Rudimentary Model Atmosphere

Olof van den Berg
Arjen Siegers

28th October 2003

3 Code for a Model

For the third part of the project we combined our strengths because of the complexity of the problem, and because we else wouldn't have been able tot finish the third part in time. For the programming we used Python, because we knew it best and it is a clear and more simpler language, though it gets slow in large calculations, such as in the part 3c and 3d.

3.1 The emergent flux

For the emergent flux in the simple Grey case, we determine the Planck function at $\tau = \frac{2}{3}$. From the lecture notes, chapter 5, we can write the flux as a function of the Planck function: $F_\lambda = \pi B(\lambda, T_{eff})$. Here, the effective temperature is the temperature found at $\tau = \frac{2}{3}$ this corresponds with $Log(\tau) = -0.1761$. From the table we find that this corresponds roughly with a temperature of 11036 K. The resulting graph is plotted in Figure 1. The relevant Python code is found in the appendix.

3.2 Opacities

3.2.1 H_{bf} vs. λ

For this part we made use of the book by Gray. For $\kappa(H_{bf})$ it gave (formula 8.5)

$$\kappa(H_{bf}) = \alpha_0 \lambda^3 10^{-\theta\chi} \sum_{n=1}^{5} \frac{1}{n^3} \tag{1}$$

in which the Gaunt factor has been taken unity and θ = 5040/T and χ = 13.6 eV $(1 - \frac{1}{n^2})$, with the extra factor due to the fact that the energy changes at the different levels. $\alpha_0 = 1.044 \times 10^{-26}$ for λ in ångstroms. We evaluated equation 1 for τ = 0.001, 2/3, and 10. To do so we took the equivalent temperatures from the data-table, because there is no τ-dependence in the function. The corresponding temperatures, as found in the table were: 7572.3 K, 11036.0 K and 16052.2 K. This resulted in the Figures 2,3 and 4.

3.2.2 H^-_{bf} vs. λ

For this part we also made use of the book by Gray. For $\kappa(H^-_{bf})$ it gave (formula 8.12)

$$\kappa(H^-_{bf}) = 4.158 \times 10^{-10} \alpha_{bf} P_e \theta^{\frac{5}{2}} 10^{0.754\theta} \tag{2}$$

with units of cm^2 per neutral hydrogen atom, and

$$\begin{aligned} \alpha_{bf} &= 1.99654 - 1.18267 \times 10^{-5}\lambda + 2.64243 \times 10^{-6}\lambda^2 - 4.40524 \times 10^{-10}\lambda^3 \\ &\quad +3.23992 \times 10^{-14}\lambda^4 - 1.39568 \times 10^{-18}\lambda^5 + 2.78701 \times 10^{-23}\lambda^6 \end{aligned}$$

This resulted in the Figures 5, 6 and 7.

3.2.3 Total k_λ vs. λ

Then these two results were combined and again plotted. The H^-_{bf} does not contribute much to the plot because the star is too hot for a large influence of H^-.
These results can be found in the Figures 8, 9 and 10.

3.3 Mean opacities

3.3.1 The Planck opacity

The Planck mean could be found in the book by Mihalas (equation 2-30)

$$\overline{\kappa}_P = \frac{\pi}{\sigma T^4} \int_0^\infty \kappa_\nu B_\nu(T) d\nu \tag{3}$$

Because it is quite impossible to do this integration, only numerical iteration is the best way to find a value for the Planck mean.

3.3.2 The Rosseland mean opacity

The Rosseland mean opacity is

$$\frac{1}{\overline{\kappa}_R} = \frac{\pi}{4\sigma T^3} \int_0^\infty \frac{1}{\kappa_\nu} \frac{dB_\nu(T)}{dT} d\nu \tag{4}$$

Doing the differentiation for $\frac{dB_\nu(T)}{dT}$ results in

$$\frac{1}{\overline{\kappa}_R} = \frac{\pi}{4\sigma T^3} \int_0^\infty \frac{1}{\kappa_\nu} \frac{2c^3h^2}{k\lambda^6 T^2} e^{\frac{hc}{\lambda kT}} \left(e^{\frac{hc}{\lambda kT}} - 1 \right)^{-2} d\nu \tag{5}$$

Because it is also quite impossible to do this integration, like before only numerical iteration is best the way to find a value for the Rosseland mean.
According to the lecture notes

$$\kappa(\nu, z) = -\frac{1}{\rho(z)} \frac{d\tau_\nu}{dz} \tag{6}$$

ρ is the same for both means, so the means are expressions of the gradient of τ over z. From the numbers we get by using the formulas it is clear that the

Planck mean opacity gives a larger gradient than the Rosseland mean opacity, which means that τ grows faster with z. The Planck mean gives a good view of the surface part of the atmosphere, while the Rosseland mean can be applied to the deeper layers of the atmosphere.

