In [1]:
```
import numpy as np
import scipy
import os
import pdb
import matplotlib.pyplot as plt
%matplotlib inline
```

Solar flux and surface brightness

1. The solar constant is a flux measuring the mean solar radiation at the Earth.

a. What are the dimensions (i.e., in terms of length, mass, time, energy, etc)?

b. In MKS/SI, what are the units?

c In cgs, what are the units?

d. Calculate its value, given the luminosity of the sun is about 3.83×1033 ergs/s and the distance to the sun is (on average) 1 au = 1.496×1013cm?

2. What is the surface brightness of the sun as seen from Earth (note that the Sun has an angular diameter of about 0.5 degrees)? Give your results per sterradian and per square arcsecond.
3. a. What is the solar constant at Jupiter? How is it related to the solar constant at Earth?

   b. What is the surface brightness of the Sun as seen from Jupiter? How is it related to the surface brightness of the Sun as seen from Earth?

In [2]:
```
Lsun=3.83e33
d=1.496e13
Fsun=Lsun/4/np.pi/d**2
print('solar constant {:e} ergs/cm2/s'.format(Fsun))
```

```
solar constant 1.361839e+06 ergs/cm2/s
```

Given angular diameter of Sun is about 0.5 degrees, calculate surface brightness both per square arcsec and per sterradian

In [3]:
```
area=np.pi*(0.5/2)**2
print('area in square degrees: ',area)
sb=Fsun/area
print('surface brightness per square degree {:e}'.format(sb))
print('surface brightness per square arcsec: ', sb/3600./3600.)

area=area*(np.pi/180)**2
print('surface brightness per sterradian {:e}:'.format(Fsun/area))
```

```
area in square degrees:  0.19634954084936207
surface brightness per square degree 6.935788e+06
surface brightness per square arcsec:  0.5351688553420817
surface brightness per sterradian 2.276885e+10:
```

How about at Jupiter?

In [4]:
```
print('flux at jupiter {:e}'.format(Fsun/5.2**2))
print('surface brightness at Jupiter is the same')
```

```
flux at jupiter 5.036386e+04
surface brightness at Jupiter is the same
```

In [5]:
```
17113859/area
180/0.25**2

1*(180/np.pi)**2
```

Out[5]:
```
3282.806350011744
```

Monochromatic flux

1. The monochromatic flux of Vega at 5500 A is F_λ=3.63x10-9 ergs/cm2/s/A

   Calculate the monochromatic flux F_v of Vega at 5500 A in units of ergs/cm^2/s/Hz

In [6]:
```
flam=3.63e-9
c=3.e10
lam=5500
fnu=lam**2/c*flam/1.e8
print(fnu)
```

```
3.66025e-20
```

Binaries and magnitudes

1. A binary star system consists of two stars of equal brightness. If the magnitude of one star is m, what is the combined magnitude of the two stars, i.e., if they were observed as an unresolved system?

2. Roughly half of the stars in our Galaxy are in binary systems. What effect do you think this has on a HR diagram of a star cluster?
3. Imagine you are observing a binary star system, but where the stars are so close together that you see them as a single object. If the two stars have magnitudes of 17 and 18 individually, what is the combined observed magnitude?
4. Write a (short!) software function that takes as input the apparent magnitude of each of two stars, and computes and returns the apparent magnitude of the two stars combined.

In [7]:
```
dm=2.5*np.log10(2)
print('combined mag = m - {:f}'.format(dm))
```

```
combined mag = m - 0.752575
```

In [8]:
```
def binary(m1,m2) :
    return -2.5*np.log10(10**(-0.4*m1)+10**(-0.4*m2))

print(binary(17,18))
```

```
16.636148842226767
```

Flux and distance

Monochromatic magnitudes

STMAG monochromatic magnitudes are defined as $mSTMAG$=−2.5log$F\lambda$−21.1, for $F\lambda$ in ergs/cm2/s/A

ABNU monochromatic magnitudes are defined as: mAB=−2.5logFv−48.6 for Fv in ergs/cm2/s/Hz

1. Given that these systems are approximately normalized to the flux of Vega at 5500 A (3.63x10-9 ergs/cm2/s/A), where do the constants (-21.1 and -48.6) come from?
2. Derive a relation between STMAG and ABNU mag, i.e., the difference between monochromatic magnitudes in the two systems as a function of wavelength. Be careful about units.

