PlanetsHW3

November 4, 2020

1 ASTR 5490 Homework 3

[36]:
```
# Import relevant modules/packages
import numpy as np
import matplotlib.pyplot as plt
from astropy import units as u
from astropy import constants as const
from astropy.timeseries import LombScargle
from PeriodicityTools import Periodicity
from GrazingTransit import GrazingTransit
from LimbDarkening import LimbDarkening
from Spectra import SpectralFeatures, Accuracy
from MathTools import Gaussian, NonRelDoppler
from TransitPlotter import LightCurveCompare
from Batman import BatmanModel

# Reload PeriodicityTools to acknowledge changes made in the script
%load_ext autoreload
%autoreload 2
```

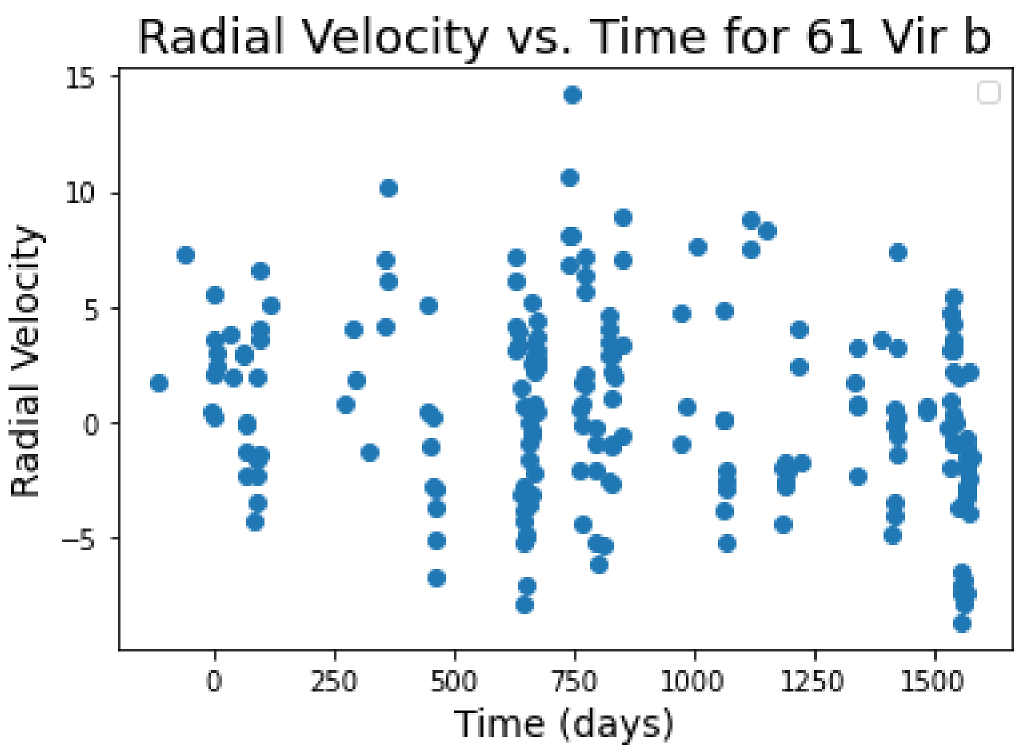
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

2 1) Exploring Radial Velocity Curve of 61 Vir b from Vogt et al. (2009)

2.0.1 https://iopscience.iop.org/article/10.1088/0004-637X/708/2/1366#apj330338s2

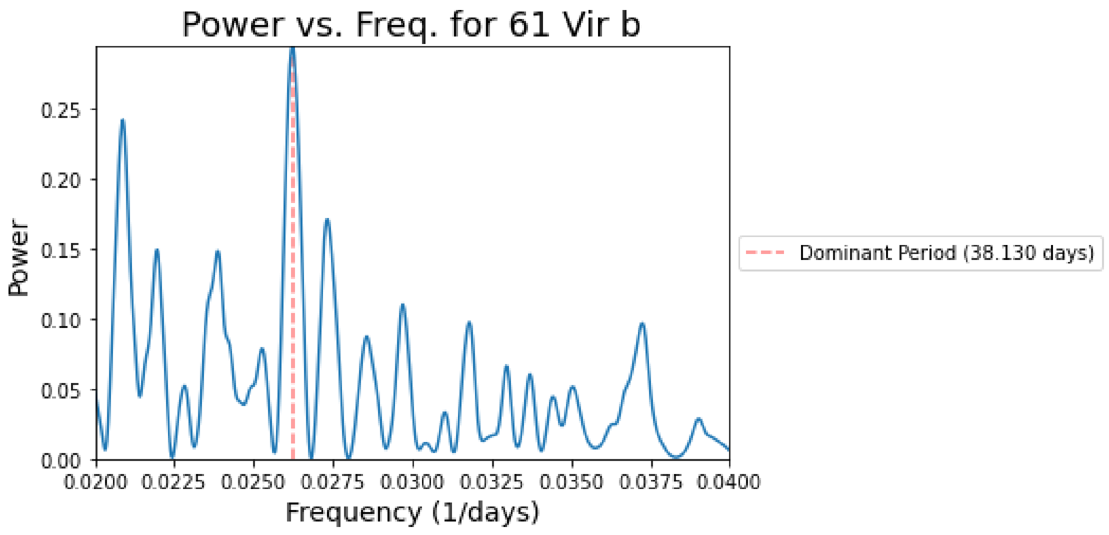
[9]:
```
# Create instance of class
Virb = Periodicity('Vogt2009_61Virb_vels.dat','61 Vir b',None)

# Generate light curve for data
times,RVs,errors = Virb.LightCurve(curve='Radial Velocity')
```

```
No handles with labels found to put in legend.
```



2.1 1a) Make a periodogram of the whole dataset

[45]:
```
# Make a Lomb Scargle Periodogram of the data
original_maxPower = Virb.LS(25,50,1000,1,flux=[],plot=True)
```



2.1.1 Looks quite similar to upper panel of Vogt et al. Figure 3, but my x-axis is the inverse of their's so mine's not exactly the same. I find the same as they do (38.13 d)

2.2 1b) Compute the false alarm probability considering the 38 day planet

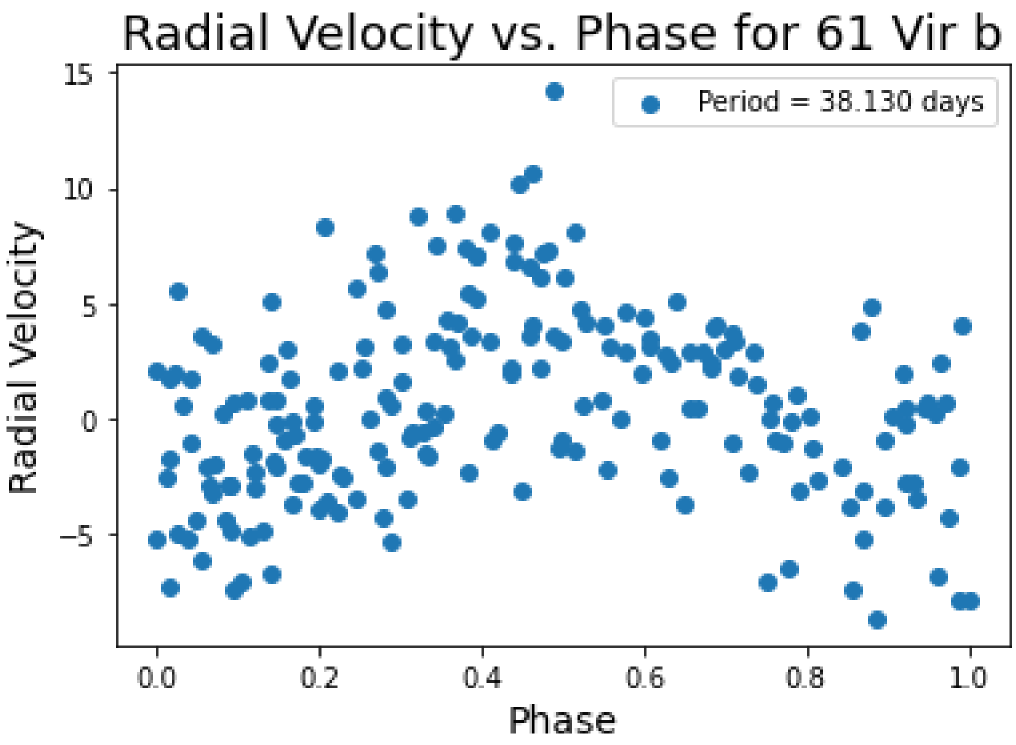
2.2.1 The F.A.P. is defined as "the fraction of trials for which the periodogram power exceeds the observed value" (Page 3 of Cumming 2004 - https://academic.oup.com/mnras/article/354/4/1165/1052087)

[15]:
```
fap = Virb.FAP(10000)
```

```
FAP = 0.000e+00
```

2.3 1c) Fold the RV data at the period determined from the periodogram

[47]:
```
Virb_folded = Periodicity('Vogt2009_61Virb_vels.dat','61 Vir b',None,period=38.
 ↪13)
new_times,new_RVs,new_errors = Virb_folded.
 ↪LightCurve(xaxis='Phase',curve='Radial Velocity')
```



2.4 1d) Pick a _________ interval and find the FAP

2.4.1 i) contiguous 100d

```
[13]: # Find FAP over first 100 days of data
      Virb_contig = Periodicity('Vogt2009_61Virb_vels.dat','61 Vir b',100,period=38.
       ↪13)
      contiguous_fap = Virb_contig.FAP(10000)
```

```
FAP = 0.000e+00
```

2.4.2 ii) sparsed 100d

```
[14]: # Find FAP using first 50 and last 50 days of data
      Virb_sparsed = Periodicity('Vogt2009_61Virb_vels.dat','61 Vir␣
       ↪b',50,contiguous=False,period=38.13)
      sparsed_FAP = Virb_sparsed.FAP(10000)
```

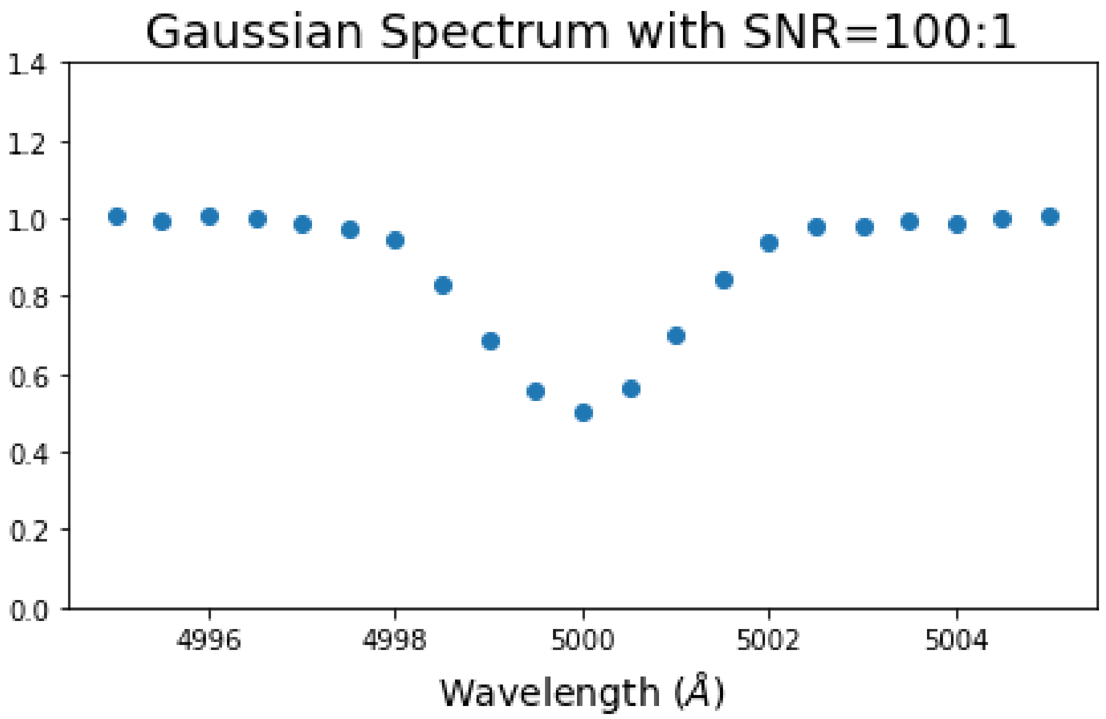
```
FAP = 0.000e+00
```

2.4.3 My calculated FAP for each method of splicing the data was 0 and I ran them all for 10000 Monte Carlo iterations so the FAP must be $\leq 10^{-4}$. This indicates a high probability ($\geq 99.99\%$) of this being a true planet detection

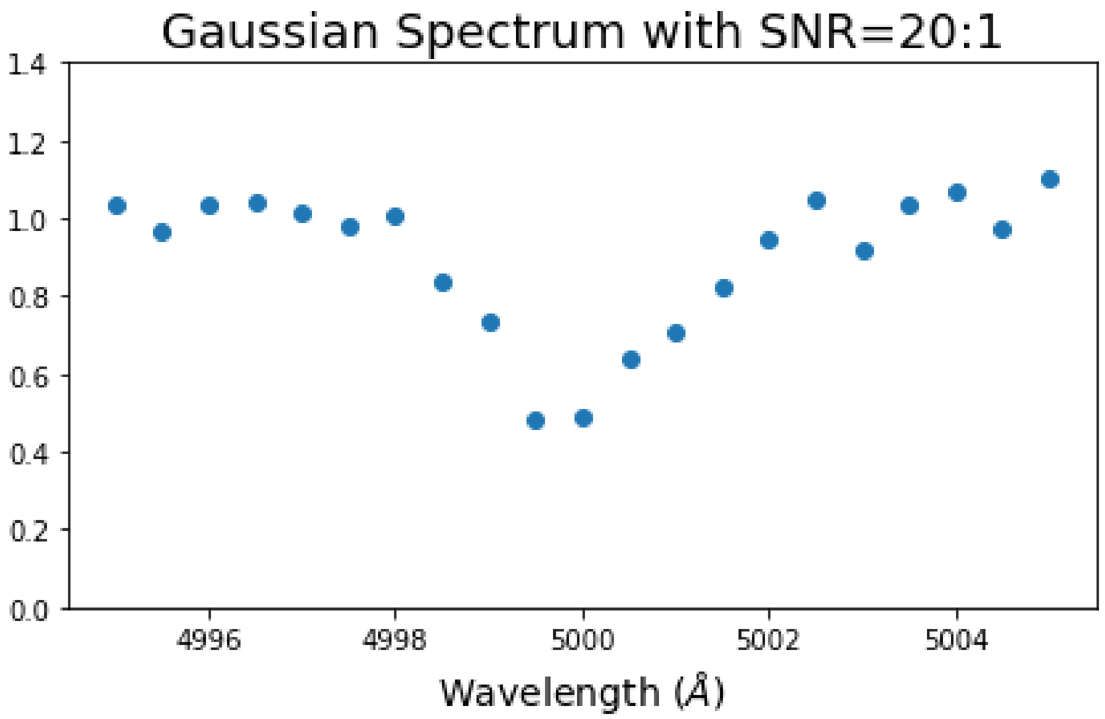
3 2) Investigating Gaussian absorption lines

3.1 2a) Generate Gaussian absorption line for normalized spectrum with: $A = 0.5$, $\sigma = 1$ angs. centered at 5000 angs. with pixels every 0.5 angs.

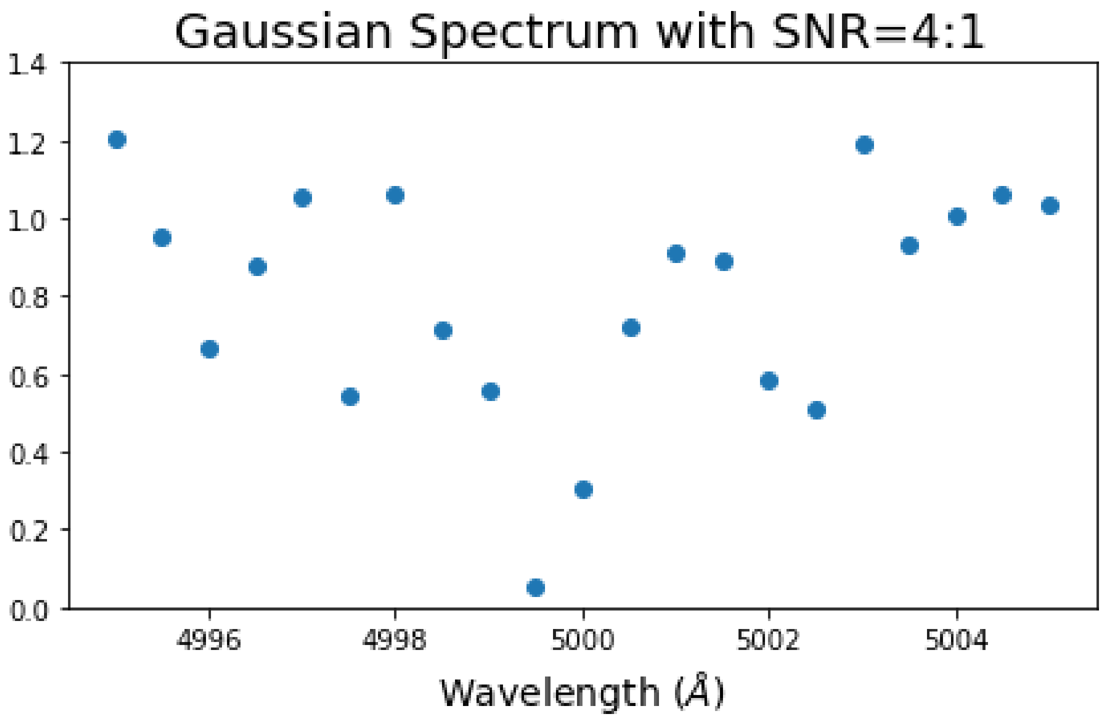
```
[3]: # Gaussian spectrium with SNR=100:1
     SNR_100 = SpectralFeatures(5000.0,1.0,0.5,0.0,1.0,0.5,100)
     x,y_100 = SNR_100.GaussianNoise()
```



[4]:
```
# Gaussian spectrium with SNR=20:1
SNR_20 = SpectralFeatures(5000.0,1.0,0.5,0.0,1.0,0.5,20)
x,y_20 = SNR_20.GaussianNoise()
```

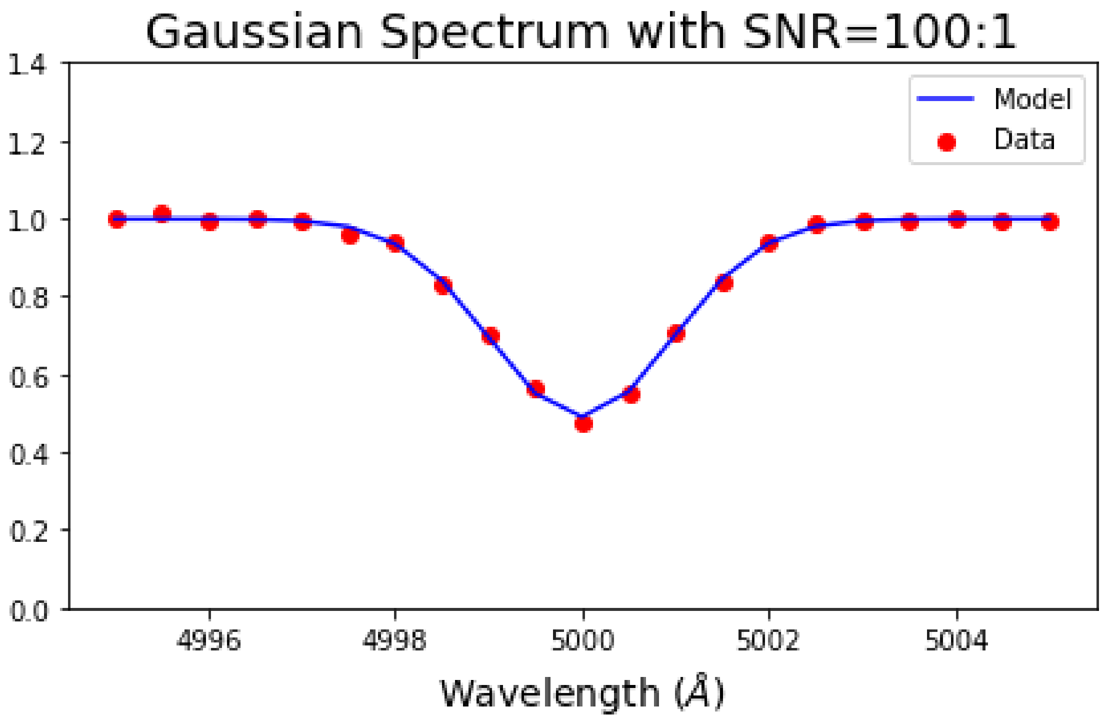


[5]:
```
# Gaussian spectrium with SNR=4:1
SNR_4 = SpectralFeatures(5000.0,1.0,0.5,0.0,1.0,0.5,4)
x,y_4 = SNR_4.GaussianNoise()
```

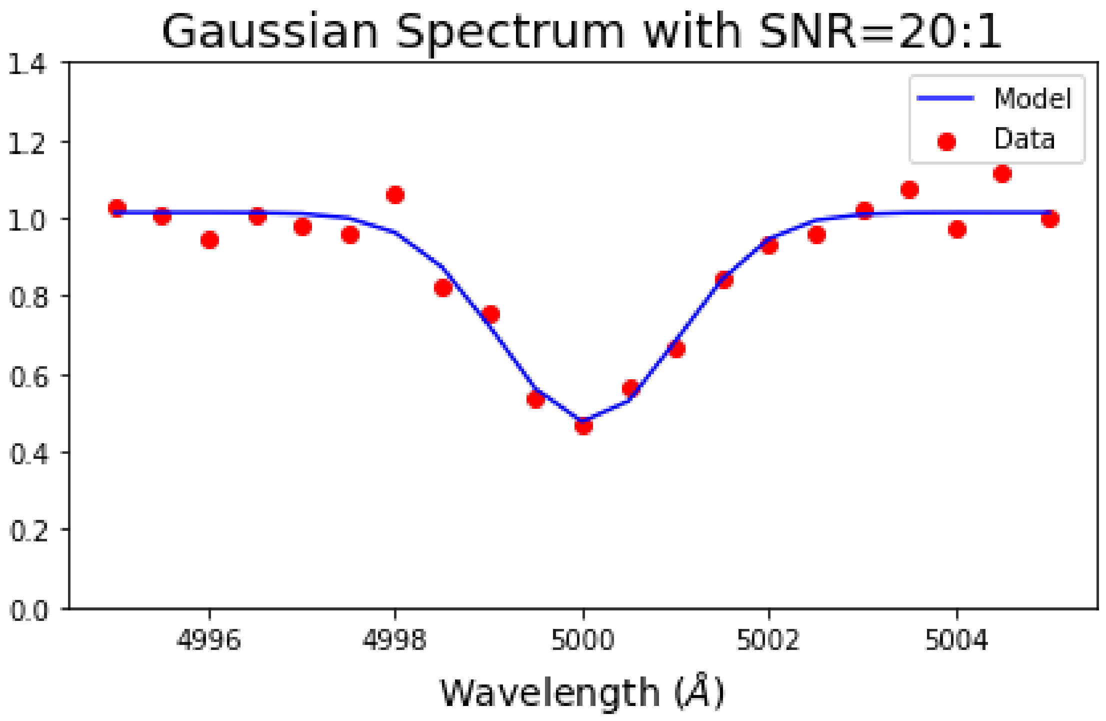


3.2 2a) Use curve_fit to fit Gaussian to noisy data from 2a. Plot accuracy of the center of the profile (mean) vs. SNR