3.4 Incorporation

For the emergent flux over all optical depths we have rewritten the formula for the Planck Mean opacity,

$$\overline{\kappa}_P = \frac{\pi}{\sigma T^4} \int_0^\infty \kappa_\lambda B_\lambda(T) d\lambda \tag{7}$$

to

$$\sigma T^4 = \pi \frac{\int_0^\infty \kappa_\lambda B_\lambda(T) d\lambda}{\overline{\kappa}_P} = F_{emergent} \tag{8}$$

In part 3a we took for the definition of the emergent flux $F_\lambda = \pi B(\lambda, T_{eff})$, so this equation is the emergent flux weighted over all κ_λ's and the Planck mean opacity. Both incorporate all optical depths. To make the calculation faster we made an extra data-file for κ_P and the results we got from part 3c. We extended the number of points to 100.000, because that gave us a better view of the plot, since there were more jumps at higher wavelengths.
To plot these data, the quickest way was to just copy the results into a new data-file and read from that to make a plot quickly, because the calculation is very slow. Figure 11 shows a plot that looks a lot like the flux from the gray case, but on the right side of the top there a clear Hydrogen excitation level jumps, caused by the influence of $\kappa(H_{bf})$ in the equation. It looks somewhat similar to Figure 5.4 of the lecture notes, which we took as an example like which our graph should look like. We find that our result is reasonable, though more time could have given us better solutions.

3.5 Conclusion

The last question for this project was to determine what kind of star this model results in. From the effective temperature of 11036 K it can already be concluded that this star should be a late B star.

References

[1] Grey, D.F., 1992, *The Observation and Analysis of Stellar Photospheres*, Cambridge University Press, U.K. (second edition)

[2] Mihalas, D., 1978, *Stellar Atmospheres*, W. H. Freeman and Co., San Francisco (second edition)

Code

A

The code for the emergent flux looked like this:

```
#!/usr/local/share/bin/python2.2
from Numeric import *
from Kplot import *

x = arrayrange(1*math.pow(10,-5), 10*math.pow(10,-5), 1*10**-8)

def planck(T,x):

    h = 6.62606*10**-27
    k = 1.38066*10** -16
    c = 2.99792*10**10
    y =  ((2*h*c**2)/(x**5) *( 1/(e**((h*c) / (x*k*T))-1)))
    y = log10(y)
    return y

T=11036
y0 = planck(T, x)
function0 = Polyline(x,y0)
box = Box()
dev = Kplot.createDevice()
panel = dev.createPanel()
panel.setWorld((min(x),min(y0)) , (max(x),max(y0)))
panel.add(function0,box)
dev.output()
dev.close()
dev.hardcopy('emergent','ps')
```