In [9]:
```
print(-2.5*np.log10(3.63e-9))
print(-2.5*np.log10(3.63e-20))
```

```
21.100233437409717
48.60023343740971
```

$$m_{st} - m_{ab} = -2.5logF_\lambda - 21.1 + 2.5logF_\nu + 48.6$$
$$m_{st} - m_{ab} = -2.5logF_\lambda/F_\nu + 27.5$$
$$m_{st} - m_{ab} = -2.5logc/\lambda^2 + 27.5$$
$$m_{st} - m_{ab} = -2.5logc + 2.5log\lambda(cm) + 2.5log\lambda(A) + 27.5 = 5log\lambda + 1.307$$
$$m_{st} - m_{ab} = -2.5logc + 5log\lambda(A) - 2.5 * 8 + 27.5 = 5log\lambda(A) - 18.69$$

In [10]:
```
print(-2.5*np.log10(3.e10)+27.5 - 20)

print(5*np.log10(5500))
print(5*np.log10(8000)-18.69)
```

```
-18.692803136799157
18.701813447471217
0.8254499349597175
```

Magnitude-flux conversion

1. Using the effective wavelengths and Vega flux zeropoints from Bessell et al (1998), e.g. as tabulated at http://www.astronomy.ohio-state.edu/~martini/usefuldata.html (http://www.astronomy.ohio-state.edu/~martini/usefuldata.html), (Links to an external site.) write a software function that will take as input an array of observed UBVRI magnitudes and return three arrays, giving the effective wavelength, Fv, and $F\lambda$ of the object at the effective wavelengths of the UBVRI filters.
2. A star has the following apparent magnitudes in the UBVRI system. Use your routine to complete the following table and to plot the derived spectral energy distribution (SED). Remembering (or looking up) Wien's law, estimate the effective temperature of this star!

Bandpass	Magnitude	Fv	$F\lambda$	λeff
U	16.74			
B	16.86			
V	16.17			
R	15.72			
I	15.28			

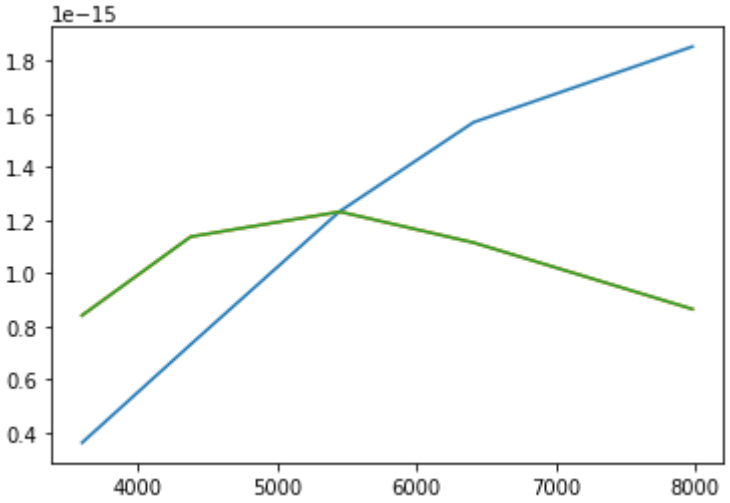
In [11]:
```
from astropy.modeling import models
from astropy import units as u
teff=5500
bb=models.BlackBody1D(teff*u.K)

def ubvritoflux(mags) :
    w=np.array([3600,4380,5450,6410,7980])
    bb(w*u.AA)
    fnu0=np.array([1.79,4.063,3.636,3.064,2.416])*1.e-20
    flam0=np.array([417.5,632,363.1,217.7,112.6])*1.e-11
    scale=10.**(-0.4*mags)
    return w, fnu0*scale, flam0*scale

ubvritoflux(np.zeros(5))
```