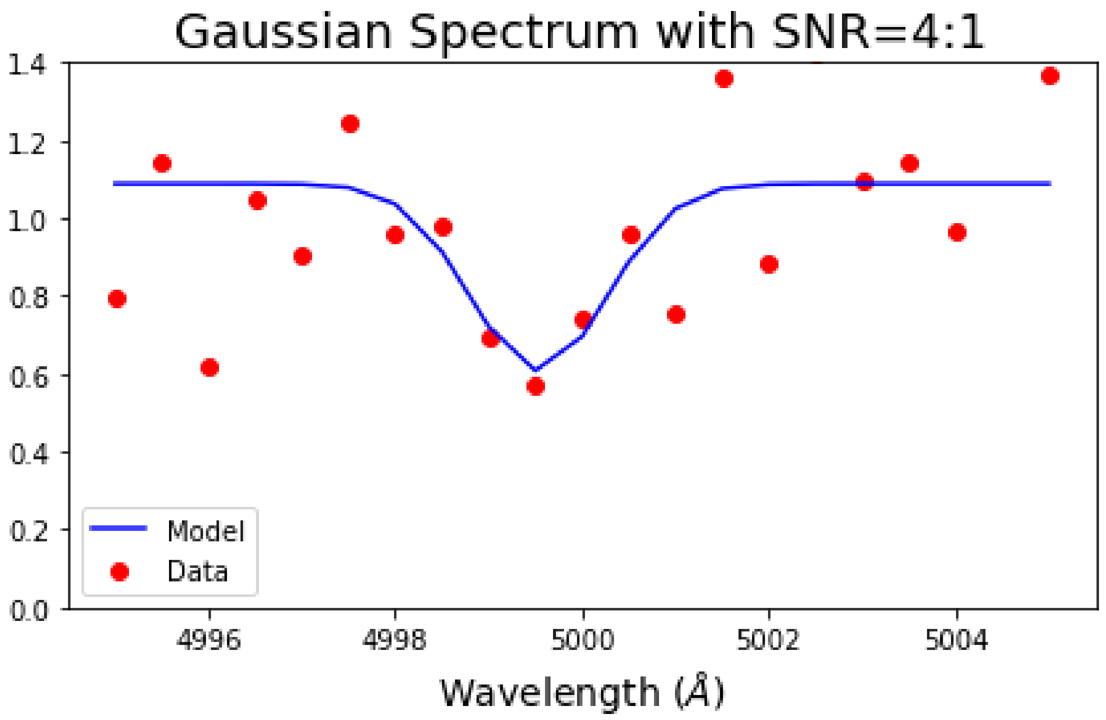
[6]:
```python
# Plot data vs. model for SNR = 100:1
accuracy_100 = SNR_100.GaussianModel(plot=True)
```



[8]:
```
# Plot data vs. model for SNR = 20:1
accuracy_20 = SNR_20.GaussianModel(plot=True)
```

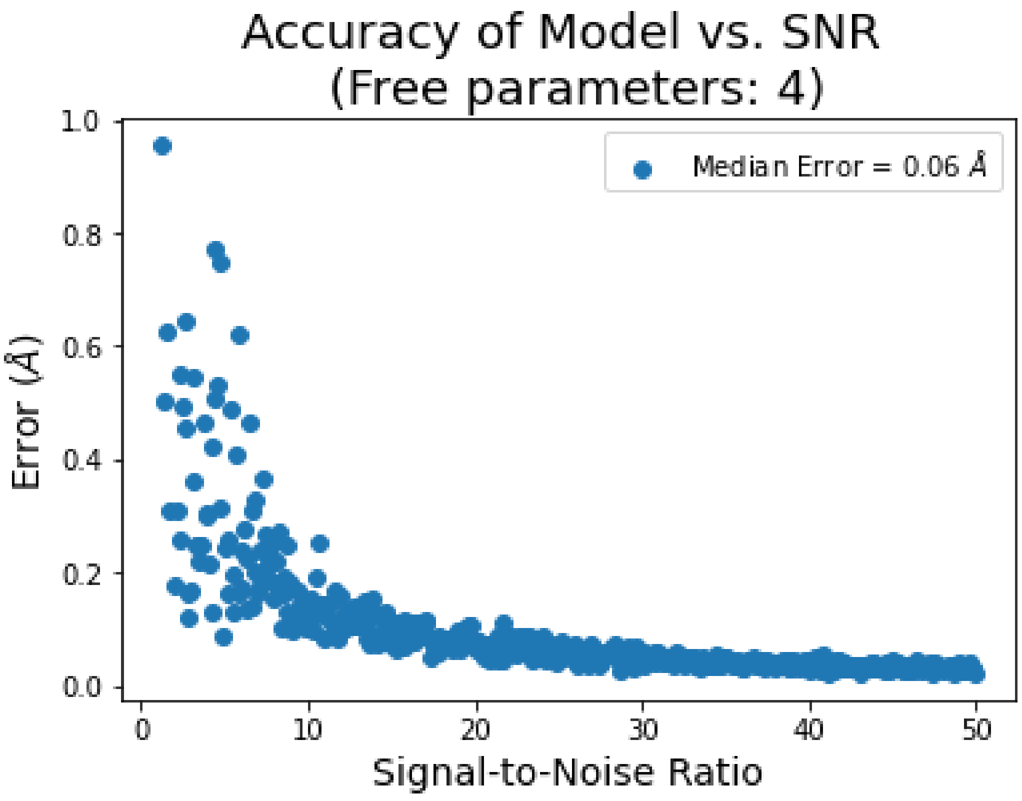


[10]:
```
# Plot data vs. model for SNR = 4:1
accuracy_4 = SNR_4.GaussianModel(plot=True)
```



[48]:
```
# Make list of SNRs and empty list of accuracies
SNRs = np.arange(1,50,.1)
Accuracy(4,SNRs)
```

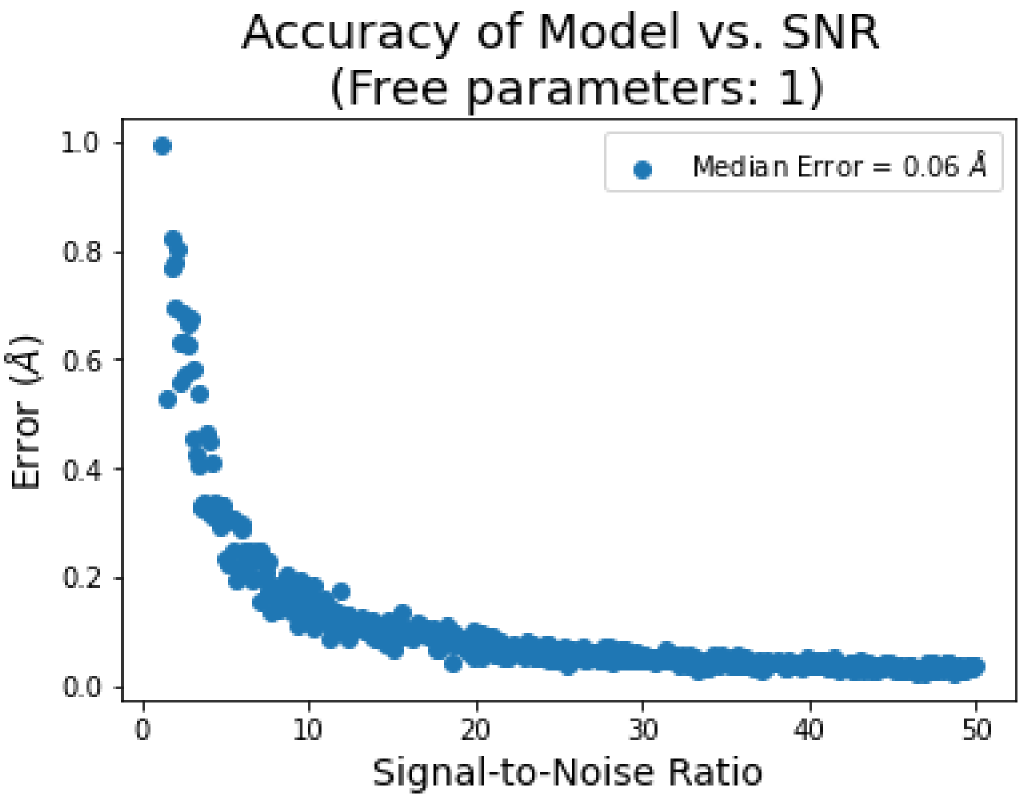
```
10 out of 490 trials had accuracy > 1
```



3.3 2c) Repeat 2b for the mean being the only free parameter

```
[49]: Accuracy(1,SNRs)
```

```
8 out of 490 trials had accuracy > 1
```

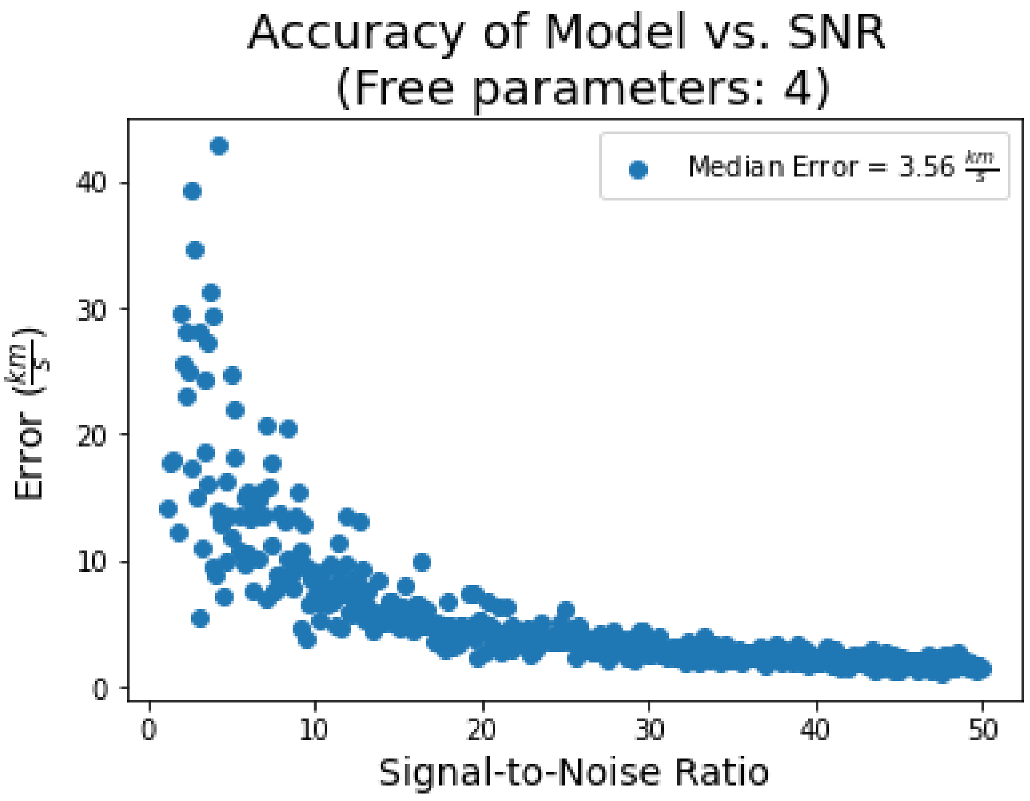


3.4 2d) Convert accuracy in angs to km/s for 2b and 2c (using non-relativistic Doppler Eqn.)

3.4.1 $\lambda' = \lambda_0 \left(1 + \frac{v}{c_0}\right)$ −> $v = c_0 \left(\frac{\lambda'}{\lambda_0} - 1\right)$

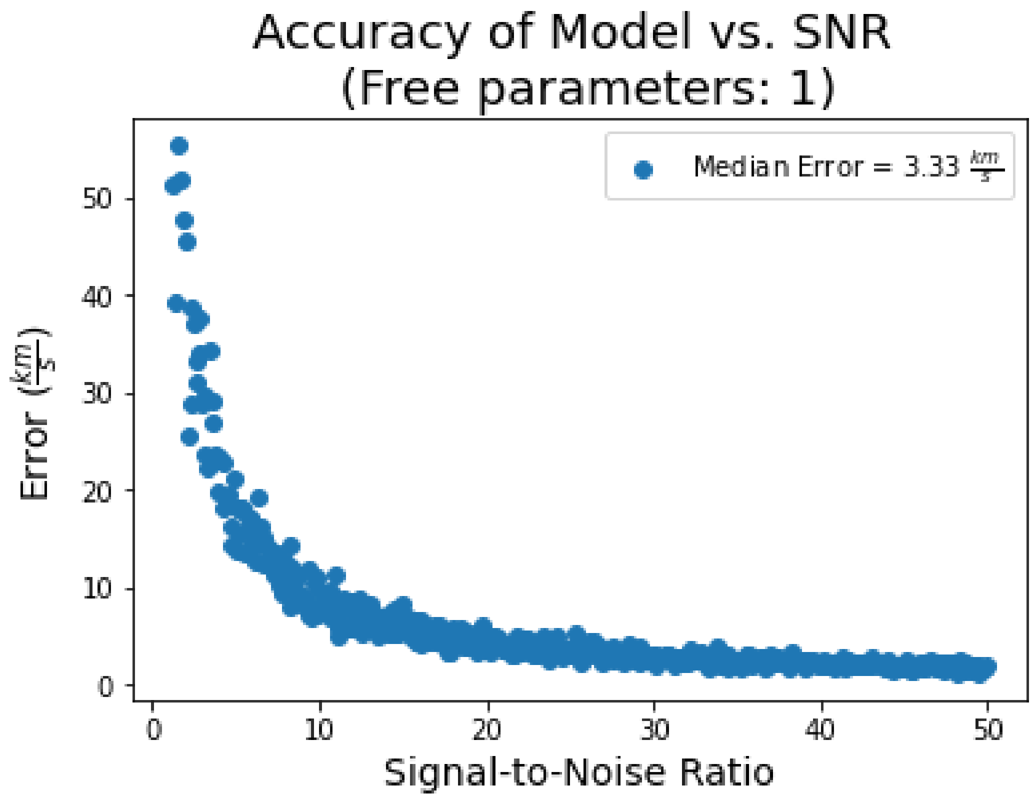
[51]:
```
Accuracy(4,SNRs,'km/s')
```

```
11 out of 490 trials had accuracy > 1
```



```
[52]: Accuracy(1,SNRs,'km/s')
```

```
7 out of 490 trials had accuracy > 1
```


Accuracy of Model vs. SNR
(Free parameters: 1)
Median Error = 3.33 $\frac{km}{s}$
Error ($\frac{km}{s}$)
Signal-to-Noise Ratio
0
10
20
30
40
50

3.4.2 Models agree much better for data sets with higher SNR. Once SNR > 30, model doesn't fit significantly better if SNR keeps getting better (increasing). Models also agree better when you fit less free parameters (less deviation this way)

3.5 2e) What mass planet could you detect if you have $N = 10^1, 10^2, 10^3$ absorption lines? Remember that accuracy improves as $\sqrt{N}$ (so $\frac{accuracy}{\sqrt{N}}$)

3.5.1 For $N = 10$: $\sqrt{10} \approx 3.16$ −> $acc. \approx 1.10\frac{km}{s} = 1110\frac{m}{s}$ −> detect $M_p > 10^{28.5}kg \approx 16.6M_{Jup}$

3.5.2 For $N = 10^2$: $\sqrt{10^2} = 10$ −> $acc. \approx 0.34\frac{km}{s} = 340\frac{m}{s}$ −> detect $M_p > 10^{28}kg \approx 5.26M_{Jup}$

3.5.3 For $N = 10^3$: $\sqrt{10^3} \approx 31.6$ −> $acc. \approx 0.10\frac{km}{s} = 100\frac{m}{s}$ −> detect $M_p > 10^{27.5}kg \approx 1.66M_{Jup}$

4 3) Plot of orbital inclination vs. semi-major axis with lines corresponding to grazing transit for planets from Earth to Jupiter-size. Also shading approx. habitable zone.

4.1 Grazing transit

4.1.1 i) Impact parameter (b): $b = acos(i)$

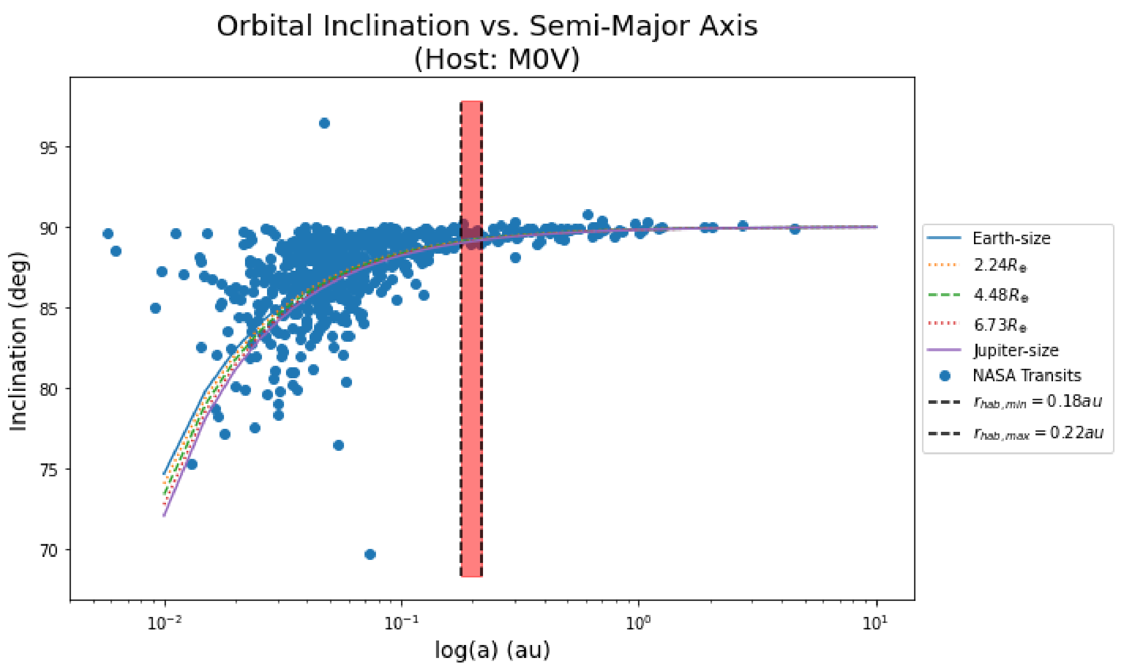
4.1.2 ii) Grazing condition: $b = acos(i) \leq R_p + R_*$

4.2 3a) M0V (dwarf) Host Star: $T_{eff} = 3870K; R_* = .559R_\odot$

4.3 3b) K0III (giant) Host Star: $T_{eff} = 4810K, R_* = 10R_\odot$ (couldn't find exact radius)

4.3.1 Property source: http://www.pas.rochester.edu/~emamajek/EEM_dwarf_UBVIJHK_colo

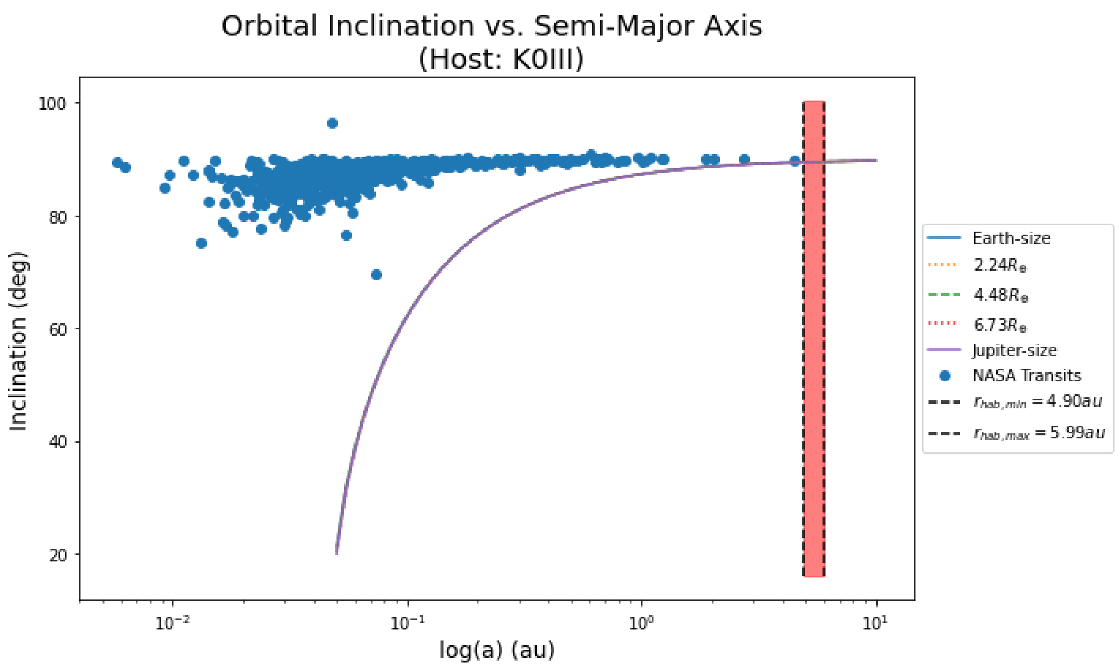
[4]:
```
# Plot grazing transits and habitable zone for MOV star
MOV = GrazingTransit('M')
MOV.InclinationSemiMajor()
```



4.3.2 It is well within observational limits to observe an Earth to Jupiter size planet in the habitable zone of an M0V star with the transit method. Many transit detections published in the NASA Exoplanet Archive fall in this detection range.

[5]:
```
# Plot grazing transits and habitable zone for MOV star
KOIII = GrazingTransit('K')
KOIII.InclinationSemiMajor()
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/GrazingTransit.py:56: RuntimeWarning:
invalid value encountered in arccos
  def InclinationSemiMajor(self):
```


Orbital Inclination vs. Semi-Major Axis
(Host: K0III)
Inclination (deg)
log(a) (au)
100
80
60
40
20
10^{-2}
10^{-1}
10^{0}
10^{1}
Earth-size
$2.24R_{\oplus}$
$4.48R_{\oplus}$
$6.73R_{\oplus}$
Jupiter-size
NASA Transits
$r_{hab,min} = 4.90au$
$r_{hab,max} = 5.99au$

4.3.3 It seems just beyond observational limits to observe an Earth to Jupiter size planet in the habitable zone of a K0III star with the transit method. There are a few transit detections published in the NASA Exoplanet Archive that lie near this detection range, but none lie within it. In both plots we see a bias in transit detections of high-inclination exoplanets likely because the exoplanet occults more of the star's light when the system is ege-on w.r.t Earth. We don't see a transit if the system is at a low inclination (i.e. face on) so it's not surprising that most transit detections have high inclinations.