B

The code for determining the opacity for H_{bf} and H^-_{bf} vs λ, and combined is:

```
#!/usr/local/share/bin/python2.2

from Kplot import *
from Numeric import *
import TableIO
import math

#First part:
reach = 20000.0
chii = 13.6
c = 2.99792458*10.0**10
lamb = arrayrange(1,reach)
```

```
def kappa_lambda(t,N):
    kappa_lambda1=arrayrange(1,reach)
    for i in range(len(lamb)):
        if (lamb[i] <= 2.43235*math.pow(10,3)):
            kappa_lambda1[i] = (1.044*math.pow(10,-26))
* 10**((-5040.0/t)*(0))*((math.pow(lamb[i],3))/1)
            kappa_lambda1[i] = kappa_lambda1[i]*N
        else:
            kappa_lambda1[i]=0

    kappa_lambda2=arrayrange(1,reach)
    for i in range(len(lamb)):
        if (lamb[i] <= 2.43235*math.pow(10,3)*4):
            kappa_lambda2[i] = (1.044*math.pow(10,-26))
* 10**((-5040.0/t)*(chii*(3/4)))*((math.pow(lamb[i],3))/8)
            kappa_lambda2[i] = kappa_lambda2[i]*N
        else:
            kappa_lambda2[i]=0

    kappa_lambda3=arrayrange(1,reach)
    for i in range(len(lamb)):

        if (lamb[i] <= 2.43235*math.pow(10,3)*9):
            kappa_lambda3[i] = (1.044*math.pow(10,-26))
* 10**((-5040.0/t)*chii*(8/9))*((math.pow(lamb[i],3))/27)
            kappa_lambda3[i] = kappa_lambda3[i]*N
        else:
            kappa_lambda3[i]=0

    kappa_lambda4=arrayrange(1,reach)
    for i in range(len(lamb)):
        if (lamb[i] <= 2.43235*math.pow(10,3)*16):
            kappa_lambda4[i] = (1.044*math.pow(10,-26))
* 10**((-5040.0/t)*chii*(15/16))*((math.pow(lamb[i],3))/64)
            kappa_lambda4[i] = kappa_lambda4[i]*N
        else:
            kappa_lambda4[i]=0

    kappa_lambda5=arrayrange(1,reach)
    for i in range(len(lamb)):
        if (lamb[i] <= 2.43235*math.pow(10,3)*25):
            kappa_lambda5[i] = (1.044*math.pow(10,-26))
* 10**((-5040.0/t)*chii*(24/25))*((math.pow(lamb[i],3))/125)
            kappa_lambda5[i] = kappa_lambda5[i]*N
        else:
            kappa_lambda5[i]=0
    kappa_lambda = kappa_lambda1+kappa_lambda2+kappa_lambda3+
+kappa_lambda4+kappa_lambda5
    return kappa_lambda
```

```
t = [7572.3,11036.0,16052.2]
Nj=[13.6020,14.6890,15.0986]
for i in range(len(t)):
    if(i==0):
        k_l1 = kappa_lambda(t[i],Nj[i])

    if(i==1):
        k_l2=kappa_lambda(t[i],Nj[i])

    if(i==2):
        k_l3=kappa_lambda(t[i],Nj[i])

#Second part:

P = [6.79371,552.193,2545.15]
T = [7572.3,11036.0,16052.2]
lamb = arrayrange(1,reach)

def holy_handgrenade(P,T):
    kappa_lambdaii = arrayrange(1,reach)
    alfa = arrayrange(1,reach)
    for i in range(len(lamb)):
        alfa[i] = 1.99654-(1.18267*math.pow(10,-5))*lamb[i]+
+(2.64243*math.pow(10,-6)*lamb[i]**2)+
-(4.40524*math.pow(10,-10)*lamb[i]**3)+
+(3.23992*math.pow(10,-14)*lamb[i]**4)+
-(1.39568*math.pow(10,-18)*lamb[i]**5)+
+(2.78701*math.pow(10,-23)*lamb[i]**6)
        kappa_lambdaii[i] = 4.158*math.pow(10,-10)*alfa[i]*
*P*math.pow((5040/T),5/2)*math.pow(10,0.754*(5040/T))*
*math.pow(10,-18)
        if (alfa[i-1]<= 0):
            alfa[i] = 0
            kappa_lambdaii[i] = 0
    return kappa_lambdaii

Nj1 = [3.68250*math.pow(10,9),9.06058*math.pow(10,6),3.02596*math.pow(10,7)]
for i in range(len(T)):
    if (i == 0):
        k_l1ii = holy_handgrenade(P[i],T[i])*Nj1[i]