Out[11]:
```
(array([3600, 4380, 5450, 6410, 7980]),
 array([1.790e-20, 4.063e-20, 3.636e-20, 3.064e-20, 2.416e-20]),
 array([4.175e-09, 6.320e-09, 3.631e-09, 2.177e-09, 1.126e-09]))
```

In [12]:
```
teff=5500
bb=models.BlackBody1D(teff*u.K)
w=np.array([3600,4380,5450,6410,7980])
bb(w*u.AA)
f0=np.array([1.79,4.063,3.636,3.064,2.416])*10e-20
fl0=np.array([417.5,632,363.1,217.7,112.6])*1.e-11
mag=-2.5*np.log10(bb(w*u.AA).value/f0)+25
plt.plot(w,bb(w*u.AA).value)
plt.plot(w,fl0*10**(-0.4*mag))
print(mag)

w,fnu,flam = ubvritoflux(mag)
plt.plot(w,flam)
```

```
[16.74021369 16.86198462 16.17402522 15.72662965 15.28725531]
```

Out[12]: `[<matplotlib.lines.Line2D at 0x11ca79290>]`



Photon flux and count equation

The monochromatic flux of Vega at 5500 A is F_λ =3.63x10-9 ergs/cm2/s/A.

1. What is the monochromatic photon flux (photons/cm2/s/A) from Vega at 5500 A?
2. a. Estimate the number of photons/second you would receive for a star with a flat $F\lambda$ spectrum with V=20 using the TMO 0.6m (diameter) telescope, assuming that the atmosphere transmits 80% of the light, and that the combined detection efficiency of the telescope and detector is 50 percent, independent of wavelength. You can assume that the V bandpass is rectangular with a central wavelength of 5500 A, a full width of 850 A, with an in-band transmission of 80 percent.

   b. Given your derived count rate, what is the zeropoint (Z) for the V bandpass for the 06m?

In [13]:
```
vega=3.63e-9/6.63e-27/3.e10*5500e-8
print(vega)
```

```
1003.7707390648569
```

In [14]:
```
m=20
counts=10.**(-0.4*m)*vega*850*.8*.5*.8*np.pi*30**2
print(counts)
z=m+2.5*np.log10(counts)
print(z)
```

```
7.719618132790185
22.218989543933947
```

In [ ]:
```
```

Photometric transformations

You observe three stars in the V filter. Two of them have known magnitudes and colors, while the third is unknown.

Star	V	B-V	V exposure time (s)	V total counts	B exposure time	B total counts
A	16	0	10	10000	20	6000
B	17	1	10	4000	20	1000
C	?	?	10	2600	20	700

1. What are the zeropoint and transformation coefficients for the B and V filters? (Note that you can do polynomial fits using numpy.polyfit() (https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html))
2. What are the B and V magnitudes of star C? (Note that you can do linear algebra with < a href=https://numpy.org/doc/stable/reference/routines.linalg.html> (https://numpy.org/doc/stable/reference/routines.linalg.html>) numpy.linalg(), e.g., numpy.linalg.solve() (https://numpy.org/doc/stable/reference/routines.linalg.solve.html).)

