velocities_transi

                # Count number of pixels in bin and add count to array
                count = np.sum(intensities_star[indices])
                count_transit = np.sum(intensities_star[indices_transit])
                num_pixels.append(count)
                num_pixels_transit.append(count_transit)

                # Calculate average velocity in bin
                avg_RV = (rad_vels[i]+rad_vels[i+1])/2.0
                avg_RVs.append(avg_RV)

            # Cast num_pixels and avg_RVs as numpy arrays
            num_pixels = np.asarray(num_pixels)*-1
            num_pixels_transit = np.asarray(num_pixels_transit)*-1
            avg_RVs = np.asarray(avg_RVs)

            # Normalize values of num_pixels array
            num_pixels -= np.min(num_pixels)
            num_pixels /= np.max(num_pixels)

            num_pixels_transit -= np.min(num_pixels_transit)
            num_pixels_transit /= np.max(num_pixels_transit)

            # Plot normalized line profiles
            label=r'$R_{star}$'
            plt.plot(avg_RVs,num_pixels,label='No Transit')
            plt.plot(avg_RVs,num_pixels_transit,label='{0}'.format(rad_planet)+label+' Transitir
            plt.xlabel(r'Radial Velocity ($\frac{km}{s}$)')
            plt.ylabel('Normalized Line Profile')
            plt.title('Line Profile of T={0}K Star \n '.format(self.star_temp)+r'($x_{cen}=$'+st
            plt.legend()
"""
# Lines to test class
star_b = LimbDarkening(5500,100)
#star_b.Star()
star_b.Transit(0.05,0.9,False)
"""
```