MQ-ASTR377-2018 / lab-1-planck-function-pemberj Private

Branch: master lab-1-planck-function-pemberj / Lab 1 – Planck Function.ipynb Find file Copy path

pemberj Finished All Excercises 19d96d0 on 19 Mar

2 contributors

1.29 MB

Introduction: The Planck Function

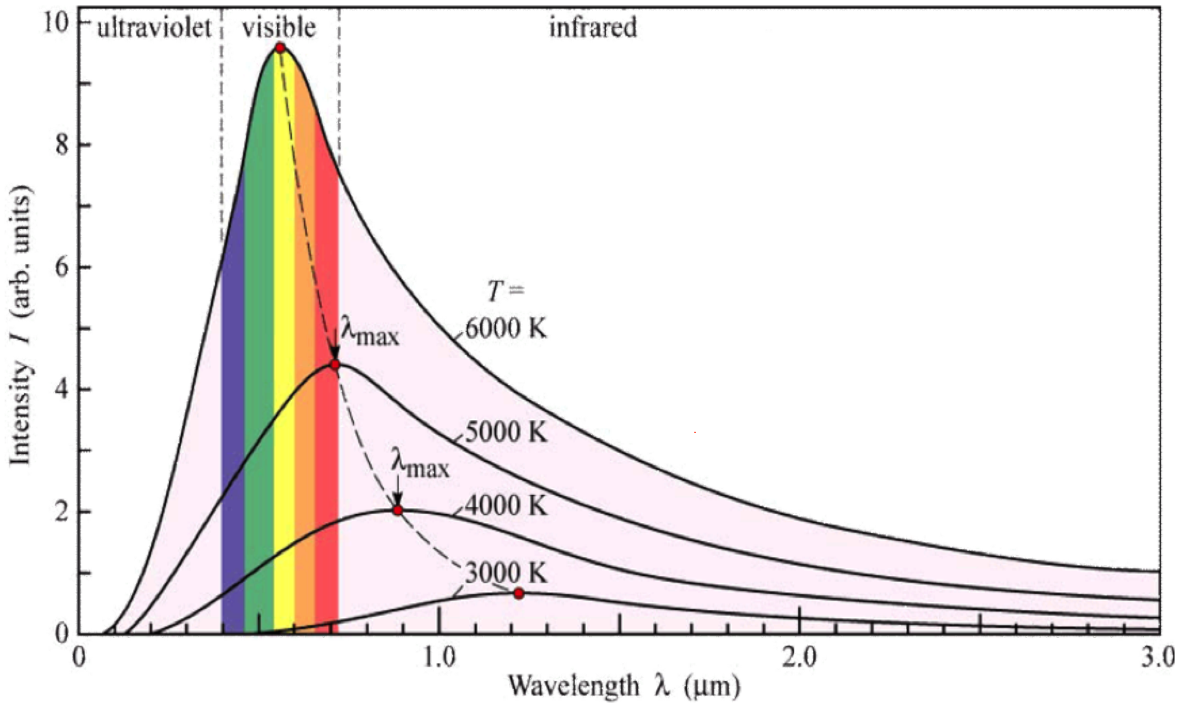
20/20

In this lab you will use python to make plots, and compute integrals and derivatives. Each task involves writing some python code, and perhaps producing a plot. Fill in your code below each task (including any plots), creating new 'cells' if you have to. Feel free to include text that explains what you are doing (for example, using headings to indicate your responses, or using <font color=blue>blue text</font> so I can see your contributions clearly), and to 'comment' your code as much as possible to explain what you are doing. I will give you feedback in <font color=red>red text</font> after marking it.

In [1]:
```
import matplotlib.pylab as plt
import numpy as np
%matplotlib inline
```

In [2]:
```
# This cell is used to load an image of the Planck function:
from IPython.display import Image
i = Image('Planck.png')
i
```

Out[2]:



In week 2 we shall discuss how a radiation field is described by its intensity I_λ or I_ν, the energy crossing unit area per unit time per unit solid angle per unit wavelength or frequency interval.

The Planck function describes the intensity of a black body radiation field. It can be defined in terms of wavelength:

$$B_\lambda(T) = \frac{2hc^2}{\lambda^5}\frac{1}{\exp(hc/\lambda kT) - 1}$$

(W m $^{-3}$ sr $^{-1}$) or frequency, B_ν:

$$B_\nu(T) = \frac{2h\nu^3}{c^2}\frac{1}{\exp(h\nu/kT) - 1}$$

(W m $^{-3}$ sr $^{-1}$).

Exercise 1

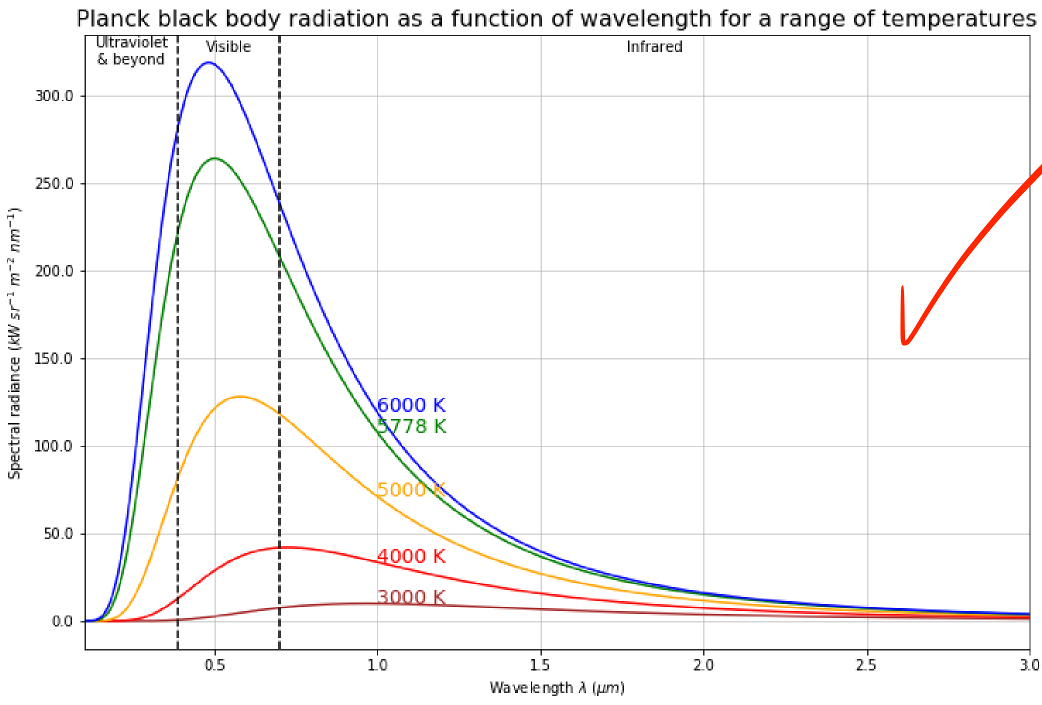
As in the figure above, plot the Planck function as a function of wavelength, B_λ, for several different values of temperature, T. To do this, you will need to create a vector of wavelengths, e.g. between 1,000Å and 30,000Å, using the numpy function `np.arange`:

```
lam = np.arange(1000,30000)*1.e-10
```

And use this as the 'x-axis' values for which to generate the intensities described by the Planck Function. Overplot curves for different temperatures. Make your plot presentable (i.e. with axis labels, legend, etc).

In [3]:
```
#Import constants for use in Planck functions
import scipy.constants
h = scipy.constants.h
c = scipy.constants.c
k = scipy.constants.k
```

In [4]:
```
#Define black body function, taking in a lambda array for the x-axis and a given temperature T in
 Kelvin
def B_lambda(lam, T):
    return ((2*h*c**2)/(lam**5))*(1/(np.exp((h*c)/(lam*k*T))-1))

#Set up figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(12,8))

#Set up lambda array for x, and set of discrete temperatures to plot functions for, with correspon
ding colours for the curves
lams = np.arange(1e3, 30e3)*1.e-10
Ts = [3000, 4000, 5000, 5778, 6000]
colours = ["brown", "red", "orange", "green", "blue"]

#Iterate over the data and plot the curves
for T, col in zip(Ts, colours):
    ax.plot(lams, B_lambda(lams, T), color=col)
    ax.text(1e-6, B_lambda(1e-6, T), str(T)+" K", size=14, color=col)

#Plot dashed lines and text to indicate the visible spectrum, as well as infrared and ultraviolet
 regions
vislow = 390e-9
vishigh = 700e-9
texty = 325e11
ax.axvline(x=vislow, color="k", ls="--")
ax.axvline(x=vishigh, color="k", ls="--")
ax.text((vislow+vishigh)/2, texty, 'Visible', horizontalalignment="center", color="k")
ax.text((min(lams)+vislow)/2, texty, 'Ultraviolet\n& beyond', horizontalalignment="center", vertic
alalignment="center", color="k")
ax.text((vishigh+max(lams))/2, texty, 'Infrared', horizontalalignment="center", color="k")

#Set the wavelength limits of the plot
ax.set_xlim(min(lams), max(lams))

#Set x ticks to be in units of microns
ticks = ax.get_xticks()*10e5
ax.set_xticklabels(ticks)

#Set y ticks to be for kW (x10^-3) and nm^-1 (x10^-9)
ticks = ax.get_yticks()*10e-12
ax.set_yticklabels(ticks)

ax.set_title("Planck black body radiation as a function of wavelength for a range of temperatures"
, size=16)
ax.set_xlabel("Wavelength $\lambda$ ($\mu m$)")
ax.set_ylabel("Spectral radiance ($kW\ sr^{-1}\ m^{-2}\ nm^{-1}$)")

ax.grid(lw=0.5)
```

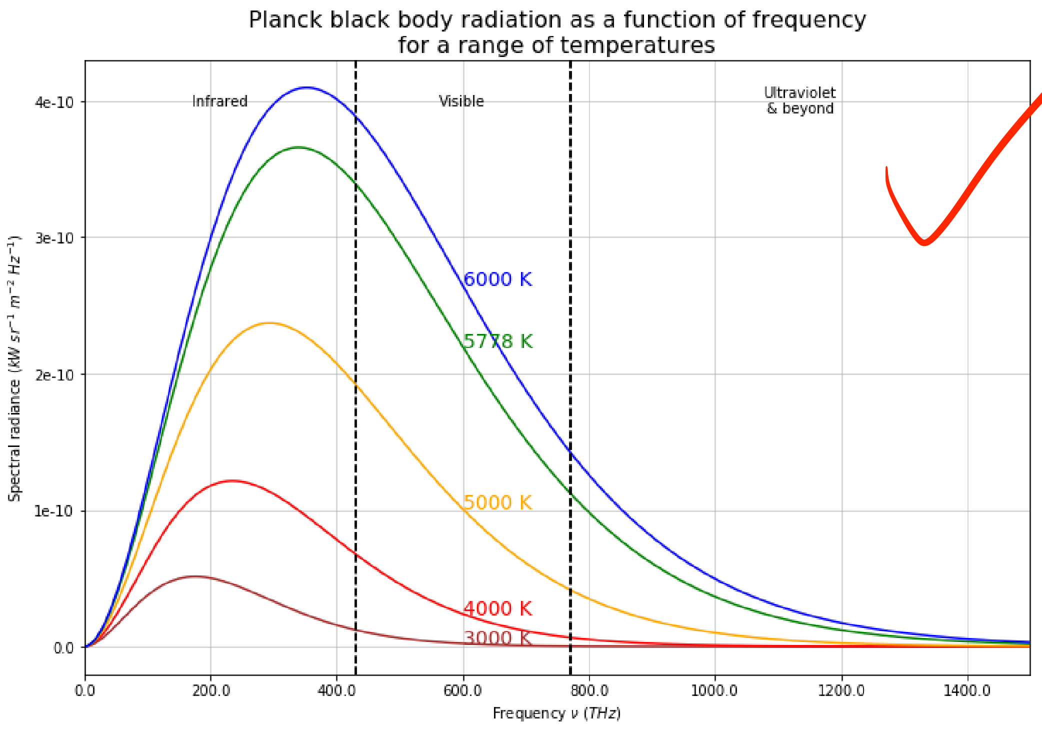


✓

Exercise 2

In a similar manner, plot the Planck Function as a function of frequency, I_ν, for the same temperatures as in the previous exercise. This time, sample frequency from around 0.01e15 – 1e15 Hz.

In [5]:
```
#Define function to generate black-body curve from frequency
def I_v(v, T):
    return ((2*h*v**3)/c**2)*(1/(np.exp((h*v)/(k*T))-1))

#Set up figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(12,8))

varr = np.arange(1, 1.5e15, 1e12)

#Iterate over the data and plot the curves
for T, col in zip(Ts, colours):
    ax.plot(varr, I_v(varr, T), color=col)
    ax.text(600e12, I_v(600e12, T), str(T)+" K", size=14, color=col)

#Plot dashed lines to indicate the visible light spectrum
ax.axvline(x=430e12, color="k", ls="--")
ax.axvline(x=770e12, color="k", ls="--")

#Plot dashed lines and text to indicate the visible spectrum, as well as infrared and ultraviolet
 regions
vislow = 430e12
vishigh = 770e12
texty = 4e-8
ax.axvline(x=vislow, color="k", ls="--")
ax.axvline(x=vishigh, color="k", ls="--")
ax.text((vislow+vishigh)/2, texty, 'Visible', horizontalalignment="center", verticalalignment="cen
ter", color="k")
ax.text((min(varr)+vislow)/2, texty, 'Infrared', horizontalalignment="center", verticalalignment=
"center", color="k")
ax.text((vishigh+max(varr))/2, texty, 'Ultraviolet\n& beyond', horizontalalignment="center", verti
calalignment="center", color="k")

#Set the frequency limits of the plot
ax.set_xlim(0, max(varr))

#Set x ticks to be in units of terahertz
ticks = ax.get_xticks()*1e-12
ax.set_xticklabels(ticks)

#Set y ticks to be for kW (x10^-3)
ticks = ax.get_yticks()*10e-3
ax.set_yticklabels(ticks)

ax.set_title("Planck black body radiation as a function of frequency\nfor a range of temperatures"
, size=16)
ax.set_xlabel("Frequency $\\nu$ ($THz$)")
ax.set_ylabel("Spectral radiance ($kW\ sr^{-1}\ m^{-2}\ Hz^{-1}$)")

ax.grid(lw=0.5)
```



✓

Exercise 3

Compute the *flux*, F (W/m ^{2}), radiating from the surface of a black body at temperature T:

$$F = \pi \int_0^\infty B_\lambda(T)\, d\lambda$$

(the factor of π comes from integrating $B_\lambda(T)$ over a hemisphere -- see Week 2 lectures). Do this integral numerically – i.e. using the `trapz` function in numpy to do the integral, instead of doing the math on paper...! To find out more about that function, type:

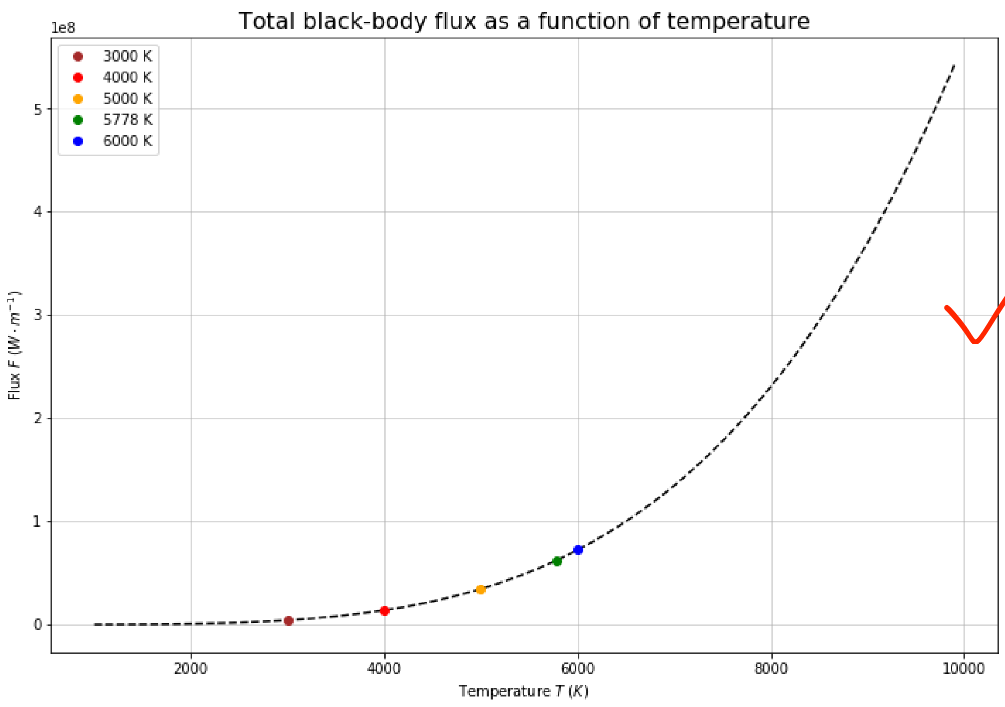
```
help(np.trapz)
```

Compute the integral for several different temperatures. If you haven't done this already, you may find it easier to define a function that will return the Planck law for a given temperature, so you can easily loop over a few different choices of T. You should end up with several values of the integral for different values of T. Plot these values. E.g.:

```
plt.plot(temp,flux,"o")
```

Here, `temp` is the array of temperatures, and `flux` is the array of integral values. The <font color=red>"o"</font> makes circular plotting symbols.

```
In [6]: fig, ax = plt.subplots(1, 1, figsize=(12,8))

        #Create an array of temperatures over which to compute the total flux of the Planck function
        temps = np.arange(1e3, 10e3, 1e2)

        #Step over the temperatures and compute the flux for each, saving these in the fluxarr array
        fluxarr = []
        for T in temps:
            fluxarr.append(np.pi*np.trapz(B_lambda(lams, T),lams))

        #And plot the fluxes as a function of temperature
        ax.plot(temps, fluxarr, color="k", ls="--")

        #Also plot the key temperatures used previously for reference
        for T, col in zip(Ts, colours):
            ax.plot(T, np.pi*np.trapz(B_lambda(lams, T),lams), color=col, lw=0, marker="o", label=str(T)+"
         K")

        ax.set_title("Total black-body flux as a function of temperature", size=16)
        ax.set_xlabel("Temperature $T$ ($K$)")
        ax.set_ylabel("Flux $F$ ($W\cdot m^{-1}$)")
        ax.legend()
        ax.grid(lw=0.5)
```



✓

Excercise 4

Now fit a polynomial to these points of flux versus temperature. Fit the polynomial using the `np.polyfit` function. This function returns the coefficients of a polynomial of a given order. To show the fit, you must recreate the polynomial using those coefficients. E.g.:

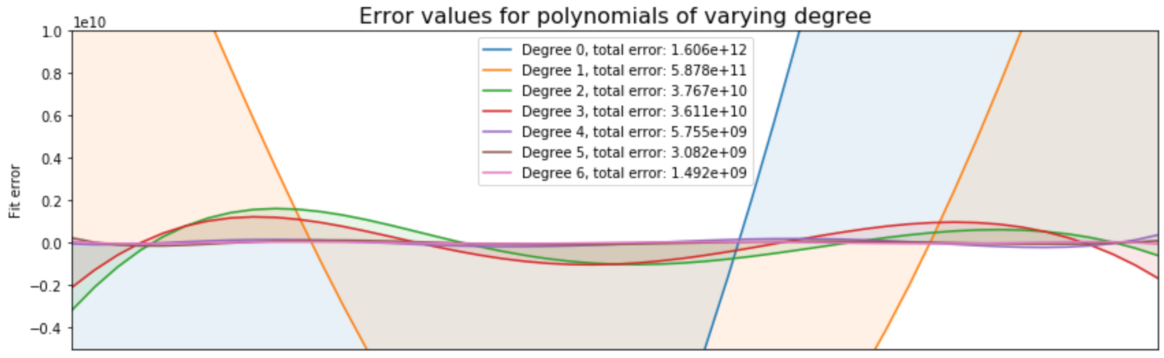
```
res = np.polyfit(x,y,2) # 2 = order of fit (quadratic in this case)
fity = res[0]*x**2 + res[1]*x + res[2] # res = coefficients of the polynomial
plot(x,fity)
```

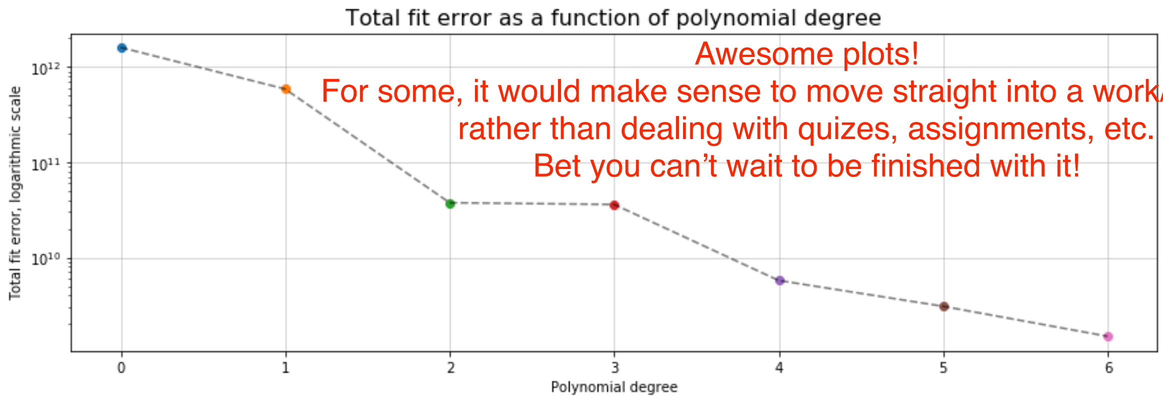
What is the order of the best-fit polynomial for these data? Note – in order to see the shape of the points clearly, make sure you have spanned a good range in temperature, e.g. at least 1000-50000 K. How can you decide which order to use?

When you have decided on the best polynomial order, use this as an exponent, γ, of the temperature, and re-plot your values of the integral against T^γ. This should show a linear relationship that shows $F = \alpha T^\gamma$. What is the name of the relation based on this proportionality? Estimate the value of α using `np.polyfit` with order = 1. How does your value of α compare with the expected constant of proportionality?

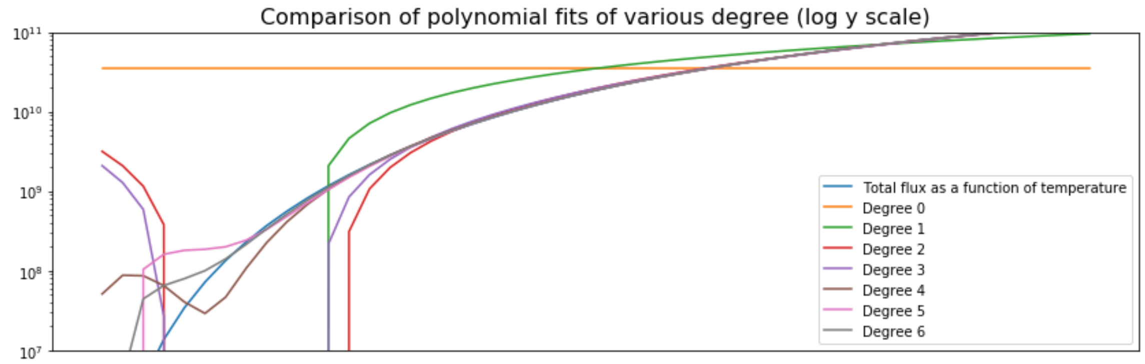
<font color=blue>In order to quantitatively determine the best fit polynomial degree we can step through several degrees and compare the resulting best fit polynomials. Here the absolute value of the difference between the found fit and the computed total flux for each temperature has been used to examine the error for the fit. By looking at the total error as a function of degree, we can find a polynomial that most closely fits the function in question.</font>

In [7]:
```python
fig, ax = plt.subplots(3, 1, figsize=(12,12))

#Set an array of a wide range of temperatures
temps = np.arange(1e3, 50e3, 1e3)

#Compute the total flux for each temperature
fluxarr = []
for T in temps:
    fluxarr.append(np.pi*np.trapz(B_lambda(lams, T), lams))

ax[2].plot(temps, fluxarr, label="Total flux as a function of temperature")

#Create an empty array to store the errors between the function and the polynomial fits
errorarray = []

#Step through several integers, using these as the polynomial degree
maxdegree = 7
for degree in np.arange(0, maxdegree):
    #Compute a best fit polynomial of the current degree
    coeff = np.polyfit(temps, fluxarr, deg=degree)

    #Create an array with the y-values of the polynomial for each temperature x-value
    yfit = 0
    for i in range(degree+1):
        yfit += coeff[i]*(temps**(degree-i))
    ax[2].plot(temps, yfit, label="Degree %.0f" % degree)

    #Compute the error corresponding to the fit and store it in the array
    remainders = fluxarr-yfit
    error = np.sum(np.abs(remainders))
    errorarray.append(error)

    #Plot the error curve in the top subplot, and the total error value in the bottom subplot
    ax[0].plot(temps, remainders, label="Degree %.0f, total error: %.3e" %(degree, error))
    ax[0].fill_between(temps, remainders, alpha=0.1)
    ax[1].plot(degree, error, marker="o")

ax[0].set_xlim(min(temps), max(temps))
ax[0].set_ylim(-5e9, 1e10)
ax[0].set_title("Error values for polynomials of varying degree", size=16)
ax[0].set_xticks([])
ax[0].set_ylabel("Fit error")
ax[0].legend()

ax[1].plot(np.arange(0, maxdegree), errorarray, color="k", ls="--", alpha=0.5)
ax[1].set_title("Total fit error as a function of polynomial degree", size=16)
ax[1].set_xlabel("Polynomial degree")
ax[1].set_ylabel("Total fit error, logarithmic scale")
ax[1].set_yscale("log")
ax[1].grid(lw=0.5)

ax[2].set_title("Comparison of polynomial fits of various degree (log y scale)", size=16)
ax[2].set_xticks([])
ax[2].set_yscale("log")
ax[2].set_ylim(10e6, 10e10)
ax[2].legend()

plt.tight_layout()
```



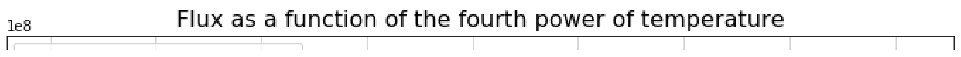


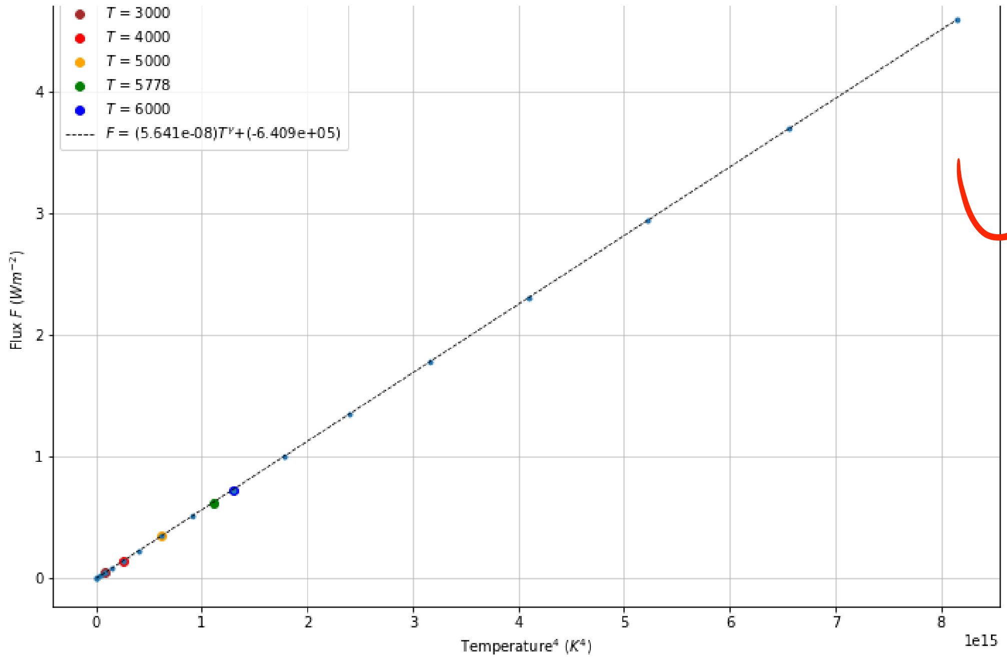
Awesome plots!
For some, it would make sense to move straight into a work/research rather than dealing with quizes, assignments, etc.
Bet you can't wait to be finished with it!



<font color=blue>While the lowest errors were found for polynomials of degree greater than or equal to 4, further experimentation (such as plotting flux as a function of temperature to higher powers and looking at linear fits below) revealed that a polynomial of degree 4 was the best fit. As higher degrees can simply have 0 or a very low coefficient of any higher powers of T, these are of necessarily equal or better fit than one of degree 4. A relationship of flux with T^4 is expected from the theory of the Stefan-Boltzmann Law. Below we look at the close linear relationship of flux as a function of the fourth power of temperature.</font>

```
In [8]: #Set the degree to 4, which was decided to have the best fit to the data, as above
        degree = 4

        #Set up lambda array for x, and set of discrete temperatures to plot functions for, with correspon
        ding colours for the curves
        lams = np.arange(1e3, 30e3)*1.e-10
        Ts = [3000, 4000, 5000, 5778, 6000]
        colours = ["brown", "red", "orange", "green", "blue"]

        fig, ax = plt.subplots(1, 1, figsize=(12,8))

        #Plot the key temperatures from earlier as a reference
        for T, col in zip(Ts, colours):
            ax.plot(T**degree, np.pi*np.trapz(B_lambda(lams, T),lams), color=col, lw=0, marker="o", label=
        "$T$ = %.0f" % T)

        #Set temperatures to use for x-values
        temps = np.arange(1e3, 10e3, 5e2)

        #Create an array of fluxes for y-values
        fluxarr2 = []
        for T in temps:
            fluxarr2.append(np.pi*np.trapz(B_lambda(lams, T),lams))

        #Raise the temperatures to the fourth power and plot the fluxes against them
        ax.plot(temps**degree, fluxarr2, lw=0, marker=".")

        #Fit a 1-degree polynomial (straight line) to the resulting plot
        coeff, var = np.polyfit(temps**degree, fluxarr2, deg=1, cov=True)

        #Create a y-value array for this linear fit
        xfit = temps
        yfit = coeff[0]*(xfit**degree)+coeff[1]

        #Plot the linear fit
        ax.plot(xfit**degree, yfit, ls="--", lw=0.8, color="k", label="$F$ = (%.3e)$T^{\gamma}$+(%.3e)" %
        (coeff[0], coeff[1]))

        ax.set_title("Flux as a function of the fourth power of temperature", size=16)
        ax.set_xlabel("Temperature$^4$ ($K^4$)")
        ax.set_ylabel("Flux $F$ ($W m^{-2}$)")
        ax.legend()
        ax.grid(lw=0.5)
```





✓

```
In [9]: print("Result from numerical calculation: %.6e" % coeff[0])

        Result from numerical calculation: 5.641086e-08
```

Exercise 5

Now calculate the integral of the Planck Function B_λ analytically. Hint:

$$0_\infty \frac{u^3 du}{e^u - 1} = \frac{\pi^4}{15}$$

You should now have an expression for the integral (which is equivalent to a total flux) as a function of temperature. What is the proportionality with temperature? Is it linear with T? Does this proportionality agree with what you determined numerically above?

<font color=blue>Analytic solution: $\pi 0_\infty B_\lambda d\lambda = \frac{2k^4}{h^3c^2}\frac{\pi^5}{15}T^4$</font> ✓

```
In [10]: sigma = (2*(k**4)*(np.pi**5))/(15*(h**3)*(c**2))
```

```
In [11]: print("Result from analytic calculation: %.6e" % sigma)

         Result from analytic calculation: 5.670367e-08
```

```
In [12]: sigmaerror = np.abs(coeff[0]-sigma)
         print("Difference between numerical and analytic calculations of sigma: %.6e" % sigmaerror)

         sigmaerrorpercent = 100*(sigmaerror/sigma)
         print("Percentage error between numerical and analytic calculations of sigma: %.3f%%" % sigmaerror
         percent)

         Difference between numerical and analytic calculations of sigma: 2.928086e-10
         Percentage error between numerical and analytic calculations of sigma: 0.516%
```

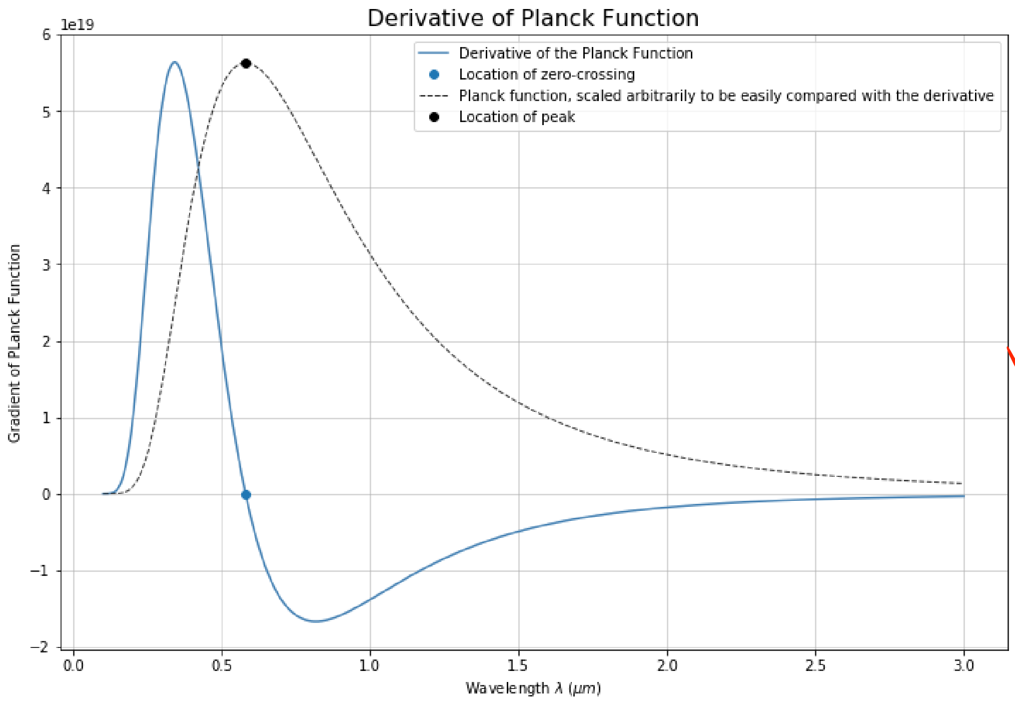
Exercise 6

Now consider the *derivative* of the Planck Function. Again, how would you compute the derivative of the Planck Function in python? Use your method to establish the wavelength of maximum intensity, λ_{max}, for a range of temperatures. Plot this peak wavelength for the different temperatures you tried. [Hint: You might find the `np.roll` function useful]

<font color=blue>In order to analyse the derivative of the function, we can define small steps dx over which to compute the function's slope. These then form the curve of the derivative. By looking at where this derivative equals zero, we can find the point at which the function peaks before decreasing again. To do this it is possible to consider the absolute value of the derivative function, so that all values are greater than or equal to zero, and find the wavelength at which the lowest value (closest to zero) can be found. Analytically the exact zero crossing can be found, but numerically there is no guarantee that one of the data points is exactly at zero, and so such a method gets a "close enough" result.</font>

```
In [13]: #Set temperature
         T = 5000

         #Set up x, dx, and y values
```

```
lams = np.arange(1000, 30000)*1.e-10
dx = lams[1]-lams[0]
y = B_lambda(lams, T)

#Generate dy/dx array, with values at steps of dx
dydx = []
for x in lams:
    dydx.append((B_lambda(x+dx, T)-B_lambda(x, T))/dx)

#Set up a plot
fig, ax = plt.subplots(1, 1, figsize=(12,8))

#Set arbitrary scalign factor to show the original Planck function alongside its derivative
scale = 4.4e6

#Plot derivative
ax.plot(lams, dydx, color="steelblue", label="Derivative of the Planck Function")
#Locate zero-crossing of the derivative and put a marker there
peaklam = (1000+np.argmin(np.abs(dydx)))*1.e-10
ax.plot(peaklam, 0, marker="o", lw=0, label="Location of zero-crossing")
#Plot scaled original function
ax.plot(lams, scale*B_lambda(lams, T), ls="--", color="k", lw=1, alpha=0.8,
        label="Planck function, scaled arbitrarily to be easily compared with the derivative")
#Plot the corresponding location of the peak (from zero-crossing)
ax.plot(peaklam, scale*B_lambda(peaklam, T), lw=0, marker="o", color="k", label="Location of peak"
)

#Set x ticks to be in units of microns
ticks = ax.get_xticks()*10e5
ax.set_xticklabels(ticks)

ax.set_title("Derivative of Planck Function", size=16)
ax.set_xlabel("Wavelength $\lambda$ ($\mu m$)")
ax.set_ylabel("Gradient of PLanck Function")
ax.legend()
ax.grid(lw=0.5)
```

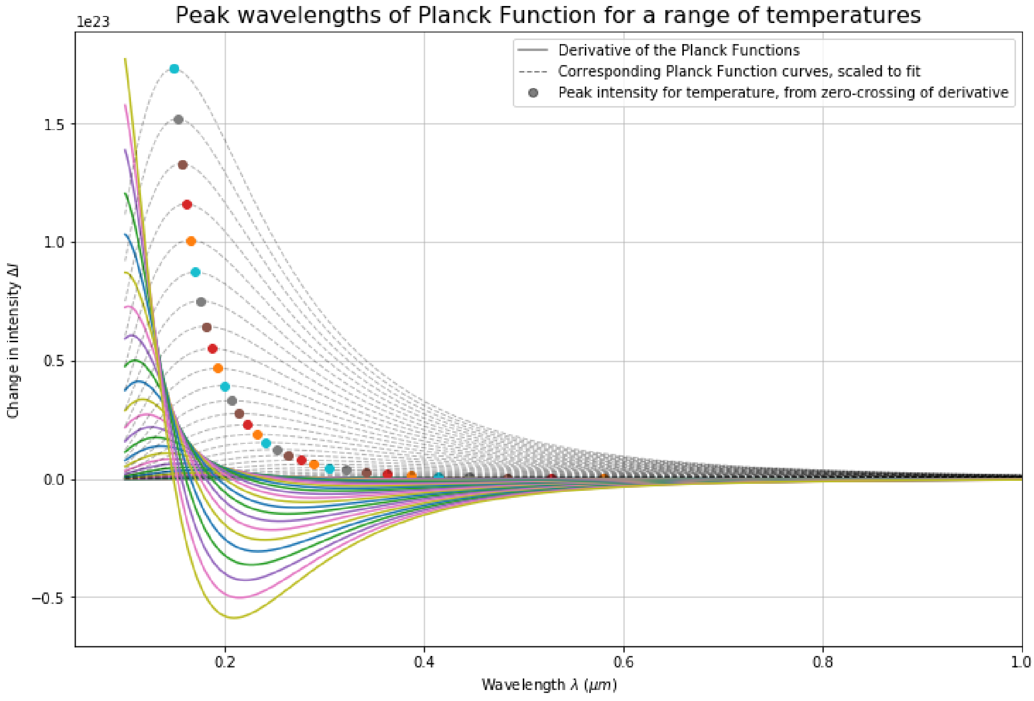


✓

<font color=blue>From the derivative of the Planck Function, we can locate the zero-crossing (as above) and thus also the peak value of the function and the wavelength at which the peak occurs. Now we repeat the above for a range of temperatures:</font>

In [14]:
```
fig, ax = plt.subplots(1, 1, figsize=(12,8))

lams = np.arange(1000, 10000)*1.e-10
dx = lams[1]-lams[0]

#Set temperature range
Ts = np.arange(5e3, 20e3, 5e2)

scale = 1.5e7

peaklams = []
for T in Ts:
    #Generate an array of y-values
    y = B_lambda(lams, T)

    #Generate dy/dx array, with values at steps of dx
    dydx = []
    for x in lams:
        dydx.append((B_lambda(x+dx, T)-B_lambda(x, T))/dx)
    ax.plot(lams, dydx)
    ax.plot(lams, scale*B_lambda(lams, T), ls="--", color="k", lw=1, alpha=0.3)
    peaklam = (1000+np.argmin(np.abs(dydx)))*1.e-10
    ax.plot(peaklam, scale*B_lambda(peaklam, T), marker="o")
    peaklams.append(peaklam)

ax.plot(0,0, color="k", alpha=0.5, label="Derivative of the Planck Functions")
ax.plot(0,0, ls="--", color="k", lw=1, alpha=0.5, label="Corresponding Planck Function curves, sca
led to fit")
ax.plot(0,0, lw=0, color="k", alpha=0.5, marker="o", label="Peak intensity for temperature, from z
ero-crossing of derivative")

ax.set_xlim(min(lams)/2, max(lams))
ticks = ax.get_xticks()*10e5
ax.set_xticklabels(ticks)
ax.set_xlabel("Wavelength $\lambda$ ($\mu m$)")
ax.set_ylabel("Change in intensity $\Delta I$")

ax.legend()
ax.grid(lw=0.5)
ax.set_title("Peak wavelengths of Planck Function for a range of temperatures", size=16)
```

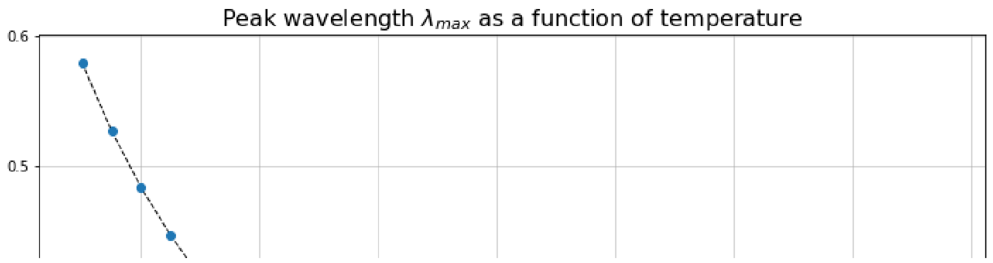
```
Out[14]: Text(0.5,1,u'Peak wavelengths of Planck Function for a range of temperatures')
```

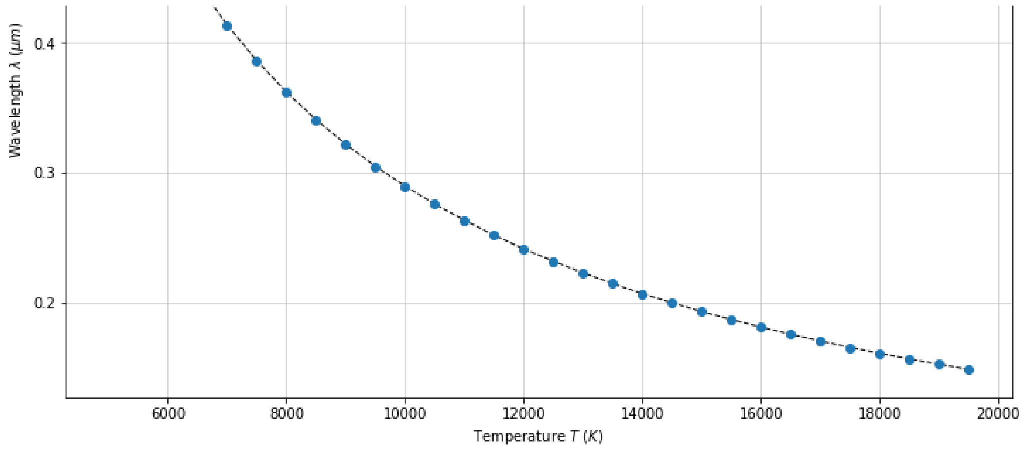


✓

<font color=blue>With this array of wavelengths at which the Planck Function peaks as a function of wavelength, we can examine the relationship between them:</font>

```
In [15]: fig, ax = plt.subplots(1, 1, figsize=(12,8))

         #Set temperature range
         Ts = np.arange(5e3, 20e3, 5e2)

         #Plot the points corresponding to temperature range, and plot a dashed line between them
         ax.plot(Ts, peaklams, marker=None, lw=1, ls="--", color="k")
         ax.plot(Ts, peaklams, marker="o", lw=0)

         ax.set_title("Peak wavelength $\lambda_{max}$ as a function of temperature", size=16)
         ax.set_xlabel("Temperature $T$ ($K$)")
         ax.set_ylabel("Wavelength $\lambda$ ($\mu m$)")
         ticks = ax.get_yticks()*10e5
         ax.set_yticklabels(ticks)
         ax.grid(lw=0.5)
```





<font color=blue>Clearly the wavelength of maximum intensity decreases as a function of black-body temperature, but the relationship between them is not clear from this analysis. This curve could be matched with a number of different functions.</font>

Exercise 7

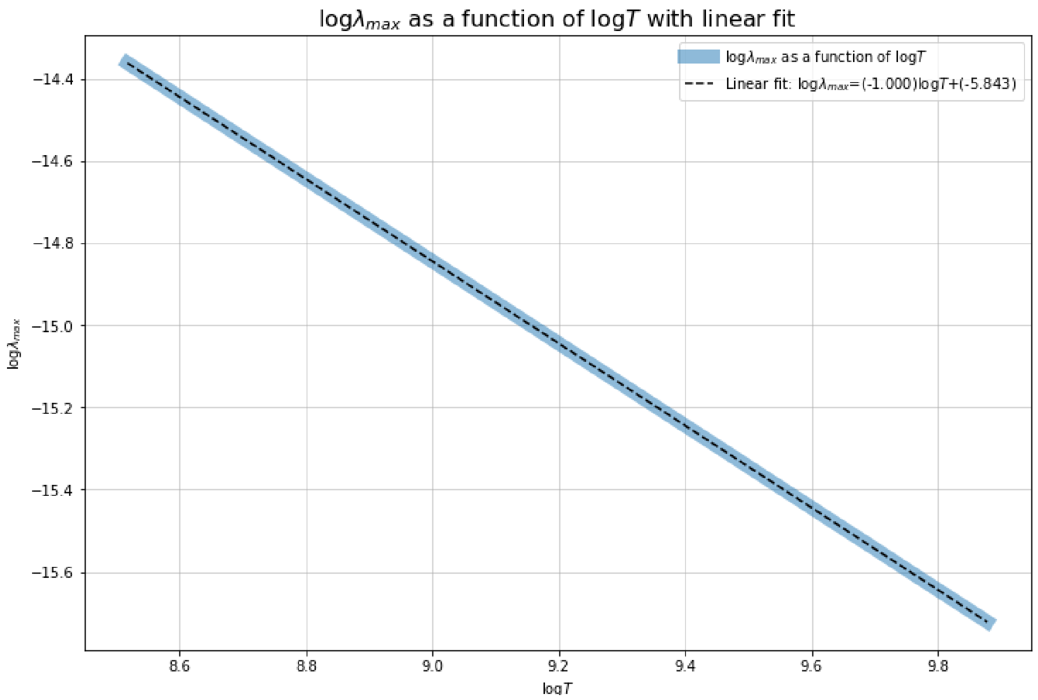
This time we'll use a different approach to analyse the trend. Is your Temperature ($T$) versus peak intensity wavelength ($\lambda_{\mathrm{m}ax}$) graph linear? Make the plot logarithmic by plotting $\log(\lambda_{\mathrm{m}ax})$ against $\log(T)$. How does it look now? The gradient of this relation gives the relevant exponent in the linear relationship between $\lambda_{\mathrm{m}ax}$ and $T$, i.e.:

$$\log(\lambda_{\mathrm{m}ax}) = \alpha \log(T) + b \Rightarrow \lambda_{\mathrm{m}ax} \propto T^{\alpha}$$

Compute the linear gradient of the log-log plot using polyfit as before. Replot your data as $(\lambda_{\mathrm{m}ax})^{\alpha}$ versus $T$, and in doing so, verify Wein's law.

```
In [16]: logpeaklams = np.log(peaklams)
         logTs = np.log(Ts)

         fig, ax = plt.subplots(1, 1, figsize=(12, 8))

         ax.plot(logTs, logpeaklams, lw=10, alpha=0.5, label="log$\lambda_{max}$ as a function of log$T$")

         coeff = np.polyfit(logTs, logpeaklams, deg=1)
         yfit = coeff[0]*logTs+coeff[1]

         ax.plot(logTs, yfit, ls="--", color="k", label="Linear fit: log$\lambda_{max}$=(%.3f)log$T$+(%.3f
         )" % (coeff[0], coeff[1]))

         ax.grid(lw=0.5)
         ax.set_title("log$\lambda_{max}$ as a function of log$T$ with linear fit", size=16)
         ax.set_xlabel("log$T$")
         ax.set_ylabel("log$\lambda_{max}$")
         ax.legend()
```

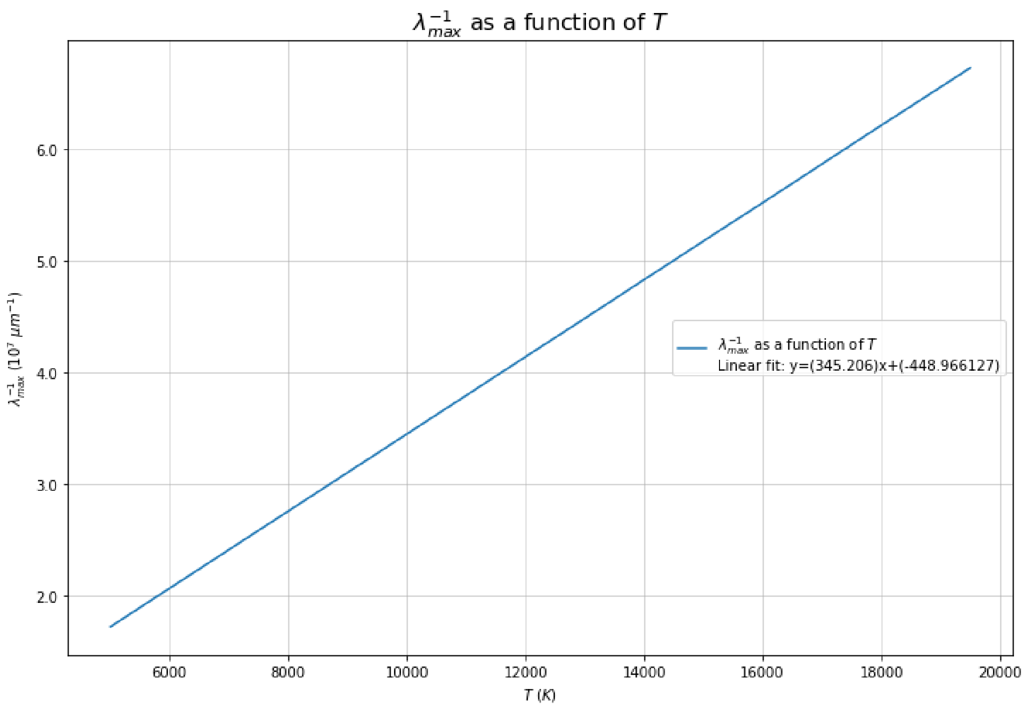
```
Out[16]: <matplotlib.legend.Legend at 0xcea07f0>
```



<font color=blue>The linear fit reveals a gradient of -1 for logλ_{max} as a function of log$T$, meaning that $\lambda_{max} \propto T^{-1}$, or $\lambda_{max} \propto \frac{1}{T}$</font>

```
In [17]: fig, ax = plt.subplots(1, 1, figsize=(12, 8))

         invpeaklams = 1/np.asarray(peaklams)
         coeff = np.polyfit(Ts, invpeaklams, deg=1)
         ax.plot(Ts, invpeaklams, label="$\lambda_{max}^{-1}$ as a function of $T$\nLinear fit: y=(%.3f)x+(
         %.6f)" % (coeff[0], coeff[1]))

         ax.grid(lw=0.5)
         ax.set_title("$\lambda_{max}^{-1}$ as a function of $T$", size=16)
         ax.set_xlabel("$T$ ($K$)")
         ax.set_ylabel("$\lambda_{max}^{-1}$ ($10^7\ \mu m^{-1}$)")
         yticks = ax.get_yticks()*10e-7
         ax.set_yticklabels(yticks)
         ax.legend(loc="center right")
```

```
Out[17]: <matplotlib.legend.Legend at 0xbafab70>
```



<font color=blue>The linear fit to this plot gives us b, Wien's displacement constant, if we take the inverse of the coefficient of x:</font>

```
In [18]: print("Wien's displacement constant, b = %.6e" % (1/coeff[0]))

         Wien's displacement constant, b = 2.896817e-03
```

<font color=blue>λ_{max} being inversely proportional to $T$ is expected from the theory and is as Wien's law states, with $\lambda_{max} \propto \frac{b}{T}$, and the factor b being roughly equal to $2.897\text{x}10^{-3} m \cdot K$.</font>

Exercise 8

Now find an analytic expression for $\lambda_{\mathrm{m}ax}$ using the analytic derivative of the Planck Function. You will not be able to find a closed-form solution for the equation, so use python to solve it iteratively and so find an expression for the wavelength of maximum intensity, $\lambda_{\mathrm{m}ax}$, as a function of temperature, $T$. Does this agree with your findings in Exercise 7?

<font color=blue>$\frac{d}{d\lambda} B_\lambda(T) = \frac{-5}{\lambda^6} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1} + \frac{1}{\lambda^5} \frac{-e^{\frac{hc}{\lambda kT}} \frac{hc}{kT} \frac{-1}{\lambda^2}}{(e^{\frac{hc}{\lambda kT}} - 1)^2}$

This expression is equal to zero at the peak wavelength. Solving for the zero points give the following:

$$\lambda T = \frac{hc}{5k(1 - e^{\frac{-hc}{\lambda kT}})}$$

✓

or

$$\lambda T = \frac{a}{5(1 - e^{\frac{-a}{\lambda T}})}$$, with $a = \frac{hc}{k_B}$

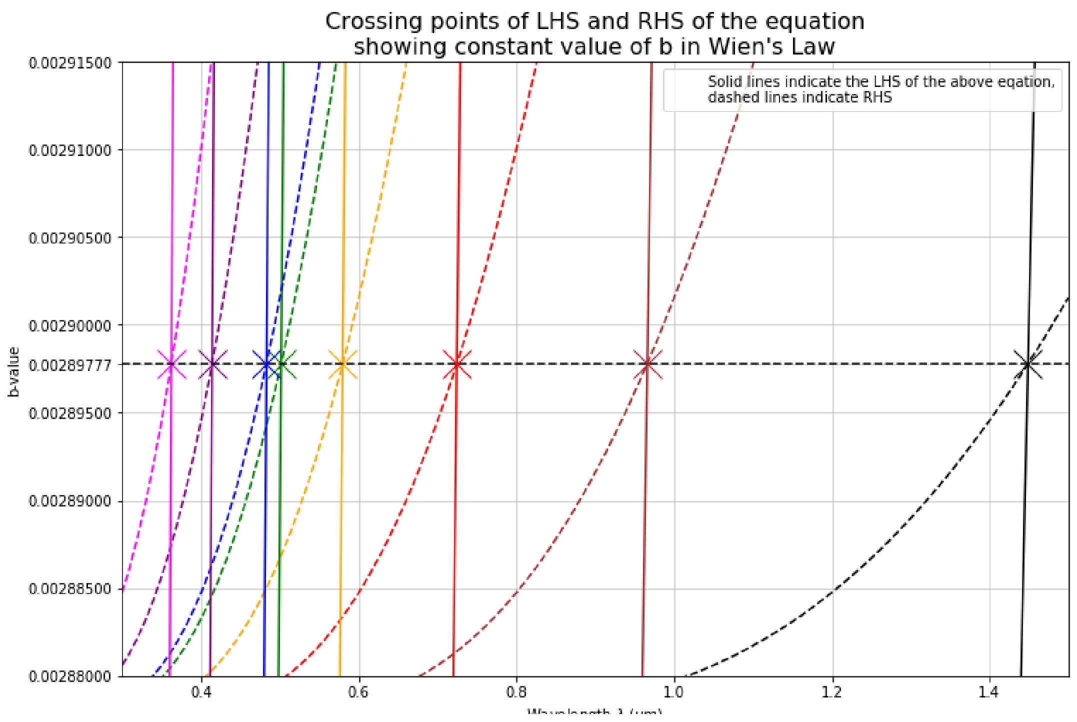
$5(1 - e^{\lambda T})$ k_B

</font>

In [19]:
```
#Take out the constants and replace with 'a'
a = (h*c)/k

#Define a function for the right-hand side of the above equation
def lambda_peak(lam, T):
    return a/(5*(1-np.exp((-1*a)/(lam*T))))

#Set up lamdas array for x-axis, temperatures, and colours
lams = np.arange(3e3, 18e3)*1.e-10
Ts = [2000, 3000, 4000, 5000, 5778, 6000, 7000, 8000]
colours = ["black", "brown", "red", "orange", "green", "blue", "purple", "magenta"]

fig, ax = plt.subplots(1, 1, figsize=(12,8))

#Step through the temperatures and plot both sides of the equation in the same colour, one line da
shed
ycrosses = []
for T, col in zip(Ts, colours):
    ax.plot(lams, lams*T, color=col)
    ax.plot(lams, lambda_peak(lams, T), color=col, ls="--")

    #Locate the crossing point by finding the minimal point of the absolute value of the differenc
e between the RHS and LHS
    xcross = (3e3+np.argmin(np.abs(lams*T - lambda_peak(lams,T))))*1.e-10
    #Plug this x value into the RHS to get the corresponding y value
    ycross = lambda_peak(xcross,T)
    ycrosses.append(ycross)
    #Plot these crossing points with an 'X'
    ax.plot(xcross, ycross, marker="x", markersize=20, color=col)

ax.plot(0, 0, lw=0, label="Solid lines indicate the LHS of the above eqation,\ndashed lines indica
te RHS")
#Find the mean value of all the crosses
b_avg = np.mean(ycrosses)
print(b_avg)

ax.set_title("Crossing points of LHS and RHS of the equation\nshowing constant value of b in Wie
n's Law", size=16)
ax.set_xlabel("Wavelength $\lambda$ ($\mu m$)")
ax.set_ylabel("b-value")

ax.set_xlim(3e-7, 15e-7)
ax.set_xticklabels(ax.get_xticks()*10e5)

ax.set_ylim(0.00288, 0.002915)
ax.set_yticks(np.append(ax.get_yticks(), b_avg))

ax.axhline(y=b_avg, ls="--", color="k")
ax.grid(lw=0.5)
ax.legend()
```

```
0.00289777371605
```

Out[19]:
```
<matplotlib.legend.Legend at 0xa3f4160>
```



✓

Wavelength λ (μm)

<font color=blue>By looking at the line crossings between λT and $\frac{hc}{5k(1-e^{\frac{-hc}{\lambda kT}})}$ and comparing the absolute value of the two lines' difference, the position in x of the lowest difference can be found (again, there is no guarantee that data point on either line lie exactly on this crossing point, so the closest minima must be used). This x-crossing can then be plugged into either side of the equation to find the corresponding y-value. In Wien's Law, this y-value is a constant, and indeed here we see that all of these crossings lie on the same horizontal line. By taking the mean value over the crossing points for several temperatures, we get a close approximation of Wien's displacement constant, with $b = 2.8977737x10^{-3}$ </font>

Exercise 9

Compute a relation between the colour index $(B - V)$ and the temperature T of a black body. Recall that:

$$(B - V) = -2.5\log\left(\frac{\int F_\lambda S_B d\lambda}{\int F_\lambda S_V d\lambda}\right) + const$$

Where F is the flux, and S is the filter transmission. We will use our own absolute system here, so we neglect the additive constant term (i.e. const = 0 above). We also simplify things by assuming that the B and V filters can be approximated as 'top-hat' functions of unity transmission, centred on $\lambda = 440$nm and $\lambda = 550$nm respectively, with widths given by:

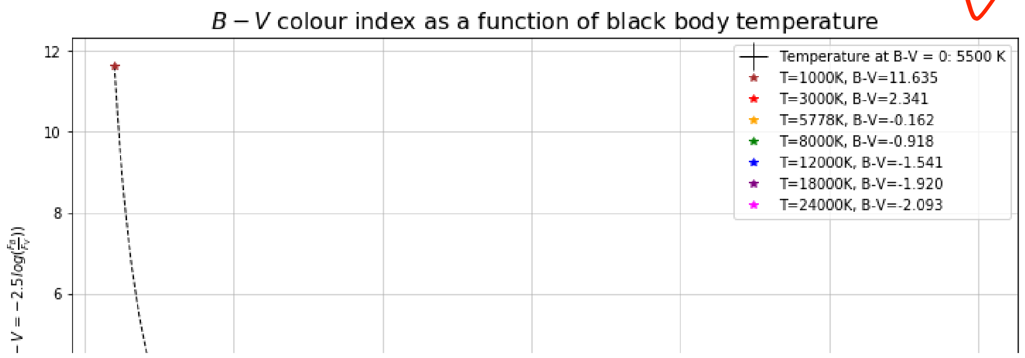
$$\Delta\lambda_B = 98\text{n}m$$
$$\Delta\lambda_V = 89\text{n}m$$

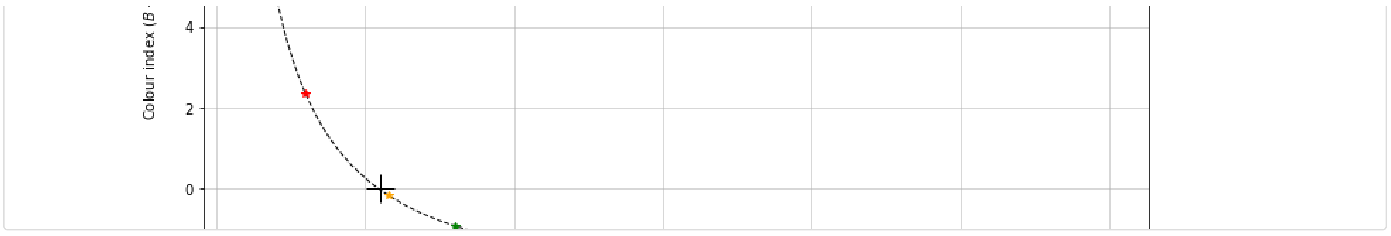
Log base 10 is always used, which is np.log10 in python

For example, the $(B - V)$ computed this way for a temperature of 42,000K = -0.98.

From your relation, at which temperature do objects change from 'red' or 'blue' for this colour index?

```
In [44]: def BminusV(T, Bcenter=440., deltaB=98., Vcenter=550., deltaV=89.):
             Blams = np.arange((Bcenter-(deltaB/2)), (Bcenter+(deltaB/2)))*1.e-9
             Bflux = np.trapz(B_lambda(Blams, T), Blams)

             Vlams = np.arange((Vcenter-(deltaV/2)), (Vcenter+(deltaV/2)))*1.e-9
             Vflux = np.trapz(B_lambda(Vlams, T), Vlams)

             return -2.5*np.log(Bflux/Vflux)

         fig, ax = plt.subplots(1, 1, figsize=(12,8))

         #Set up lambda array for x, and set of discrete temperatures to plot functions for, with correspon
         ding colours for the curves
         lams = np.arange(1e3, 30e3)*1.e-10
         Ts = np.arange(1e3, 30e3, 1e2)
         keyTs = [1000, 3000, 5778, 8000, 12000, 18000, 24000]
         colours = ["brown", "red", "orange", "green", "blue", "purple", "magenta"]

         BminusVarray = []
         for T in Ts:
             BminusVarray.append(BminusV(T))
         ax.plot(Ts, BminusVarray, color="k", ls="--", lw=1)

         crossX = (1000+np.argmin(np.abs(BminusVarray))*100)
         ax.plot(crossX, 0, marker="+", markersize=20, lw=0, color="k", label="Temperature at B-V = 0: %.0f
          K" % crossX)

         for T, col in zip(keyTs, colours):
             ax.plot(T, BminusV(T), marker="*", lw=0, color=col, label="T=%.0fK, B-V=%.3f" % (T, BminusV(T
         )))

         ax.set_title("$B-V$ colour index as a function of black body temperature", size=16)
         ax.set_xlabel("Temperature $T$ ($K$)")
         ax.set_ylabel("Colour index ($B-V$ = $-2.5log(\\frac{F_B}{F_V})$")
         ax.grid(lw=0.5)
         ax.legend()

Out[44]: <matplotlib.legend.Legend at 0x1567d358>
```




Colour index (B
4
2
0