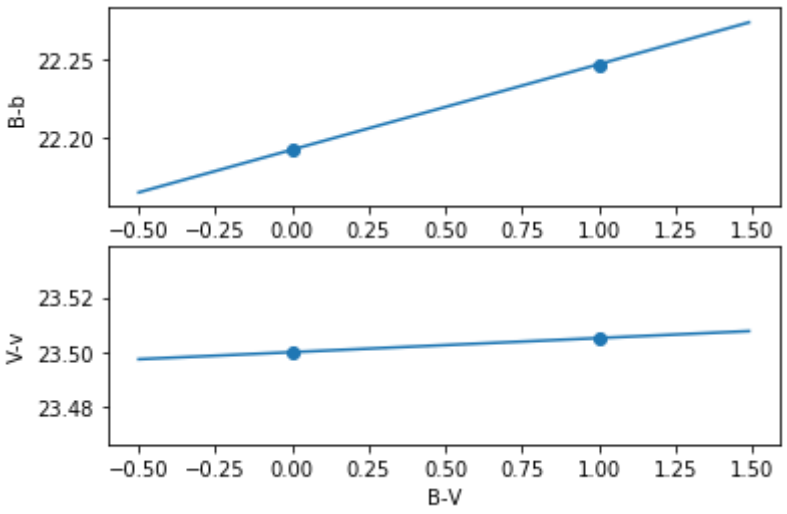
instrumental magnitudes, b and v, are -2.5 log (counts/s)

$$B = b + t_B(B-V) + z_B$$
$$V = v + t_V(B-V) + z_V$$

$$B - b = t_B(B-V) + z_B$$
$$V - v = t_V(B-V) + z_V$$

In [15]:
```
# load data into arrays
vstan=np.array([16,17])
bvstan=np.array([0,1])
bstan=vstan+bvstan

vinst=-2.5*np.log10(np.array([10000,4000])/10)
binst=-2.5*np.log10(np.array([6000,1000])/20)

# get the B coefficients and plot data and fit
bcoef=np.polyfit(bvstan,bstan-binst,1)
print('bcoef: ',bcoef)
plt.subplot(2,1,1)
plt.scatter(bvstan,bstan-binst)
plt.ylabel('B-b')
bv=np.arange(-0.5,1.5,0.01)
plt.plot(bv,bcoef[0]*bv+bcoef[1])

# get the V coefficients and plot data and fit
vcoef=np.polyfit(bvstan,vstan-vinst,1)
plt.subplot(2,1,2)
plt.scatter(bvstan,vstan-vinst)
plt.xlabel('B-V')
plt.ylabel('V-v')
plt.plot(bv,vcoef[0]*bv+vcoef[1])
print('vcoef: ',vcoef)
```

```
bcoef:  [ 0.05462187 22.19280314]
vcoef:  [5.14997832e-03 2.35000000e+01]
```



Going back to the original photometry table (phot), now we want to solve for the standard magnitudes of star 1. We have

$$B = b + t_B(B - V) + z_B$$

$$V = v + t_V(B - V) + z_V$$

This isn't quite as trivial as plugging in stuff on the right hand side, because the right hand side uses the *standard* magnitudes, not the observed ones. Still, it is two equations for two unknowns. You can do the algebra to solve for B and V, or, probably easier, set it up as a linear set of equations and solve using matrix arithmetic. With unknowns on the left:

$$B - t_B(B - V) = b + z_B$$
$$V - t_V(B - V) = v + z_V$$

$$\begin{pmatrix} 1 - t_B & t_B \\ -t_V & 1 + t_V \end{pmatrix} \begin{pmatrix} B \\ V \end{pmatrix} = \begin{pmatrix} b + z_B \\ v + z_V \end{pmatrix}$$

In [16]:
```
# set up the lhs square matrix, which is the same for all objects
vinst=-2.5*np.log10(np.array([10000,4000,2600])/10)
binst=-2.5*np.log10(np.array([6000,1000,700])/20)

lhs=np.matrix([[1-bcoef[0],bcoef[0]],[-vcoef[0],1+vcoef[0]]])
print('lhs: ')
print(lhs)
rhs=np.matrix([binst+bcoef[1],vinst+vcoef[1]])
print('rhs: ')
print(rhs)

out=(np.linalg.solve(lhs,rhs))
print('out: ')
print(out)
print('B-V: ',out[0,2]-out[1,2])
```

```
lhs:
[[ 0.94537813  0.05462187]
 [-0.00514998  1.00514998]]
rhs:
[[16.         17.94537813 18.33263303]
 [16.         16.99485002 17.46256663]]
out:
[[16.         18.         18.38263119]
 [16.         17.         17.46728067]]
B-V:  0.9153505213930124
```

