Draft version December 9, 2006
Preprint typeset using LaTeX style emulateapj v. 12/01/06

STELLAR ATMOSPHERE MODEL

Ryan T. Hamilton
Department of Astronomy, New Mexico State University, P. O. Box 30001, MSC 4500, Las Cruces, New Mexico 88003-8001, e-mail: rthamilt@nmsu.edu

Draft version December 9, 2006

ABSTRACT

Using a 1-D, plane-parallel, LTE, Grey stellar atmosphere, we examine the results of a simulation running from $10^{-3} \leq \tau \leq 10^{2.5}$ for a solar type star. This restricted range is chosen for computational simplicity, as well as having readily available partition functions for the range of temperatures encountered in the model. Behavior of the model vs. physical behavior will be discussed.

Subject headings: Stars: General, Methods: Numerical

1. INTRODUCTION

Using the framework of a plane parallel grey atmosphere, we can apply a few relationships and conservation principles to arrive at a physical and numerical simulation of the outer layers of a stellar atmosphere. This model can then shed some light on the underlying physics and observed properties of a given star.

While not nearly as advanced as other models or techniques, our method nevertheless includes opacity effects as well as detailed balancing of the excitation and ionization species to generate the internal conditions of the model. It is this application of balancing and the use of iterative methods of convergence that are arguably the most important application and result of our work, as they have far reaching consequences to topics in astronomy outside that of stellar atmospheres.

2. STELLAR MODEL PROCEDURE

We set out on our quest for the model by first taking a step back and thinking about what exactly we wanted to get out of it. By looking at how various properties change as we start at the top of the given star and march our way in, the various processes show themselves through the changes in our observed and recorded quantities.

The model is basically set up as an iterative process, starting with only two guessed quantities for a given stellar model. Using both the T_{eff} and initial P_g at the top of our stellar atmosphere, we are able to step through the outer layers of our model by applying the basic physical relationships such as the Boltzman equation, Saha equation, particle conservations, and hydrostatic equilibrium to generate the conditions at the layer of the star immediately below the current.

For calculational simplicity and to be able to check against a known source, our model assumes a solar composition and a solar T_{eff} of 5770.0 K. After that was up and running, the T_{eff} was increased to see what would happen to the basic physical properties shown in Figure 1. A comparison between the results for a solar type star and a typical main sequence A star is shown in Figure 2, and will be discussed in § 3.

2.1. *Physical Assumptions*

Our model is somewhat simple, and depends on a few basic assumptions. Perhaps the most important is that our model is plane-parallel, which restricts us to traveling relatively shallow into the atmosphere of the star. This also allows us to use an important tool such as the Hopf function, which is a correction to the derived temperature profile following from application of Eddington's approximation. Also, we assume that as we proceed in towards the center of the star the gravity felt will not change appreciably. This allows us to use the surface gravity g as our balancing gravity when evaluating hydrostatic equilibrium. We assume a solar composition of $X = 0.7$ $Y = 0.28$ and $Z = 0.02$. For simplicity we also assume only 1 metal present, Calcium, and up to two ionizations for Helium and Calcium.

The partition functions we use for the calculation of the various ionization states are valid only for $0.2 \leq \theta = 5040.0/T \leq 2.0$ or for $2520 \leq T \leq 25200$ are are given with the code attached in the appendix. It is this narrow range of θ that restricts the reach of our model, as deviations into temperatures higher than 25000 K can be reached rather quickly for a star with a higher T_{eff}. Even in the initial solar case, deviations from this temperature range can be seen in the sharp downturn of neutral Calcium in Figure 4, and turns off quicker in the A star shown as well. Toying with different values of $\theta \sim 0.1$ led to dramatically different results in the neutral calcium population at $\tau \gtrsim 1.0$, showing its sensitivity at very high temperatures.

2.2. *The Physics*

We step through our model in units of $\log \tau$, where our step size $d\tau$ is defined below. We also starts with an initial guess at the surface pressure, and a T_{eff} from which to generate the temperature profile. The temperature profile is taken from using the Hopf function, which is defined such that

$$d\tau = \chi(\rho, T)\rho_{tot} dx \tag{1}$$

$$\frac{T(\tau)}{T_{eff}} = \left[\frac{3}{4}\{\tau + q(\tau)\}\right]^{1/4} \tag{2}$$

where $q(\tau)$ is a smoothly varying correction for small τ that converges to the usual $T(\tau)$ relationship from Eddington's approximation. After the initial guess for the pressure is made, it is used in the equation of state to determine the total number of particles N_{tot}. This is then used the iterative rootsolving routine ZBRENT (see § 2.3), which iterates through different guesses of n_e in the Saha equation until it converges on a solution. Once