    if (i==1):
        k_l2ii = holy_handgrenade(P[i],T[i])*Nj1[i]
    if (i==2):
        k_l3ii = holy_handgrenade(P[i],T[i])*Nj1[i]

#part three: summation

for i in range(len(T)):
    if (i==0):
        k_l1iii = k_l1 + k_l1ii
```

```
    if (i==1):
        k_l2iii = k_l2 + k_l2ii
    if (i==2):
        k_l3iii = k_l3 + k_l3ii

#The plotting section:
function1 = Polyline(lamb,k_l1)
function2 = Polyline(lamb,k_l2)
function3 = Polyline(lamb,k_l3)

function1ii = Polyline(lamb,k_l1ii)
function2ii = Polyline(lamb,k_l2ii)
function3ii = Polyline(lamb,k_l3ii)

function1iii= Polyline(lamb,k_l1iii)
function2iii= Polyline(lamb,k_l2iii)
function3iii= Polyline(lamb,k_l3iii)
box1 = Box()
box1.leftaxis.textangle = pi/2
dev1 = createDevice()
xlab = Label('lambda', 'bottom')
ylab = Label('absorption', 'left')
panel1 = dev1.createPanel()
panel1.setWorld((min(lamb),min(k_l1)),(max(lamb),max(k_l1)))
panel1.add(function1,box1,xlab,ylab)
dev1.output()
dev1.close()
dev1.hardcopy('3bi1','ps')
box2 = Box()
box2.leftaxis.textangle = pi/2
dev2 = createDevice()
panel2 = dev2.createPanel()
panel2.setWorld((min(lamb),min(k_l2)),(max(lamb),max(k_l2)))
panel2.add(function2,box2,xlab,ylab)
dev2.output()
dev2.close()
dev2.hardcopy('3bi2','ps')
box3 = Box()
box3.leftaxis.textangle = pi/2
dev3 = createDevice()
panel3 = dev3.createPanel()
panel3.setWorld((min(lamb),min(k_l3)),(max(lamb),max(k_l3)))
panel3.add(function3,box3,xlab,ylab)
dev3.output()
dev3.close()
dev3.hardcopy('3bi3','ps')
box1ii = Box()
box1ii.leftaxis.textangle = pi/2
dev1ii = createDevice()
panel1ii = dev1ii.createPanel()
```

```
panel1ii.setWorld((min(lamb),min(k_l1ii)),(max(lamb),max(k_l1ii)))
panel1ii.add(function1ii,box1ii,xlab,ylab)
dev1ii.output()
dev1ii.close()
dev1ii.hardcopy('3bii1','ps')
box2ii = Box()
box2ii.leftaxis.textangle = pi/2
dev2ii = createDevice()
panel2ii = dev2ii.createPanel()
panel2ii.setWorld((min(lamb),min(k_l2ii)),(max(lamb),max(k_l2ii)))
panel2ii.add(function2ii,box2ii,xlab,ylab)
dev2ii.output()
dev2ii.close()
dev2ii.hardcopy('3bii2','ps')
box3ii = Box()
box3ii.leftaxis.textangle = pi/2
dev3ii = createDevice()
panel3ii = dev3ii.createPanel()
panel3ii.setWorld((min(lamb),min(k_l3ii)),(max(lamb),max(k_l3ii)))
panel3ii.add(function3ii,box3ii,xlab,ylab)
dev3ii.output()
dev3ii.hardcopy('3bii3','ps')
box1iii = Box()
dev1iii = createDevice()
box1iii.leftaxis.textangle = pi/2
xlab = Label('lambda', 'bottom')
ylab = Label('absorption', 'left')
panel1iii = dev1iii.createPanel()
panel1iii.setWorld((min(lamb),min(k_l1iii)),(max(lamb),max(k_l1iii)))
panel1iii.add(function1iii,box1iii)
dev1iii.output()
dev1iii.close()
dev1iii.hardcopy('3biii1','ps')
box2iii = Box()
box2iii.leftaxis.textangle = pi/2
dev2iii = createDevice()
panel2iii = dev2iii.createPanel()
panel2iii.setWorld((min(lamb),min(k_l2iii)),(max(lamb),max(k_l2iii)))
panel2iii.add(function2iii,box2iii,xlab,ylab)
dev2iii.output()
dev2iii.close()
dev2iii.hardcopy('3biii2','ps')
box3iii = Box()
box3iii.leftaxis.textangle = pi/2
dev3iii = createDevice()
panel3iii = dev3iii.createPanel()
panel3iii.setWorld((min(lamb),min(k_l3iii)),(max(lamb),max(k_l3iii)))
panel3iii.add(function3iii,box3iii,xlab,ylab)
dev3iii.output()
dev3iii.hardcopy('3biii3','ps')
```

C

For the Planck and Rosseland mean opacity we had to rewrite the code and read in all data from the file, which resulted in a much shorter code. The part for the actual calculation of the means looks somewhat massive, but we had to incorporate the Planck function and its derivative to T in it completely to make the program work. Python seems to still have some flaws in its applications.
For convenience and more clarity I will leave now the functions for H_{bf} and H^-_{bf} out of the code, since these parts will be repeatedly used and will only cause extensive lines of code.

```
#!/usr/local/share/bin/python2.2

from Kplot import *
from Numeric import *
from TableIO import *
import TableIO
import math
import copy

chii = 13.6
c = 2.99792458*math.pow(10, 10)
k = 1.3806503*math.pow(10,-16)
h = 6.62606876*math.pow(10,-27)

reach=20000.0
points=200.0
step=reach/points

lamb = arrayrange(200,reach,step)
sigma = 5.6704*math.pow(10,-5)

file = 'sap3data.txt'
[Tau,z,T,NE,NA] = TableIO.readColumns(file, '#!', [1,2,3,5,6])

#function for k_Hbf
def kappa_lambda(T,N)

[SEE PART B]

#function for k_Hbf-
def holy_handgrenade(NE,T):

[SEE PART B]

#function for Planck mean opacity
kappa_p = 0
for i in range(len(T)):
   for j in range(1,len(lamb)):
     kappa_p = kappa_p + (math.pi*(kappa_lambda(T[i])+
```

```
+ holy_handgrenade(NE[i],T[i]))*
*((2*h*(c**2))/((e**((h*c)/((lamb[j]*math.pow(10,-8))*k*T[i]))-1)*
*((lamb[j]*math.pow(10,-8))**5)))
*((lamb[j]*math.pow(10,-8))+
-(lamb[j-1]*math.pow(10,-8))))/(sigma*T[i]**4)
   print "at T =", T[i] ,"or tau =",
10**(Tau[i]),"the Planck Mean Opacity =", kappa_p[j]

#function for Rosseland mean opacity
kappa_r = 0
for i in range(len(T)):
   for j in range(1,len(lamb)):
     kappa_r = kappa_r +
((math.pi*((e**((h*c)/((lamb[j]*math.pow(10,-8))*
*k*T[i])))))) *(((c**3)*(h**2)*2)/(((e**((h*c)/((lamb[j]*
*math.pow(10,-8))*k*T[i]))-1)**2)*k*((T[i])**2)*
*((lamb[j]*math.pow(10,-8))**6)))
*(((lamb[j]*math.pow(10,-8))+
-(lamb[j-1]*math.pow(10,-8)))/((kappa_lambda(T[i])+
+ holy_handgrenade(NE[i],T[i]))*4*sigma*(T[i]**3)))
   print "at T =", T[i] ,"or tau =", 10**(Tau[i]),
"the Rosseland Mean Opacity =", 1/kappa_r[j]
```

D

For the complete incorporation of everything to get the emergent flux against λ we only had to implement the extra function for the emergent flux.

```
#!/usr/local/share/bin/python2.2

from Kplot import *
from Numeric import *
from TableIO import *
import TableIO
import math
chii = 13.6

k = 1.3806503*math.pow(10,-16)
h = 6.62606876*math.pow(10,-27)
c = 2.99792458*math.pow(10, 10)

reach=100000.0
points=1000.0
step=reach/points
lamb = arrayrange(200,reach,step)
sigma = 5.6704*math.pow(10,-5)

file = 'sap3data.txt'
file1 = 'p3cdata.txt'
[Tau,z,T,NE,NA] = TableIO.readColumns(file, '#!', [1,2,3,5,6])
```

```
[kappa_p,kappa_r] = TableIO.readColumns(file1,'#!',[0,1])

#function for k_Hbf
def kappa_lambda(T,N):

[SEE PART B]

#function for k_Hbf-
#def holy_handgrenade(T,NE)

[SEE PART B]

#function for the emergent flux
for j in range(len(lamb)):
    Integral = 0
    for i in range(len(T)):
        Integral = Integral + (math.pi*(kappa_lambda(T[i])[j]+
+ holy_handgrenade(NE[i],T[i]))[j]*
*((2*h*(c**2))/((e**((h*c)/((lamb[j]*math.pow(10,-8))*
*k*T[i]))-1)*((lamb[j]*math.pow(10,-8))**5)))*
*((lamb[j]*math.pow(10,-8))-(lamb[j-1]*math.pow(10,-8))))/kappa_p[i]
    print Integral
```

For the plotting part we made an extra program, because it only involved reading data from files. We only read until 60800 Å, because no results came after that because of the failsaves we put in our programs to prevent of blowing up our calculations.

```
#!/usr/local/share/bin/python2.2

from Kplot import *
from Numeric import *
from TableIO import *
import TableIO
import math
import copy

chii = 13.6
c = 2.99792458*math.pow(10, 10)
k = 1.3806503*math.pow(10,-16)
h = 6.62606876*math.pow(10,-27)

reach=60800.0
points=608.0
step=reach/points
lamb = arrayrange(200,reach,step)
sigma = 5.6704*math.pow(10,-5)

file = 'p3ddata0.txt'
[flux_lamb] = TableIO.readColumns(file, '#!', [0])
```

```
#the plotting section
function = Polyline(log10(lamb),flux_lamb)

box = Box()
box.leftaxis.textangle = pi/2
dev = createDevice()
xlab = Label('Wavelength (log(10^-8cm))', 'bottom',displacement = 10)
ylab = Label('Emergent flux(erg', 'left', displacement = 15)
panel = dev.createPanel()
panel.setWorld((min(log10(lamb)),min((flux_lamb))),
(max(log10(lamb)),max((flux_lamb))))
panel.add(function,box, xlab, ylab)
dev.output()
dev.close()
dev.hardcopy('emerflux','ps')
```

The next part contains all the figures made for Project 3.

Figures

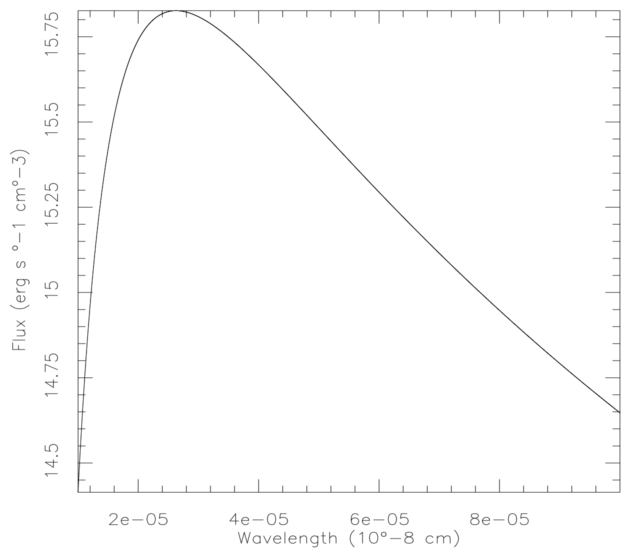

Figure 1: The emergent flux in the simple Grey case. Units on the vertical axis are erg

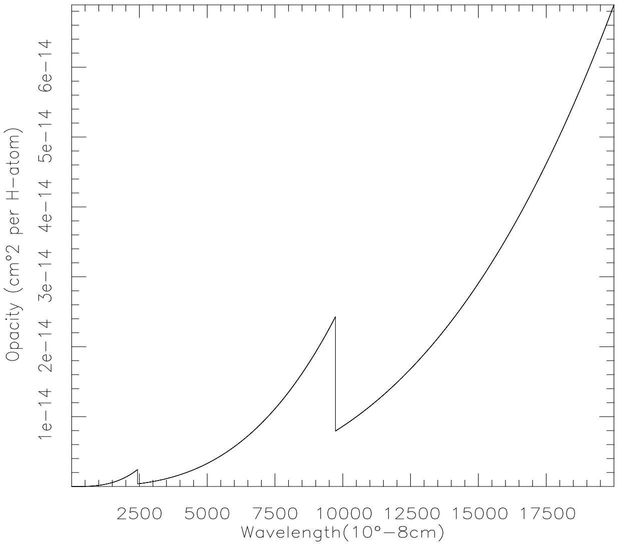

Figure 2: The H_{bf} opacity at τ=0.001

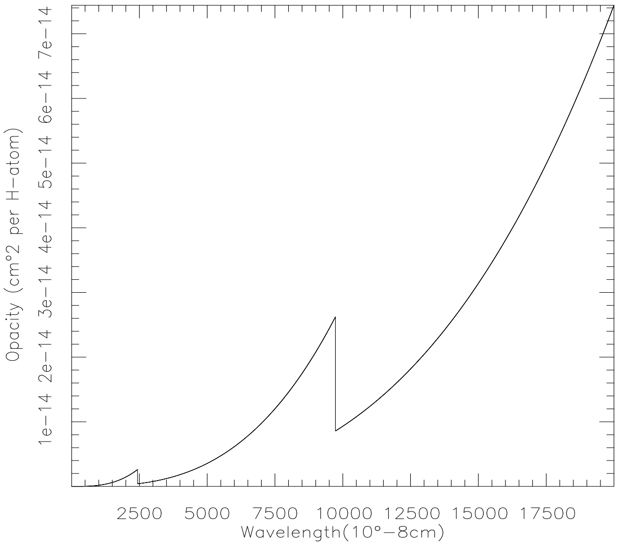

Figure 3: The H_{bf} opacity at τ=2/3

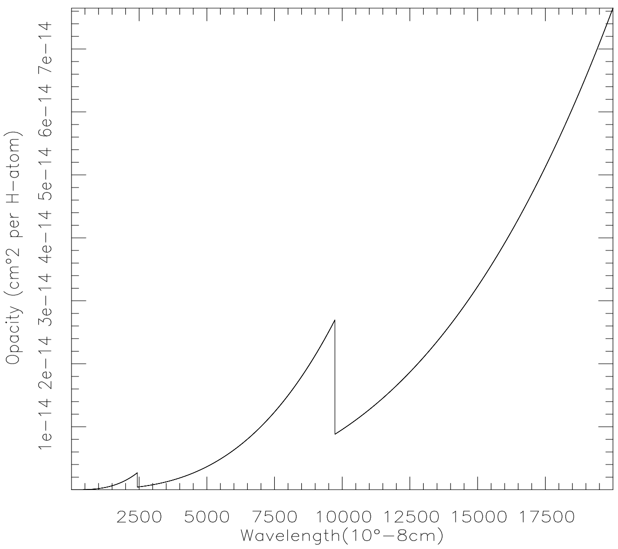