Star	V	B-V	V exposure time (s)	V total counts	B exposure time	B total counts
A	16	0	10	10000	20	6000
B	17	1	10	4000	20	1000
C	18.38	0.915	10	2600	20	700

Exposure time calculator

The next cell demonstrates components for a signal calculator using functions. These just implement some simple possible inputs, but can be generalized to accept additional inputs.

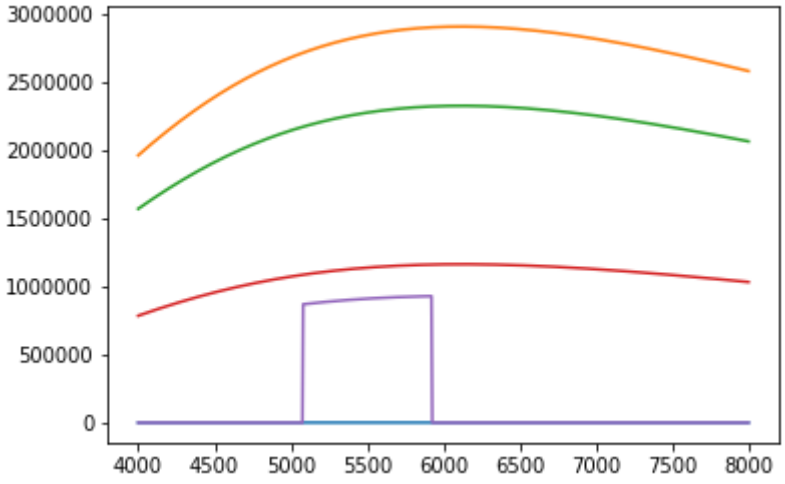
In [17]:
```
from astropy.modeling import models
from astropy import units as u
import astropy.constants as c
from astropy.modeling.models import BlackBody

@u.quantity_input(wave=u.AA,teff=u.K)
def sed(wave,type='bb',teff=6000*u.K) :
    """ routine to return photon spectrum of desired type at desired wavele
    """
    if type == 'bb' :
        bb = BlackBody(temperature=teff)
        norm= 3.63e-9*u.erg/u.cm**2/u.s/u.AA / \
              (bb(5500*u.AA)*u.sr*c.c/(5500*u.AA)**2).to(u.erg/u.cm**2/u.s/
        return (bb(wave)*c.c/wave**2*wave/c.h/c.c*u.sr).to(1/u.cm**2/u.s/u.
    else :
        raise ValueError('Input type {:s} not yet implemented'.format(type)

@u.quantity_input(wave=u.AA)
def telescope(wave,diameter=100*u.cm,name=None,trans=0.9,eps=0.):
    """ return telescope area * throughput at input wavelengths
        if name is specified, read information from file, else use
        simple prescription with input diameter=, trans=, and eps=
        eps is the fractional radius of a central obscuration
    """
    area = (np.pi*(diameter/2*(1-eps))**2).to(u.cm**2)
    if name==None :
        out = np.ones(len(wave)) * trans
        return area*out
    else :
        raise ValueError('Name {:s} not yet implemented'.format(name))

@u.quantity_input(wave=u.AA)
def filt(wave,name=None,cent=5500*u.AA,wid=1000*u.AA,trans=0.8) :
    """ return filter throughput at input wavelengths
        if name is specified, read from a file, else use simple
        square filter profile with cent=, wid=, and trans= keywords
    """
    if name==None :
        out = np.zeros(len(wave))
        out[np.where((wave>cent-wid/2)&(wave<cent+wid/2))] = trans
        return out
    else :
        raise ValueError('Name {:s} not yet implemented'.format(name))

@u.quantity_input(wave=u.AA)
def inst(wave,name=None,trans=0.8) :
    """ return instrument throughput at input wavelengths
        if name is specified, read from file, else use simple
        single transmission with value from trans= keyword
    """
    if name==None :
        out = np.ones(len(wave)) * trans
        return out
    else :
        raise ValueError('Name {:s} not yet implemented'.format(name))
```

```
@u.quantity_input(wave=u.AA)
def det(wave,name=None,trans=0.8) :
    """ return detector throughput at input wavelengths
    """
    if name==None :
        out = np.ones(len(wave)) * trans
        return out
    else :
        raise ValueError('Name {:s} not yet implemented'.format(name))


def readfile(name,columns=['col1','col2']) :
    """ function to read input data file
    """
    try: dat=ascii.read(name)
    except FileNotFoundError :
        print('File {:s} not found'.format(file))
    except:
        raise ValueError('Error in readfile for {:s}'.format(file))
    return dat[columns[0]],dat[columns[1]]
```