N_{tot} and n_e have been determined, they are used to find the number of neutrals N_n and the hard parts are mostly over. The electron and neutral densities can be determined to give the density at the current layer, which is also then used to determine the opacity, among other things. The equations governing these basic processes are outlined below.

$$N_{tot} = N_n + n_e \tag{3}$$

$$N_k = \sum_{j=0}^{J_k} N_{jk} \tag{4}$$

$$N_n = \sum_k N_k \tag{5}$$

$$\rho_{tot} = \rho_n + \rho_e \tag{6}$$

$$\rho_n = m_H N_n \sum_k \alpha_k A_k \tag{7}$$

$$\rho_e = m_e n_e \tag{8}$$

While solving for n_e, the rootsolving routine ZBRENT steps through the different ionization species to see what n_e gives the best agreement for balancing the ionization species against one another. After convergence is achieved, the excitation states can be determined by application of the Boltzman equation, Equation 9 below. Equation 10 is the all important Saha equation, for determining the population of the ionization states. It is a relative population of adjacent states, but one can apply some tricks such as in Equation 12 to determine the population of a given ionization state relative to the ground state.

$$\frac{n_{ijk}}{N_{jk}} = \frac{g_{ijk}}{U_j(T)} exp \frac{\chi_{ijk}}{kT} \tag{9}$$

$$\frac{N_{jk}}{N_{j+1k}} = n_e \frac{U_j(T)}{U_{j+1}(T)} C_I T^{3/2} exp \frac{\chi_{Ijk}}{kT} \tag{10}$$

$$n_e = (N_{tot} - n_e) \sum_k \alpha_k \sum_{j=1}^{J_k} j f_{jk}(n_e, T) \tag{11}$$

$$f_j k(n_e, T) = \frac{N_{jk}}{N_k} = \frac{N_{jk}/N_{0k}}{\sum_{j=0}^{J_k} (N_{jk}/N_{0k})} \tag{12}$$

With all this firm in hand, the final step in the model is to balance the pressures and gravity to see if hydrostatic equilibrium is achieved. One can determine the equations of state in the usual way as shown in Equations 13 & 14 for the electrons and neutrals, then find the total gas pressure to see how things turned out.

$$P_n = N_n kT \tag{13}$$

$$P_e = n_e kT \tag{14}$$

$$P_g = P_n + P_e \tag{15}$$

$$dP_g = g \frac{d\tau}{\chi(\rho, T)} \tag{16}$$

At the very first layer, since we do not start directly at $\tau = 0.0$, when we calculate the change in pressure dP_g as in Equation 15 we should see that dP_g and our first pressure guess should be equivalent, since at $\tau = 0.0$, $P_g = 0.0$. They will most likely not be, so another guess is made at P_g in the current layer and *most* of the routine up to now is run again with a new P_g, thus a new N_{tot}, etc. Optimizations can be made to cut out most of the computation time during this process, which will be discussed further in § 2.3. This process continues until convergence in P_g is achieved, then we progress to the next layer in $\log \tau$ and repeat.

2.3. *Modeling Techniques*

The code, attached in the appendix, is written in FORTRAN77. The general method has been described above, although several steps warrant further discussion. The first of which is the method of interpolating the various lookup tables, such as the partition functions (Grey 2005) and the Rossland Mean Opacities (Cox & Tabor 1976).

For the partition functions, included in Table 1, a smoothly varying natural cubic spline (Burden & Faires 2001) was used to interpolate between values of θ. Outside the parameter range for θ, however, it is dangerous to proceed as was mentioned in the case of Ca I. To interpolate the Rossland Mean Opacities, however, a 2-D interpolation scheme was created since the grid of opacities is in discreet values of ρ_{tot} and T. The input file of opacities was split into three parts; the densities, temperatures, and the grid of opacities itself. Given a new ρ_{tot} and T, the interpolator zeros in on the temperature and pressures just above and below those it is passed, and then interpolates in density then temperature to finally get the desired opacity value.

Since n_e is somewhat recursive, we must use an iterative rootsolving technique like ZBRENT (Press et. al. 1996) to converge on a solution within a given tolerance with minimum effort. ZBRENT itself converges rather quickly, and in the hopes of optimization some steps were moved around to maximize efficiency. One particular example is the calculations of the ionization fractions; ZBRENT is called within the hydrostatic equilibrium balancing, but nothing is done with the computed value of n_e until after pressure has been balanced. This cuts out an enormous chunk of redundant computing that would have been done otherwise, and cut the runtime of the program from its first incarnation by a large degree. The full model itself including all output runs in $\sim$ 9 *sec.*, a respectable and good result for a program such as this.

