PlanetsHW4_JimmyLilly

November 26, 2020

1 ASTR 5490 Homework 4 (Jimmy Lilly 11/26/20)

[3]:
```
# 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 Blackbody import SED
from scipy.integrate import quad
from MathTools import EquilTemp, MagDiff
from MaxwellBoltzmann import MaxwellBoltzmann
from ReadFile import ReadBandpass
import bisect

# Reload scripts I may have changed
%load_ext autoreload
%autoreload 2
```

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

2 1) Create a blackbody spectral energy distribution for a star like the sun with $T_{eff} = 5780K$ and $1R_{\odot}$. Include wavelengths between the X-ray (1 angs) and far infrared (100 μm) [$10^{-10}m$ to $ 10^{-4}m$]

##

$B_\lambda(T) = \frac{2hc^2}{\lambda^5}\frac{1}{\left(e^{\frac{hc}{\lambda kT}}-1\right)} \Bigg| B_\nu(T) = \frac{2h\nu^3}{c^2}\frac{1}{\left(e^{\frac{h\nu}{kT}}-1\right)}$

2.1 1a) Plot this SED. Integrate it over wavelength and multiply by the surface area of the sun $(4\pi R_{\odot}^2)$ and another factor of π (the angular integral over azimuthal and polar angle) and verify that you recover the luminosity of the sun: $4\pi R^2\sigma T^4 = 2\cdot 10^{33}\frac{erg}{s}$

##

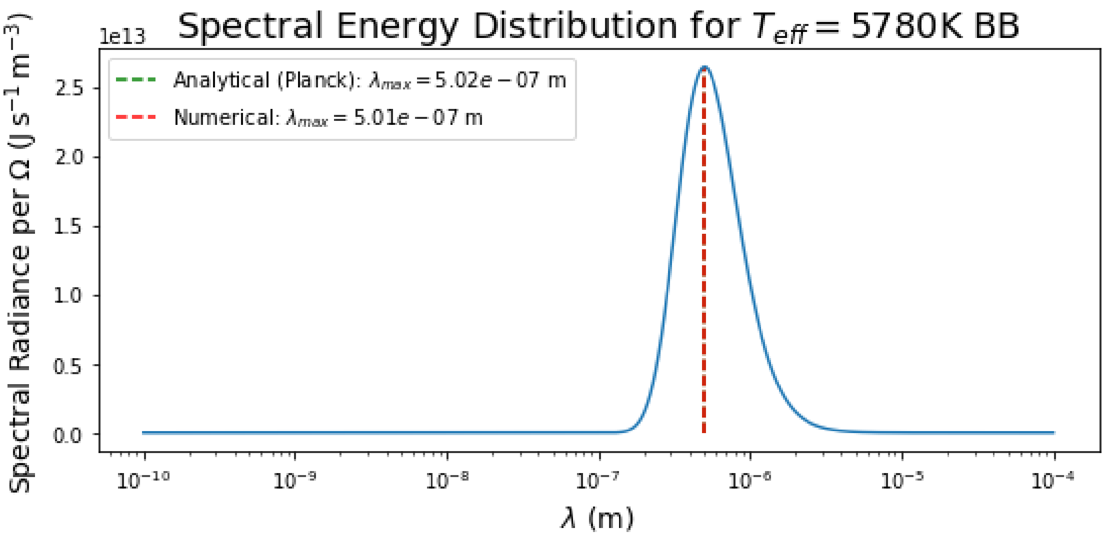
$L_{\nu,\lambda} = 4\pi^2 R_*^2 B_{\nu,\lambda}$

2.2 1b) What fraction of the energy is emitted shorter than the peak versus longer than the peak?

[64]:
```
Sun_like = SED('wavelen','planck')
luminosity_1a = Sun_like.SEDStar(True)
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
Luminosity = 1.006 Lsun
24.91% of energy emitted below peak
75.03% of energy emitted above peak

Program took 8.00 sec to run
```

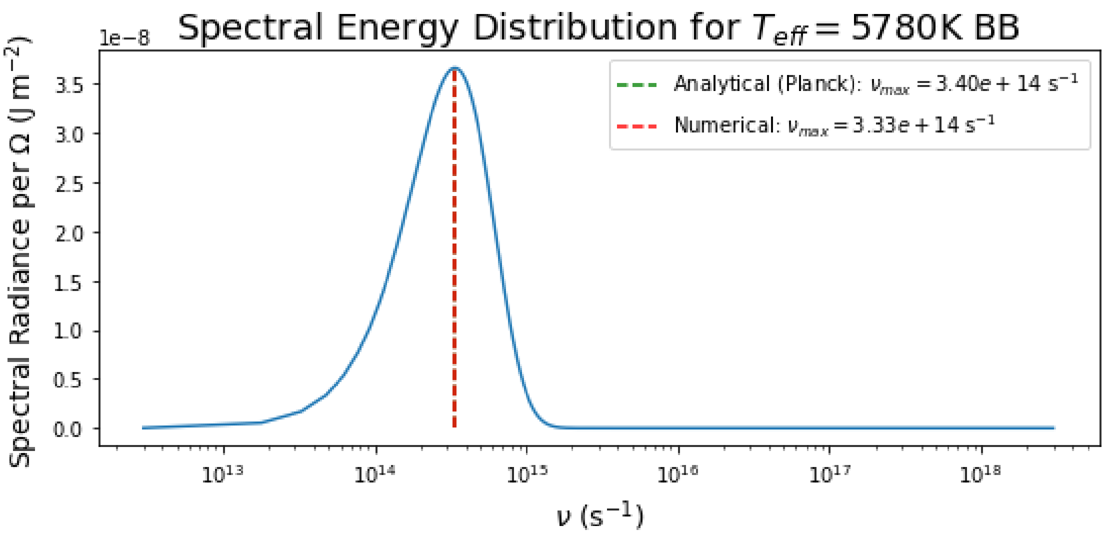


[126]:
```
Sun_like_freq = SED('freq','planck',N=2*10**5)
luminosity_1a_2 = Sun_like_freq.SEDStar(True)
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
Lstar = 1.006 Lsun
Luminosity = 1.006 Lsun
31.41% of energy emitted below peak
65.88% of energy emitted above peak
```

```
Program took 7.92 sec (0.132 min) to run
```

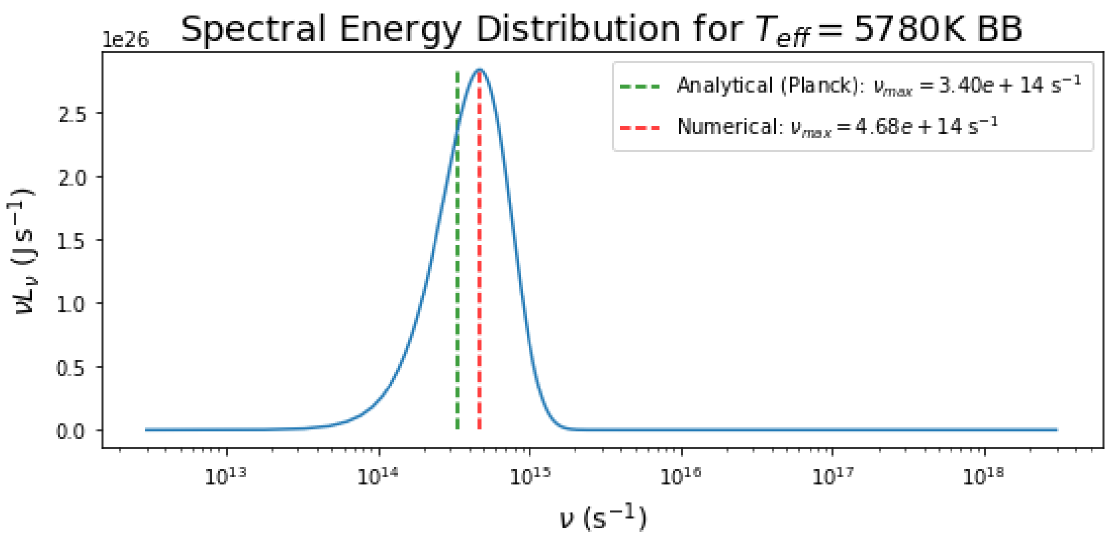


2.2.1 The units on the y-axis of my frequency Planck function are correct, the s and Hz in the denominatior just cancel out in the calculation with astropy units. Also, my luminosity calculations are correct, the percentages just don't add to 100% because I'm not integrating over an infinite domain.

2.3 1c) Spectral energy distributions in astronomy are often plotted as νL_ν on the y-axis versus ν on the x axis because the total energy scales as ν (i.e., $E = h\nu$). Make such an SED and compare it to L_ν on the y-axis versus ν. What changes?

[66]:
```python
# Plot nu*L_nu vs. nu
plot_a_1c = SED('freq','xvar_luminos')
xdata_a,ydata_a = plot_a_1c.SEDStar(True)
```

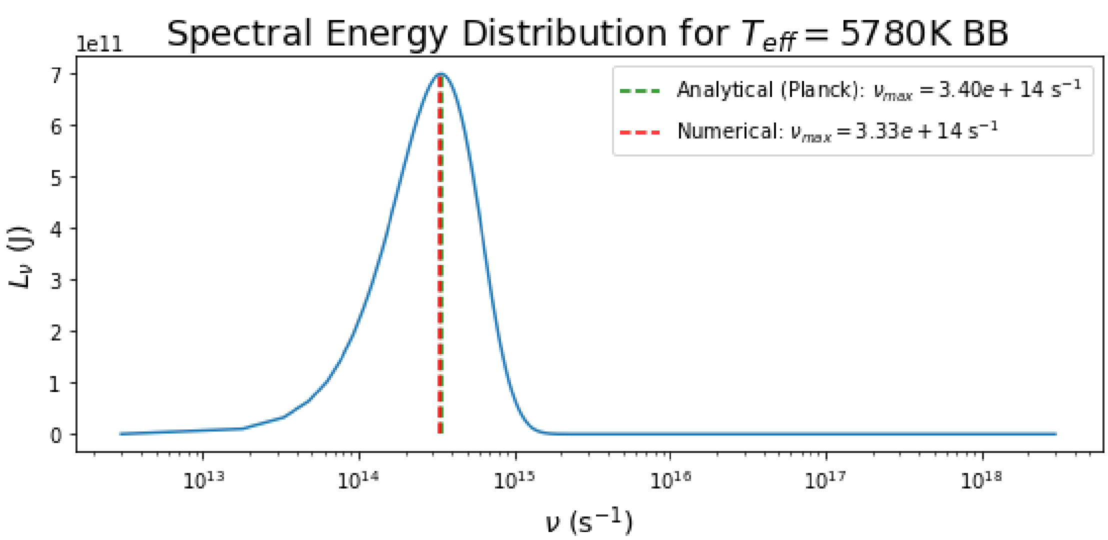
```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)

Program took 8.08 sec to run
```




```
[67]: # Plot L_nu vs. nu
plot_b_1c = SED('freq','luminosity')
xdata_b,ydata_b = plot_b_1c.SEDStar(True)
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
Program took 7.91 sec to run
```

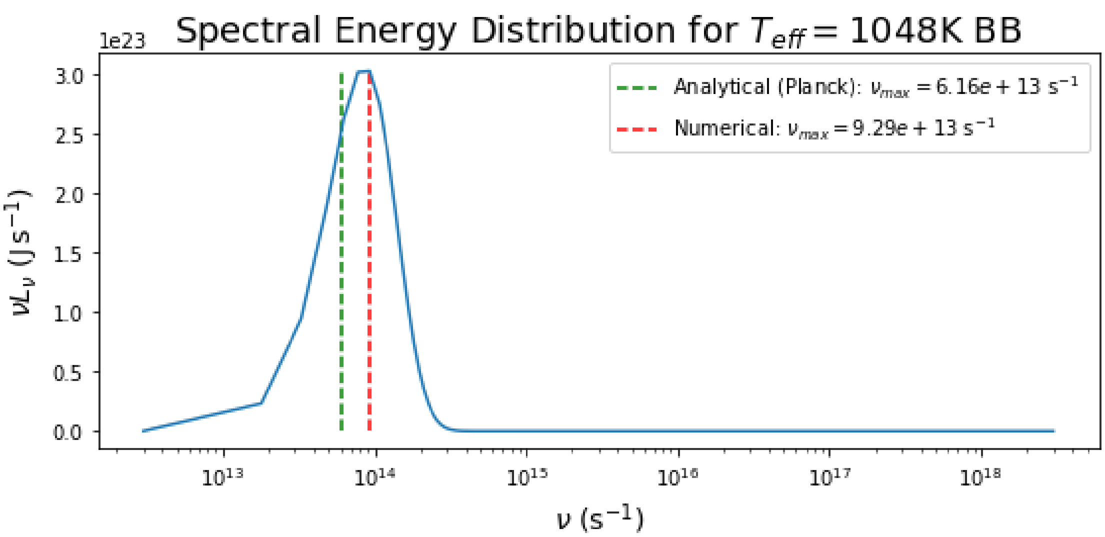


2.3.1 The plot of νL_ν is narrower arround the peak frequency than the L_ν plot. Also, the νL_ν plot peaks further from the peak Planck frequency than the plot of just L_ν does.

2.4 1d) Create an SED that consists of two components: a sun-like blackbody and a hot super-Jupiter with radius $R = 3R_{Jup}$ at 0.05 au from the star. Plot each component separately. Plot the contrast ratio, the ratio of L_λ (or L_ν) versus wavelength (or frequency) to illustrate where this contrast ratio peaks.

[68]:
```
# Calculate temperature of hot super-Jupiter (HSJ)
T_HSJ = EquilTemp(0.5,1.0*const.R_sun,0.05*u.au,5780)
HSJ_SED = SED('freq','xvar_luminos',Teff=T_HSJ)
x_HSJ, y_HSJ = HSJ_SED.SEDStar(True)
```

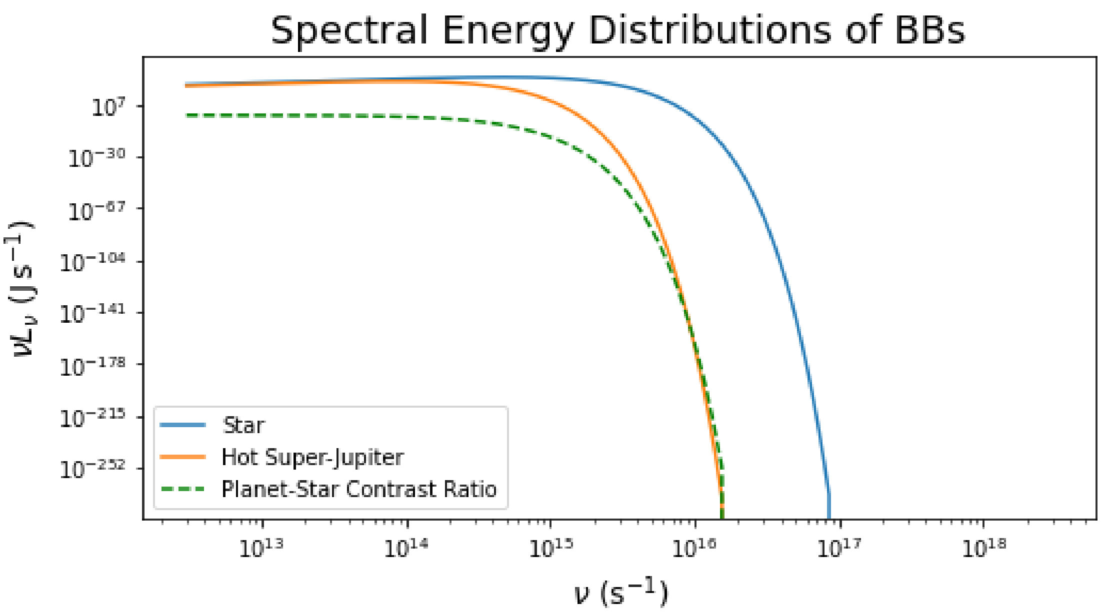
```
Body Temperature = 1048.14 K

/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)

Program took 7.91 sec to run
```



[69]:
```
# Plotting SED components separately
plt.figure(figsize=(8,4))
plt.plot(xdata_a,ydata_a,label='Star') # plot SED of Sun-like star
plt.plot(x_HSJ,y_HSJ,label='Hot Super-Jupiter') # plot SED of hot super-Jupiter
contrast_ratio = np.divide(y_HSJ,ydata_a)
plt.plot(x_HSJ,contrast_ratio,label='Planet-Star Contrast␣
↪Ratio',linestyle='dashed',color='green')
```

```
plt.xscale('log')
plt.yscale('log')
plt.xlabel(r'$\nu$ ({0:latex_inline})'.format(x_HSJ.unit),fontsize=14)
plt.ylabel(r'$\nu L_{{\nu}}$ ({0:latex_inline})'.format(y_HSJ.unit),fontsize=14)
plt.title('Spectral Energy Distributions of BBs',fontsize=18)
plt.legend()
```

[69]: <matplotlib.legend.Legend at 0x7f0b136d3c18>



2.4.1 The contrast ratio peaks at shorter wavelengths near the far-infrared. This agrees with our previous discussions in this class about direct imaging needing to be done in the infrared as this is where the planet is most visible in front of the star.

2.5 1e) Fold (that is, multiply the F_ν by the efficiency at each wavelength and integrate over the bandpass) the SED of the sun (working in frequency units) through each bandpass (Kepler broad optical band and the 4.5 micron Spitzer band) to add up the energy that would be observed in each bandpass.

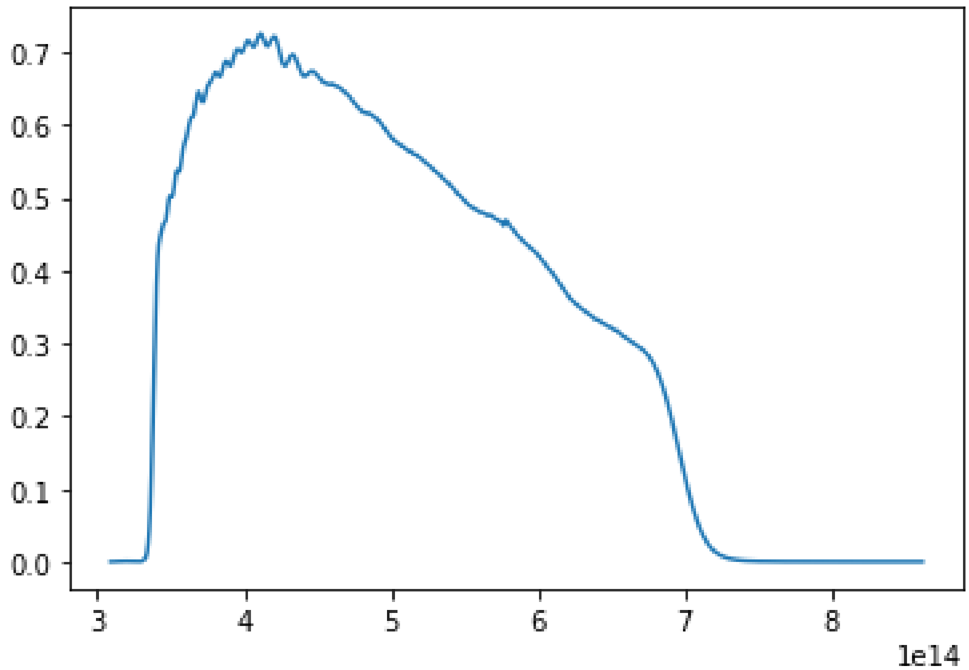
2.6 Compare the magnitude difference to the approximate V-W2 color from Pecaut & Mamajek (2013, ApJS, 208, 9) ($\Delta m = 1.584$ according to their table) for a solar type star

##

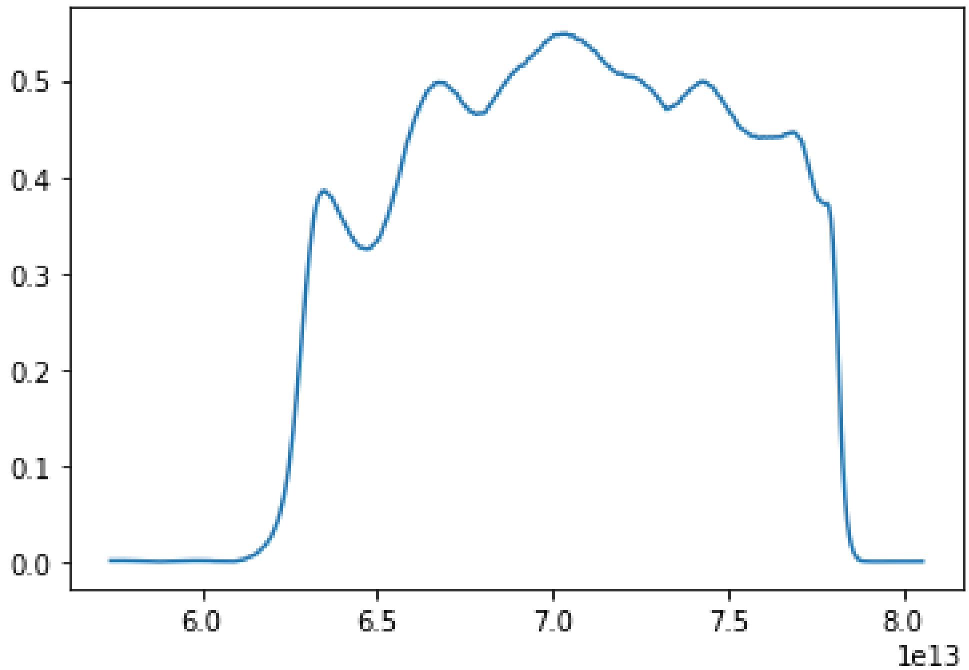
$\Delta m = -2.5log\left(\frac{F(Kp)}{F(4.5)}\right)$

2.7 Do the same for the super-Jupiter to find how many magnitudes of difference there are between the host star and planet at each wavelength.

[128]:
```
# Extract frequencies and efficiences from Kepler bandpass data
Kepler_freq, Kepler_effic = ReadBandpass('Bandpass_Kepler.dat','freq','nm')
Kepler_freq = Kepler_freq[::-1] # List elements in ascending frequency

# Calculate flux over Kepler bandpass for Sun-like star
Kepler_flux_Sun = Sun_like_freq.
 ↪ResponseFunction(Kepler_freq,Kepler_effic,plot=True)

# Calculate flux over Kepler bandpass for A0V star (Teff=9700K)
SED_A0 = SED('freq','planck',Teff=9700)
Kepler_flux_A0 = SED_A0.ResponseFunction(Kepler_freq,Kepler_effic)
```



[133]:
```
# Extract frequencies and efficiences from Spitzer 4.5um bandpass data
Spitzer_freq, Spitzer_effic = ReadBandpass('Bandpass_SpitzerI2_4.5microns.
↪dat','freq','um')
Spitzer_freq = Spitzer_freq[::-1] # List elements in ascending frequency

# Calculate flux over Spitzer bandpass for Sun-like star
Spitzer_flux_Sun = Sun_like_freq.
↪ResponseFunction(Spitzer_freq,Spitzer_effic,plot=True)

# Calculate flux over Spitzer bandpass for Sun-like star
Spitzer_flux_A0 = SED_A0.ResponseFunction(Spitzer_freq,Spitzer_effic)
```



[140]:
```
# Calculate magnitude difference considering both a G2V (Sun-like) star and an␣
↪A0V star
delta_m_Sun = MagDiff(Kepler_flux_Sun,Spitzer_flux_Sun)
delta_m_A0 = MagDiff(Kepler_flux_A0,Spitzer_flux_A0)
delta_m = delta_m_Sun-delta_m_A0
print("My calculated V-W2 color for a Sun-like star is {0:.3f}".format(delta_m))
```

```
Apparent magnitude difference = -5.176
Apparent magnitude difference = -6.335
My calculated V-W2 color for a Sun-like star is 1.159
```