Example using functions for V=20, 6000K blackbody, with 0.6m telescope, instrument transmission 0.8 (proxy for atmosphere), detector transmission 0.5, filter transmission 0.8 in square bandpass centered at 5500 A, width 850A

In [18]:
```
# set desired wavelength array
wave=np.arange(4000,8000,5)*u.AA

# add the various terms one by one and plot
vmag=0
photflux=10.**(-0.4*vmag)*sed(wave,teff=6000*u.K)
plt.plot(wave,photflux)
photflux*=telescope(wave,diameter=0.6*u.m,trans=1)
plt.plot(wave,photflux)
photflux*=inst(wave,trans=0.8)
plt.plot(wave,photflux)
photflux*=det(wave,trans=0.5)
plt.plot(wave,photflux)
photflux*=filt(wave,trans=0.8,cent=5500*u.AA,wid=850.*u.AA)
plt.plot(wave,photflux)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x11cc92390>]
```



If this is an imaging instrument, we'd want to integrate over the spectrum to get the signal. Can do this with np.trapz (or other alternatives)

In [19]:
```
np.trapz(photflux,wave)
```

Out[19]: $7.6507656 \times 10^{8} \ \frac{1}{\mathrm{s}}$

OK, now let's try an implementation using object oriented programming, which makes a lot of sense for this application. We can add extra flexibility here.