3. MODEL RESULTS

Looking at Figure 1, we can immediately tell that our model has one good thing going for it - temperature increases as τ increases, as per the hopf function. We can also see that as τ gets very large, we begin to cover more ground in the atmosphere as our physical depth increases. At approximately 10^7 or 10^8 cm, however, we still are covering only a tiny fraction of a 1 $R_\odot = 6.96 \times 10^{10}$ cm star. The opacity increases as τ and ρ_{tot} increase, but a lag in the opacity is seen compared to the density enhancement. This local increase in opacity is most likely due to the creation of a partial ionization zone of Hydrogen starting around $\tau = 40$ and 15000 K, increasing the opacity as the thermal energy goes into ionizing Hydrogen. This continues until about $\tau = 100$, after which all the Hydrogen has been ionized and the opacity starts to decline slightly as seen in Figure 1.

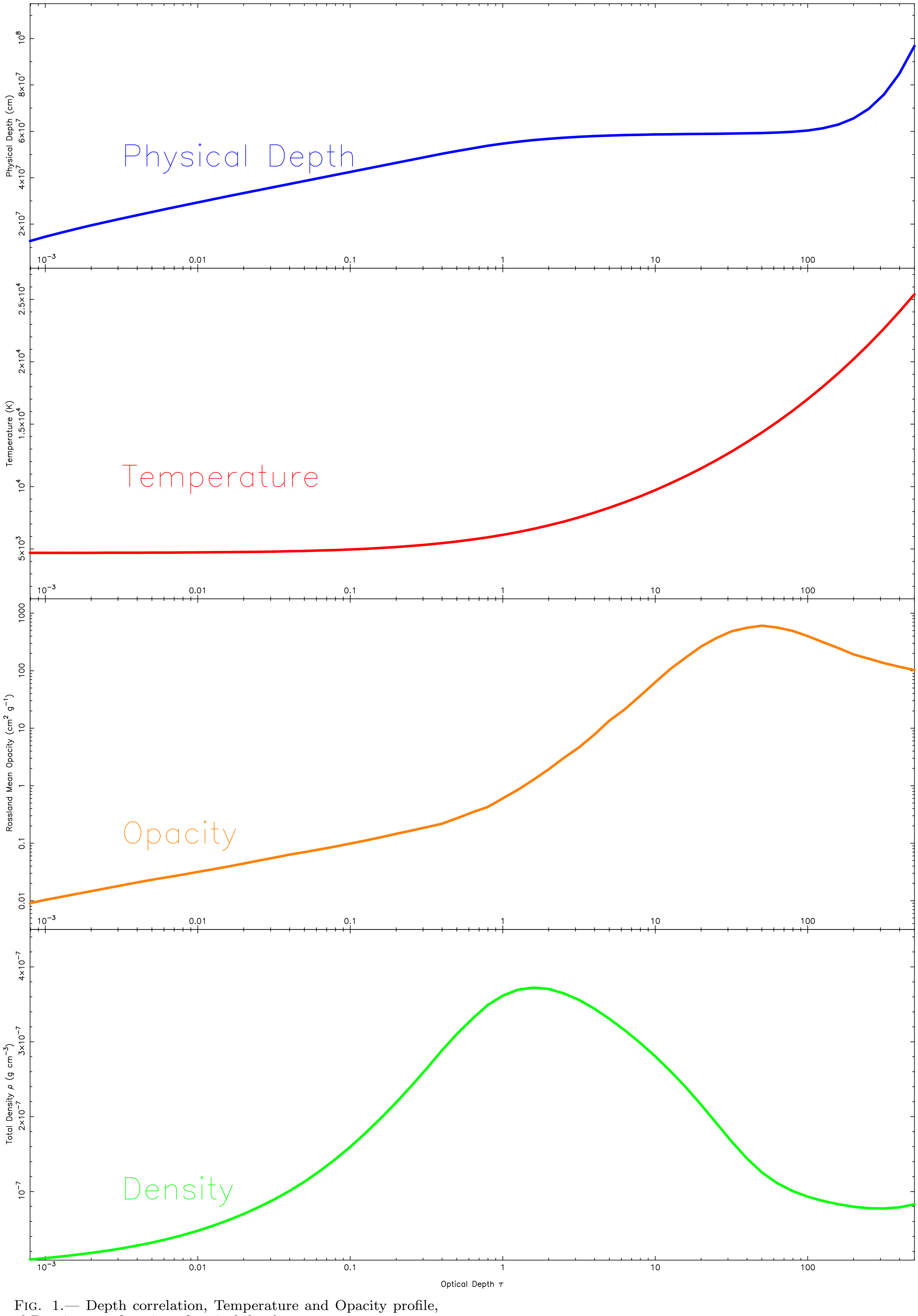

FIG. 1.— Depth correlation, Temperature and Opacity profile, and Density as a function of optical depth τ.