2.7.1 My result for a Sun-like star is relatively close to the 1.584 from the table, but not quite. I think this can be attributed to using different reference stars to define the magnitude scale

[148]:
```
# To do this for Jupiter, need to calculate temperature of Jupiter using␣
↪radiative equilibrium
T_Jup_Sun = EquilTemp(5.2*u.au,0.34)
T_Jup_A0 = EquilTemp(5.2*u.au,0.34,Rstar=2.09*const.R_sun,Teff=9700*u.K)

# Create instance of SED class for Jupiter
SED_Jup = SED('freq','planck',Teff=T_Jup_Sun.value)
SED_Jup_A0 = SED('freq','planck',Teff=T_Jup_A0.value)

# Calculate fluxes over bandpasses for Sun-like star
```

In [148]:
```
# To do this for Jupiter, need to calculate temperature of Jupiter using radiati
T_Jup_Sun = EquilTemp(5.2*u.au,0.34)
T_Jup_A0 = EquilTemp(5.2*u.au,0.34,Rstar=2.09*const.R_sun,Teff=9700*u.K)

# Create instance of SED class for Jupiter
SED_Jup = SED('freq','planck',Teff=T_Jup_Sun.value)
SED_Jup_A0 = SED('freq','planck',Teff=T_Jup_A0.value)

# Calculate fluxes over bandpasses for Sun-like star
Kepler_flux_Jup = SED_Jup.ResponseFunction(Kepler_freq,Kepler_effic)
Spitzer_flux_Jup = SED_Jup.ResponseFunction(Spitzer_freq,Spitzer_effic)

# Calculate fluxes over bandpasses for A0V star
Kepler_flux_Jup_A0 = SED_Jup_A0.ResponseFunction(Kepler_freq,Kepler_effic)
Spitzer_flux_Jup_A0 = SED_Jup_A0.ResponseFunction(Spitzer_freq,Spitzer_effic)

# Calculate magnitude differences for Jupiter
delta_m_Jup_Sun = MagDiff(Kepler_flux_Jup,Spitzer_flux_Jup)
delta_m_Jup_A0 = MagDiff(Kepler_flux_Jup_A0,Spitzer_flux_Jup_A0)
delta_m_Jup = delta_m_Jup_Sun-delta_m_Jup_A0
print("My calculated V-W2 color for a super Jupiter is {0:.3f}".format(delta_m_J
```

```
Apparent magnitude difference = 119.215
Apparent magnitude difference = 48.046
My calculated V-W2 color for a super Jupiter is 71.169
```

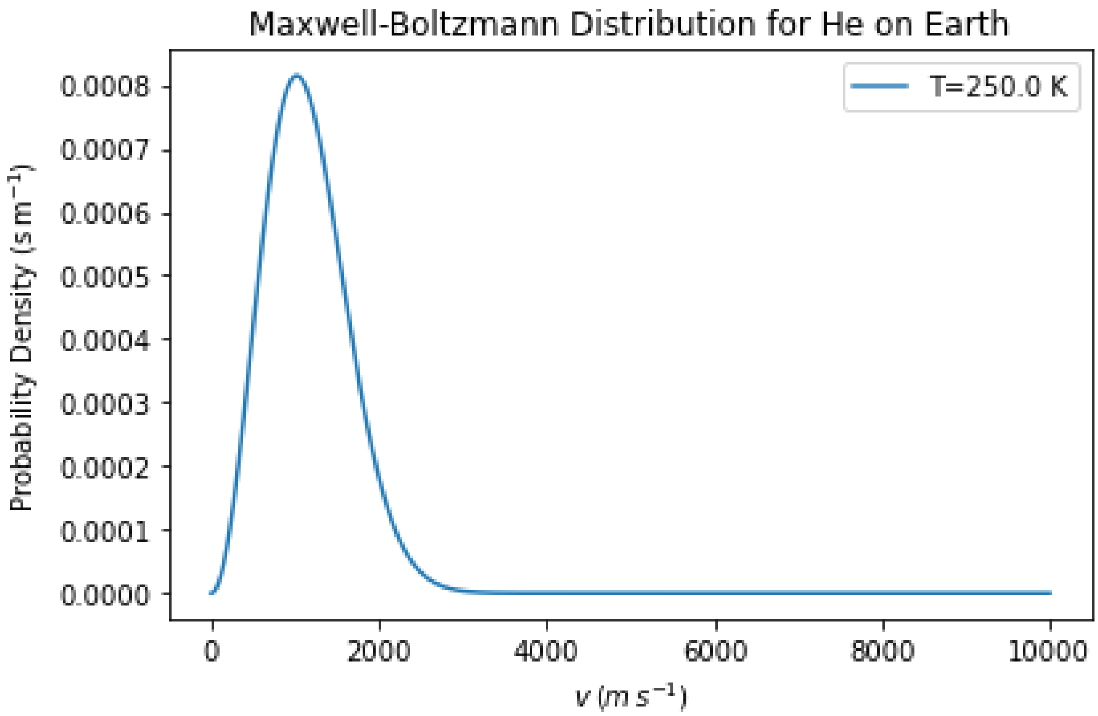
There is almost 2 orders-of-magnitude difference between the star and super Jupiter's apparent magnitudes. Requires precise observations to detect the planet!

2) Maxwell-Boltzmann Distributions

2a) Plot a Maxwell-Boltzmann distribution of speeds for He in the Earth's atmosphere. Integrate this distribution between the v_{esc} for Earth and $v = \infty$ to find what fraction of He atoms at any given time have speeds greater than escape velocity. Do the same for molecular nitrogen and compare fractions

```
Kepler_flux_Jup = SED_Jup.ResponseFunction(Kepler_freq,Kepler_effic)
Spitzer_flux_Jup = SED_Jup.ResponseFunction(Spitzer_freq,Spitzer_effic)

# Calculate fluxes over bandpasses for A0V star
Kepler_flux_Jup_A0 = SED_Jup_A0.ResponseFunction(Kepler_freq,Kepler_effic)
Spitzer_flux_Jup_A0 = SED_Jup_A0.ResponseFunction(Spitzer_freq,Spitzer_effic)

# Calculate magnitude differences for Jupiter
delta_m_Jup_Sun = MagDiff(Kepler_flux_Jup,Spitzer_flux_Jup)
delta_m_Jup_A0 = MagDiff(Kepler_flux_Jup_A0,Spitzer_flux_Jup_A0)
delta_m_Jup = delta_m_Jup_Sun-delta_m_Jup_A0
print("My calculated V-W2 color for a super Jupiter is {0:.3f}".
 ↪format(delta_m_Jup))
```

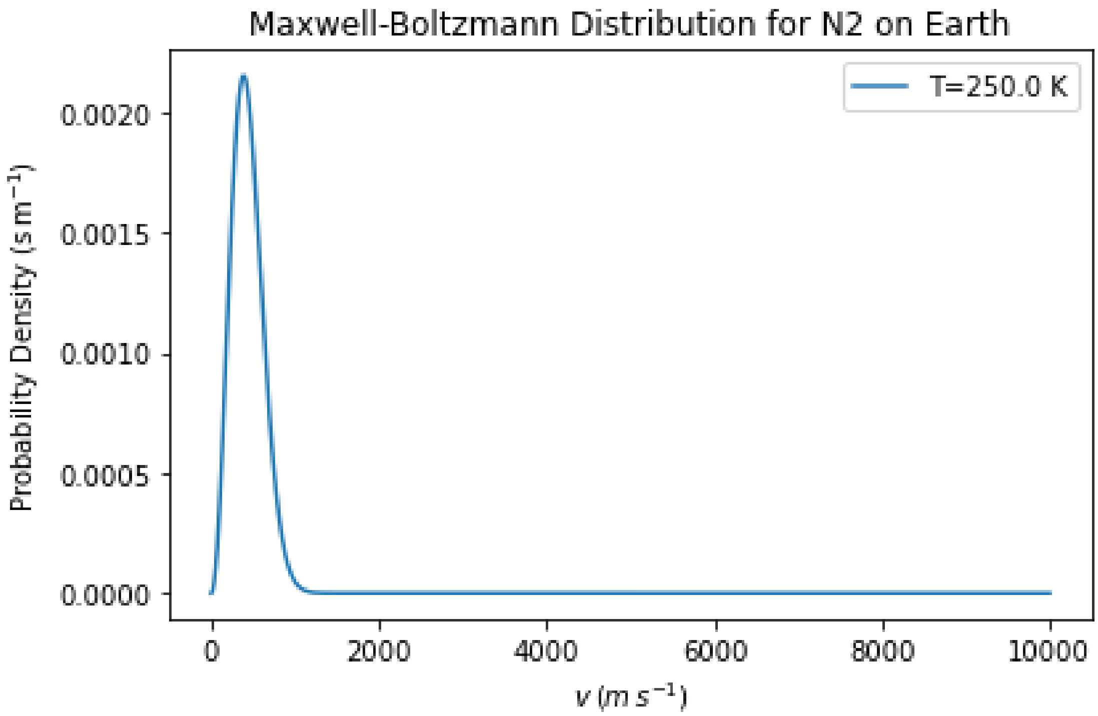
[28]:
```
# Plot MB dist. for He at T=250K on Earth
He_dist = MaxwellBoltzmann(4,250.0,'He')
speeds, He_probs = He_dist.MBDistribution('Earth')
```



3.1.1 $7.36 \cdot 10^{-50}$% of He atoms exceed escape velocity of Earth at T=250.0 K. This chance is much higher than for N_2 which makes sense since He is only a trace element in the atmosphere and is 7x less massive than N_2 so it can reach higher speeds more easily.

[22]:
```
# Plot MB dist. for N at T=250K on Earth
N_dist = MaxwellBoltzmann(28,250.0,'N2')
speeds, N_probs = N_dist.MBDistribution('Earth')
```

```
0.00e+00% of N2 atoms exceed escape velocity of Earth at T=250.0 K
Program took 0.07 sec to run
```

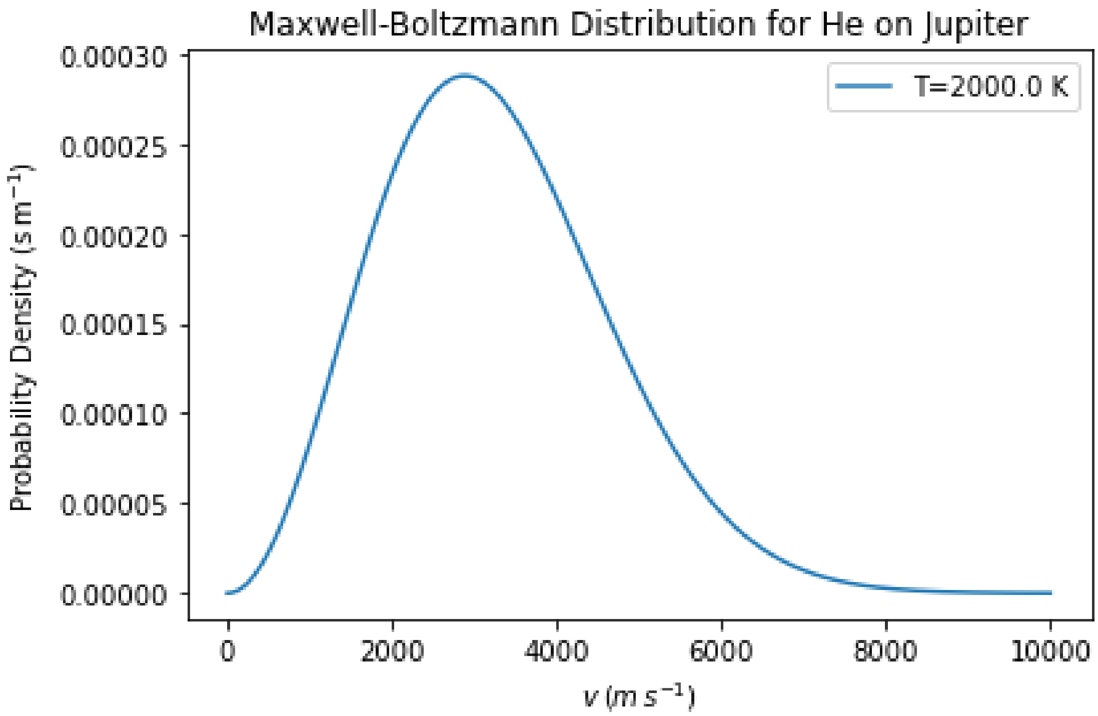


3.1.2 0% of N_2 atoms exceed escape velocity of Earth at T=250.0 K. This makes sense because most of the atmosphere is comprised of N_2 so we'd expect it's chance of escape at standard atmospheric temperatures to be extremely low

3.2 2b) Do the same thing for a hot $T = 2000K$ Jupiter-sized planet, considering both atomic He and molecular CO.

[12]:
```
# Plot MB dist. for He at T=2000K on Jupiter
He_dist_Jup = MaxwellBoltzmann(4,2000.0,'He')
speeds, He_probs_Jup = He_dist_Jup.MBDistribution('Jupiter')
```

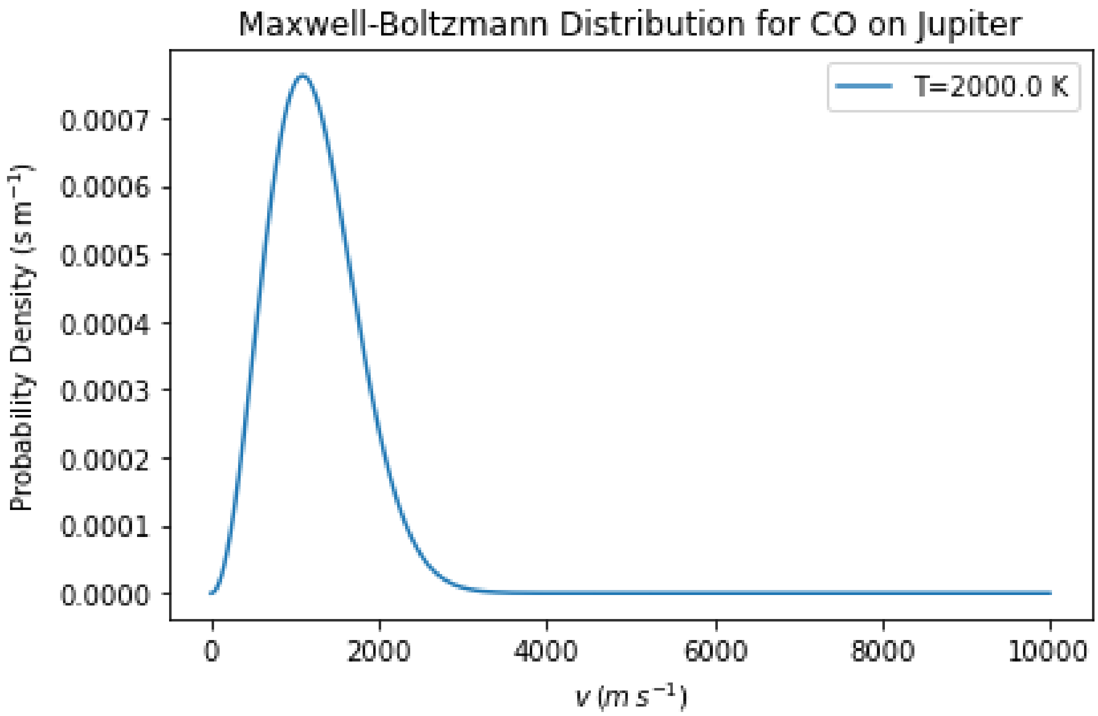
```
1.73e-182% of He atoms exceed escape velocity of Jupiter at T=2000.0 K
Program took 0.11 sec to run
```



3.2.1 $1.73 \cdot 10^{-182}\% \approx 0$ of He atoms exceed escape velocity of Jupiter at T=2000.0 K. Agrees with consensus that He is a major component of Jupiter's atmosphere

[19]:
```
# Plot MB dist. for CO at T=2000K on Jupiter
CO_dist_Jup = MaxwellBoltzmann(28,2000.0,'CO')
speeds, CO_probs_Jup = CO_dist_Jup.MBDistribution('Jupiter')
```

```
0.00e+00% of CO atoms exceed escape velocity of Jupiter at T=2000.0 K
Program took 0.08 sec to run
```



3.2.2 0% of CO atoms exceed escape velocity of Jupiter at T=2000.0 K. Presence of CO in Jupiter's atmosphere has been suggested to stem from meteoroid imacts that bring water which then gets converted to CO by various chemical processes (Prather, Logan, Mcelroy 2008 ApJ)

https://ui.adsabs.harvard.edu/abs/1978ApJ...223.1072P/abstract

4 3) Create a model of a protoplanetary debris disk system consisting of a central solar-type star and a disk of dust extending from the dust sublimation radius (where $T \sim 2000K$) to $1000au$. Break the disk into thin rings of radius dr and compute the area of each ring ($2\pi r dr$) and the temperature of each ring under the assumption of radiative equilibrium

##

$r_{sub} = \frac{R_*}{2}\sqrt{1-A}\left(\frac{T_{eff}}{T_{eq}}\right)^2$

##

$T_{ring} = \left(\frac{(1-A)R_*^2}{4r^2}\right)^{1/4} T_{eff}$

4.1 3a) Take $1M_{\oplus}$ of dust and spread it uniformly over the model protoplanetary disk (use this to calculate surface density σ_{disk} in g/cm^2).

##

$\sigma_{disk} = \frac{M_{\oplus}}{\pi R^2} = \frac{M_{\oplus}}{\pi(r_{max}-r_{sub})^2} \rightarrow [\sigma_{disk}] = \frac{g}{cm^2}$

4.2 Take the mean mass of a spherical grain to be $M_{gr} = \frac{4\pi}{3}a^3\rho_{gr}$ where the mean grain radius is $a = 0.1mm$ and the mean grain density is $\rho_{gr} = 2g/cm^3$. The grain then radiates as a blackbody with that surface area $(4\pi a^2)$. Given these grain parameters, compute a grain surface density (σ_{grain}) in number of grains per area in each ring...

##

$\eta_{grain} = \frac{\sigma_{disk}}{M_{gr}} \rightarrow [\eta_{grain}] = \frac{N_{grains}}{cm^2}$

##

$N_{grains/ring} = \eta_{grain} \cdot A_{ring}$

4.3 and add up the flux from each grain within each ring

##

$L_{\nu,ring} = \sum_{i=1}^{N_{grains/ring}} L_{\nu,i} = \sum_{i=1}^{N_{grains/ring}} \pi(4\pi a^2)B_\nu(T_{ring}) = N_{grains/ring} \cdot 4\pi^2 a^2 \cdot B_\nu(T_{ring})$

##

$L_{\nu,disk} = \sum_{i=1}^{N_{rings}} (L_{\nu,ring})_i$

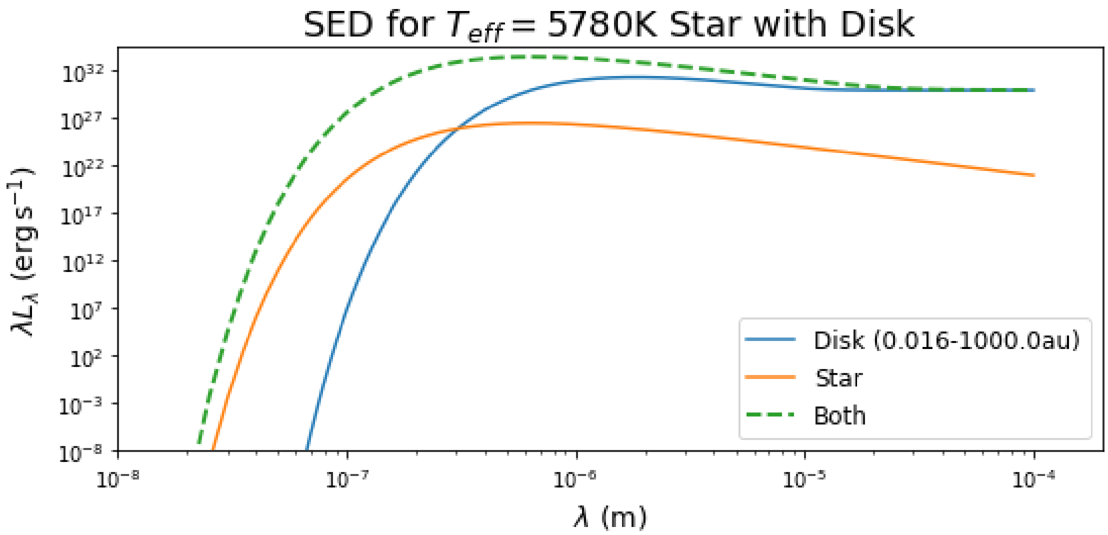
4.4 Plot the SED of the star, the disk, and the sum of the two. What is the ratio of luminosity in the disk to the luminosity in the star?

[ ]:
```
# Calculate disk-to-star luminosity ratio for 3a
SED_Disk_1000au_freq = SED('freq','xvar_luminos')
freq_ydata_3a = SED_Disk_1000au_freq.StarDiskProfile()
```

[108]:
```
# Plot disk, star, and combined SED for #3a
SED_Disk_1000au_wavelen = SED('wavelen','xvar_luminos')
SED_Disk_1000au_wavelen.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
StarDiskProfile took 10.72 sec (0.179 min) to run
```

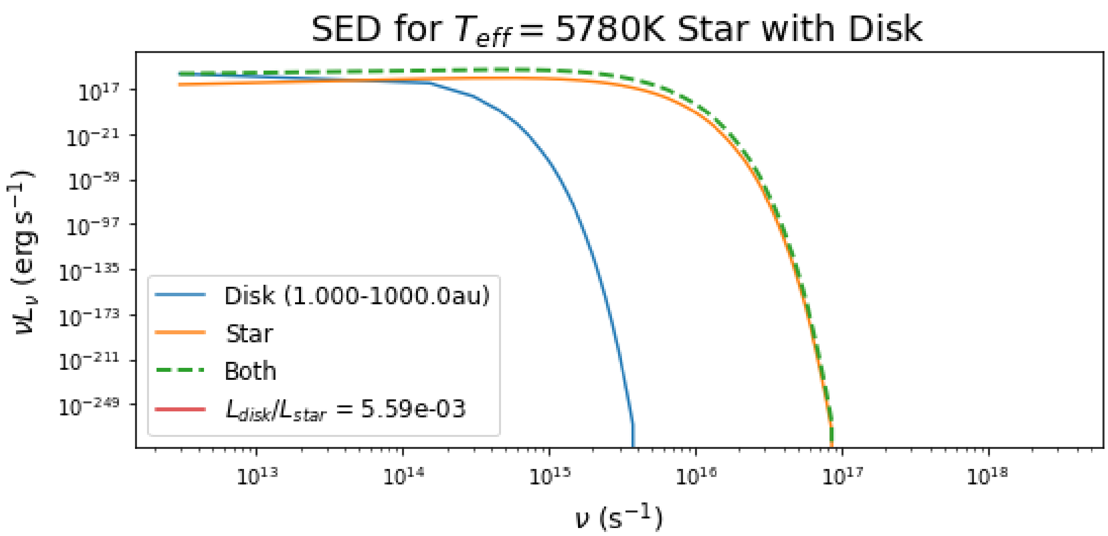


4.5 3b) Now suppose planets have cleared out all the dust inside 1 au or inside 10 au. Generate and plot the resulting spectral energy distributions and ratios $L_{disk}/L*$. Comment on how these spectral energy distributions might be used to infer the presence of gaps in the disk.

[111]:
```
# Calculate disk-to-star luminosity ratio for 3b,i
SED_Disk_clear_1au_freq = SED('freq','xvar_luminos',r_min=1.0)
freq_ydata_3b_i = SED_Disk_clear_1au_freq.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

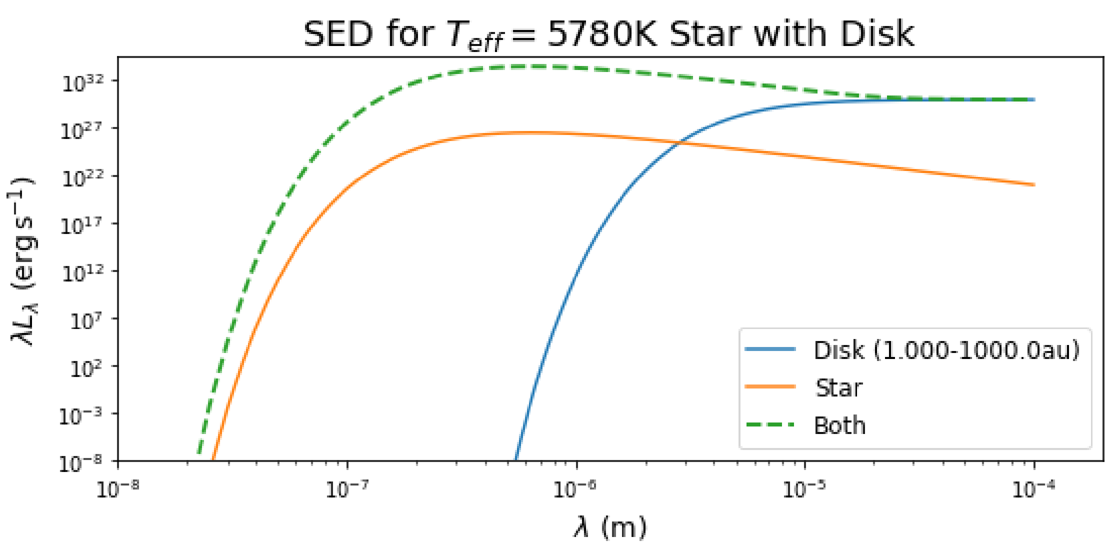
```
Lstar = 1.003 Lsun
Disk-to-Star Luminosity Ratio = 5.59e-03
StarDiskProfile took 10.32 sec (0.172 min) to run
```



[112]:
```
# Plot disk, star, and combined SED for #3b,i (dust cleared out from within 1␣
↪au)
SED_Disk_clear_1au_wavelen = SED('wavelen','xvar_luminos',r_min=1.0)
SED_Disk_clear_1au_wavelen.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

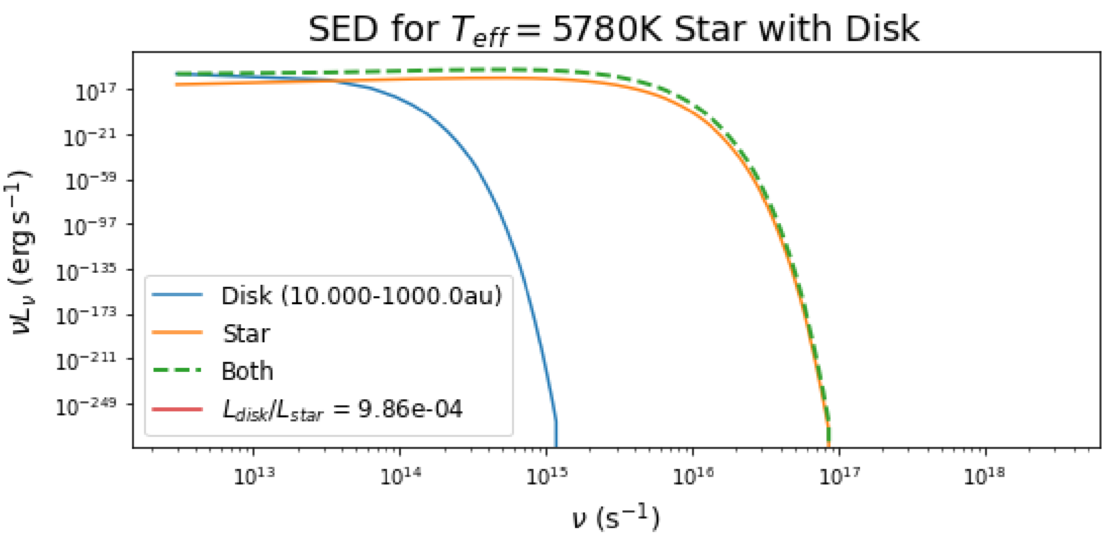
```
StarDiskProfile took 10.92 sec (0.182 min) to run
```



[114]:
```
# Calculate disk-to-star luminosity ratio for 3b,ii
SED_Disk_clear_10au_freq = SED('freq','xvar_luminos',r_min=10.0,N=10**5)
freq_ydata_3b_ii = SED_Disk_clear_10au_freq.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

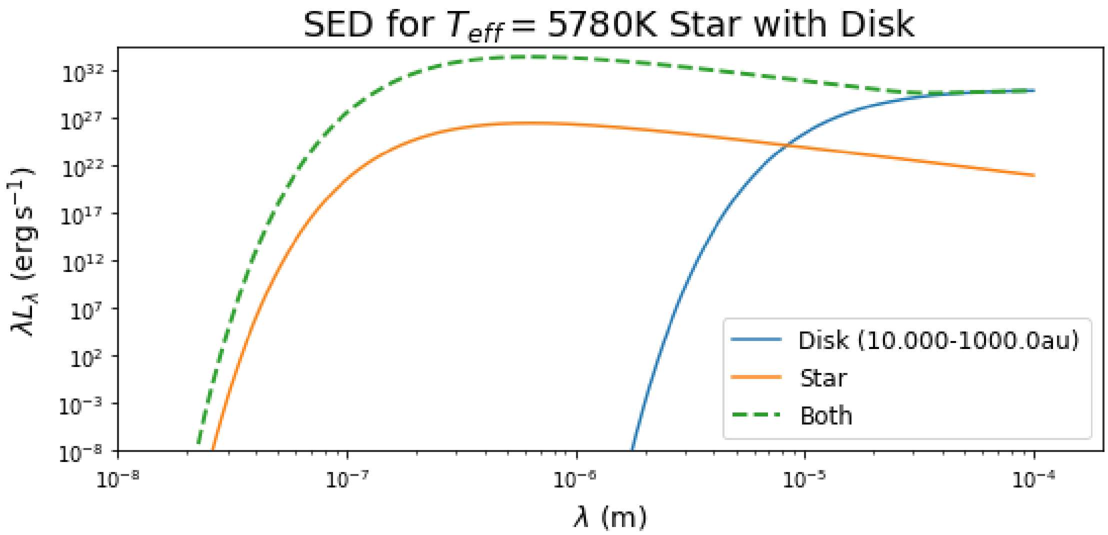
```
Lstar = 1.006 Lsun
Disk-to-Star Luminosity Ratio = 9.86e-04
StarDiskProfile took 50.29 sec (0.838 min) to run
```



[122]:
```
# Plot disk, star, and combined SED for #3b,ii (dust cleared out from within 10␣
↪au)
SED_Disk_clear_10au_wavelen = SED('wavelen','xvar_luminos',r_min=10.0,N=10**5)
SED_Disk_clear_10au_wavelen.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
StarDiskProfile took 53.25 sec (0.888 min) to run
```



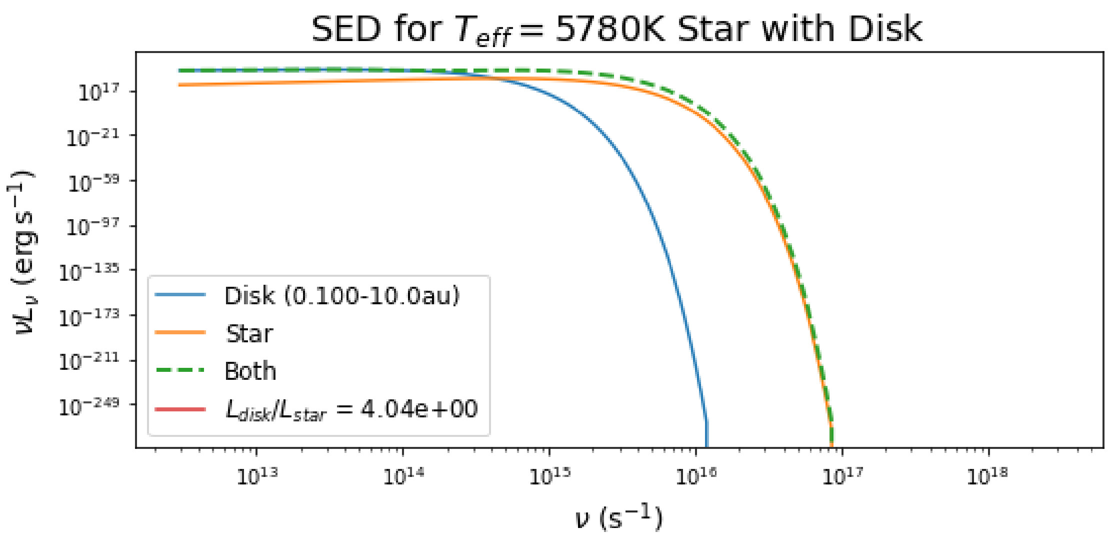
4.5.1 Comparing the SEDs for dust cleared out within 1 and 10 au of the host star to the SED with no dust cleared out past 0.02 au (r_{sub}), it's clear that the former peak at longer wavlengths ($\sim 10-100\mu m$) than the latter at shorter wavelengths ($\sim 1\mu m$).

4.5.2 It appears that you could infer the presence of a planet (dust cleared out within a certain distance from the host star) by looking at the frequency gap between where the star+disk and disk SEDs begin to be detected (i.e. where they first appear above the x-axis). Additionally, the star+disk SED begins to match the disk SED at short wavelengths and where this overlap begins is also different for the 1au and 10au cases (e.g. the green dashed line matches the blue line over a broader frequency range for the 1au case than the 10au case)

4.6 3c) Suppose the disk inner edge were now 0.1 au and the outer edge of the disk was truncated at 10au due to the gravitational effects of a distant planet or perhaps a companion star. How does the resulting SED change compared to the full-disk case?

[120]:
```python
# Initialize SED class for case in #3c (dust lies from .1-10au)
SED_Disk_truncated_freq = SED('freq','xvar_luminos',r_min=0.10,r_max=10.0,dr=.
↪01,N=10**5)
freq_ydata_3c = SED_Disk_truncated_freq.StarDiskProfile()
```

```
Lstar = 1.006 Lsun
Disk-to-Star Luminosity Ratio = 4.04e+00
StarDiskProfile took 51.44 sec (0.857 min) to run
```

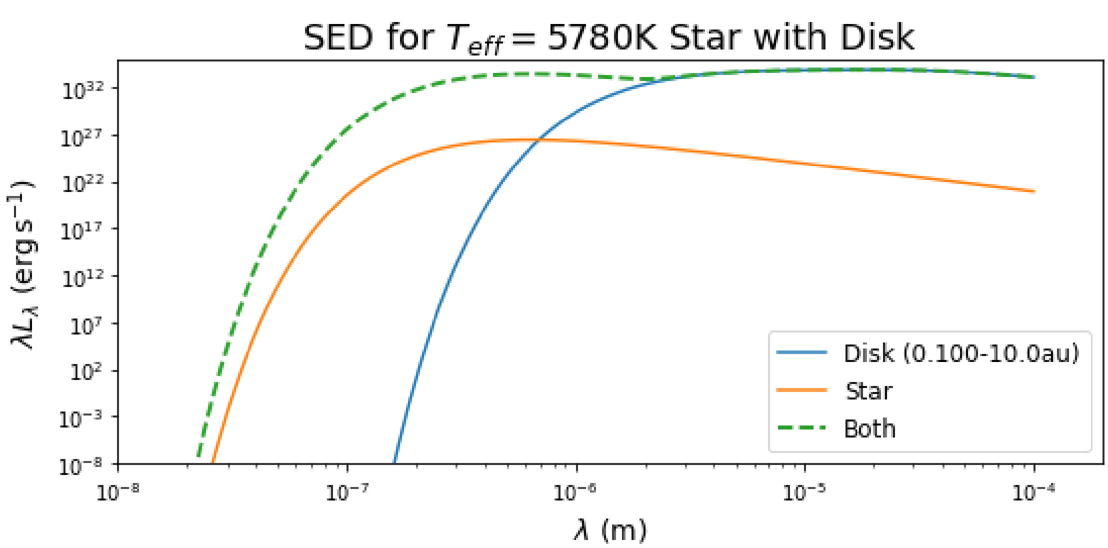


[121]:

```
# Plot disk, star, and combined SED for #3b,ii (dust cleared out from within 10␣
↪au)
SED_Disk_truncated_wavelen = SED('wavelen','xvar_luminos',r_min=0.10,r_max=10.
↪0,dr=.01,N=10**5)
SED_Disk_truncated_wavelen.StarDiskProfile()
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
StarDiskProfile took 52.22 sec (0.870 min) to run
```



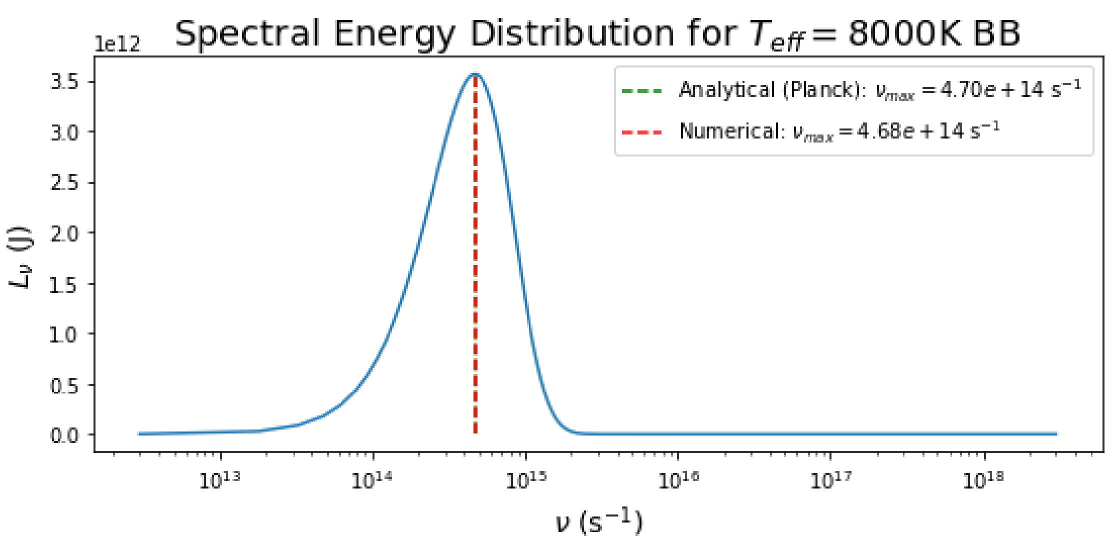
4.6.1 The truncated disk SED peaks at about $10\mu m$ instead of $1\mu m$ for the full disk SED and also the star+disk SED is dominated by the disk at . Additionally, the truncated SED has a consistently higher luminosity and dominates the star+disk SED across a wider frequency range (begins at $\sim 1\mu m$ for the truncated case and at $\sim 10\mu m$ for the full disk case).

4.7 3d) Use actual data from the Beta Pictoris system from the near and far infrared to create a model for an A6V star ($T_{eff} = 8000K, R = 1.92R_{\odot}$) surrounded by a dusty disk (assume a distance of 19pc)

[184]:
```
# Get first look at SED for A6V star with disk from sublimation radius to 1000au
SED_A6V_initial_freq = SED('freq','luminosity',Teff=8000,Rstar=1.92,N=2*10**5)
freq_A6V_initial,luminos_A6V_initia = SED_A6V_initial_freq.SEDStar(plot=True)
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
Lstar = 7.085 Lsun
Program took 7.81 sec (0.130 min) to run
```

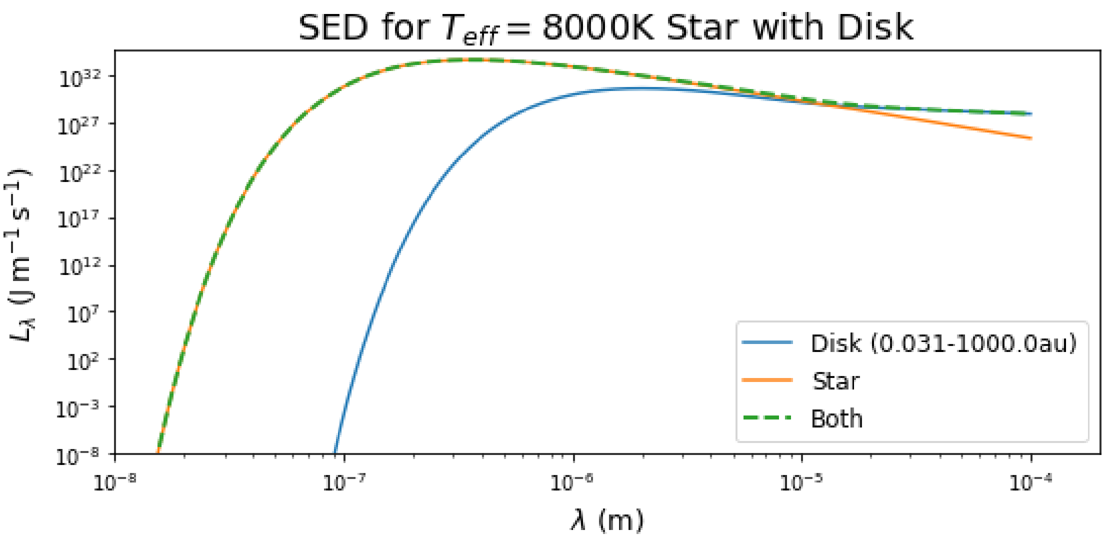


[233]:
```
SED_A6V_initial_wave = SED('wavelen','luminosity',Teff=8000,Rstar=1.
↪92,N=2*10**5)
A6V_initial_combined = SED_A6V_initial_wave.StarDiskProfile()
```

```
Total dust mass = 1.001 M_Earth
```

```
/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)
```

```
StarDiskProfile took 105.57 sec (1.760 min) to run
```



[157]:
```python
# Function to determine wavelength limits of different bands
def BandLimits(midpoint,FWHM):
    # Inputs:
    #   midpoint: central frequency of band
    #   FWHM: FWHM of band
    # Returns:
    #   limits: 2-element array of lower and upper wavelengths of band (in␣
 ↪meters)

    # Redefine midpoint and FWHM in meters
    midpoint = midpoint.to(u.m)
    FWHM = FWHM.to(u.m)

    # Calculate sigma of wavelength distribution
    sigma = FWHM/2.35

    # Calculate limits
    lower = np.round(midpoint-(sigma*3.0),9)
    upper = np.round(midpoint+(sigma*3.0),9)

    return([lower.value,upper.value])
```

[171]:
```python
# Calculate wavelength limits for each band
band_limits = []
for params in band_params:
    limits = BandLimits(params[0],params[1])
    band_limits.append(limits)
```

4.7.1 I originally wrote this function because I thought the HST tool would be more optimized to provide monochromatic fluxes for a range of wavelengths. Since it can really only do one wavelength at a time, I didn't use this function

[191]:
```
# Define midpoints and FWHMs (in nm) for different bands from SIMBAD␣
↪(V,R,I,J,H,K)
# Data from https://en.wikipedia.org/wiki/Photometric_system
V_params = [551.0,88.0]
R_params = [658.0,138.0]
I_params = [806.0,149.0]
J_params = [1220.0,213.0]
H_params = [1630.0,307.0]
K_params = [2190.0,390.0]
band_params = np.asarray([[551.0,88.0],[658.0,138.0],[806.0,149.0],[1220.0,213.
↪0],[1630.0,307.0],[2190.0,390.0]])*u.nm
band_midpoints = (np.asarray([551.0,658.0,806.0,1220.0,1630.0,2190.0])*u.nm).
↪to(u.m).value
```

[ ]:
```
# Calculate monochromatic luminosity at midpoints of different bands (full disk)
my_luminosities_fulldisk = []
for i in range(len(band_midpoints)):
    testSED = SED('freq','luminosity',Teff=8000,Rstar=1.
↪92,lambda_min=band_midpoints[i],lambda_max=band_midpoints[i],N=1)
    test_StarDiskdata = testSED.StarDiskProfile(plot=False)
    my_luminosities_fulldisk.append(test_StarDiskdata[0])
```

[180]:
```
# Projected surface area at distance to Beta Pictoris
denominator = 4*np.pi*(19*u.pc).to(u.m)**2
```

[202]:
```
# Make list of band magnitudes
# Data from http://simbad.u-strasbg.fr/simbad/sim-id?mescat.
↪iso=on&Ident=@2997572&Name=*+bet+Pic&submit=display+selected+measurements#lab_meas
band_magnitudes = [3.86,3.74,3.58,3.57,3.51,3.48]

# Convert to fluxes/luminosities using this HST/STScI tool: https://colortool.
↪stsci.edu/unit-conversion/#output
band_flux = np.asarray([1.01e-24,9.16e-25,8.32e-25,6.23e-25,3.87e-25,2.
↪51e-25])*((u.J/u.s)/(u.Hz*u.m**2))
band_luminosity = (band_flux*denominator).value
```

[210]:
```
# Find ratio of luminosities from my calculations (full disk) and HST calculator
print(my_luminosities_fulldisk/band_luminosity)
```

```
[0.79397265 0.8990482  0.93184301 0.88621074 1.00168922 1.01178437]
```

4.7.2 Considering a full disk from the sublimation distance to 1000au, my luminosity values agree reasonably well with the converted magnitudes from SIMBAD (i.e. ratios near 1). This model seems to work best towards the H and K bands.

[ ]:
```
# Calculate monochromatic luminosity at midpoints of different bands (truncated␣
↪disk)
my_luminosities_truncated = []
for i in range(len(band_midpoints)):
    testSED = SED('freq','luminosity',Teff=8000,Rstar=1.92,r_min=0.1,r_max=100.
↪0,lambda_min=band_midpoints[i],lambda_max=band_midpoints[i],N=1)
    test_StarDiskdata = testSED.StarDiskProfile(plot=False)
    my_luminosities_truncated.append(test_StarDiskdata[0])
```

[226]:
```
# Find ratio of luminosities from my calculations (full disk) and HST calculator
print(my_luminosities_truncated/band_luminosity)
```

```
[0.79397251 0.89904655 0.93182734 0.88600737 1.00498679 1.06211009]
```

4.7.3 Considering a truncated disk makes my ratios worse so this doesn't appear to match the published data better

[ ]:
```
# Now decrease dust size by a factor of 10 (.01mm instead of .1mm) for full disk
my_luminosities_fulldisk_red_size = []
for i in range(len(band_midpoints)):
    testSED = SED('freq','luminosity',Teff=8000,Rstar=1.
↪92,lambda_min=band_midpoints[i],lambda_max=band_midpoints[i],N=1)
    test_StarDiskdata = testSED.StarDiskProfile(plot=False)
    my_luminosities_fulldisk_red_size.append(test_StarDiskdata[0])
```

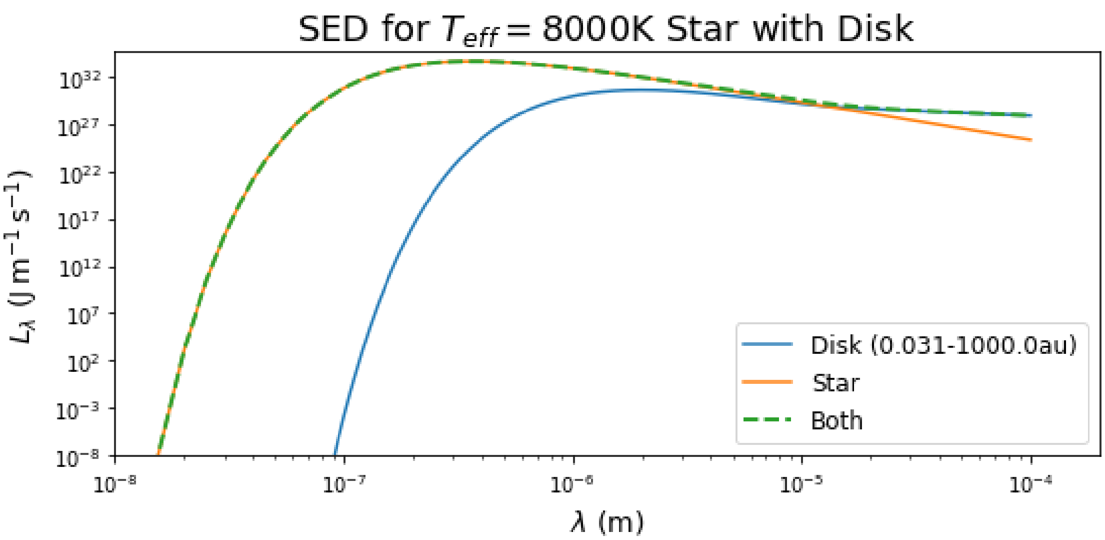
[228]:
```
# Find ratio of luminosities from my calculations (full disk) and HST calculator
print(my_luminosities_fulldisk_red_size/band_luminosity)
```

```
[0.79397387 0.8990631  0.93198666 0.88987169 1.02097695 1.07074759]
```

4.7.4 This slightly improves the smaller wavelength bands but worsens the larger bands

[231]:
```
SED_A6V_smalldust = SED('wavelen','luminosity',Teff=8000,Rstar=1.92)
A6V_smalldust_combined = SED_A6V_smalldust.StarDiskProfile(plot=True)
```

```
Total dust mass = 1.001 M_Earth

/usr/local/Anaconda3/lib/python3.6/site-packages/astropy/units/quantity.py:477:
RuntimeWarning: overflow encountered in exp
  result = super().__array_ufunc__(function, method, *arrays, **kwargs)

StarDiskProfile took 11.06 sec (0.184 min) to run
```



4.7.5 **Regarding the degeneracy between the mass of dust, inner radius of the disk, and outer radius of the disk: if the mass is decreased, but the size of the disk increased, you may see the same effects as if the mass was increased, but the size of the disk was decreased. So to investigate how modifying the mass changes the SED, the disk size must remain the same and vice versa to investigate how modifying the disk size changes the SED.**

4.7.6 **Perhaps due to the nature of the scales of my graphs, I don't notice a significant difference in the SEDs for smaller vs. larger dust size (i.e. smaller vs larger dust mass). The dust mass/size assumption is important though because it dictates how the grain radiates and which wavelengths of light it most effectively absorbs/emits.**

4.7.7 **My luminosity values being relatively similar to the published values indicates to me that a model disk with our assumed properties ($1M_\oplus$ spread over r_{sub} to 1000au, a_{grain}=.1mm) is a decent model for Beta Pictoris' disk.**

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 12 16:15:10 2020

@author: jimmy
"""
import numpy as np
import matplotlib.pyplot as plt
import bisect
import time
from astropy import units as u
from astropy import constants as const
from MathTools import EquilTemp

# Class to generate and analze spectral energy distributions (SEDs)
class SED:

    def __init__(self,xvariable,yvariable,r_min=None,r_max=None,dr=1.0,Teff=5780,Rstar=1.0,lambd
        # Inputs:
        #   xvariable: 'freq' or 'wavelen' to determine which Planck form to use
        #   yvariable: 'planck' or 'luminosity' or 'xvar_lumin'
        #   r_min: where disk starts w.r.t star (in au)
        #   r_max: where disk ends w.r.t star (in au)
        #   Teff: effective temperature of host star (in K, default is T_Sun)
        #   Rstar: radius of host star (in Rsun, defualt is 1 R_Sun)
        #   dr: differential radius between rings (in au, default is 1.0)
        #   lambda_min: minimum wavelength to calc. Planck function over (in m)
        #   lambda_max: minimum wavelength to calc. Planck function over (in m)
        #   Number of subintervals to integrate over

        # Cast initial parameters as global variables
        self.xvariable = xvariable
        self.yvariable = yvariable
        self.Teff = Teff*u.K
        self.lambda_min = lambda_min*u.m
        self.lambda_max = lambda_max*u.m
        self.N = N

        # Calculate surface area of sun for later use
        self.sun_SA = 4*np.pi*(Rstar*const.R_sun**2)

        # Define differential radius (distance between rings of disk)
        self.dr = (dr*u.au).to(u.m)

        # If user doesn't specify starting dist. of disk, use r_sub
        if r_min == None:
            # Calculate the dust sublimation radius using rad. equil. temp. eqn.
            self.r_sub = (const.R_sun/2*np.sqrt(1-0.3)*(self.Teff.value/2000)**2).to(u.au)
            self.r_min = self.r_sub.to(u.m)
        # Otherwise, use their starting point and convert it to meters
        else:
            self.r_min = (r_min*u.au).to(u.m)

        # If user doesn't specify ending dist. of disk, use 1000 au
        if r_max == None:
            self.r_max = (1000.0*u.au).to(u.m)
        else:
            self.r_max = (r_max*u.au).to(u.m)

        # Calculate wavelength at which blackbody peaks (in m)
        self.lambda_peak = ((2.90*(10**3)*u.micron*u.K)/self.Teff).to(u.m)

        # Set boundaries of analysis depending on x variable
        if self.xvariable == 'wavelen':

            # Set xdata boundaries and make list of x values (self.x)
```

```
            self.x_min = self.lambda_min
            self.x_max = self.lambda_max
            self.x = np.linspace(self.x_min,self.x_max,self.N)

            # Calculate wavelength at which blackbody peaks (in m)
            self.y_peak = self.lambda_peak

            # Define x-axis label
            self.xlabel = r'$\lambda$ ({0:latex_inline})'.format(self.x_min.unit)

            # Define v-line labels
            self.vlabel = r'Analytical (Planck): $\lambda_{{max}}={0:.2e}$ {1:latex_inline}'.for
            self.num_vlabel = r'Numerical: $\lambda_{{max}}={0:.2e}$ {1:latex_inline}'

            if self.yvariable == 'planck':
                self.ylabel = r'Spectral Radiance per $\Omega$ ({0:latex_inline})'
            elif self.yvariable == 'luminosity':
                self.ylabel = r'$L_{{\lambda}}$ ({0:latex_inline})'
            elif self.yvariable == 'xvar_luminos':
                self.ylabel = r'$\lambda L_{{\lambda}}$ ({0:latex_inline})'

        elif self.xvariable == 'freq':

            # Convert wavelength range to frequency range and make x value list
            self.x_min = self.lambda_min.to(u.s**(-1), equivalencies=u.spectral())
            self.x_max = self.lambda_max.to(u.s**(-1), equivalencies=u.spectral())
            self.x = np.linspace(self.x_max,self.x_min,self.N)

            # Calculate frequency at which blackbody peaks (in s^-1)
            self.y_peak = (5.88*(10**10)*(u.s**(-1))/u.K)*self.Teff

            # Define x-axis label
            self.xlabel = r'$\nu$ ({0:latex_inline})'.format(self.x_min.unit)

            # Define v-line labels
            self.vlabel = r'Analytical (Planck): $\nu_{{max}}={0:.2e}$ {1:latex_inline}'.format(
            self.num_vlabel = r'Numerical: $\nu_{{max}}={0:.2e}$ {1:latex_inline}'

            if self.yvariable == 'planck':
                self.ylabel = r'Spectral Radiance per $\Omega$ ({0:latex_inline})'
            elif self.yvariable == 'luminosity':
                self.ylabel = r'$L_{{\nu}}$ ({0:latex_inline})'
            elif self.yvariable == 'xvar_luminos':
                self.ylabel = r'$\nu L_{{\nu}}$ ({0:latex_inline})'

        else:
            print("Valid entries are 'wavelen' or 'freq'")

    # Function to calculate main part Planck function at given wavelength
    def Planck(self,x,T,units=True):
        # Inputs:
        #   x: value of x-variable to calculate Planck function at
        #   T: temperature of blackbody (in K)
        #   units: boolean to decide if quantities should have units
        #           (no units is preferable if using func. to integrate)
        # Returns:
        #   B: value of Planck function at that wavelength

        # Define temperature with Kelvin units
        #T = T*u.K

        if self.xvariable == 'wavelen':
            # Calculate 2hc^2 (prefactor in Planck's function)
            prefactor = (2*const.h*const.c**2)

            # Calculate hc/kT (constant in exponential of Planck's function)
            exp_factor = (const.h*const.c/(const.k_B*T))
```

```
        if units == False:
            # Calculate value of Planck function at this wavelength
            B = prefactor.value*x.value**(-5)/(np.exp(exp_factor.value/x.value)-1)
        else:
            B = prefactor*x**(-5)/(np.exp(exp_factor/x)-1)

    elif self.xvariable == 'freq':

        # Calculate 2h/c^2 (prefactor in Planck's function)
        prefactor = 2*const.h/const.c**2

        # Calculate h/kT (constant in exponential of Planck's function)
        exp_factor = const.h/(const.k_B*T)

        if units == False:
            # Calculate value of Planck function at this wavelength
            B = prefactor.value*x.value**3/(np.exp(exp_factor.value*x.value)-1)
        else:
            B = prefactor*x**3/(np.exp(exp_factor*x)-1)

    return(B)

# Function to insert values and sort them
def insert_list(self,main_list, new_list):
    # Inputs:
    #   main_list: primary list that new_list will be inserted into
    #   new_list: list of new values to insert into main_list
    # Returns:
    #   main_list(updated): primary list with new values correctly sorted

    # Place each value of new_list into correct position
    # main_list.tolist() converts numpy array to reg. list for indexing
    for i in range(len(new_list)):
        bisect.insort(main_list.tolist(),new_list[i])

    return(main_list)

# Function to plot spectral energy distribution of star
def SEDStar(self,plot=False):
    # Inputs:
    #   plot: boolean to decide to make plot of SED
    # Returns:
    #   Plot of star's SED
    #   xdata and ydat used to plot SED

    # Determine when function began running
    start_time = time.time()

    # Calculate Planck function at each wavelength/freq
    y = self.Planck(self.x,self.Teff)

    # Calculate total Sun luminosity
    self.star_luminosity = np.pi*self.sun_SA*np.trapz(y,self.x)

    if self.xvariable == 'freq':
        print("Lstar = {0:.3f} Lsun".format(self.star_luminosity/const.L_sun.to(u.J/u.s)))

    # Convert Planck function to luminosity
    if self.yvariable == 'luminosity':
        y *= np.pi*self.sun_SA

    # Convert Planck function to luminosity*xvariable (planck * x**2)
    elif self.yvariable == 'xvar_luminos':
        y *= np.pi*self.sun_SA
        y = np.multiply(self.x,y)
```

```
        elif self.yvariable == 'flux':
            y *= np.pi

        # Find where blackbody peaks from my calculations
        peak_loc = np.argmax(y)
        numerical_max = self.x[peak_loc]

        if self.yvariable == 'planck' and plot == True:
            # Calculate stellar luminosity (np.trapz integrates Planck func.)
            luminosity = np.pi*self.sun_SA*np.trapz(y,self.x)
            print("Luminosity = {0:.3f} Lsun".format(luminosity.to(u.erg/u.s)/const.L_sun.to(u.e

            # Calculate fraction of luminosity from below peak wavelength
            lumin_before = np.pi*self.sun_SA*np.trapz(y[:peak_loc],self.x[:peak_loc])
            frac_before = lumin_before/luminosity*100
            print("{0:.2f}% of energy emitted below peak".format(frac_before))

            # Calculate fraction of luminosity from peak wavelength and beyond
            lumin_after = np.pi*self.sun_SA*np.trapz(y[peak_loc:],self.x[peak_loc:])
            frac_after = lumin_after/luminosity*100
            print("{:.2f}% of energy emitted above peak \n".format(frac_after))

        # Decide whether to plot SED or not
        if plot == True:

            # Create wide figure (w=8in,h=4in)
            plt.figure(figsize=(8,4))

            # Plot data with x-axis on log scale
            plt.plot(self.x,y)
            plt.xscale('log')

            # Plot analytical and numerical wavelength peaks
            plt.vlines(self.y_peak.value,min(y).value,max(y).value,colors='green',\
                       linestyles='dashed',label=self.vlabel)
            plt.vlines(numerical_max.value,min(y).value,max(y).value,colors='red',\
                       linestyles='dashed',label=self.num_vlabel.format(numerical_max.value,nume
            plt.legend()

            # Axes labels and titles
            plt.xlabel(self.xlabel,fontsize=14)
            plt.ylabel(self.ylabel.format(y.unit),fontsize=14)
            plt.title(r'Spectral Energy Distribution for $T_{eff}=$%dK BB'%self.Teff.value,fonts
            plt.tight_layout()

            # Tell user how long function took to run
            end_time = time.time()-start_time
            print('Program took %.2f sec (%.3f min) to run'%(end_time,end_time/60.0))

        return(self.x,self.y)

    # Function to calculated total flux over bandpass
    def ResponseFunction(self,bandpass,effic,plot=False):
        # Inputs:
        #   bandpass: frequency ranges of bandpass
        #   effic: efficiencies/response functions of bandpass
        # Returns:
        #   flux_total: total integrated flux over full freq. range

        # Plot efficiency of bandpass
        if plot==True:
            plt.plot(bandpass,effic)

        # Multiply Planck function at each frequency by efficiency
        planck = np.multiply(self.Planck(bandpass,self.Teff),effic)

        # Integrate Planck function over bandpass
```

```python
        flux_total = np.trapz(planck,bandpass)

        return(flux_total)

    def SEDDisk(self,a=.1*10**-3,rho=2.0,plot=False):
        # Inputs:
        #   a: mean grain radius (in m)
        #   rho: mean grain density (in g/cm^3 = kg/m^3)
        # Returns:
        #   disk_ydata: user-selected ydata for disk (planck, luminos, etc.)

        # Define global variables from input variables
        self.grain_size = a*u.m
        self.rho = (rho*u.g/(u.cm**3)).to(u.kg/u.m**3)

        # Calculate mean dust grain mass (kg)
        self.m_grain = 4/3*np.pi*self.grain_size**3*self.rho

        # Calculate grain surface area (m^2)
        self.SA_grain = 4*np.pi*self.grain_size**2

        # Define total disk area (m^2) and mass surface density (kg/m^2)
        self.area_total = np.pi*(self.r_max-self.r_min)**2
        self.surf_dens = const.M_earth/self.area_total

        # Calculate grain surface density (in # of grains/m^2)
        self.grain_dens = self.surf_dens/self.m_grain

        # Make list of radii to describe each ring's distance from star
        radii = np.arange(self.r_min.value,self.r_max.value+self.dr.value,self.dr.value)*u.m

        # Make list of ring areas (2*pi*r*dr)
        areas = 2*np.pi*radii*self.dr

        # Make list of temperatures of each ring
        temps = EquilTemp(radii)

        # Calculate number of grains within each ring
        numGrains = self.grain_dens*areas

        # Calculate mass in each ring and total dust mass in disk
        masses = numGrains*self.m_grain
        dust_mass = np.sum(masses)/const.M_earth # in Earth masses
        print('Total dust mass = {0:.3f} M_Earth'.format(dust_mass))

        # Establish empty array of sums of mono. ydata for each temp
        disk_plancks = []
        disk_luminosities = []

        # Loop over temperatures to calc. Planck at each freq for each temp.
        for i in range(len(self.x)):

            # Track progress of code by printing loop numbers
            #print("Now executing loop {0} of {1}".format(i+1,len(self.x)))

            # Calculate Planck function for each ring
            ring_plancks = numGrains*self.Planck(self.x[i],temps)

            # Calculate monochromatic luminosity for each ring
            ring_luminosities = np.pi*self.SA_grain*ring_plancks

            # Sum Planck + luminosity value of each ring and add sums to lists
            disk_plancks.append(np.sum(ring_plancks))
            disk_luminosities.append(np.sum(ring_luminosities))

        # Recast disk planck and luminosity arrays to better configuration
        disk_plancks = np.asarray([y.value for y in disk_plancks])*disk_plancks[0].unit
```

```
        disk_luminosities = np.asarray([y.value for y in disk_luminosities])*disk_luminosities[0

        # Calculate disk luminosity
        self.disk_luminosity = np.pi*self.SA_grain*np.trapz(disk_plancks,self.x)

        # Save planck array as numpy array w/ proper units
        if self.yvariable == 'planck':
            disk_ydata = disk_plancks

        # Save luminosity array as numpy array w/ proper units
        elif self.yvariable == 'luminosity':
            disk_ydata = disk_luminosities

        # Save nu*L_nu array as numpy array w/ proper units
        elif self.yvariable == 'xvar_luminos':
            disk_ydata = disk_luminosities*self.x

        # Plot SED of disk if requested by user
        if plot == True:
            # Create wide figure (w=8in,h=4in)
            plt.figure(figsize=(8,4))

            plt.plot(self.x,disk_ydata,label=('Disk: ({0:.3f}-{1:.1f}au)'\
                      .format(self.r_min.to(u.au).value,self.r_max.to(u.au).value)))
            plt.xscale('log')

            # Axes labels and titles
            plt.xlabel(self.xlabel,fontsize=14)
            plt.ylabel(self.ylabel.format(disk_ydata.unit),fontsize=14)
            plt.title(r'SED for Disk Around $T_{{eff}}=${0}K Star'.format(self.Teff.value),fonts
            plt.tight_layout()

        return(disk_ydata)

    # Function to plot star and disk SEDs
    def StarDiskProfile(self,plot=True):

        # Determine when function started running
        start_time = time.time()

        # Generate disk ydata values and convert to numpy array
        disk_ydata = self.SEDDisk()#.to(u.erg/u.s)

        # Generate frequencies and star flux values
        xdata,star_ydata = self.SEDStar()
        #star_ydata.to(u.erg/u.s)

        # Calculate sum of disk and Sun flux
        combined_system = np.add(disk_ydata,star_ydata)

        # Calculate ratio of total luminosities of disk and star
        self.luminosity_ratio = self.disk_luminosity.value/self.star_luminosity.value

        if self.xvariable == 'freq':
            print("Disk-to-Star Luminosity Ratio = {0:.2e}".format(self.luminosity_ratio))

        # Plot SEDs if user wants
        if plot == True:
            # Create wide figure (w=8in,h=4in)
            plt.figure(figsize=(8,4))

            # Plotting data
            plt.plot(xdata,disk_ydata,label=r'Disk ({0:.3f}-{1:.1f}au)'\
                     .format(self.r_min.to(u.au).value,self.r_max.to(u.au).value))
            plt.plot(xdata,star_ydata,label='Star')
            plt.plot(xdata,combined_system,label='Both',linestyle='dashed',linewidth=2)
            if self.xvariable == 'freq':
```

```
            plt.plot([],[],'',label=r'$L_{{disk}}/L_{{star}}$ = {0:.2e}'.format(self.luminos
        plt.xscale('log')
        plt.yscale('log')

        if self.xvariable == 'wavelen':
            plt.xlim(xmin=10**-8)
            plt.ylim(ymin=10**-8,ymax=np.max(combined_system).value*10)

        # Axes labels and titles
        plt.xlabel(self.xlabel,fontsize=14)
        plt.ylabel(self.ylabel.format(disk_ydata[0].unit),fontsize=14)
        plt.title(r'SED for $T_{eff}=$%dK Star with Disk'%self.Teff.value,fontsize=18)
        plt.tight_layout()
        plt.legend(prop={'size': 12})

        # Tell user how long function took to run
        end_time = time.time()-start_time
        print('StarDiskProfile took %.2f sec (%.3f min) to run'%(end_time,end_time/60.0))

    return(combined_system.value)
"""
# Code to test class and functions
test = SED('freq','xvar_luminos',r_min=1.0,N=10**4)
#test.Planck(10**-6*u.m,units=False)
#test.SEDStar(True)
test.StarDiskProfile(plot=True)
#"""
```

```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 oserations
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)

# Trapezoidal integration function
def TrapIntegrate(f,a,b,N=500):
    # Inputs:
    #   a: lower bound
    #   b: upper bound
    #   N: # of trapezoids
    # Returns:
    #   s*h: value of integral
    # Example:
    # f = lambda x: 3*x**2
```

```
    # >> TrapIntegrate(f,0,1,10000)
    # >>>> Integral = 1.0000000050000002

    # Step size
    h = (b-a)/float(N)

    # First and last terms of trapezoidal sum calculated directly
    s = 0.5*(f(a)+f(b))

    # Integrate over number of trapezoids
    for theta in range(0,N):

        # Evaluate function at one step further from previous step
        s += f(a+(theta*h))
    print('Integral = {0}'.format(s*h))
    return(s*h)

# Function to calculate radiative equilibrium tempertature
def EquilTemp(dist,albedo=0.3,Rstar=1.0*const.R_sun,Teff=5780*u.K,show=False):
    # Inputs:
    #   albedo: albedo of body of interest (usually planet; default=0.3)
    #   Rstar: radius of host star (in Rsun; default is 1)
    #   dist: distance of body of interest from host star (usually SMA in au)
    #   Teff: effective temperature of host star (in K; default is Sun temp.)
    # Returns:
    #   Teq: radiative equilibrium temperature of body of interest

    # Convert Rstar and dist to same units (m)
    Rstar = Rstar.to(u.m).value
    dist = dist.to(u.m).value

    # Calculate radiative equilibrium temperature
    Teq = (((1-albedo)*Rstar**2)/(4*dist**2))**(1/4)*Teff

    # Print temperature if desired
    if show == True:
        print("Body Temperature = {0:.2f} K".format(Teq))

    return(Teq)

# Function to calculate apparent magnitude difference
def MagDiff(flux1,flux2):
    # Inputs:
    #   flux1,2: integrated fluxes over different freq/wavelen. baselines
    # Returns:
    #   delta_m: apparent magnitude difference (m1-m2)

    # Calculate delta_m
    delta_m = -2.5*np.log10(flux1/flux2)

    print("Apparent magnitude difference = {0:.3f}".format(delta_m))
    return(delta_m)

# Function to calculate main part Planck function at given wavelength
def Planck(x,T,units=True):
    # Inputs:
    #   x: value of x-variable to calculate Planck function at
    #   T: temperature of blackbody (in K)
    #   units: boolean to decide if quantities should have units
    #           (no units is preferable if using func. to integrate)
    # Returns:
    #   B: value of Planck function at that wavelength

    # Define temperature with Kelvin units
    #T = T*u.K

    # Calculate 2h/c^2 (prefactor in Planck's function)
```

```
prefactor = 2*const.h/const.c**2

# Calculate h/kT (constant in exponential of Planck's function)
exp_factor = const.h/(const.k_B*T)

if units == False:
    # Calculate value of Planck function at this wavelength
    B = prefactor.value*x.value**3/(np.exp(exp_factor.value*x.value)-1)
else:
    B = prefactor*x**3/(np.exp(exp_factor*x)-1)

return(B)
```

```
#!/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
from astropy import units as u

# 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

# Function to read HW4 bandpass data (Kepler and Spitzer 4.5um)
def ReadBandpass(filename,xvariable,unit,delim='\t',skip=8):

    # Extract data from file
    data = np.genfromtxt(filename,delimiter=delim,skip_header=skip)

    # Extract data into 2 separate lists
    xdata, ydata = zip(*data)

    # wavelengths in micron
    if unit == 'um':
        # Save lists as numpy arrays (convert first list to m from um)
        xdata, ydata = np.asarray(xdata)/10**6*u.m, np.asarray(ydata)
    # wavelengths in nanometers
    elif unit == 'nm':
        # Save lists as numpy arrays (convert first list to m from nm)
        xdata, ydata = np.asarray(xdata)/10**9*u.m, np.asarray(ydata)

    # Convert wavelengths to frequencies
    if xvariable == 'freq':
        xdata = xdata.to(u.s**(-1), equivalencies=u.spectral())

    return(xdata,ydata)
```