In [20]:
```
class Object:
    """ Class representing an object
        Given a type (blackbody or input spectrum) and a magnitude,
        provides method for Fnu, Flam, photon flux
    """
    def __init__(self,type='bb',teff=10000,mag=0,refmag='V'):
        self.type = type
        self.teff = teff
        self.mag = mag
        self.refmag = refmag

    @u.quantity_input(wave=u.m)
    def sed(self,wave):
        """ Return sed in Fnu or Flambda, with units (depending on source c
        """
        if self.type == 'bb' :
            bb = BlackBody(temperature=self.teff*u.K)
            norm= 3.63e-9*u.erg/u.cm**2/u.s/u.AA / \
                  (bb(5500*u.AA)*u.sr*c.c/(5500*u.AA)**2).to(u.erg/u.cm**2/
            return bb(wave)*norm*u.sr*10.**(-0.4*self.mag)
        else :
            raise ValueError('Input type {:s} not yet implemented'.format(t

    @u.quantity_input(wave=u.m)
    def photflux(self,wave):
        """ Return SED in photon flux"""
        return (self.flam(wave)*wave/c.h/c.c).to(1/u.cm**2/u.s/u.AA)

    @u.quantity_input(wave=u.m)
    def flam(self,wave):
        """ Return SED in Flambda"""
        sed=self.sed(wave)
        if sed.unit.is_equivalent(u.erg/u.cm**2/u.s/u.AA) :
            # sed is already Flambda
            return sed
        elif sed.unit.is_equivalent(u.erg/u.cm**2/u.s/u.Hz) :
            # convert from Fnu to Flambda
            return (sed*c.c/wave**2).to(u.erg/u.cm**2/u.s/u.AA)
        else:
            print('sed has incorrect dimensions')

    @u.quantity_input(wave=u.m)
    def fnu(self,wave):
        """ Return SED in Fnu"""
        sed=self.sed(wave)
        if sed.unit.is_equivalent(u.erg/u.cm**2/u.s/u.Hz) :
            # sed is already Fnu
            return sed
        if sed.unit.is_equivalent(u.erg/u.cm**2/u.s/u.AA) :
            # convert from Flambda to Fnu
            return (sed*wave**2/c.c).to(1/u.cm**2/u.s/u.Hz)
        else:
            print('sed has incorrect dimensions')


class Telescope :
```

```
    """ Class representing a telescope
        Given a name (or diameter and mirror array), provide methods
        area
        throughput
    """
    def __init__(self,name='',diameter=1*u.m,mirrors=['Al']) :
        self.name = name
        if name == 'ARC3.5m' :
            self.diameter=3.5*u.m
            self.mirrors=['Al','Al','Al']
            self.eps=0.4
        elif name == 'TMO' :
            self.diameter=0.6*u.m
            self.mirrors=['Al','Al']
            self.eps=0.4
        elif name == '' :
            self.diameter=diameter
            self.eps=0.
            self.mirrors=mirrors
        else :
            raise ValueError('unknown telescope')

    def area(self) :
        """ Return telescope collecting area
        """
        return (np.pi*(self.diameter/2)**2*(1-self.eps**2)).to(u.cm**2)

    def throughput(self,wave) :
        """ Return telescope throughput
        """
        t=np.ones(len(wave))
        for mir in self.mirrors :
            if type(mir) is float :
                # if mirrors is a float, use it as a constant throughput (b
                t *= mir
            else :
                tmp = Mirror(mir)
                t *= tmp.reflectivity(wave)
        return t

class Mirror :
    """ class representing a mirror
        given coating name, provide method for reflectivity
    """
    def __init__(self,type,const=0.9) :
        self.type = type
        self.const = const

    def reflectivity(self,wave) :
        if self.type == 'const' :
            return np.ones(len(wave)) + self.const

        else :
            # read data file depending on mirror coating type
            try: dat=ascii.read(os.environ['ETC_DIR']+'/data/mirror/'+self.
            except FileNotFoundError :
                raise ValueError('unknown coating type: {:s}',self.type)
```