Figure 4: The H_{bf} opacity at τ=10

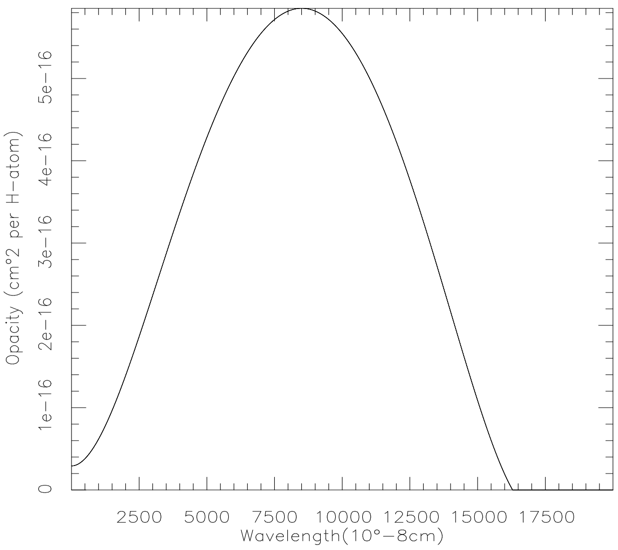

Figure 5: The H^{-}_{bf} opacity at τ=10

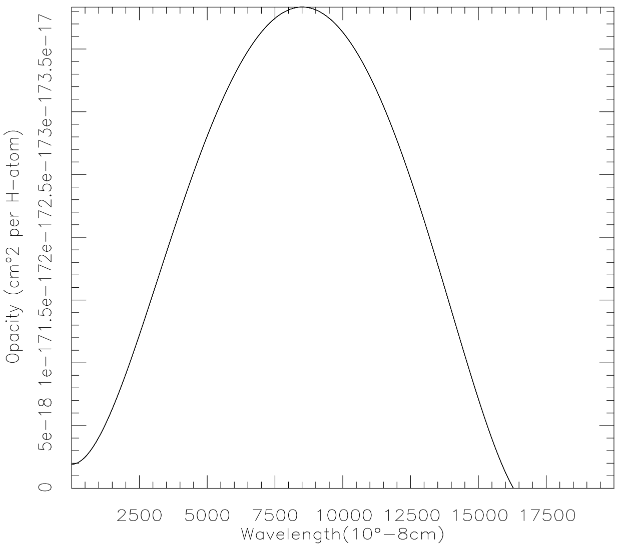

Figure 6: The H^{-}_{bf} opacity at τ=10

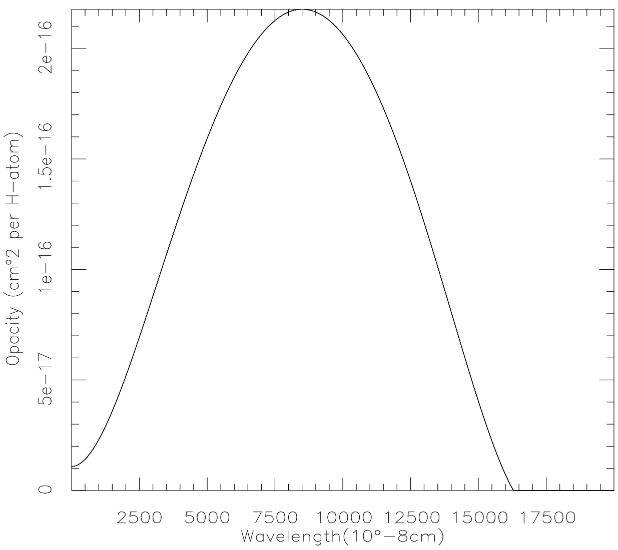

Figure 7: The H^{-}_{bf} opacity at τ=10

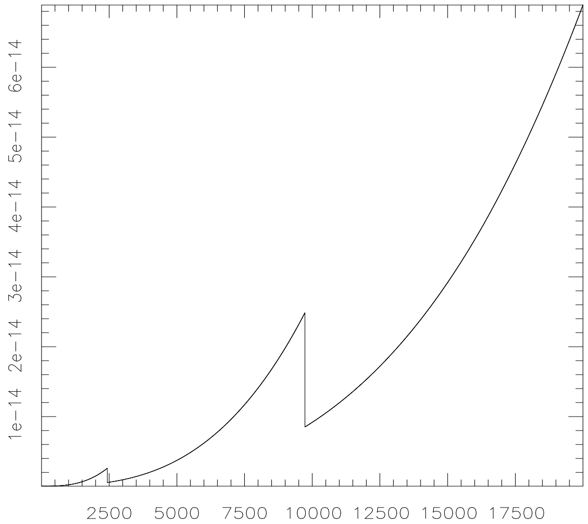

Figure 8: The combined opacity at $\tau = 0.001$

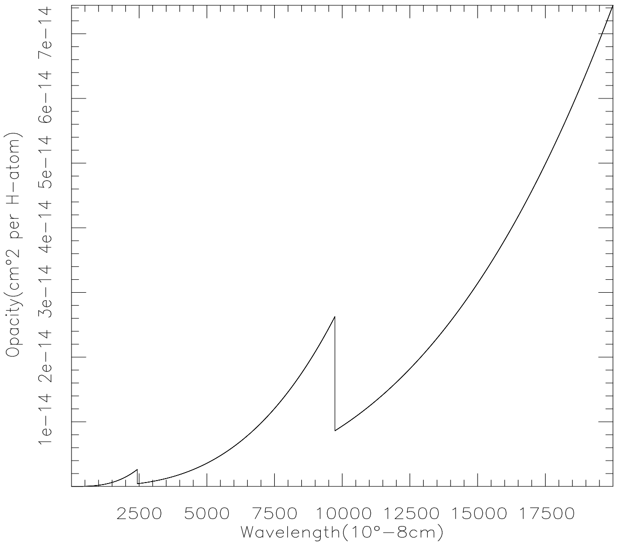

Figure 9: The combined opacity at $\tau = 2/3$

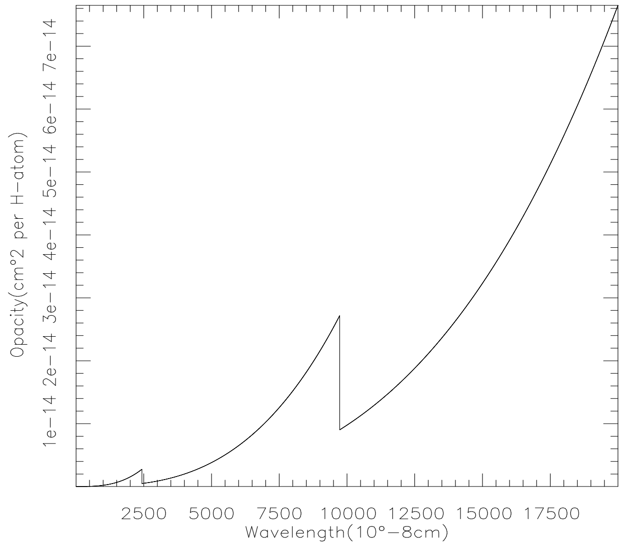

Figure 10: The combined opacity at $\tau = 10$

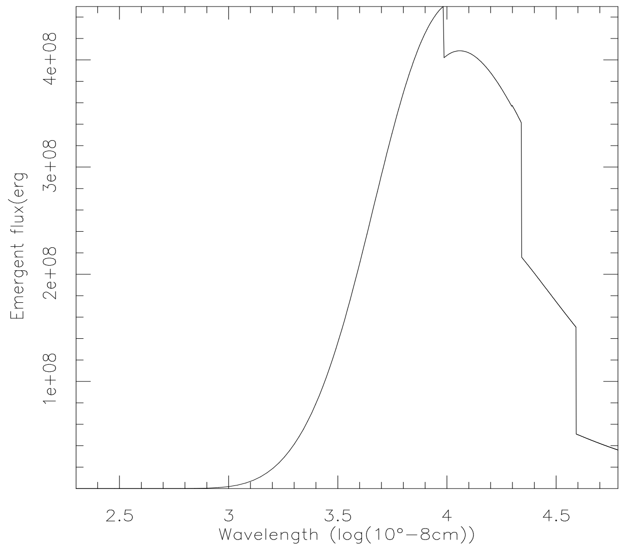

Figure 11: The emergent flux over all opacities againt λ