5 4) Simulate a star as a solid face-on disk by breaking the stellar surface into a grid of 1000x1000 square pixels

5.1 4a) Assign each pixel a surface brightness with 1.0 at the center. Use quadratic limb darkening tables-Van Hamme (1993)-to assign a relative brightness to other pixels. Assume G2V star with $T_{eff} = 5500K$ and $\lambda_{obs} = 5000$ angs

5.1.1 Van Hamme (1993): https://ui.adsabs.harvard.edu/abs/1993AJ....106.2096V/abstract

5.1.2 Source for log(g) of dwarf (V) stars: http://www.astro.sunysb.edu/metchev/PHY521/lecture

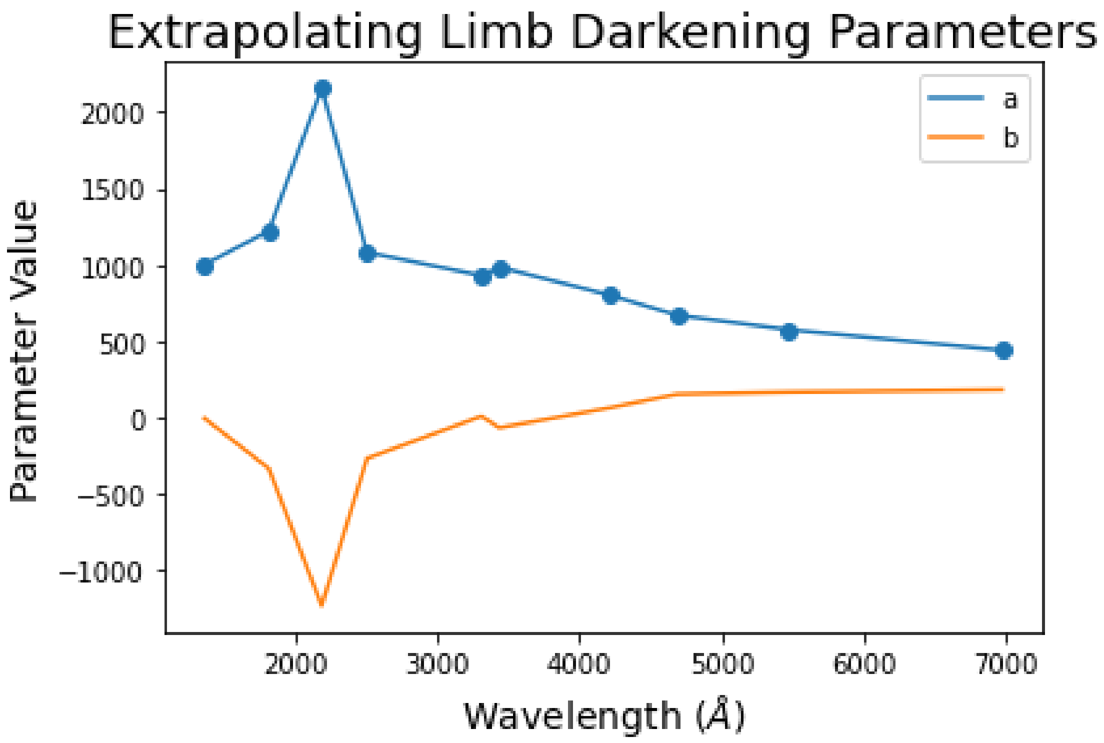
5.2 Q.L.D.: $I(\mu) = I(0)\left[1 - a(1-\mu) - b(1-\mu)^2\right]$

5.3 Substituting for μ: $I(\mu) = I(0)\left[1 - a\left(1 - \sqrt{\frac{R_*^2 - r^2}{R_*^2}}\right) - b\left(1 - \sqrt{\frac{R_*^2 - r^2}{R_*^2}}\right)^2\right]$ where a is the radius of the disc and r is the distance from the center of the disc (guidance from http://orca.phys.uvic.ca/~tatum/stellatm/atm6.pdf)

5.4 I'm sourcing a and b from Table 5 on the 10th page of Wade & Rucinski (1985): http://articles.adsabs.harvard.edu/pdf/1985A%26AS...60..471W

```
[57]: # We're using 5000 angstroms so this is my attempt to extrapolate proper a and␣
      ↪b values
      lambdas = [1362,1815,2187,2506,3312,3437,4212,4687,5475,6975]
      a_list = [1002,1222,2157,1082,930,985,803,671,576,442]
      b_list = [-3,-332,-1230,-266,11,-65,65,154,168,183]
      plt.plot(lambdas,a_list,label='a')
      plt.plot(lambdas,b_list,label='b')
      plt.scatter(lambdas,a_list)
      plt.xlabel(r'Wavelength ($\AA$)',fontsize=14)
      plt.ylabel('Parameter Value',fontsize=14)
      plt.title('Extrapolating Limb Darkening Parameters',fontsize=18)
      plt.legend()
```

```
[57]: <matplotlib.legend.Legend at 0x7fc85429eeb8>
```



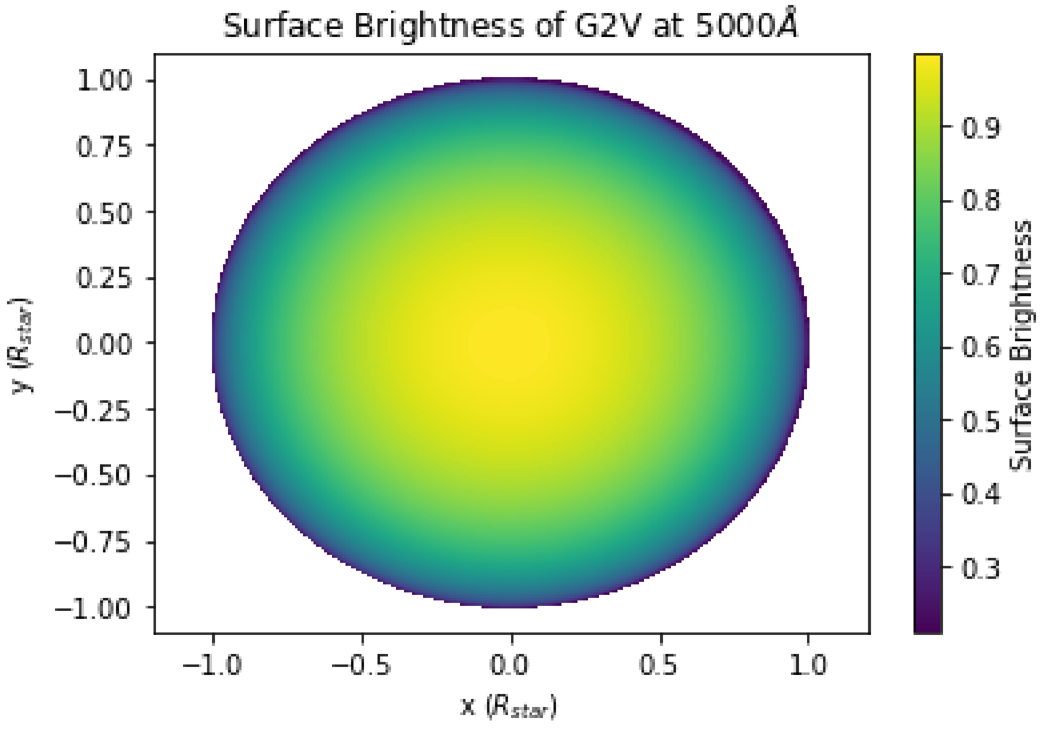
5.4.1 In range of 5000 angs., the relationship for both parameters seems pretty linear so I'll use a linear extrapolation to find a and b at 5000 angs

[58]:
```
# List indexes of lambdas that lie above and below 5000 angs.
min_index = 7
max_index = 8

# Define sublists of points below and above point of interest
a_sub = [a_list[min_index],a_list[max_index]]
b_sub = [b_list[min_index],b_list[max_index]]
lambdas_sub = [lambdas[min_index],lambdas[max_index]]

# Interpolate to find proper values at 5000 angs
a_new = np.interp(5000,lambdas_sub,a_sub)
b_new = np.interp(5000,lambdas_sub,b_sub)
print(a_new,b_new)
```

```
633.2652284263959 159.56091370558374
```

[59]:
```
# Use LimbDarkening function to plot surface of G2V star
star = LimbDarkening(633.27/1000,159.56/1000)
x,y,intensities = star.Star(1000)
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:36: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
```

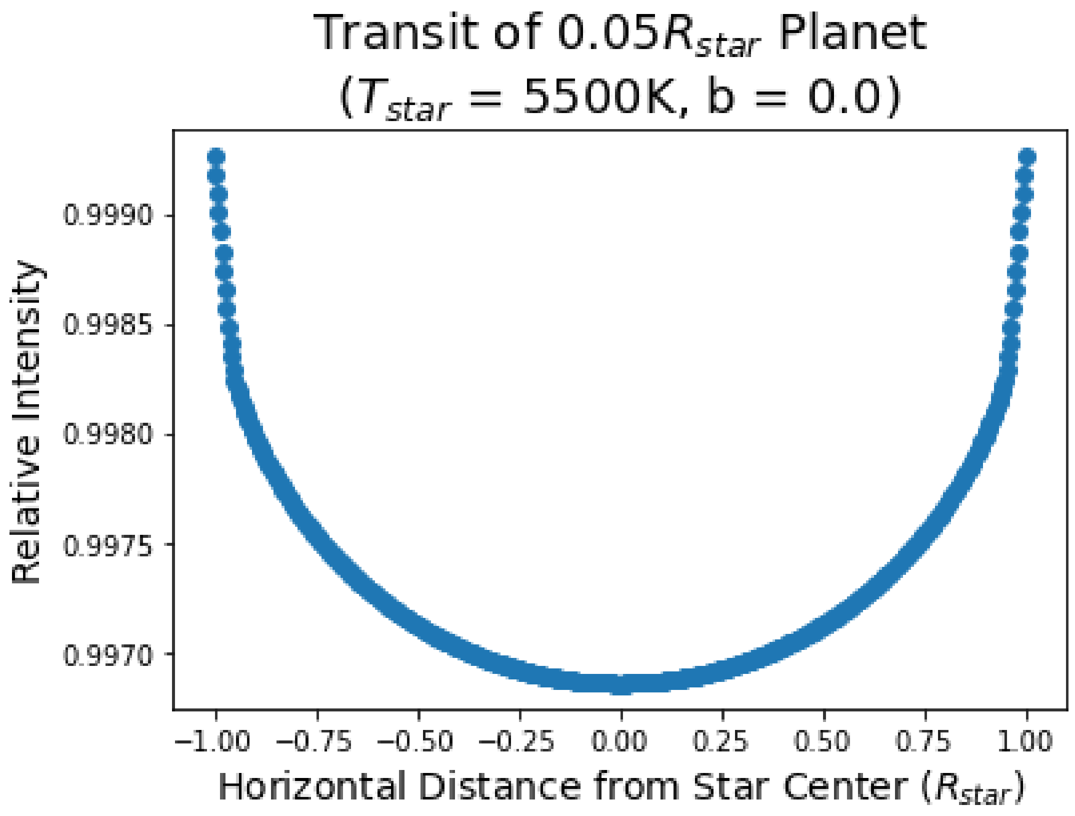


5.5 4b) Plot light curve of transit of .05R_{star} at impact parameter 0

```
[10]: star_b = LimbDarkening(5500,500)
```

```
[27]: star_b.Transit(0.05,0.0,False)
```

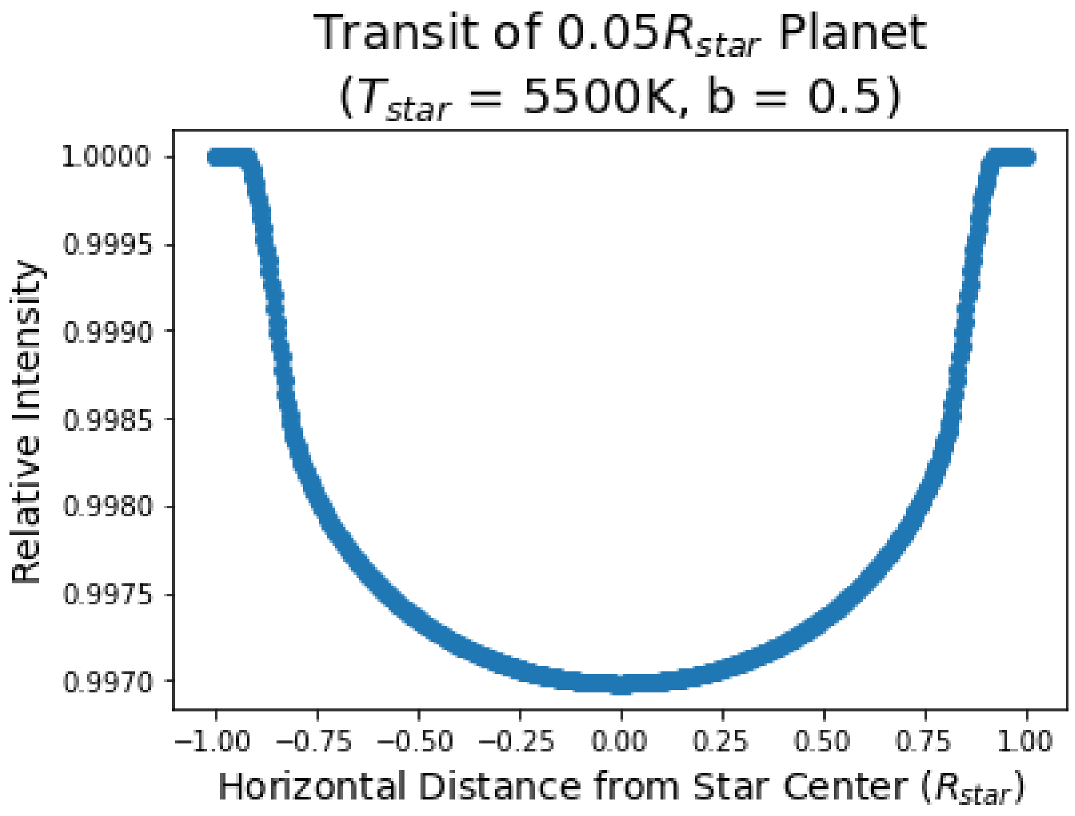
```
My program took 240.88 seconds to run
```



5.6 4c) Plot light curve of transit of $.05R_{star}$ at impact parameter 0.5

[11]:
```
star_b.Transit(0.05,0.5,False)
```

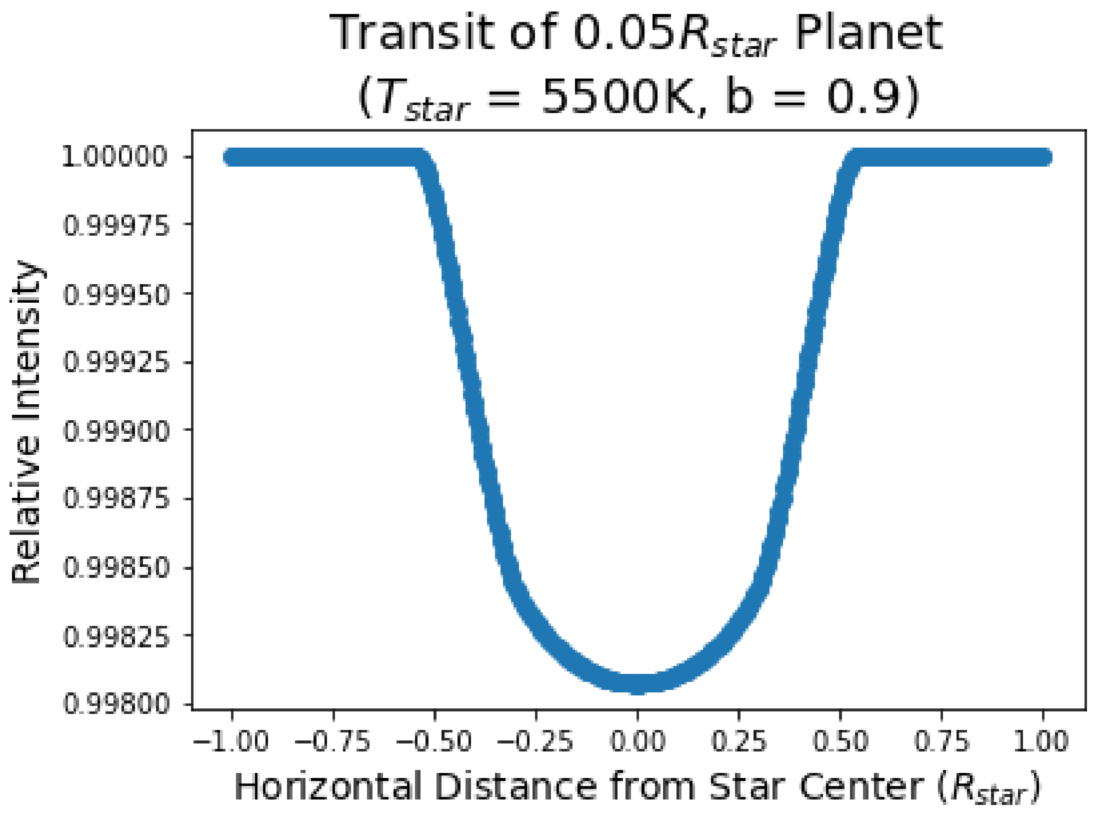
```
My program took 3.75 minutes to run
```



5.7 4d) Plot light curve of transit of $.05R_{star}$ at impact parameter 0.9 (grazing transit)

[12]:
```
star_b.Transit(0.05,0.9,False)
```

```
My program took 3.84 minutes to run
```



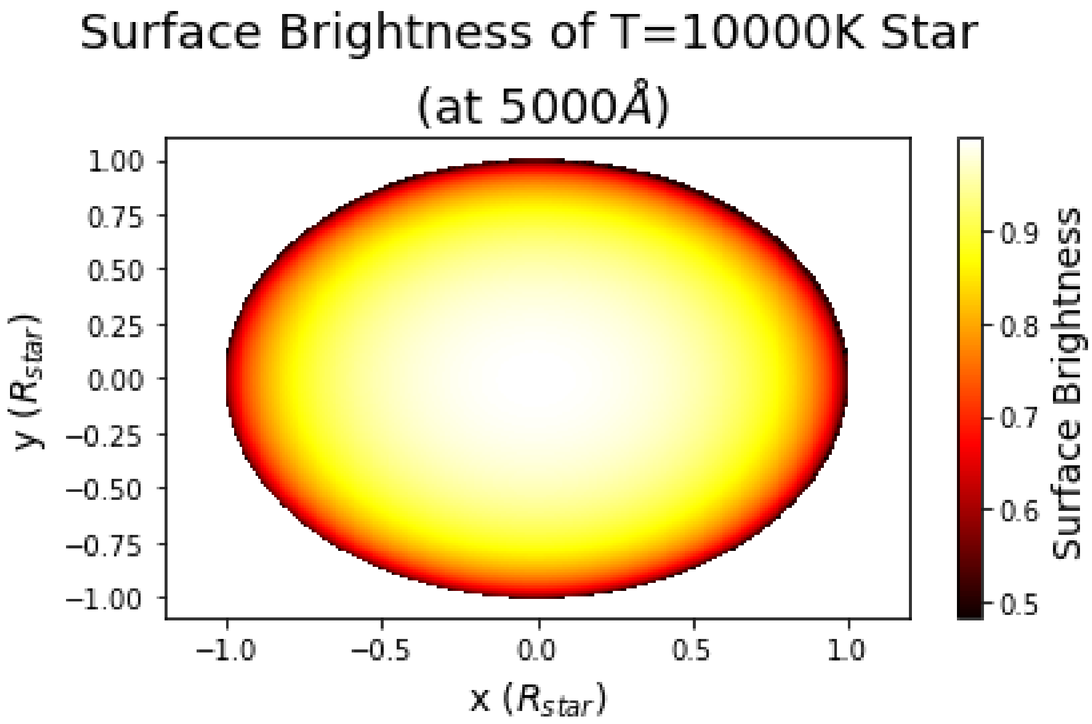
5.8 4e) Overplot on the plots for B/C/D the light curve for a T=10,000 K host star and a 3600 K host star and summarize the differences

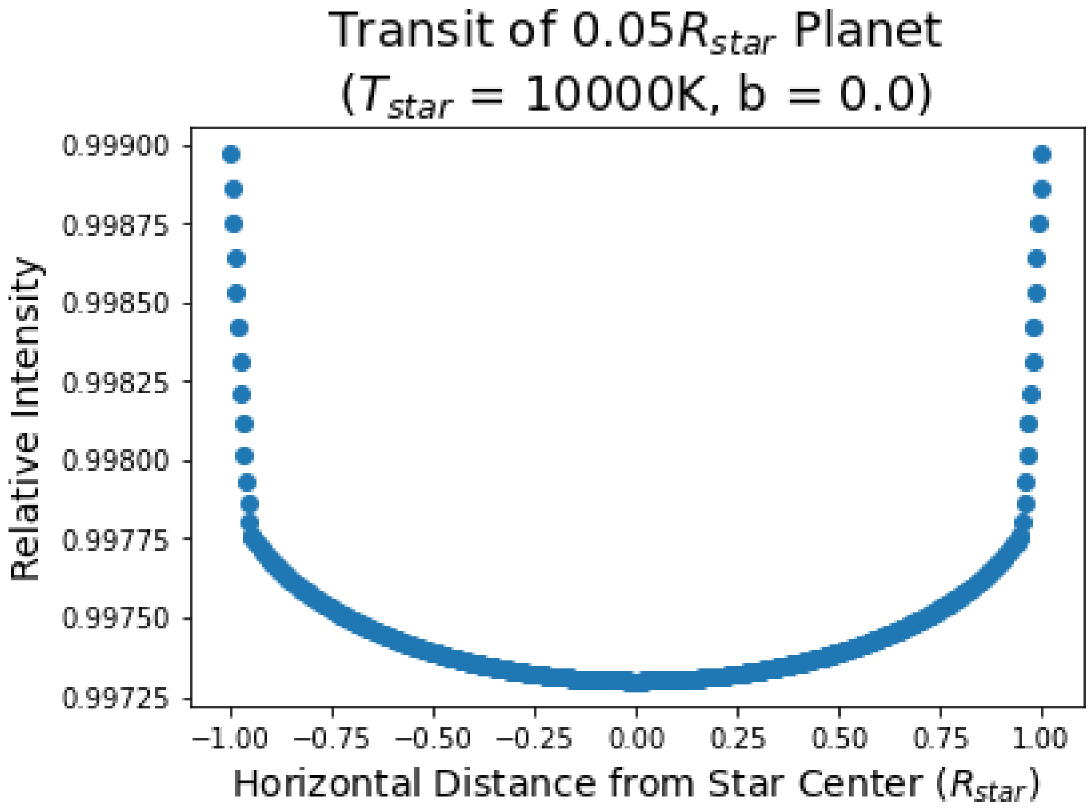
5.8.1 For 10000K, got parameters from: First two rows at relevant temp. in Table 3 of Claret (2013) http://cdsarc.u-strasbg.fr/ftp/cats/J/A+A/552/A16/ori/Table3

[13]:
```
star_10000K = LimbDarkening(10000,500)
```

[30]:
```
# impact parameter = 0
star_10000K.Star(plot=True)
star_10000K.Transit(0.05,0.0,False)
```

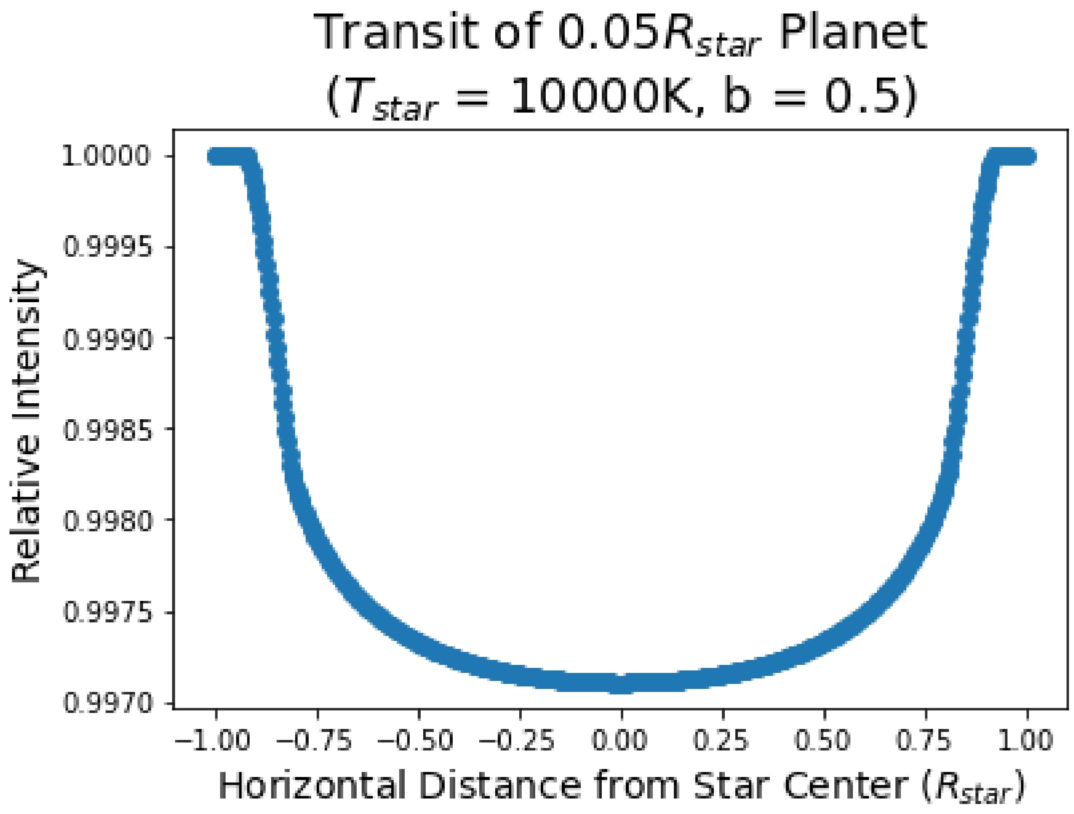
```
My program took 4.87 minutes to run
```


Surface Brightness of T=10000K Star
(at 5000Å)
y (R_{star})
x (R_{star})
Surface Brightness
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
0.9
0.8
0.7
0.6
0.5


Transit of 0.05R_{star} Planet
(T_{star} = 10000K, b = 0.0)
Relative Intensity
Horizontal Distance from Star Center (R_{star})
0.99900
0.99875
0.99850
0.99825
0.99800
0.99775
0.99750
0.99725
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

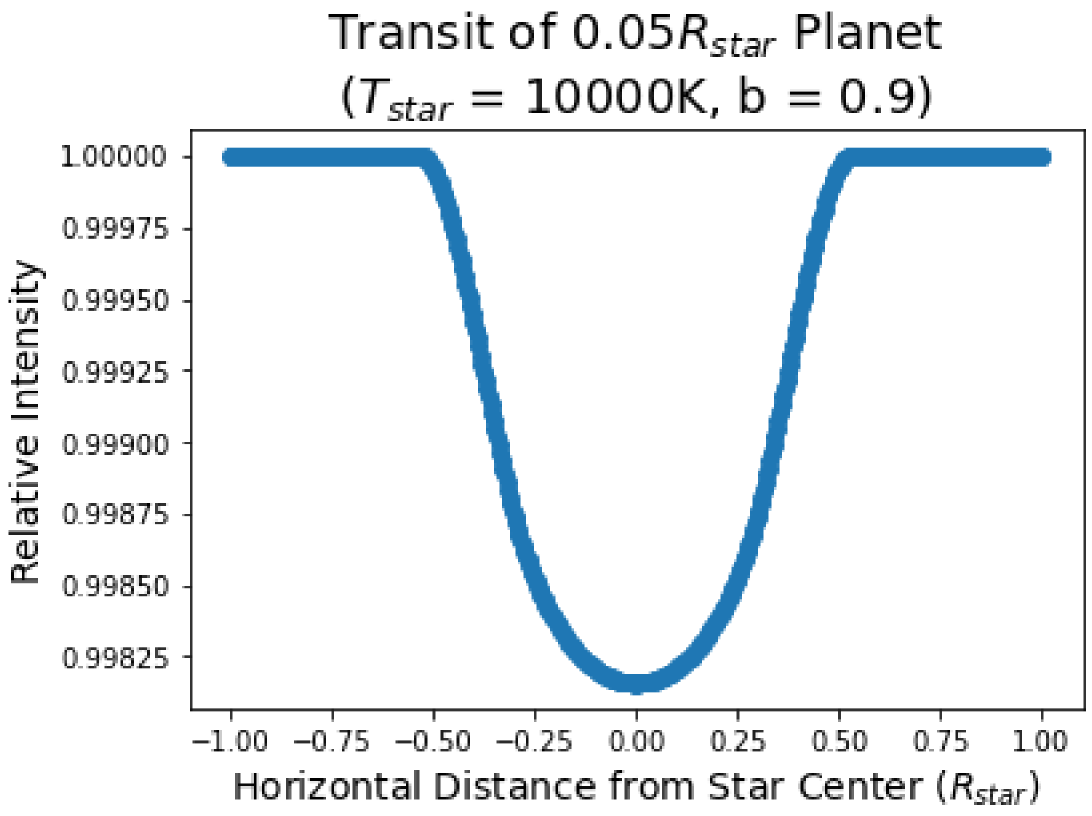
[31]:
```
# impact parameter = 0.5
star_10000K.Transit(0.05,0.5,False)
```

```
My program took 4.15 minutes to run
```



[32]:
```
# impact parameter = 0.9
star_10000K.Transit(0.05,0.9,False)
```

```
My program took 4.15 minutes to run
```

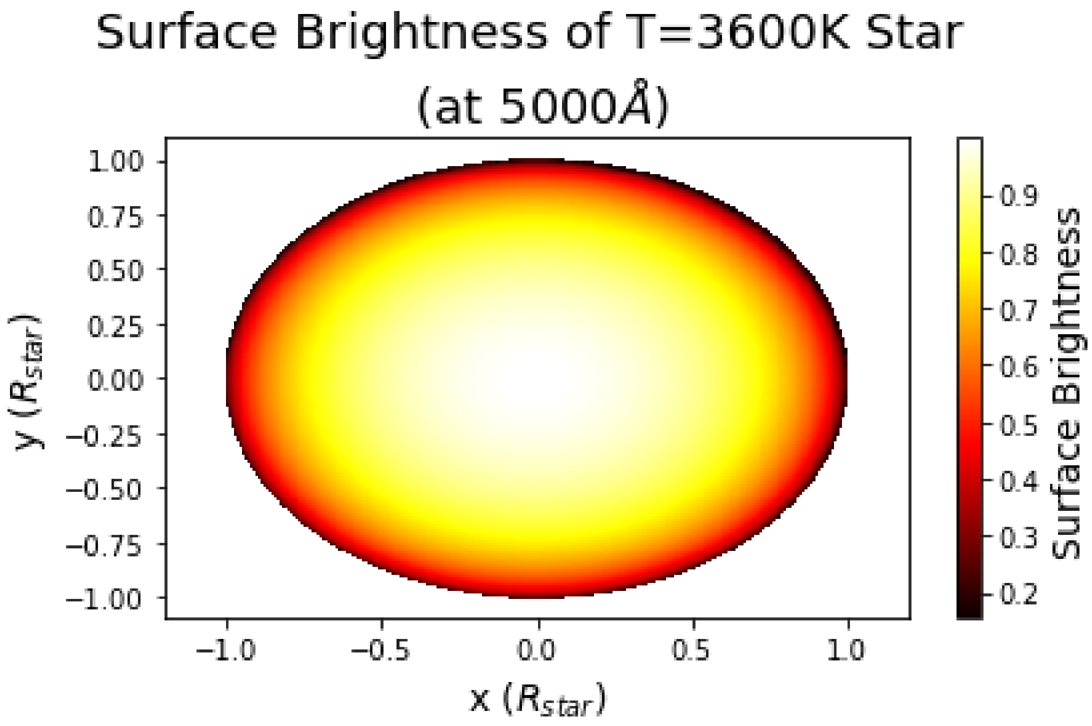


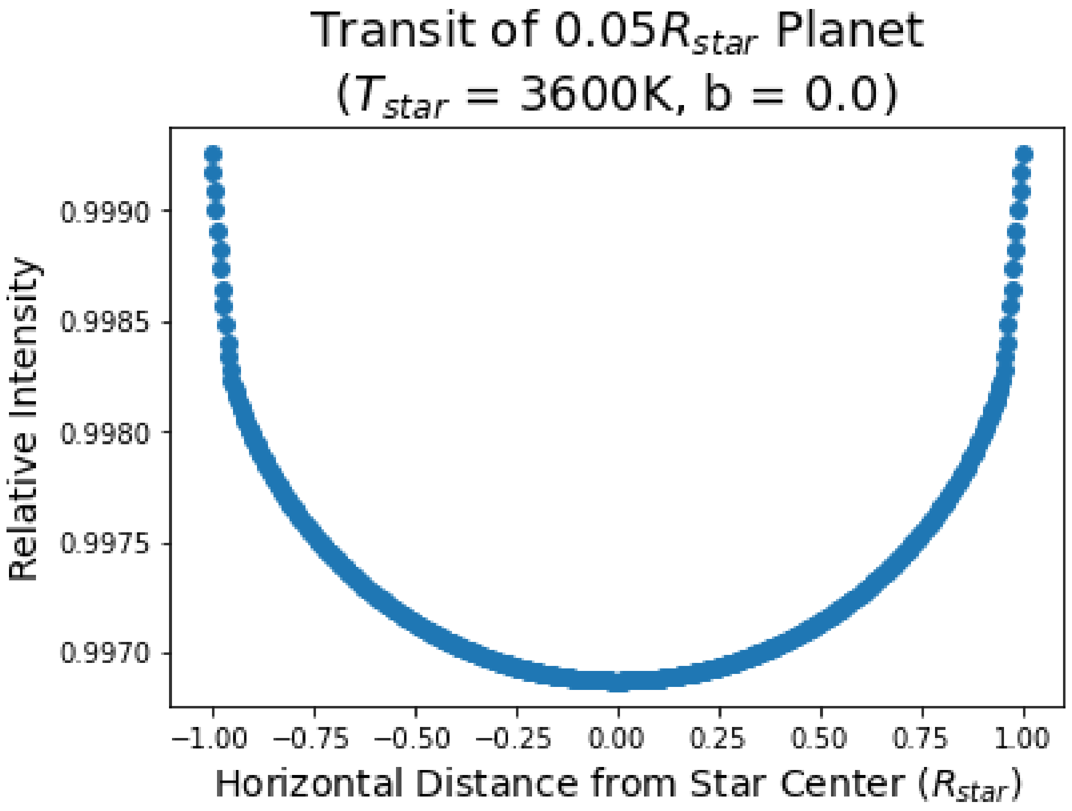
5.8.2 For 3600K, got parameters from: Column 10 (V filter) of Table 2 from Claret (1998) https://cdsarc.unistra.fr/viz-bin/ReadMe/J/A+A/335/647?format=html&tex=true

[14]:
```
star_3600K = LimbDarkening(3600,500)
```

[35]:
```
# impact parameter = 0
star_3600K.Star(plot=True)
star_3600K.Transit(0.05,0.0,False)
```

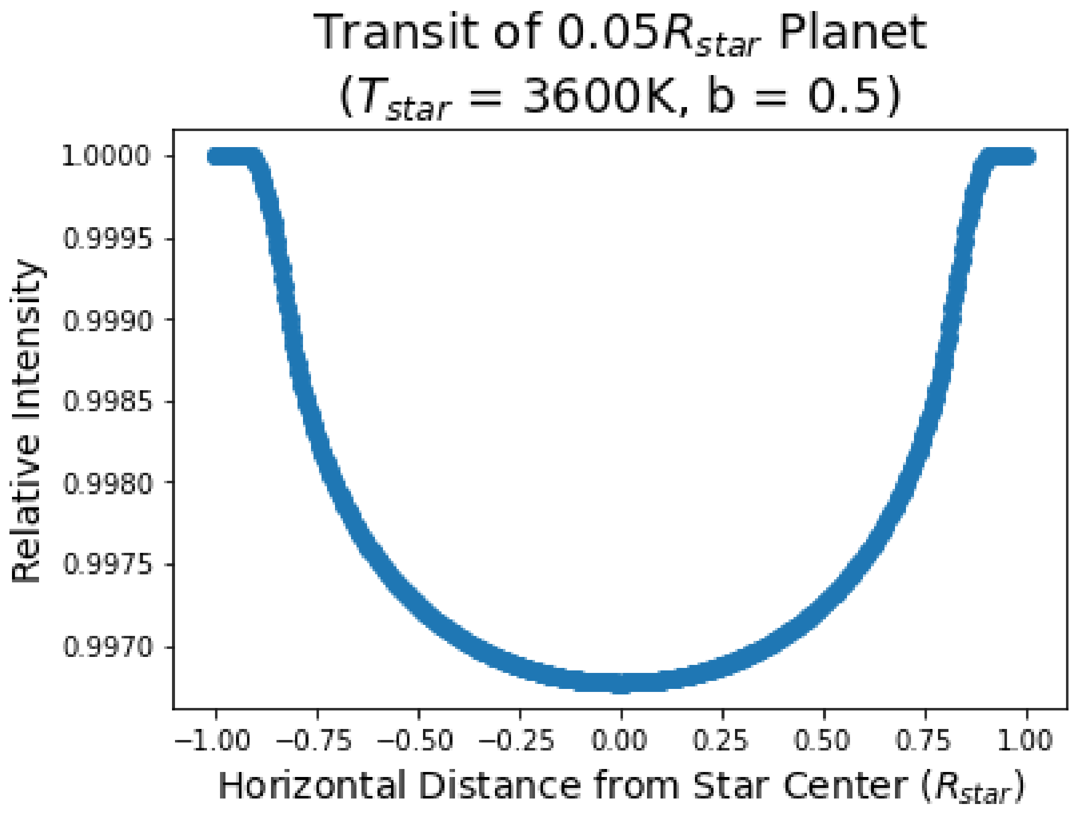
```
My program took 4.13 minutes to run
```


Surface Brightness of T=3600K Star (at 5000Å)
y (R_{star})
x (R_{star})
Surface Brightness
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2


Transit of 0.05R_{star} Planet (T_{star} = 3600K, b = 0.0)
Relative Intensity
Horizontal Distance from Star Center (R_{star})
0.9990
0.9985
0.9980
0.9975
0.9970
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

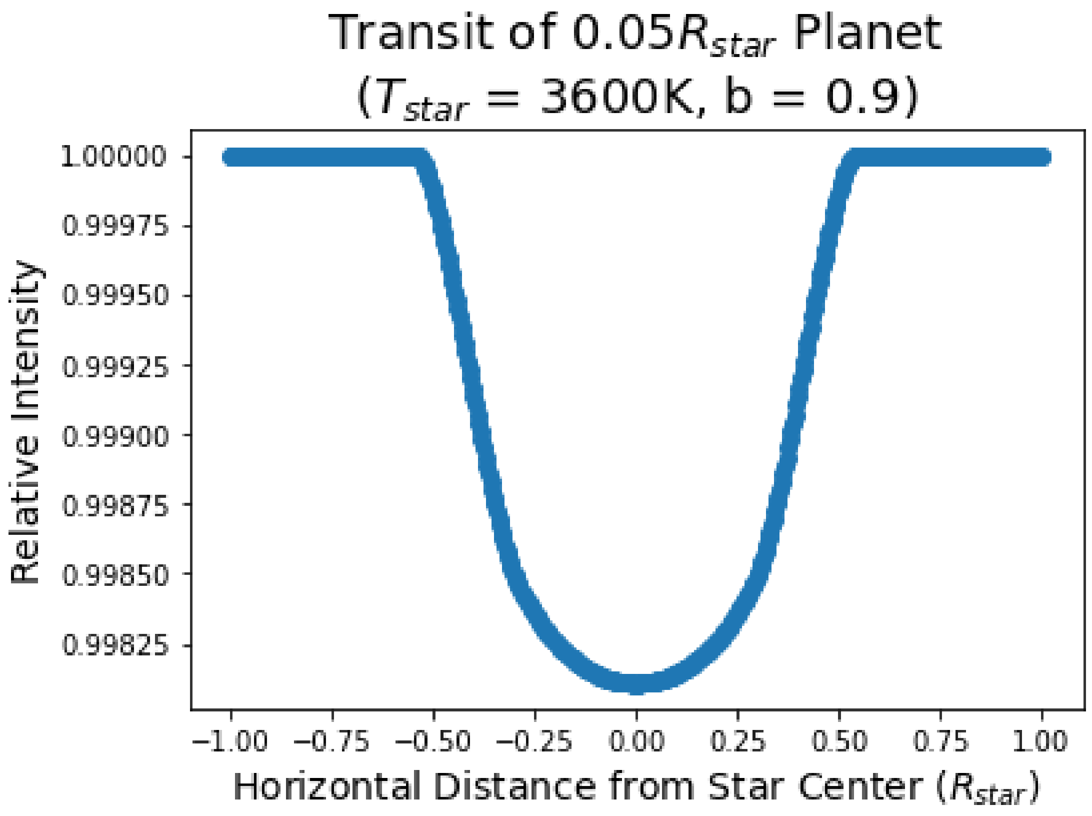
[36]:
```
# impact parameter = 0.5
star_3600K.Transit(0.05,0.5,False)
```

```
My program took 4.20 minutes to run
```

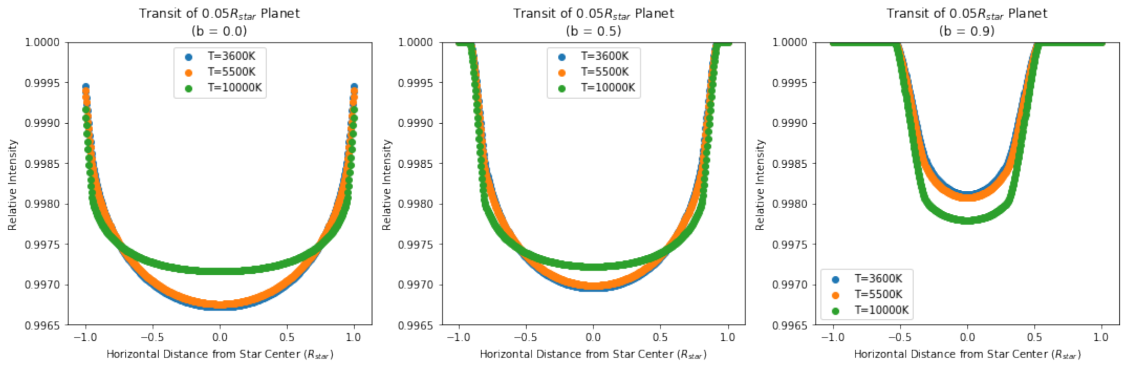


[15]:
```
# impact parameter = 0.9
star_3600K.Transit(0.05,0.9,False)
```

```
My program took 3.74 minutes to run
```



[30]:
```
LightCurveCompare()
```



5.9 4f) Comparing result from 4c (b=0.5) to results from Kreidberg 2015 who used 'batman'

5.9.1 Figure 2 of Kreidberg 2015 https://iopscience.iop.org/article/10.1086/683602/pdf

5.9.2 My plots are very similar qualititatively with the same wide U-shaped curve with flat edges at the beginning and end of the transit. Quantitatively, my model dooes not dip as low as Kreidberg (0.997 for mine, 0.990 for theirs). Overall, I'd say my model replicates theirs quite well. For b=0, my code stars when the planet is just on the edge which is why there's no horizontal lines at the top of the transit 'well' in this case.

6 5) Now let your star rotate with an equatorial velocity of $v_R = 10\frac{km}{s}$. The projected radial velocity of each surface element is a function of stellar latitude ϕ and longitude θ

##

$v_{rad} = v_R sin\theta cos\phi$

6.1 5a) Using your pixelated numerical star surface code, produce an intensity-weighted velocity profile plot for the star with no transit. This is also known as the rotational velocity profile (in the following, I use conventions define in Table 1 here https://en.wikipedia.org/wiki/Del_in_cylindrical_and_spherical_coordinates)

##

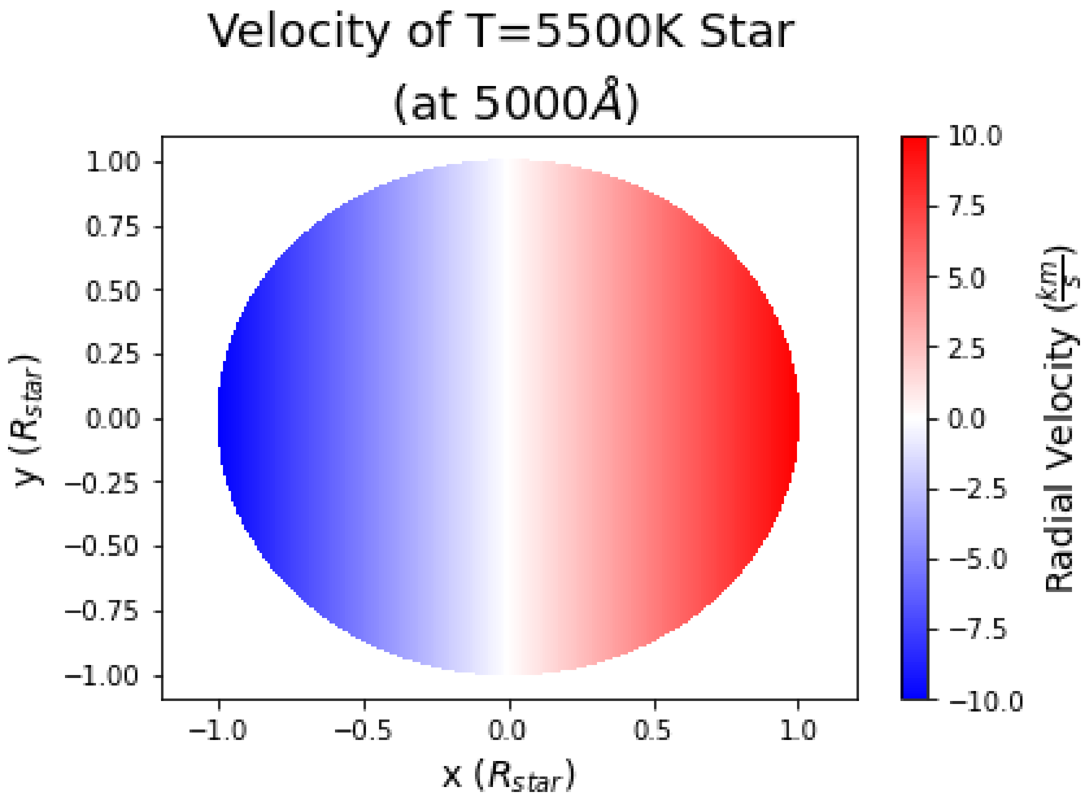
$\theta = arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right)$, $\phi = arctan\left(\frac{y}{x}\right)$

##

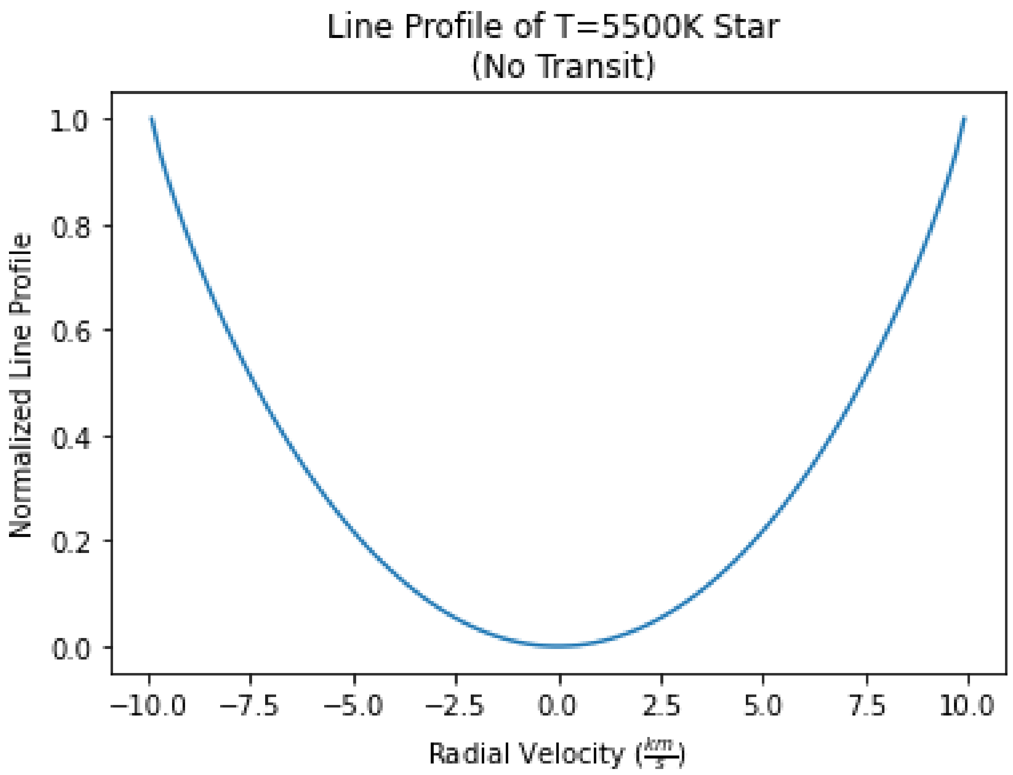
$v_{rad} = v_R sin\left(arctan\left(\frac{\sqrt{x^2+y^2}}{\sqrt{R_{star}^2-x^2-y^2}}\right)\right) cos\left(arctan\left(\frac{y}{x}\right)\right)$

[5]:
```python
# Tests of class and functions within the class
RotatingStar = LimbDarkening(5500,1000)
RotatingStar.RVProfile(plot='star')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```



```
[6]: RotatingStar.RVProfile(plot='profile')
```



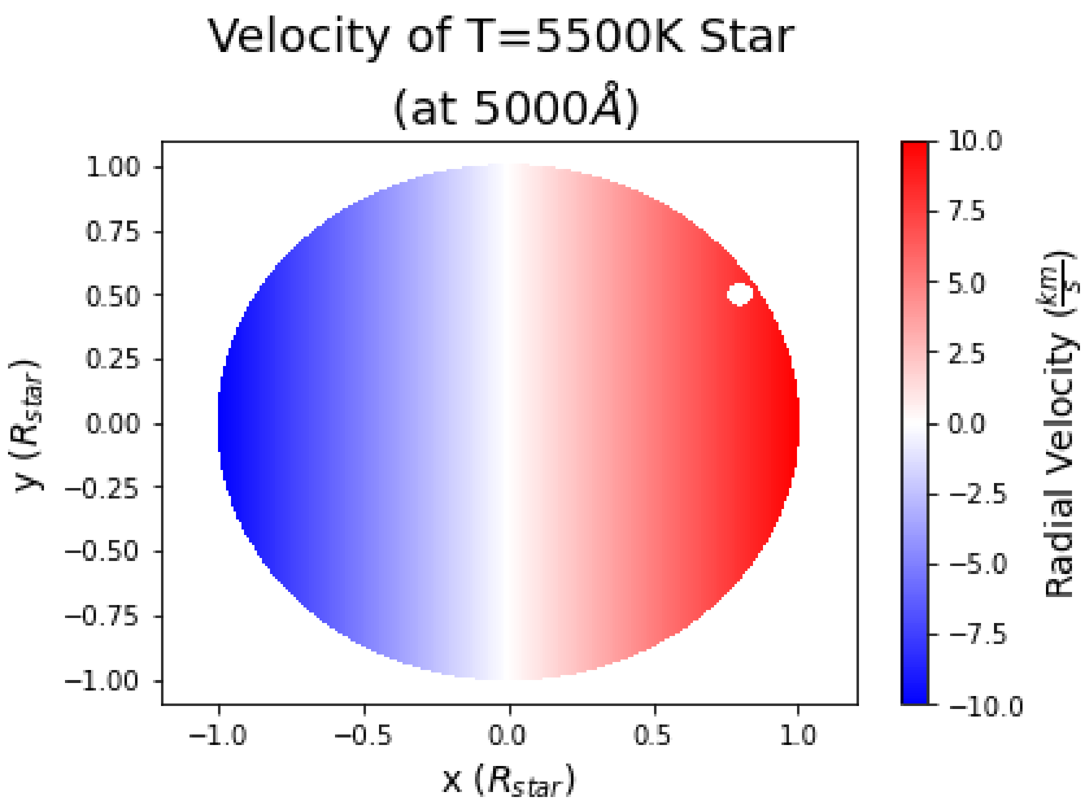
6.2 5b) Produce the same proifle for the star for when the planet is at four different positions across the disk of the star at an impact parameter of b=0.5

6.3 5c) Plot the difference between the 5a and 5b plots. Compare quantitatively to those in Figure 1 of Gaudia & Winn (2007, ApJ).

6.4 https://ui.adsabs.harvard.edu/abs/2007ApJ...655..550G/abstract

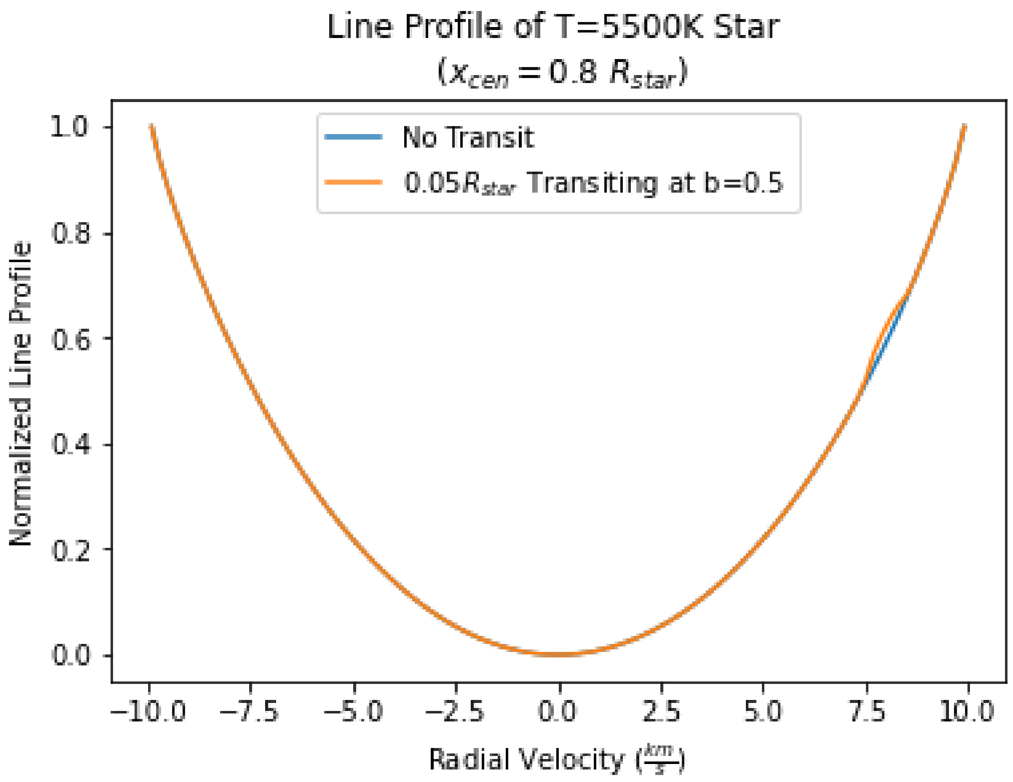
[10]:
```
RotatingStar.RVProfileTransit(x_center=0.8,plot='star')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```



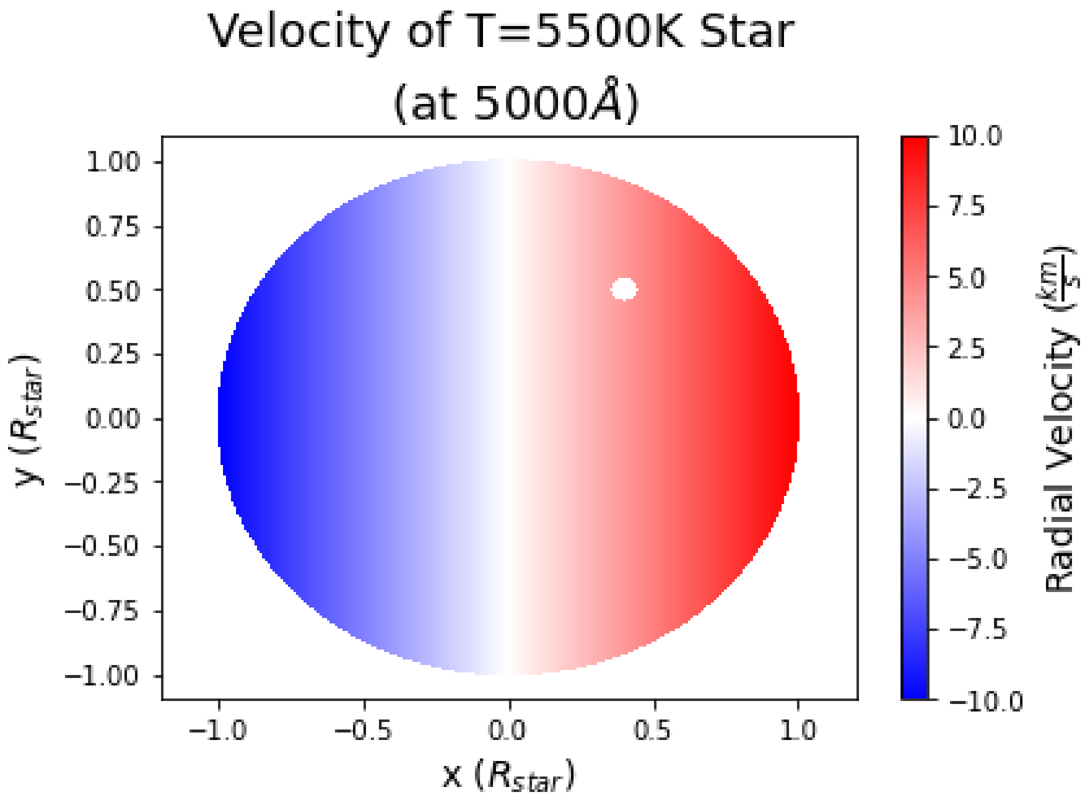
```
[14]: RotatingStar.RVProfileTransit(x_center=0.8,plot='profile')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```

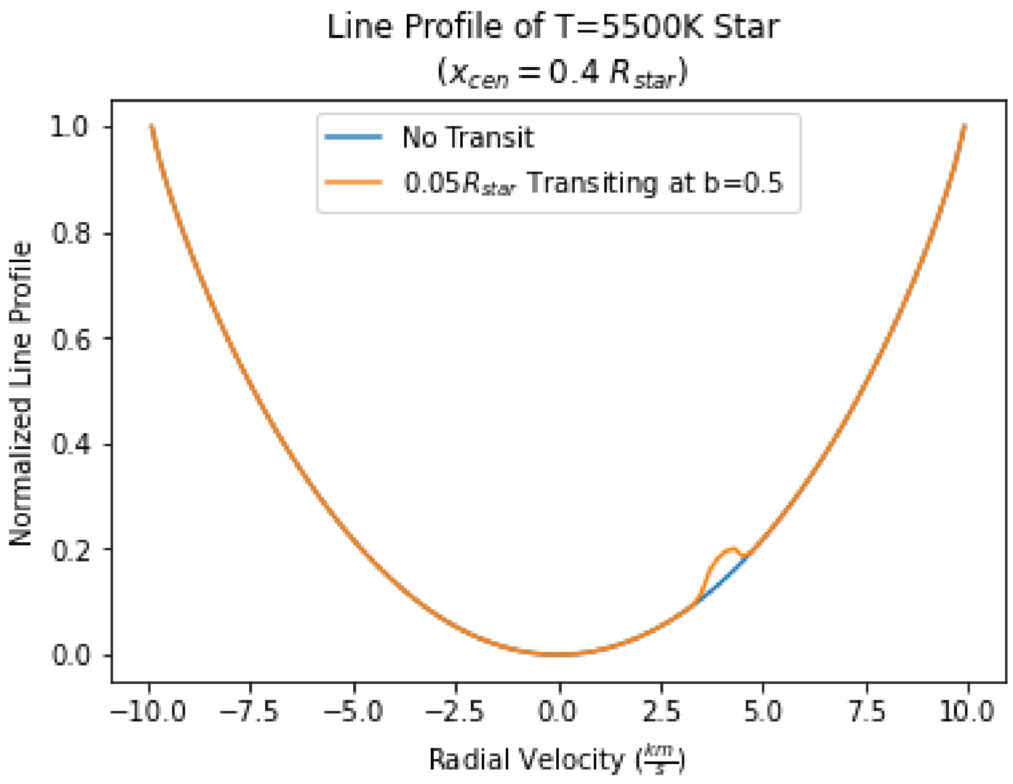


[15]:
```
RotatingStar.RVProfileTransit(x_center=0.4,plot='star')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```

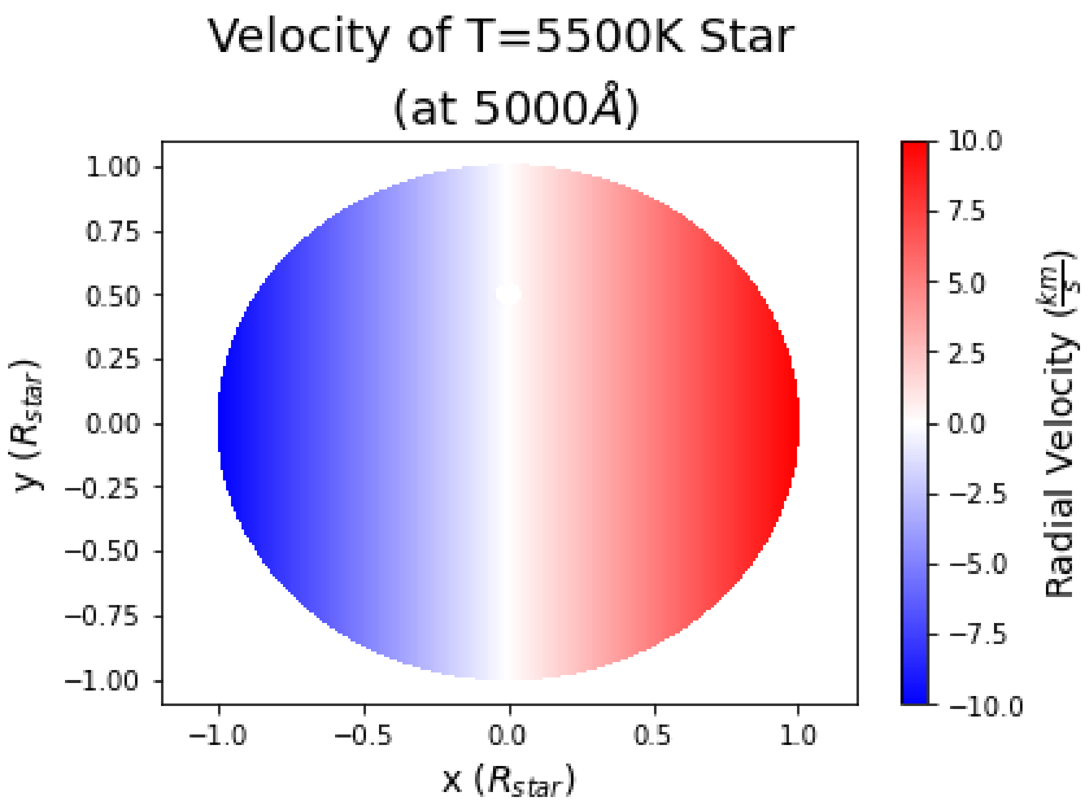


```
[16]: RotatingStar.RVProfileTransit(x_center=0.4,plot='profile')
```

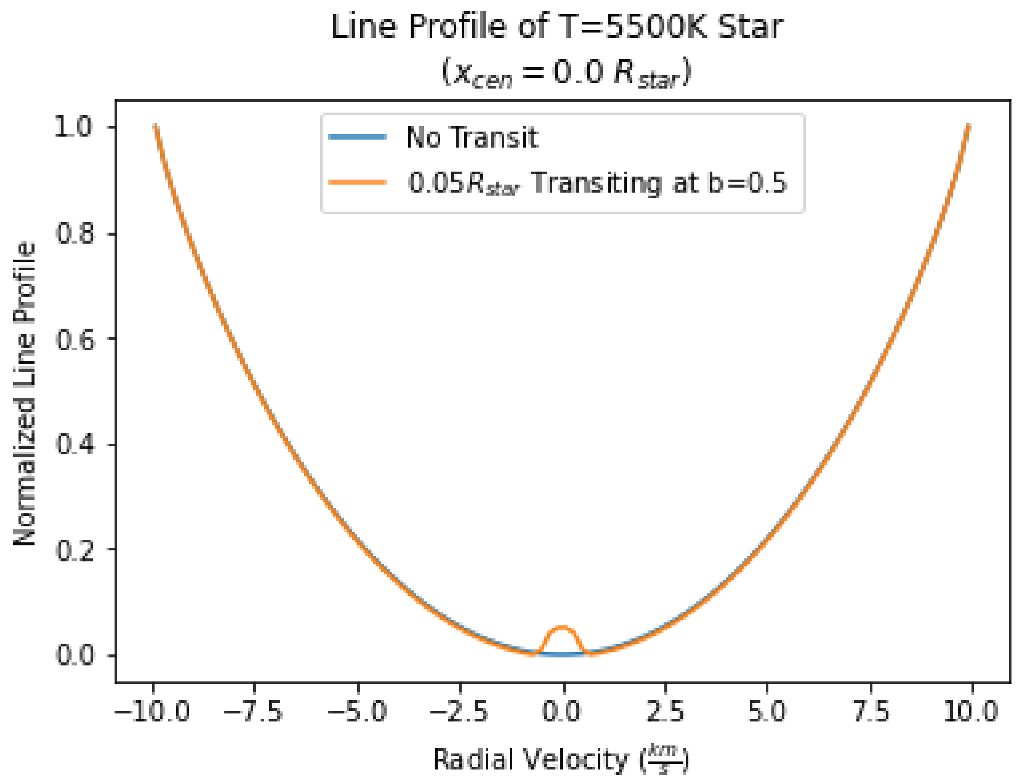


[17]:
```
RotatingStar.RVProfileTransit(x_center=0.0,plot='star')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```

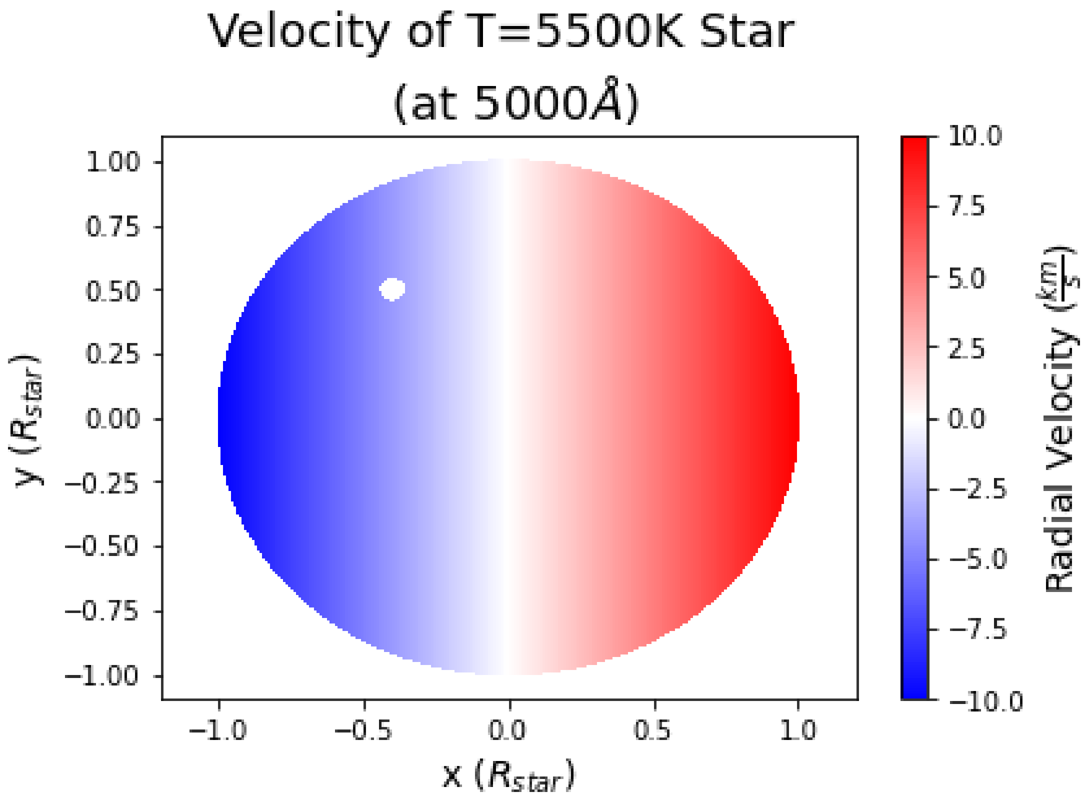


```
[18]: RotatingStar.RVProfileTransit(x_center=0.0,plot='profile')
```

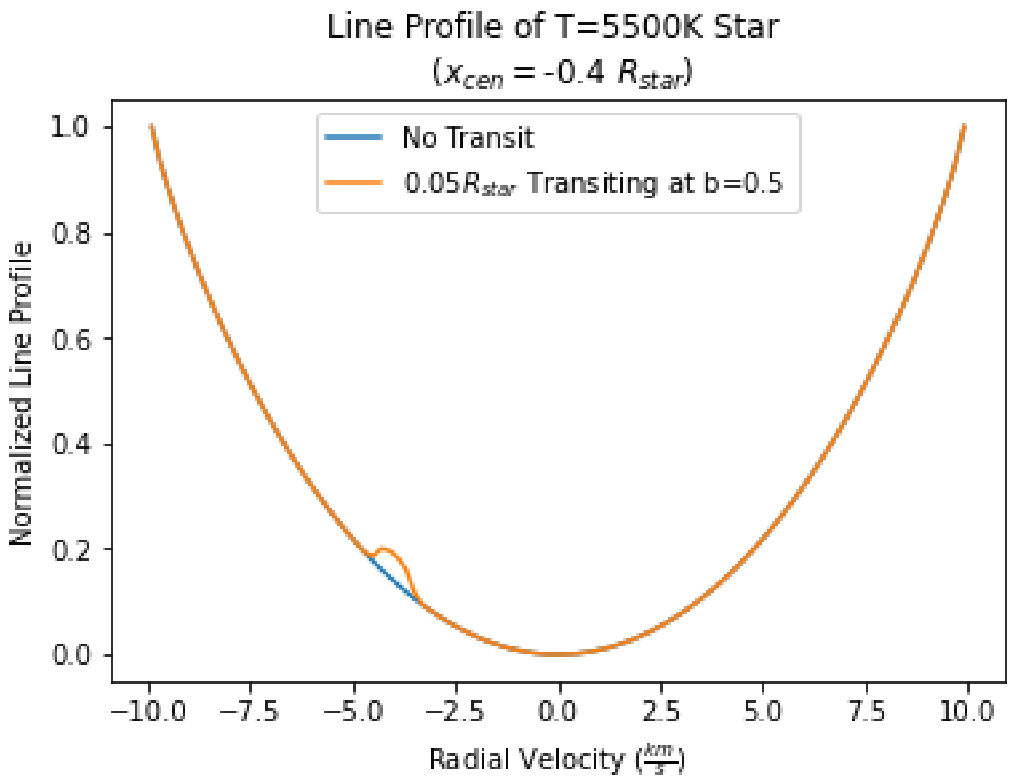


```
[19]: RotatingStar.RVProfileTransit(x_center=-.4,plot='star')
```

```
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:53: RuntimeWarning:
invalid value encountered in sqrt
  mu = np.sqrt(1-abs(r**2/R_star**2))
/d/users/jimmy/Documents/ASTR5490/HW3/LimbDarkening.py:213: RuntimeWarning:
invalid value encountered in sqrt
  theta = np.sqrt((x**2+y**2)/(1.0-x**2-y**2))
```



[20]:
```
RotatingStar.RVProfileTransit(x_center=-.4,plot='profile')
```



6.5

My plots are qualitatively very similar to those in Gaudi & Winn (2007, ApJ). My plots only include when the planet is just out-of-transit and fully in-transit so I don't have the zero-slope edges at the top-right and top-left of my plots like they do.

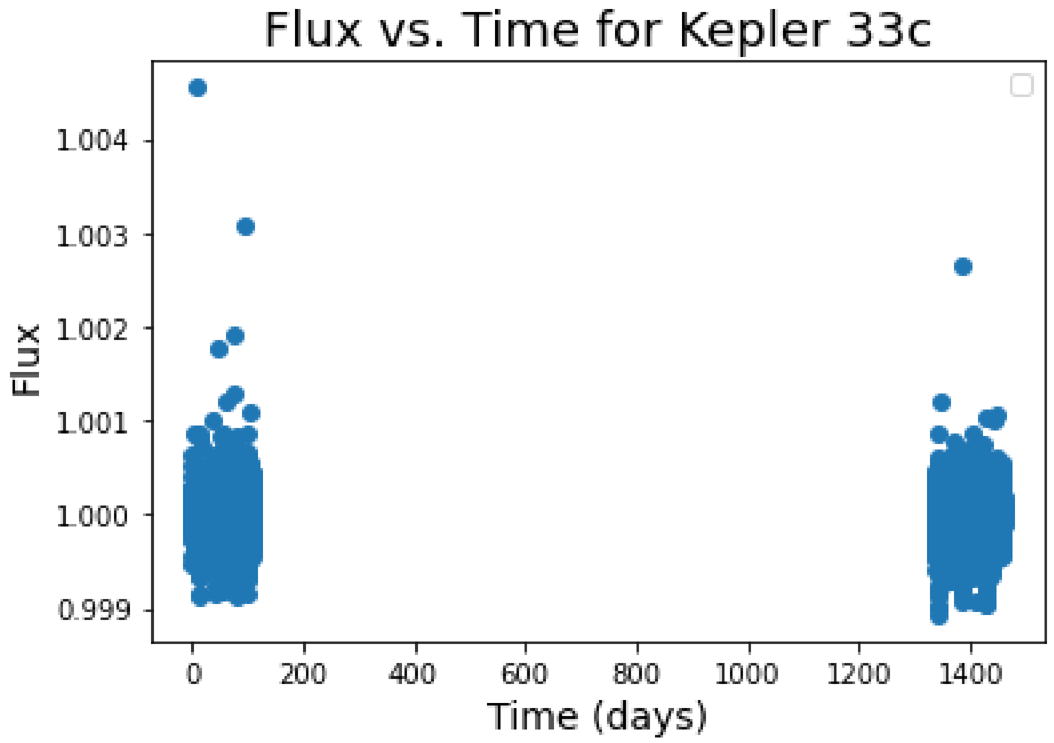
7 6) Fold Kepler 33 data at period of Planet C. Use 'batman' light curve tool (https://www.cfa.harvard.edu/~lkreidberg/batman/tutorial.html) to fit a light transit curve to the data and recover planet parameters from Lissauer et al. (2012; https://ui.adsabs.harvard.edu/abs/2012ApJ...750..112L/abstract)

7.1 Raw Light Curve (chose subset of phase interval to zoom in on dip)

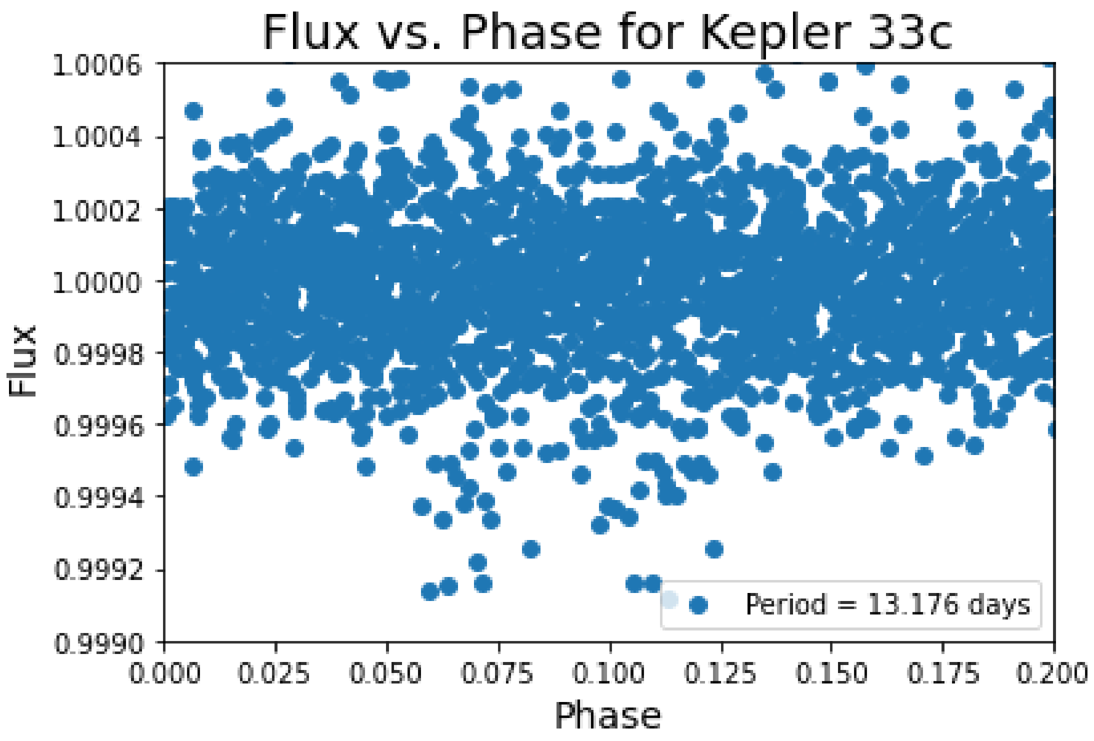
[4]:
```
# Initialize my Periodicity class and plot the light curve
Kepler33c = Periodicity('Kepler33.dat','Kepler␣
↪33c',['col0','col1','col2'],5000,contiguous='Bookend',period=13.17562)
```

[5]:
```
time,rel_flux,err = Kepler33c.LightCurve(plot=True,xaxis='Time')
```

```
No handles with labels found to put in legend.
```



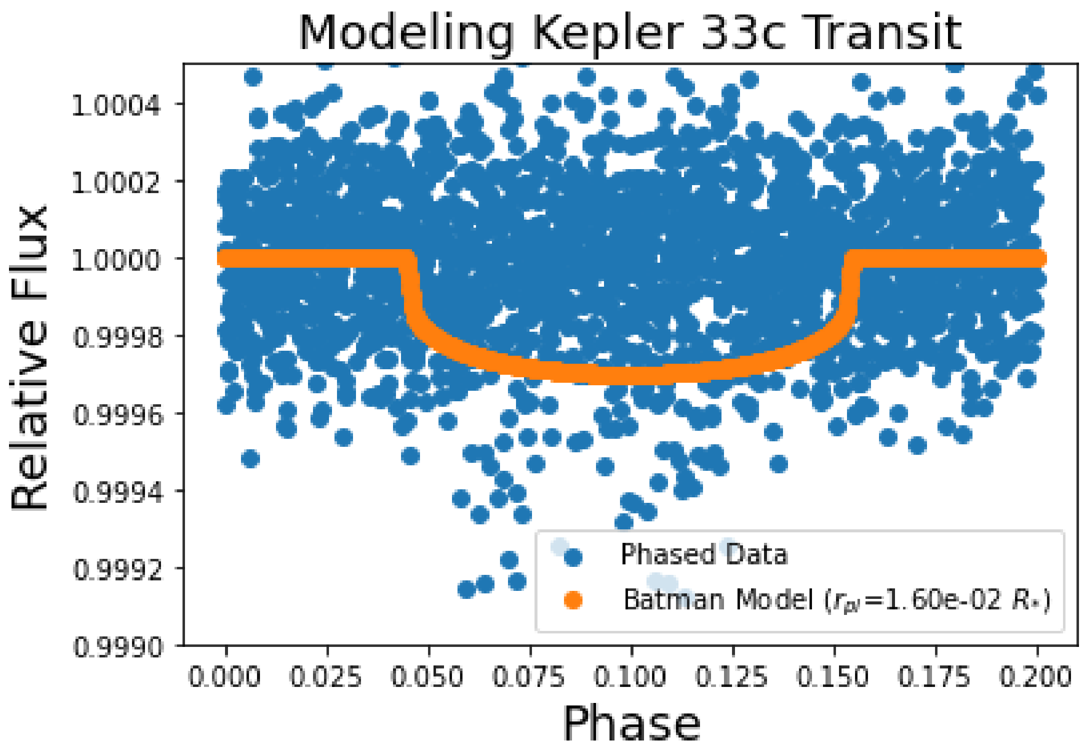
[6]:
```
phase,rel_flux,err = Kepler33c.LightCurve(plot=True,xaxis='Phase')
```



[37]:
```
# Using actual parameters from paper
batman_time_actual,batman_flux_actual,rad_pl_actual = BatmanModel(P=13.
↪17562,rad_pl=0.01602,a=13.8,i=88.19,e=0.2,w=0.0,coeff=[0.4899, 0.
↪1809],plot=False)
```

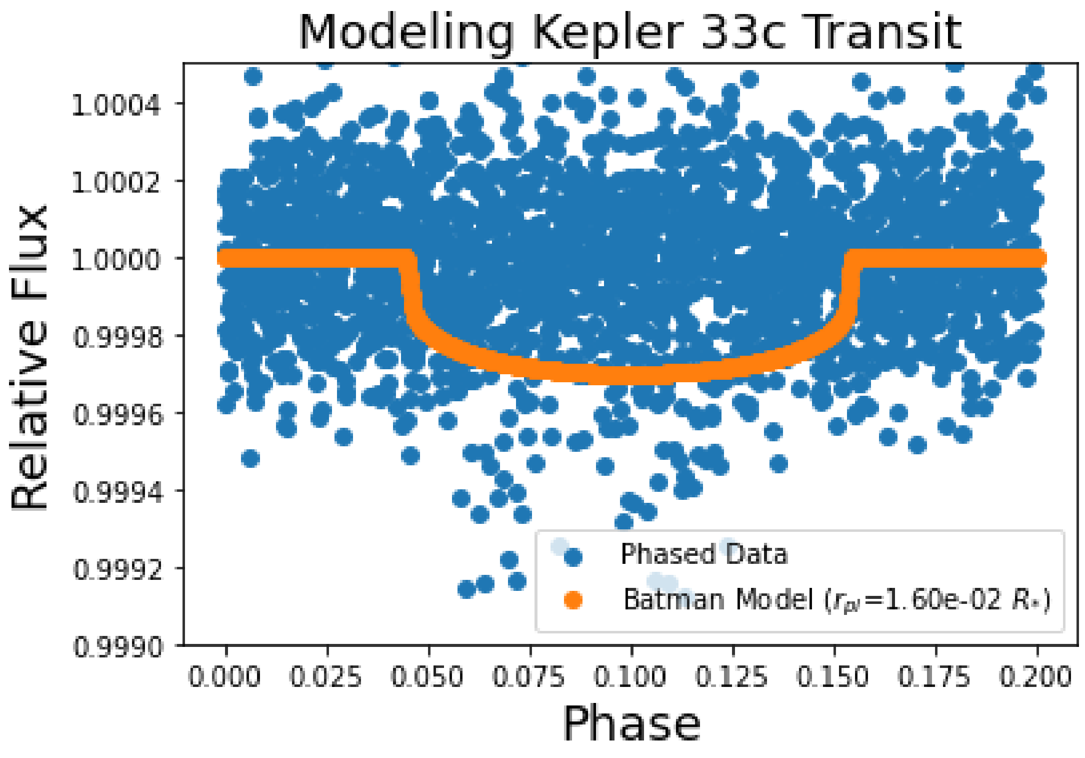
[38]:
```
# Cut real data to zoom in on transit
phase_cut = [(i,x) for (i,x) in enumerate(phase) if x <= 0.2]
cuts = [x[0] for x in phase_cut]
phase_cuts = [x[1] for x in phase_cut]
rel_flux_cuts = []
for index in cuts:
    flux = rel_flux[index]
    rel_flux_cuts.append(flux)

def BatmanCompare(xdata,ydata,rad_pl):
    # Plot real phased data and Batman model
    fig,ax = plt.subplots()
    ax.scatter(phase_cuts,rel_flux_cuts,label='Phased Data')
    ax.scatter(xdata,ydata,label=r'Batman Model ($r_{pl}$=%.2e $R_{*}$)'%rad_pl)
    ax.get_yaxis().get_major_formatter().set_useOffset(False)
    ax.set_ylim(0.999,1.0005)
    ax.set_xlabel('Phase',fontsize=18)
    ax.set_ylabel('Relative Flux',fontsize=18)
    ax.set_title('Modeling Kepler 33c Transit',fontsize=18)
    ax.legend()
```

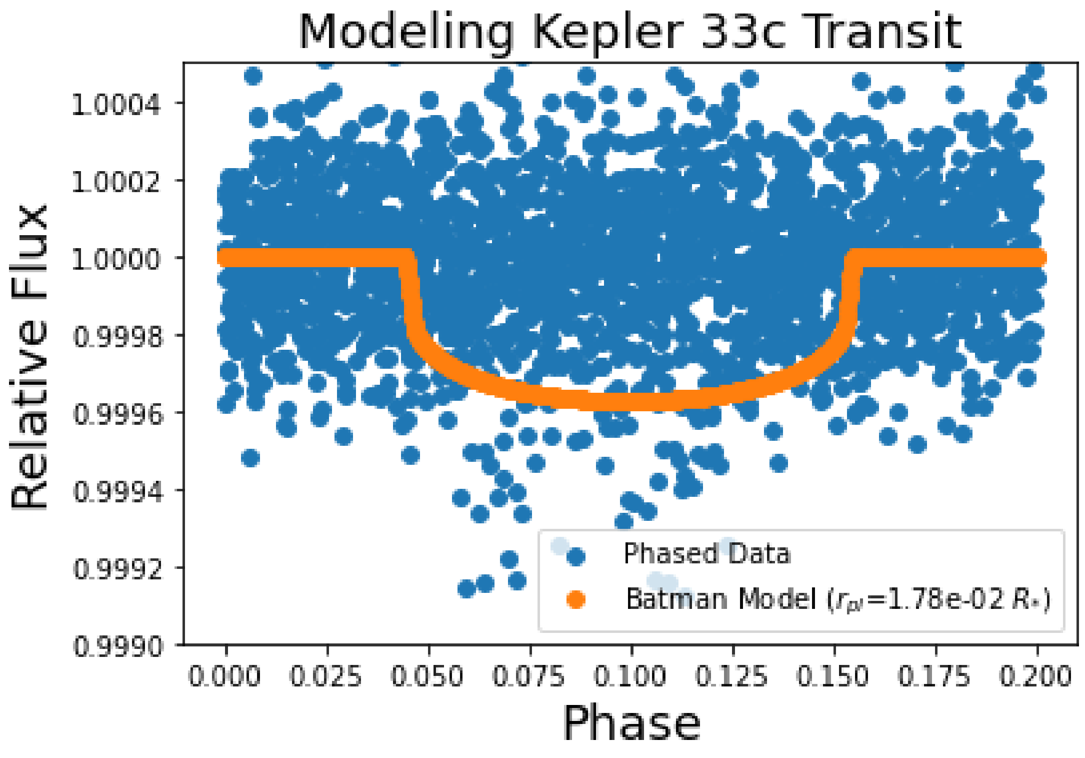
[39]:
```
BatmanCompare(batman_time_actual,batman_flux_actual,rad_pl_actual)
```

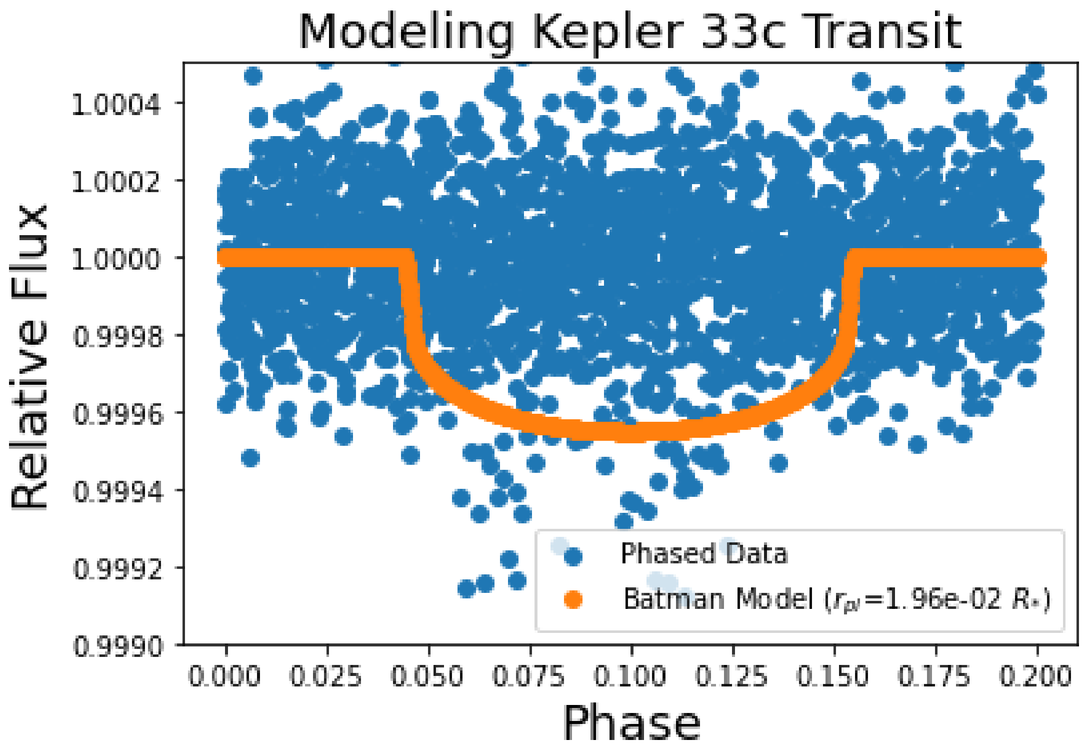


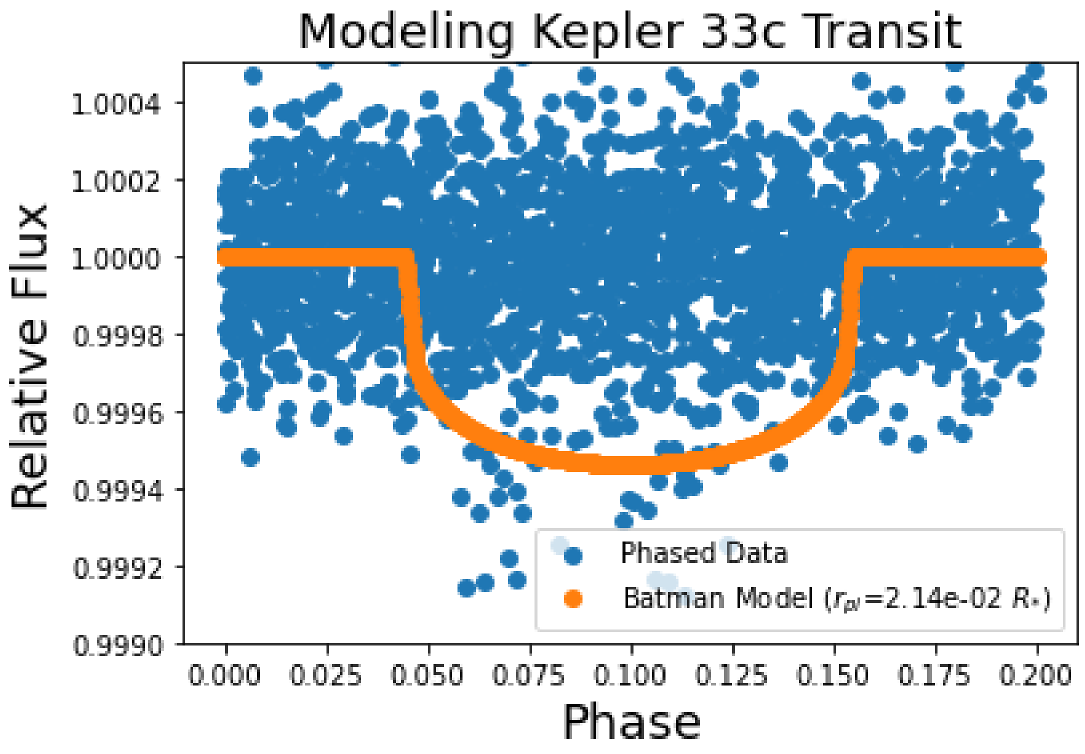
7.1.1 Actual radius from paper doesn't look like it quite fits. I'll try looping over radii to test better versions

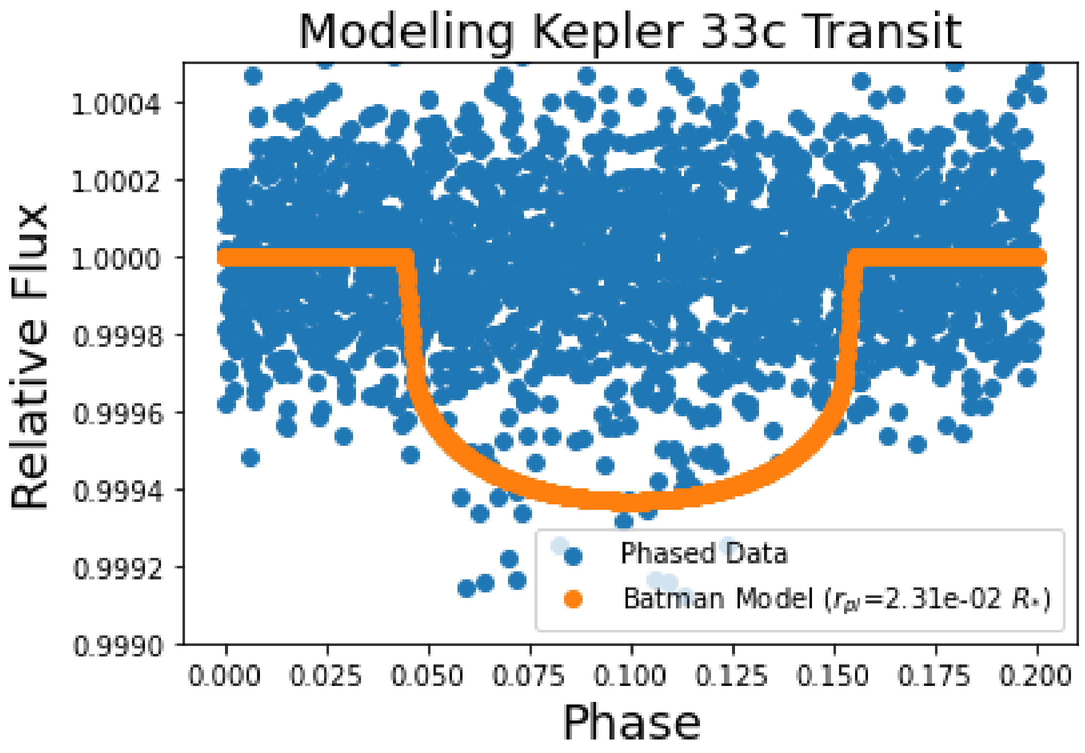
[45]:
```
test_radii = np.linspace(rad_pl_actual,2*rad_pl_actual,10)
for radius in test_radii:
    times, fluxes,rad_pl = BatmanModel(P=13.17562,rad_pl=radius,a=13.8,i=88.
↪19,e=0.2,w=0.0,coeff=[0.4899, 0.1809],plot=False)
    BatmanCompare(times,fluxes,rad_pl)
```


Modeling Kepler 33c Transit
Relative Flux
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Phase
Phased Data
Batman Model (r_{pl}=1.60e-02 R_*)

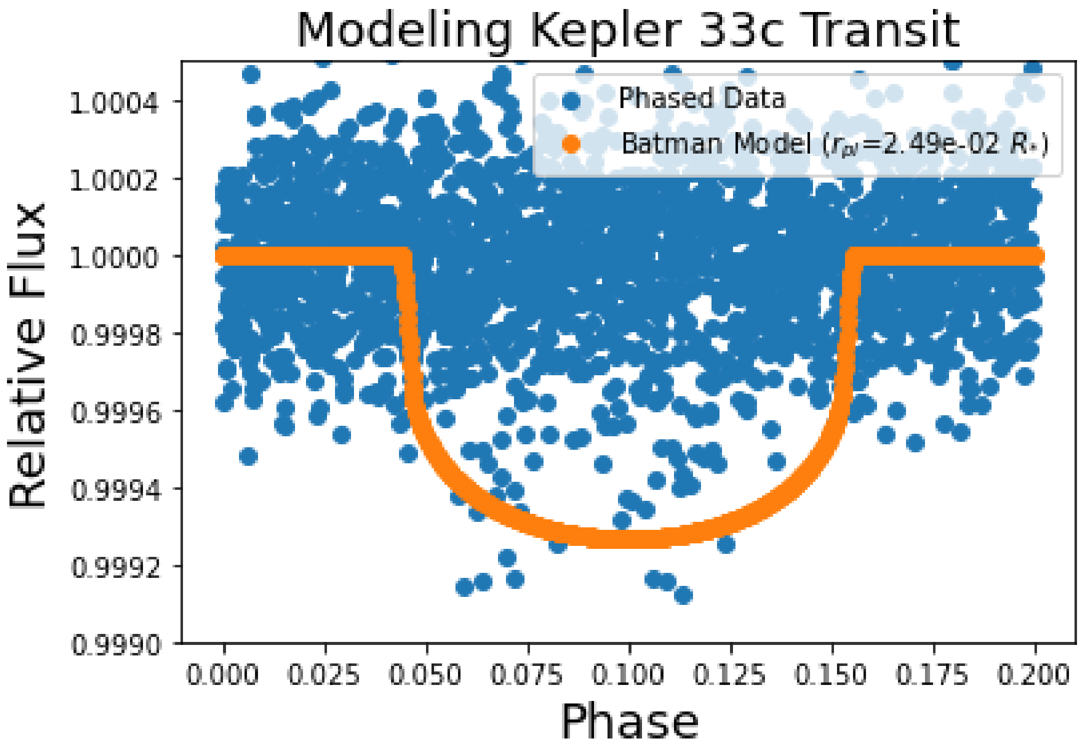

Modeling Kepler 33c Transit
Relative Flux
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Phase
Phased Data
Batman Model (r_{pl}=1.78e-02 R_*)

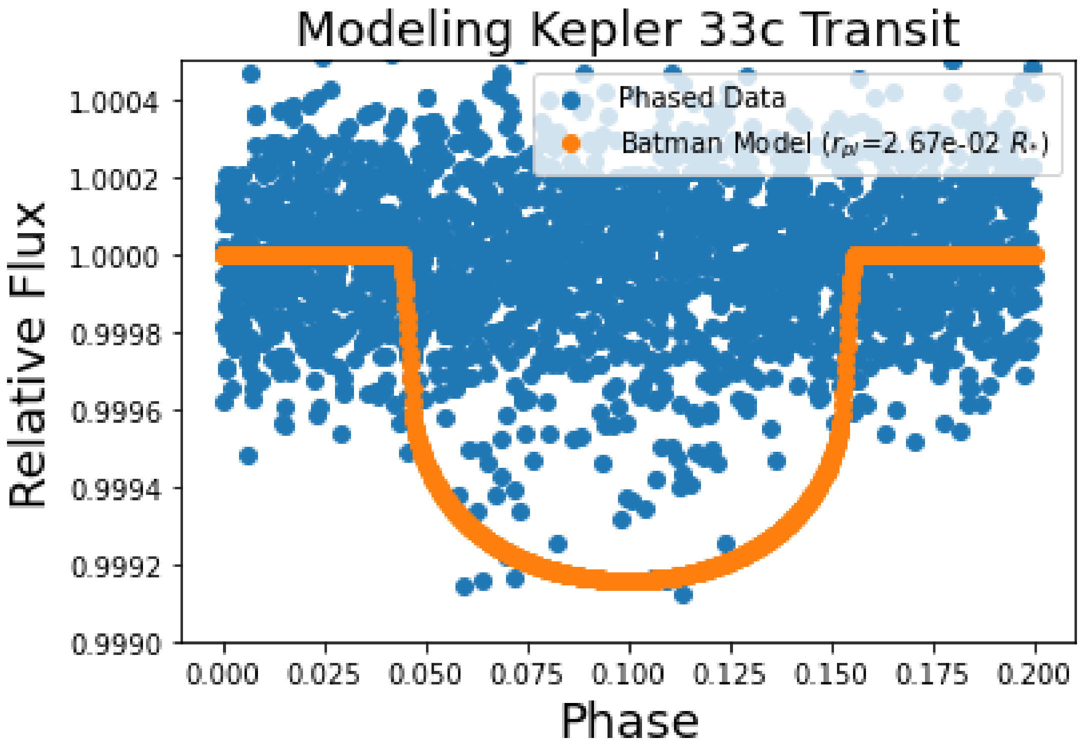

Modeling Kepler 33c Transit
Relative Flux
Phase
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Phased Data
Batman Model (r_{pl}=1.96e-02 R_*)

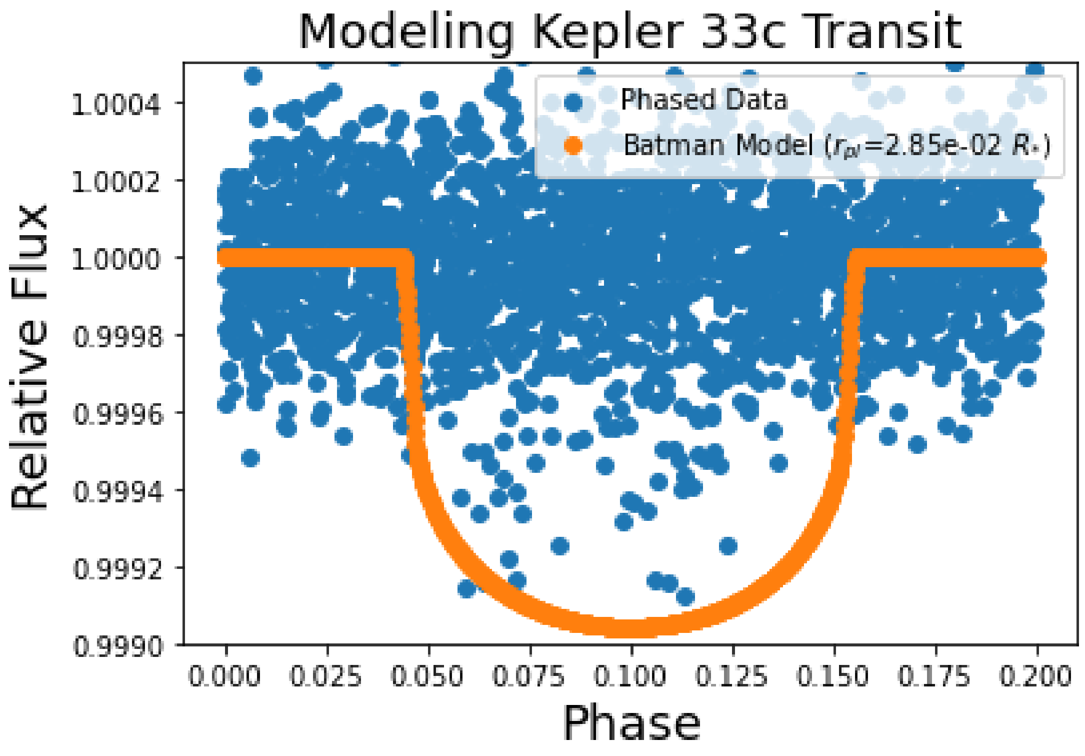

Modeling Kepler 33c Transit
Relative Flux
Phase
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Phased Data
Batman Model (r_{pl}=2.14e-02 R_*)

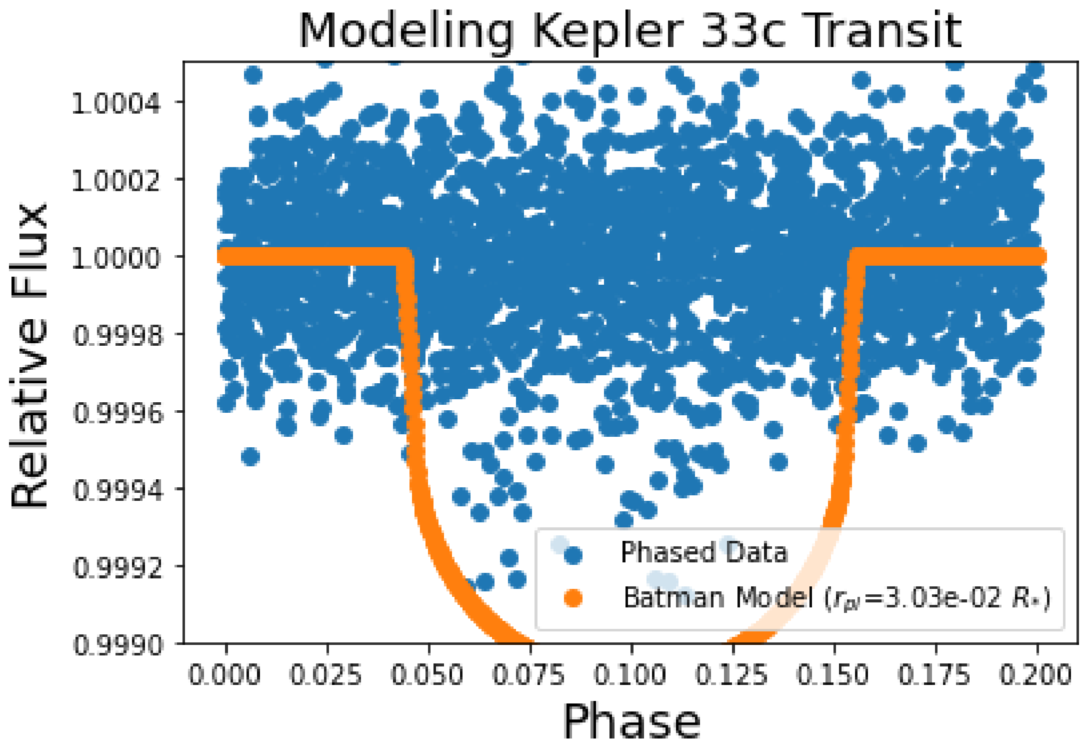


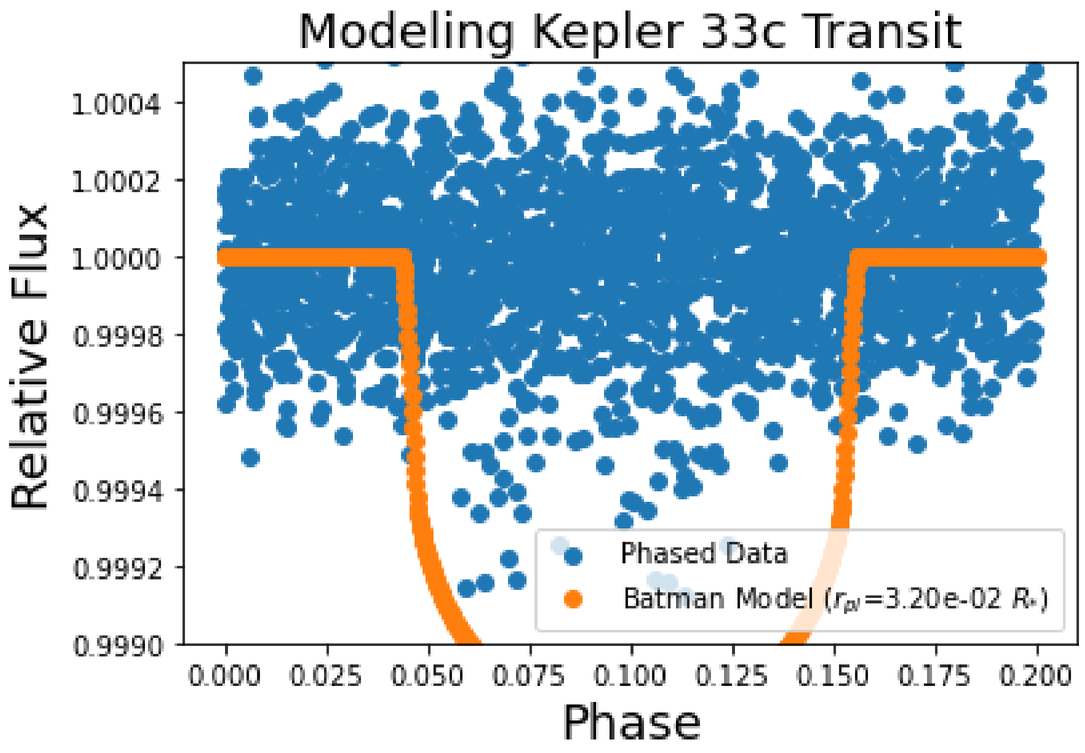
Modeling Kepler 33c Transit
Relative Flux
Phase
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Phased Data
Batman Model (r_{pl}=2.31e-02 R_*)


Modeling Kepler 33c Transit
Phased Data
Batman Model (r_{pl}=2.49e-02 R_*)
Relative Flux
Phase
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

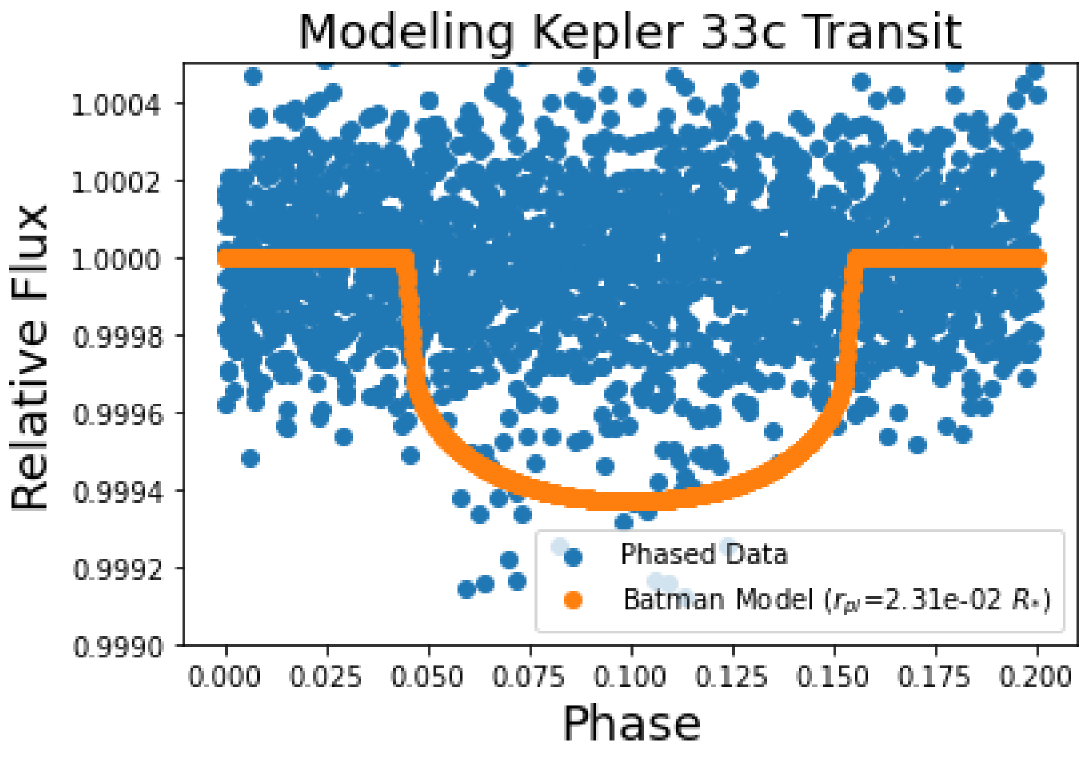

Modeling Kepler 33c Transit
Phased Data
Batman Model (r_{pl}=2.67e-02 R_*)
Relative Flux
Phase
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Modeling Kepler 33c Transit
Relative Flux
Phase
Phased Data
Batman Model (r_{pl}=2.85e-02 R_*)
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Modeling Kepler 33c Transit
Relative Flux
Phase
Phased Data
Batman Model (r_{pl}=3.03e-02 R_*)
1.0004
1.0002
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200



[47]:
```
# Plot best radius
finalx,finaly,rad = BatmanModel(P=13.17562,rad_pl=.0231,a=13.8,i=88.19,e=0.
↪2,w=0.0,coeff=[0.4899, 0.1809],plot=False)
BatmanCompare(finalx,finaly,rad)
```



7.1.2 My best-fit used a radius of $0.0231R_*$ which is 1.44 times greater than the published value of $0.01602R_*$. Not bad at all!

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  5 13:16:09 2020

@author: jimmy
"""
# Import numpy module
import numpy as np
import matplotlib.pyplot as plt

# Function to read simple text files
def Read(filename,col_names,unpack_bool=True):
    # Use numpy's loadtxt function to read columns of text file
    data = np.genfromtxt(filename,delimiter=' ',names=True)
    col1, col2, col3 = data[col_names[0]],data[col_names[1]],data[col_names[2]]
    return(col1,col2,col3)

# Function to read NASA Exoplanet Archive text files
def ReadNASA(filename,skip):
    # Read and return data of confirmed exoplanets from NASA Exoplanet Archive
    data = np.genfromtxt(filename,dtype=None,delimiter=',',skip_header=skip,names=True,invalid_r
    return data
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  5 12:56:37 2020

@author: jimmy
"""
# Import relevant modules/packages
import numpy as np
import matplotlib.pyplot as plt
from astropy.timeseries import LombScargle
from ReadFile import Read

class Periodicity:

    def __init__(self,filename,objectname,colNames,numPoints=None,contiguous='True',period=None
        # Inputs:
        #     filename: file path or file name (if file in same folder as notebook)
        #     objectname: name of object you're plotting curve of
        #     numPoints: number of data points you want to use from the file
        #     period: period of plot feature (only used if xaxis='Phase' to fold the data)
        self.file = filename
        self.object = objectname
        self.numPoints = numPoints
        self.period = period

        # Extract time, flux, and error data from text file
        self.times,self.fluxes,self.errors = Read(self.file,colNames)

        # Decide what times array to make
        if self.numPoints == None:
            self.times = [time - self.times[0] for time in self.times]
        else:
            if contiguous == 'True':
                # Select contiguous interval of points
                self.times = [time - self.times[0] for time in self.times[:self.numPoints]]
                self.fluxes = self.fluxes[:self.numPoints]
                self.errors = self.errors[:self.numPoints]
            elif contiguous == 'Bookend':
                # Select sparsed interval of data
                self.times_i = [time - self.times[0] for time in self.times[:self.numPoints]]
                self.times_f = [time - self.times[0] for time in self.times[-self.numPoints:]]
                self.times = self.times_i + self.times_f

                # Concatenate first and last n elements of flux list
                self.fluxes_i = self.fluxes[:self.numPoints]
                self.fluxes_f = self.fluxes[-self.numPoints:]
                self.fluxes = [*self.fluxes_i,*self.fluxes_f]

                # Concatenate first and last n elements of error list
                self.errors_i = self.errors[:self.numPoints]
                self.errors_f = self.errors[-self.numPoints:]
                self.errors = [*self.errors_i,*self.errors_f]
            elif contiguous == 'Random':
                # Generate random list of indices
                randoms = np.random.randint(0, high=len(self.times), size=numPoints)
                #print(randoms)
                # Select random sparsed interval of data
                self.times = self.times[randoms]

                # Concatenate first and last n elements of flux list
                self.fluxes = self.fluxes[randoms]

                # Concatenate first and last n elements of error list
                self.errors = self.errors[randoms]
```

```python
# Function for plotting light curves from text file
def LightCurve(self,plot=True,xaxis='Time',curve='Flux'):
    # Inputs:
    #     plot: boolean to decide whether to plot the data (True) or not (False)
    #     xaxis: decide which x parameter to calculate/plot ('Time or Phase')
    #     curve: string that indicates y parameter being plotted (used in axis label)
    # Returns:
    #     xdata: array with data from x-axis
    #     fluxes: array with associated y-axis data
    #     errors: measurement errors read from text file

    # Define list of data to plot on x-axis (time or phase)
    xdata = []

    if plot == True:
        # Initialize axis figure and axis
        fig = plt.figure()
        ax = fig.add_subplot(111)

        # Decide what x-axis should be
        if xaxis == 'Time':
            xlabel = 'Time (days)'

            # Plot flux vs time
            ax.scatter(self.times,self.fluxes)

            xdata = self.times

        elif xaxis == 'Phase':
            xlabel = 'Phase'

            # Calculate phase from time data
            phases = [(time%self.period)/self.period for time in self.times]

            # Make a scatter plot of flux vs. phase
            ax.scatter(phases,self.fluxes,label='Period = {0:.3f} days'.format(self.period)

            xdata = phases
            ax.set_ylim(0.999,1.0006)
            ax.set_xlim(0.0,0.2)

        # Add plot features
        ax.set_xlabel(xlabel,fontsize=14)
        ax.set_ylabel('{0}'.format(curve),fontsize=14)
        ax.set_title('{0} vs. {1} for {2}'.format(curve,xaxis,self.object),fontsize=18)
        ax.legend()
    else:
        # Decide what x-axis should be
        if xaxis == 'Time':
            xdata = self.times
        elif xaxis == 'Phase':
            # Calculate phase from time data
            phases = [(time%self.period)/self.period for time in self.times]
            xdata = phases

    return(xdata,self.fluxes,self.errors)

# Function to generate power spectrum from a flux vs. time dataset
def LS(self,minP,maxP,numIntervals,i,flux,plot=False,trueP=None):

    if flux == []:
        flux = self.fluxes

    # Define range of frequencies to search over
    minfreq = 1./maxP
    maxfreq = 1./minP
```

```
        # Make list of frequencies within the range
        frequency = np.linspace(minfreq,maxfreq,numIntervals)

        # Use LombScargle method to calculate power as a function of those frequencies
        power = LombScargle(self.times,flux,self.errors,nterms=i).power(frequency)

        # Find maximum power and frequency/period of maximum power
        maxp = np.max(power)
        maxind = np.argmax(power)

        maxfreq = frequency[maxind]
        best_period = 1./maxfreq

        if plot == True:
            # Plot power spectrum using lists from above
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.plot(frequency,power)

            # Set axes limits
            ax.set(xlim=(frequency[0],frequency[-1]), ylim=(0,np.max(power)))
            ax.set_xlabel('Frequency (1/days)',fontsize=14)
            ax.set_ylabel('Power',fontsize=14)
            ax.set_title('Power vs. Freq. for {0}'.format(self.object),fontsize=18)

            # Plot line indicating period of system from SIMBAD
            if trueP != None:
                ax.vlines(1./trueP,0,1,linestyle='dashed',label='Published Period ({0:.3f} days

            # Plot vertical line of best period
            ax.vlines(1./best_period,0,1,linestyle='dashed',label='Dominant Period ({0:.3f} day
            ax.legend(loc='center left',bbox_to_anchor=(1, 0.5))

        return(maxp)

    # Function to calculate the false alarm probability of a radial velocity detection
    def FAP(self,numIterations):
        # Calculate stats of errors on RV measurements
        meanErr = np.mean(self.errors)
        stddevErr = np.std(self.errors)
        length = len(self.errors)

        # Empty list of maximum powers
        maxPowers = []
        numExceed = 0

        # Set number of iterations for loop
        numIterations = 10000

        # Calculate max power from non-noisy data
        original_maxPower = self.LS(35,45,1000,1,flux=self.fluxes)

        # Monte Carlo simulation of 10000 noise profiles
        # Used to calculate False Alarm Probability (FAP)
        for i in range(numIterations):

            # Generate Gaussian noise to add to RV measurements
            noise = np.random.normal(meanErr,stddevErr,length)

            # Add noise to RV measurements
            #newRVs = np.add(self.fluxes,noise)

            # Calculate maximum power in the Lomb-Scargle periodogram
            maxPower = self.LS(35,45,1000,1,flux=noise)
            maxPowers.append(maxPower)
            # Check if maximum power exceeds that of period in non-noisy data
            if maxPower > original_maxPower:
                numExceed += 1

        # Calculate FAP
        FAP = numExceed/numIterations
        print('FAP = {0:.3e}'.format(FAP))
        return(FAP)
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  5 12:51:52 2020

@author: jimmy
"""
# Import relevant modules/packages
import numpy as np
from astropy import units as u
from astropy import constants as const
import matplotlib.pyplot as plt

# Function to make a list of a descending geometric series
def DescendingGeometric(length):

    # Make list of coefficients that are all 1
    c = np.ones(length)

    # Multiply each component by another factor of 1/2
    for i in range(1,len(c)):
        c[i] *= .5/i

    return(c)

# Function to numerically solve differentiable equation
# Resource that helped me: https://www.math.ubc.ca/~pwalls/math-python/roots-optimization/newtor
def NewtonRaphson(f,df,x0,precision,numSteps):
    # Inputs:
    #    f: function to evaluate
    #    df: derivative of function
    #    x0: initial guess at solution
    #    precision: answer won't exactly be 0, so set a tolerance
    #    numSteps: maximum number of times to iterate

    # Establish first guess at solution
    x = x0

    # Iterate over number of steps
    for i in range(0,numSteps):

        # Evaluate function
        func = f(x)

        # If f(x) is within precision, declare that value of x as the solution
        if abs(func) <= precision:
            #print('A solution of {0:.2e} was found in {1} iterations'.format(x,i))
            break

        # If f(x) is not within precision, continue searching for solution
        elif abs(func) > precision:

            # Evaluate derivative
            deriv = df(x)

            # Adjust guess of solution by subtracting quotient of function and derivative from t
            x -= func/deriv

    return(x)

# Function to compute Chi Squared and reduced Chi Squared to compare models to obserations
def ChiSquared(model,observation,error,free):
    # Inputs:
    #     model = list of values from model
    #     observation = list of values from actual observations
    #     error = list of errors (sigma) for each observation
    #     free = number of free parameters in the model
    # Returns:
    #     Chi Squared and reduced Chi squared to indicate goodness of fit for the model

    # Initialize Chi Squared as 0
    ChiSq = 0.0

    # Calculate number of degrees of freedom (# of data points - free)
    nu = len(model) - free

    # For each data point:
    for i in range(len(model)):
        # Calculate the difference between the obsrevation and model (residual)
        residual = observation[i] - model[i]

        # Calculate square of quotient of residual and error value for particular data point
        term = (residual/error[i])**2

        # Add this term to the overall Chi Squared value
        ChiSq += term

    # Calculate reduced Chi Squared (just Chi Squared / # of DoF)
    RedChiSq = ChiSq/nu

    return(ChiSq,RedChiSq,nu)

# Function to calculate Gaussian
def Gaussian(x,offset,amplitude,mean,stddev,wavelength=5000.0):
    # Inputs:
    #   x: point at which to calculate Gaussian (can be a list of values)
    #   offset: set continuum level of Gaussian
    #   amplitude: peak depth of function
    #   mean: center of Gaussian
    #   stddev: width of Gaussian
    #   wavelength: reference wavelength for spectrum

    # Returns:
    #   Value of Gaussian function at x

    # Define exponent
    exponent = (-1.0*(x-mean-wavelength)**2)/(2*stddev)

    # Calculate function value
    function = offset-(amplitude*np.exp(exponent))

    return(function)

# Function to calculate non-relativistic doppler shift
def NonRelDoppler(new_value,rest=5000.0):

    # Convert speed of light to km/s
    c = const.c.to(u.km/u.s).value

    # Calculate new velocity
    velocity = ((new_value/rest)-1)*c

    return(velocity)

"""wavelength = 5000.0
offset = 1.0
P = 0.5
mean = 0.0
sigma = 1.0
sample = 0.5

# Generate list of wavelengths to draw spectrum over
x = wavelength+np.arange(mean-5.0,mean+5.0+sample,sample)
y = Gaussian(x,offset,P,mean,sigma)
```

```
plt.scatter(x,y)"""
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 15:58:58 2020

@author: jimmy
"""
import numpy as np
import matplotlib.pyplot as plt
from MathTools import Gaussian, NonRelDoppler
from scipy.optimize import curve_fit
from astropy import units as u

class SpectralFeatures():

    # Initialize input parameters
    def __init__(self,wavelength,offset,P,mean,sigma,sampleInterval,SNR):
        # Inputs:
        #   P: peak depth of spectrum
        #   sigma: width of spectrum (in Angstroms)
        #   mean: center of spectrum (in Angstroms)
        #   sampleInterval: spacing of pixels (in Angstroms)
        #   SNR: signal-to-noise ratio you want to generate

        self.SNR = SNR
        self.wavelength = wavelength
        self.offset = offset
        self.P = P
        self.mean = mean
        self.sigma = sigma
        self.sample = sampleInterval

    def GaussianNoise(self,plot=True):
        # Returns:
        #   Plot of Gaussian spectrum with user-defined properties

        # Generate list of wavelengths to draw spectrum over
        x = self.wavelength+np.arange(self.mean-5.0,self.mean+5.0+self.sample,self.sample)

        # Calculate value of Gaussian function at each x
        function = Gaussian(x,self.offset,self.P,self.mean,self.sigma)

        # Calculate photometric precision from SNR (SNR=1/sigma_prec)
        sigma_prec = 1.0/self.SNR

        # Generate Gaussian noise depending on SNR
        noise = np.random.normal(0,sigma_prec,len(x))

        # Add Gaussian noise to Gaussian function values
        noiseData = np.add(function,noise)

        if plot == True:
            # Plot noisy data
            plt.scatter(x,noiseData)
            plt.xlabel(r'Wavelength ($\AA$)',fontsize=14)
            plt.title('Gaussian Spectrum with SNR={0}:1'.format(self.SNR),fontsize=18)
            plt.ylim(0,1.4)
            plt.tight_layout()

        return(x,noiseData)

    # Function to fit a Gaussian model to Gaussian with noise
    def GaussianModel(self,free=4,plot=False):
        # Inputs:
        #   init_val: array of initial guesses for free parameters
        #   free: number of free parameters (4=all,1=mean only free param.)
        #   plot: choose to plot the model vs. data or not
        # Returns:
        #   best-fit parameters to model noisy Gaussian

        # Set 'free' as global parameter for later use
        self.free = free

        # Extract wavelength and Gaussian values from GaussianNoise
        x, y = self.GaussianNoise(plot=False)

        # Use curve_fit to find best parameters to fit model
        self.guesses = [0.5,0.25,0.2,0.5]

        # Fit different functions for different # of free parameters
        if free == 4:

            #  Use curve_fit to find best-fit parameters and their covariances
            best_vals, covar = curve_fit(Gaussian, x, y, p0=self.guesses)
            #print('best_vals: {}'.format(best_vals))

            # Calculate y values of model
            y_model = Gaussian(x,best_vals[0],best_vals[1],best_vals[2],best_vals[3])

            # Extract center of Gaussian from best-fit model parameters
            mean_model = best_vals[2]
            errors = np.abs(np.diag(covar))
            accuracy_model = np.sqrt(errors[2])

        elif free == 1: # fix all Gaussiain parameters but the mean
            custom_Gaussian = lambda x, mean: Gaussian(x,self.offset,self.P,mean,self.sigma)
            best_vals, covar = curve_fit(custom_Gaussian, x, y, p0=self.guesses[2])
            #print('best_vals: {}'.format(best_vals))

            # Calculate y values of model
            y_model = Gaussian(x,self.offset,self.P,best_vals[0],self.sigma)

            # Extract center of Gaussian from best-fit model parameters
            mean_model = best_vals[0]
            accuracy_model = np.sqrt(np.diag(covar))

        # Plot data vs. model
        if plot == True:
            # Plot data vs. model
            plt.scatter(x,y,label='Data',color='red')
            plt.plot(x,y_model,label='Model',color='blue')
            plt.legend()
            plt.xlabel(r'Wavelength ($\AA$)',fontsize=14)
            plt.title('Gaussian Spectrum with SNR={0}:1'.format(self.SNR),fontsize=18)
            plt.ylim(0,1.4)
            plt.tight_layout()

        # Calculate relative accuracy
        #rel_acc = 1-np.abs(mean_model)/self.wavelength*100
        return(accuracy_model)

# Function to calculate the accuracy of Gaussian model at partic. SNR
def Accuracy(free,signals,units='angs'):
    # Inputs:
    #   signals: array of SNRs
    #   free: number of free parameters (4=all,1=mean only free param.)
    # Returns:
    #   plot of accuracy of model vs. SNR

    # Empty list of accuracies and SNRs that give realistic errors
    accuracies = []
    best_signals = []

    i,j = 0,0
```

```
# Loop over many SNR cases to find relationship between accuracy vs. SNR
for value in signals:
    j += 1
    # Create instance of class for particular SNR
    SNR_instance = SpectralFeatures(5000.0,1.0,0.5,0.0,1.0,0.5,value)

    # Calculate accuracy of model for particular SNR and add to list
    accuracy = SNR_instance.GaussianModel(free,plot=False)

    if accuracy <= 1.0:
        best_signals.append(value)
        accuracies.append(accuracy)
    else: # count number of outliers that give very high errors
        i += 1

# Tell user how many trials had excessive errors
print('{0} out of {1} trials had accuracy > 1'.format(i,j))

# Convert errors to km/s if requested
if units=='km/s':
    # Add errors to center wavelength and convert to km/s
    accuracies = [NonRelDoppler(x+5000.0) for x in accuracies]

    # Set units for y label
    unit_string = r'$\frac{km}{s}$'
else: # if units='angs'(default)
    # Set units for y label
    unit_string = r'$\AA$'

# Plot accuracy vs SNR.
plt.scatter(best_signals,accuracies,label='Median Error = {0:.2f} '.format(np.median(accurac
plt.xlabel('Signal-to-Noise Ratio',fontsize=14)
plt.ylabel(r'Error ({0})'.format(unit_string),fontsize=14)
plt.title('Accuracy of Model vs. SNR \n (Free parameters: {0})'.format(free),fontsize=18)
plt.legend()
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  2 12:55:40 2020

@author: jimmy
"""
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from ReadFile import ReadNASA
from astropy import units as u
from astropy import constants as const
from matplotlib.patches import Rectangle

# Class to plot grazing transits and compare to real data
class GrazingTransit:

    def __init__(self,star_type):

        # Set star type as global variable
        self.star_type = star_type

        # Define stellar host parameters
        if star_type == 'M':
            self.multiple = .559 # fractional size of Sun for M0V star
            self.star_temp = 3870.0
            self.host = 'M0V'
        elif star_type == 'K':
            self.multiple = 10.0 # fractional size of Sun for K0III star
            self.star_temp = 4810.0
            self.host = 'K0III'

        # Calculate radius of star in au
        self.rad_star = self.multiple*const.R_sun.to(u.au).value

        # Define min and max liquid water temperatures in K
        # Source (slide 3):https://www.astro.umd.edu/~miller/teaching/astr380f09/slides14.pdf
        self.TempWaterMin = 274.15
        self.TempWaterMax = 303.15

    # Function to calculate inclinations of minimally observable transits
    def Graze(self,rad_pl,a):
        # acos(i) = R_pl + R_star
        # Inputs:
        #   rad_pl: radius of planet (in au)
        #   rad_star: radius of host star (in au)
        #   a: arbitray semi-major axis
        # Returns:
        #   i: inclination

        # Maximum acos(i) value to observe transit
        threshold = rad_pl+self.rad_star

        # Calculate inclination if acos(i) = threshold (grazing condition)
        i = np.degrees(np.arccos(threshold/a))

        return(i)

    # Function to calculate habitable zone around star
    def HabitableZone(self):
        # Returns: min and max habitable zone distances (liquid water temps)

        # Calculate minimum and maximum habitable zone distances
        HZMax = self.rad_star/2*np.sqrt(1-0.3)*((self.star_temp/self.TempWaterMin)**2)
        HZMin = self.rad_star/2*np.sqrt(1-0.3)*((self.star_temp/self.TempWaterMax)**2)
        return(HZMin,HZMax)

    # Function to plot lines of minimally detectable transits
    def InclinationSemiMajor(self):

        # Define radii of Earth and Jupiter in au
        EarthRad = const.R_earth.to(u.au).value
        JupRad = const.R_jup.to(u.au).value
        increment = JupRad/EarthRad/5

        # Make array of planet radii from Earth's to Jupiter's
        PlRad = np.linspace(EarthRad,JupRad,5)

        # Make array of semi-major axes
        a = np.linspace(.01,10,2000) # in au

        # Generate array of inclinations
        inclinations = [self.Graze(PlRad[i],a) for i in range(len(PlRad))]
        inclinations = np.asarray(inclinations)

        # Create figure and axis object
        plt.figure(figsize=(10,6))
        ax = plt.subplot(111)

        # Plot grazing transit lines for each planet (with different linestyles)
        linestyles = ['-',':','--',':','-']
        for i in range(len(PlRad)):
            if i == 0:
                phrase = 'Earth-size'
            elif i == len(PlRad)-1:
                phrase = 'Jupiter-size'
            else:
                phrase = str(np.round(increment*i,2))+r'$R_{\oplus}$'
            ax.plot(a,inclinations[i],label=phrase,linestyle=linestyles[i])
        ax.set_xscale('log')

        # Read in data from NASA Exoplanet Archive
        ExoplanetData = ReadNASA('NASAExo.csv',skip=76)

        # Identifty locations of data points for each discovery method
        detections = np.where(ExoplanetData['pl_discmethod'] == 'Transit')

        # Define x (mass) and y (semi-major axis) datasets to plot
        x = ExoplanetData['pl_orbsmax'][detections]
        y = ExoplanetData['pl_orbincl'][detections]

        # Plot NASA Data
        ax.scatter(x,y,label='NASA Transits')
        ax.set_xlabel(r'log(a) (au)',fontsize=14)
        ax.set_ylabel('Inclination (deg)',fontsize=14)
        ax.set_title('Orbital Inclination vs. Semi-Major Axis \n (Host: {0})'.format(self.host),

        # Get limits of axes for adding plots
        ymin,ymax = ax.get_ylim()

        # Shade habitable zone
        minD, maxD = self.HabitableZone()
        ax.vlines(minD,ymin,ymax,label=r'$r_{hab,min} = %.2f au$'%minD,linestyle='--',color='bla
        ax.vlines(maxD,ymin,ymax,label=r'$r_{hab,max} = %.2f au$'%maxD,linestyle='--',color='bla
        ax.add_patch(Rectangle((minD,ymin),maxD-minD,ymax-ymin,fill=True,color='r',alpha=0.5))
        ax.legend(loc='center left',bbox_to_anchor=(1, 0.5))
        plt.tight_layout()
"""
# Lines to test function (comment out when not testing)
test = GrazingTransit('K')
test.InclinationSemiMajor()
"""
```

```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
        # 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))
```

```python
# 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)
"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 21 17:32:14 2020

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

# Function to overplot transit light curves for different b's and star temps
def LightCurveCompare(rad_planet=0.05,b_values=[0.0,0.5,0.9],temperatures=['3600','5500','10000'
    # Inputs:
    #   b_values: impact parameters of data you simulated
    #   temperatures: temperatures of stars you simulated
    # Establish figure and axes objects
    fig, (ax1,ax2,ax3) = plt.subplots(ncols=len(b_values), figsize=(15,5))

    # Set up empty lists of x and y data to plot
    xdata = []
    ydata = []

    # Loop over all impact parameter and star temp combinations
    for i in range(len(b_values)):
        for j in range(len(b_values)):
            # Define filename to extract data from
            filename = 'C:/Users/Jimmy/ASTR5490/HW3/TransitData/Transit_0.05Rstar_b={0}_{1}K.dat

            # Extract data from file and assign to x and intens
            data = np.loadtxt(filename,skiprows=1,unpack=True)
            x, intens = data[-2],data[-1]

            # Add data lists to empty list (makes a list of lists)
            xdata.append(x)
            ydata.append(intens)

    # Very inelegantly plot all data combinations in row of subplots
    # Loop over b=0 data
    for i in range(3):
        ax1.scatter(xdata[i],ydata[i],label='T={0}K'.format(temperatures[i]))
        j += 1
    ax1.set_ylim(0.9965,1)
    ax1.legend()
    ax1.set_xlabel(r'Horizontal Distance from Star Center ($R_{star}$)')
    ax1.set_ylabel('Relative Intensity')
    ax1.set_title('Transit of {0}'.format(rad_planet)+r'$R_{star}$ Planet'\
              +'\n'r'(b = {0})'.format(b_values[0]))
    # Loop over b=0 data
    for i in range(3,6):
        ax2.scatter(xdata[i],ydata[i],label='T={0}K'.format(temperatures[i-3]))
    ax2.set_ylim(0.9965,1)
    ax2.legend()
    ax2.set_xlabel(r'Horizontal Distance from Star Center ($R_{star}$)')
    ax2.set_ylabel('Relative Intensity')
    ax2.set_title('Transit of {0}'.format(rad_planet)+r'$R_{star}$ Planet'\
              +'\n'r'(b = {0})'.format(b_values[1]))
    # Loop over b=0 data
    for i in range(6,9):
        ax3.scatter(xdata[i],ydata[i],label='T={0}K'.format(temperatures[i-6]))
    ax3.set_ylim(0.9965,1)
    ax3.legend()
    ax3.set_xlabel(r'Horizontal Distance from Star Center ($R_{star}$)')
    ax3.set_ylabel('Relative Intensity')
    ax3.set_title('Transit of {0}'.format(rad_planet)+r'$R_{star}$ Planet'\
              +'\n'r'(b = {0})'.format(b_values[2]))

    # Give the plots some room to breathe
```

```
plt.tight_layout()
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 22 15:34:25 2020

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

def BatmanModel(t0=0.0,P=1.0,rad_pl=0.1,a=15.0,i=87.0,e=0.0,w=90.0,coeff=[0.5,0.1],plot=False):

    # Initialize the model
    params = batman.TransitParams()       #object to store transit parameters
    params.t0 = t0 #76.6764                    #time of inferior conjunction
    params.per = P #13.17562                       #orbital period
    params.rp = rad_pl #0.01602                       #planet radius (in units of stellar radii)
    params.a = a #13.8                         #semi-major axis (in units of stellar radii)
    params.inc = i #88.19                        #orbital inclination (in degrees)
    params.ecc = e #0.0                        #eccentricity
    params.w = w #90.                         #longitude of periastron (in degrees)
    params.limb_dark = "quadratic"        #limb darkening model
    params.u = coeff #[0.4899, 0.1809]     #limb darkening coefficients [u1, u2]

    t = np.linspace(-.25, .25, 1000)  #times at which to calculate light curve
    t_new = np.interp(t, (t.min(), t.max()), (0, 0.2))
    m = batman.TransitModel(params, t)    #initializes model

    flux = m.light_curve(params)                    #calculates light curve

    if plot==True:
        fig,ax = plt.subplots()
        ax.get_yaxis().get_major_formatter().set_useOffset(False)
        ax.plot(t_new,flux)

    return (t_new,flux)
#BatmanModel()
```