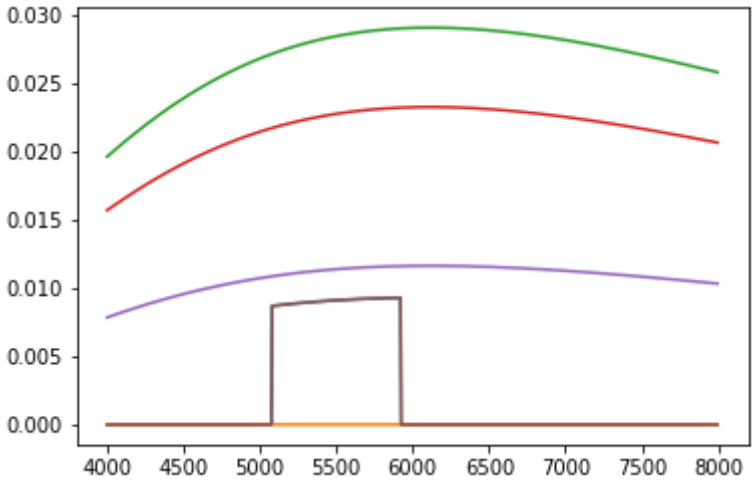
```
            wav=dat['col1']
            ref=dat['col2']/100.
            interp=scipy.interpolate.interp1d(wav,ref)
            return interp(wave.to(u.nm).value)

class Instrument :
    """ class representing an Instrument
    """
    def __init__(self,name='',efficiency=0.8) :
        self.name=name
        self.efficiency=efficiency
        self.detector = Detector(efficiency=1.)

    def throughput(self,wave) :
        if self.name == '' :
            return np.ones(len(wave))*self.efficiency
        else :
            raise ValueError('need to implement instrument {:s}',self.name)

    def filter(self,wave,filter='',cent=5500*u.AA,wid=850*u.AA,trans=0.8) :
        if filter=='' :
            out = np.zeros(len(wave))
            out[np.where((wave>cent-wid/2)&(wave<cent+wid/2))] = trans
            return out
        else :
            try: dat=ascii.read(os.environ['ETC_DIR']+'/data/inst/'+self.na
            except FileNotFoundError :
                raise ValueError('cant find filter file: {:s} for instrumen
                                 filter, self.name)

            raise ValueError('Name {:s} not yet implemented'.format(name))

class Detector :
    """ class representing a detector
    """
    def __init__(self,name='',efficiency=0.8) :
        self.name=name
        self.efficiency=efficiency

    def throughput(self,wave) :
        if self.name == '' :
            return np.ones(len(wave))*self.efficiency
        else :
            raise ValueError('need to implement detector {:s}',self.name)


class Atmosphere :
    """ class representing the Earth's atmosphere
    """
    def __init__(self,name='',transmission=0.8) :
        self.name=name
        self.transmission=transmission

    def throughput(self,wave) :
        if self.name == '' :
            return np.ones(len(wave))*self.transmission
        else :
```

```
            raise ValueError('need to implement atmosphere {:s}',self.name)

class Observation :
    """ Object representing an observation
    """
    def __init__(self,obj=None,atmos=None,telescope=None,instrument=None,wa
        self.obj = obj
        self.atmos = atmos
        self.telescope = telescope
        self.instrument = instrument
        self.wave=wave

    def photonflux(self) :
        """ Return photon flux
        """
        photflux = (obj.photflux(self.wave)*
                self.atmos.throughput(self.wave) *
                self.telescope.area()*self.telescope.throughput(self.wave)*
                self.instrument.throughput(self.wave)*self.instrument.filte
        return photflux

    def counts(self) :
        """ Return integral of photon flux
        """
        return np.trapz(self.photonflux(),self.wave)
```

In [21]:
```
os.environ['ETC_DIR'] = '/Users/holtz/classes/ay535/fall21/etc/'
```

In [22]:
```
# wavelength array to do calculations on
wave=np.arange(4000,8000,5)*u.AA

# set up object, telescope, instrument, atmosphere
obj=Object(type='bb',teff=6000,mag=20)
tel=Telescope(diameter=0.6*u.m,mirrors=[1.0])
inst=Instrument(efficiency=0.5)
atmos=Atmosphere(transmission=0.8)

obs=Observation(obj=obj,atmos=atmos,telescope=tel,instrument=inst,wave=wave
plt.plot(wave,obs.photonflux())
print(obs.counts())

#individual components for plotting, demonstrating use of object methods
photflux=obj.photflux(wave)
plt.plot(wave,photflux)
photflux*=tel.throughput(wave)*tel.area()
plt.plot(wave,photflux)
photflux*=atmos.throughput(wave)
plt.plot(wave,photflux)
photflux*=inst.throughput(wave)
plt.plot(wave,photflux)
photflux*=inst.filter(wave,trans=0.8)
plt.plot(wave,photflux)

counts=np.trapz(photflux,wave)
print(counts)
```

```
7.650765577073331 1 / s
7.650765577073331 1 / s
```



In [ ]:

In [ ]:

In [ ]: