Problem Set 1 solutions

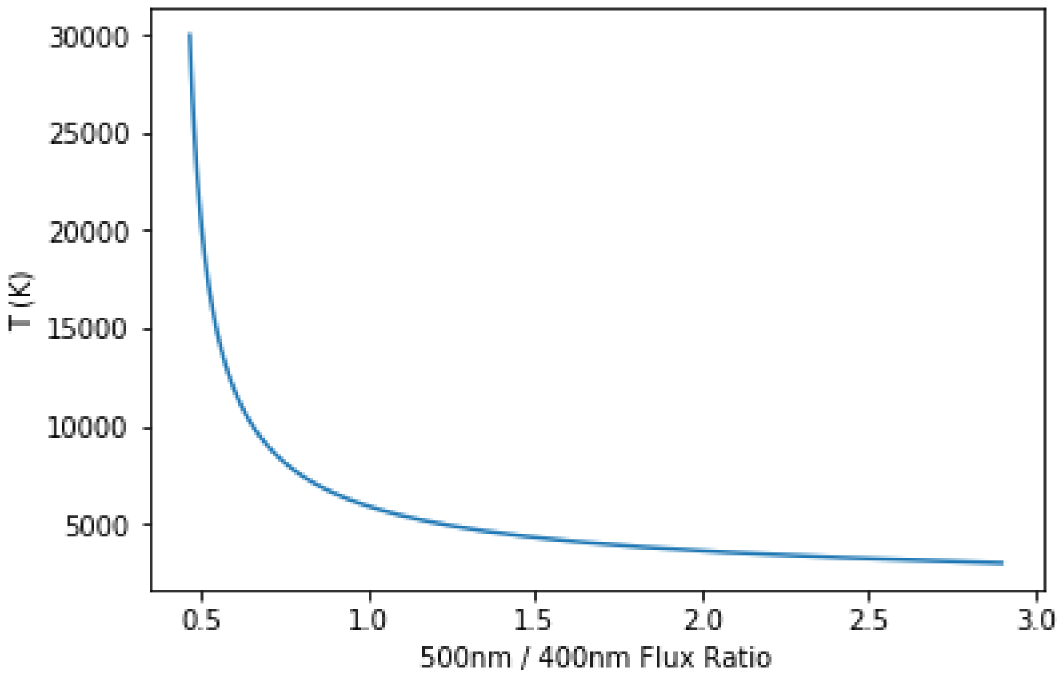
August 9, 2019

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import astropy.constants as c
        import astropy.units as u
```

1. (2 marks) We use the chain rule here, noting that: $\lambda = \frac{c}{\nu}$
$f_\nu = |\frac{df}{d\nu}| = |\frac{df}{d\lambda}\frac{d\lambda}{d\nu}| = \frac{c}{\nu^2}|\frac{df}{d\lambda}|$
The absolute value is not important to get the mark. The derivative notation implies differentiation of the cumulative flux distribution in opposite directions for wavelength and frequency.

2. (7 marks in total) **(a)** (2 marks) Create a numpy array defining the temperature, find the flux for each wavelength. 1 mark for the correct plot. 1/2 mark for axis labels and 1/2 mark for using the formula for wavelength.

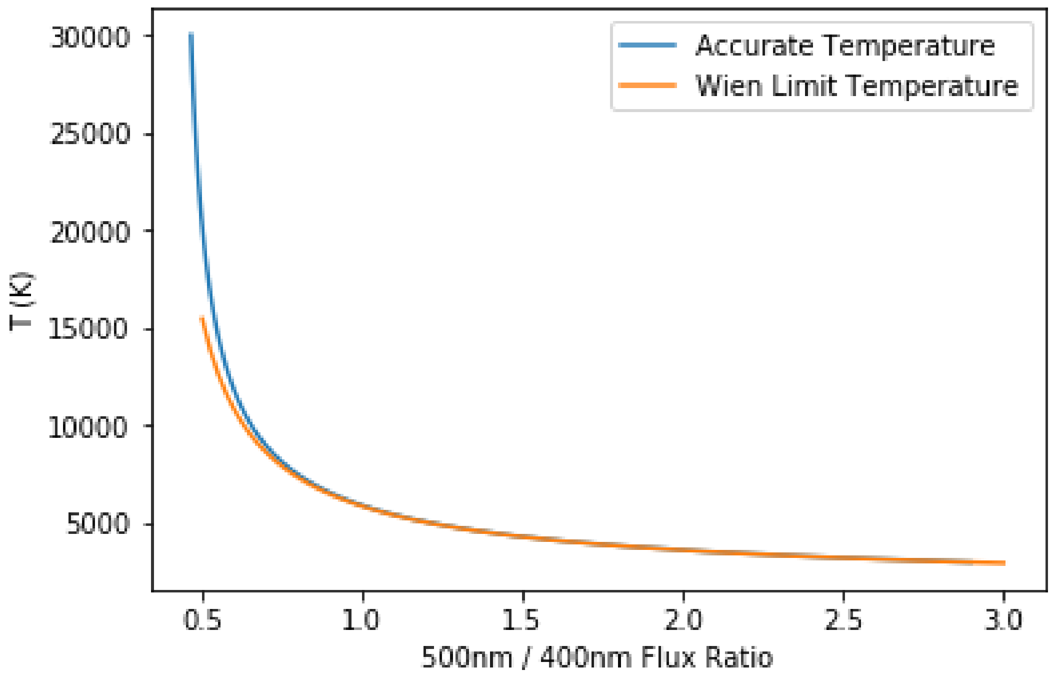
```
In [2]: wave1 = 440*u.nm
        wave2 = 550*u.nm
        T = np.linspace(3000,30000,100)*u.K
        #Flux density, which is pi times the Blackbody specific intensity.
        f1 = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/T)-1)
        f2 = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/T)-1)
        plt.clf()
        plt.plot(f2/f1, T)
        plt.xlabel('500nm / 400nm Flux Ratio')
        plt.ylabel('T (K)')
        plt.show()
```



(b) (1 mark, and 1 mark for the plot) In the Wien limit, we can write: $f_1 = \frac{2\pi hc^2}{\lambda_1^5} \exp(-hc/\lambda_1 k_B T)$ Taking the flux ratio: $\frac{f_2}{f_1} = \left(\frac{\lambda_1}{\lambda_2}\right)^5 \exp\left(hc/k_B T \times \left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2}\right)\right)$ Taking the logarithm of both sides: $\ln(f_2/f_1) = 5\ln(\lambda_1/\lambda_2) + \frac{1}{T}\frac{hc}{k_B}\left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2}\right)$

Re-arranging gives the resulting formula.

```
In [3]: #Re-do the plot and add the approximation.
        plt.clf()
        plt.plot(f2/f1, T, label='Accurate Temperature')
        fratio = np.linspace(0.5,3,100)
        plt.plot(fratio, (c.h*c.c/c.k_B*(1/wave1 - 1/wave2)/(np.log(fratio) + 5*np.log(wave2/wa
                 label='Wien Limit Temperature')
        plt.xlabel('500nm / 400nm Flux Ratio')
        plt.ylabel('T (K)')
        plt.legend()
        plt.show()
```



(c) (1 mark for Poisson argument) With a total number of photons N_p, the signal-to-noise is $N_p/\sqrt{N_p}$, which is 100:1. This is a precision of 1%. Their flux ratio is measured to a precision of $\sqrt{2} \approx 1.4\%$. Using an interactive python prompt, we can zoom in to the plot and see how much a 1.4% change in flux ratio changes with temperature and use `ginput`. In notebook form, lets compute the derivative explicitly, by changing temperature by 1K.

```
In [4]: #(2 marks for one of several ways to get a rough answer here)
        #Fluxes at 4000K and slightly more.
        f1 = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/(4000*u.K))-1)
        f2 = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/(4000*u.K))-1)
        f1_plus = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/(4001*u.K))-1)
        f2_plus = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/(4001*u.K))-1)
        #Approximate the derivative of the flux ratio with temperature using a finite differen
        dfratio_DT = (f2/f1 - f2_plus/f1_plus)/(1*u.K)
        Delta_fratio = (0.01*np.sqrt(2) * f2/f1)
        Delta_T_4000 = Delta_fratio/dfratio_DT

        #Fluxes at 20,00K and slightly more.
        f1 = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/(20000*u.K))-1)
        f2 = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/(20000*u.K))-1)
        f1_plus = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/(20001*u.K))-1)
        f2_plus = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/(20001*u.K))-1)
        #Approximate the derivative of the flux ratio with temperature using a finite differen
        dfratio_DT = (f2/f1 - f2_plus/f1_plus)/(1*u.K)
        Delta_fratio = (0.01*np.sqrt(2) * f2/f1)
```

```
Delta_T_20000 = Delta_fratio/dfratio_DT

#Now print out the precision
print("Temperature Precision at 4000K: {:.2f}".format(Delta_T_4000))
print("Temperature Precision at 20000K: {:.2f}".format(Delta_T_20000))
```

```
Temperature Precision at 4000K: 34.77 K
Temperature Precision at 20000K: 1187.16 K
```

3. (7 marks in total) Firstly, note that additional information was given in this question that wasn't used. The orbit of the binary could have been used via Kepler's law to obtain the stellar masses, and see that the stellar radius and temperature made sense. However, the problem set was long enough without asking this as well.

** (a)** (2 marks) We made a plot in question 1. Here we just need to read off the value. We'll use numpy's `interp` in this model solution. From an interactive prompt, zooming in to the plot is also fine.

```
In [5]: wave1=532*u.nm
        wave2=797*u.nm
        f1 = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/T)-1)
        f2 = 2*np.pi*c.h*c.c**2/wave2**5/(np.exp(c.h*c.c/wave2/c.k_B/T)-1)
        #One trick here - the x axis when interpolating has to be monotonic. So we use f1/f2,
        star_T = np.interp(5.9e-13/4.8e-13, f1/f2, T)*u.K
        print("Star temperature is {:5.1f}".format(star_T))
```

```
Star temperature is 5020.2 K
```

(b) (1 mark) The question tells us the parallax is 19 milli-arcsec. This is the inverse of the distance in pc.

```
In [6]: distance = (1/0.019)*u.pc
        print("Distance : {:.1f}".format(distance.to(u.pc)))
        print("Distance : {:.1e}".format(distance.to(u.km)))
```

```
Distance : 52.6 pc
Distance : 1.6e+15 km
```

** (c) ** (2 marks) Use the inverse-square law, which we can write as: $\frac{f_1(d)}{f_1(R_*)} = \left(\frac{R_*}{d}\right)^2$ You can derive this by noting that the luminosity seen through a sphere of radius d_1 and d_2 are the same. We scale the distance by the square root of the flux ratio: $d = R_* \sqrt{f_1(d)/f_1(R_*)}$ This works for either fluxes at either wavelength, if th star temperature is correct.

```
In [7]: f1_star_surface = 2*np.pi*c.h*c.c**2/wave1**5/(np.exp(c.h*c.c/wave1/c.k_B/star_T)-1)
        star_radius = distance * np.sqrt((5.9e-13*u.erg/u.s/u.cm**2/u.AA)/f1_star_surface)
        print("Star Radius: {:.2f}".format(star_radius.to(u.R_sun)))
```

```
Star Radius: 0.89 solRad
```

(d) (2 marks) We obtain the luminosity by multiplying the blackbody flux according to the Stefan-Boltzmann law by the surface area of the star. This is a dwarf star - slightly less luminous than the sun and slightly smaller.

```
In [8]: star_luminosity = (4 * np.pi * star_radius**2 * c.sigma_sb * star_T**4).to(u.L_sun)
        print("Star luminosity: {:.2f}".format(star_luminosity))
```

```
Star luminosity: 0.46 solLum
```