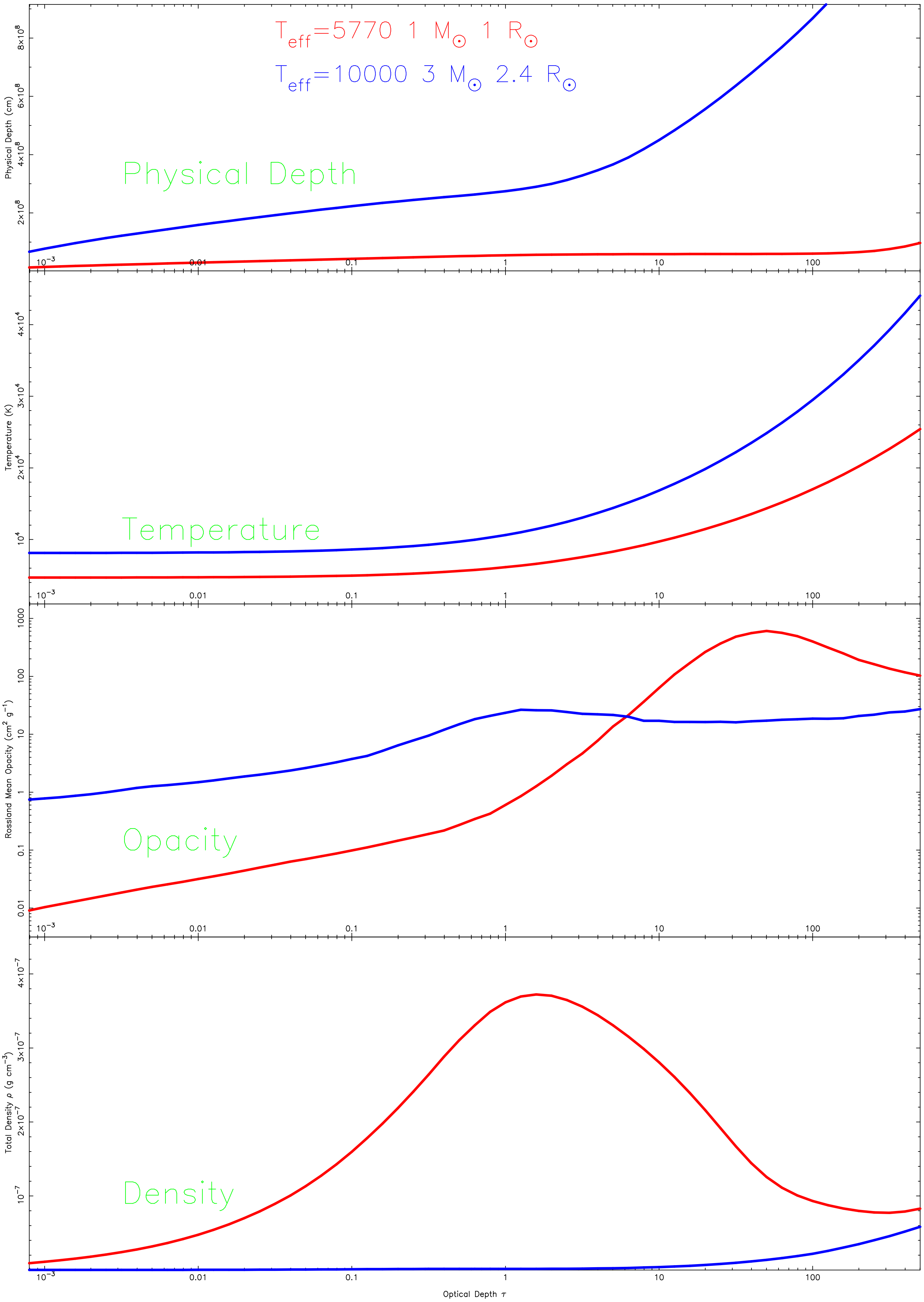

FIG. 2.— Comparison between solar type star (red) and hotter A star (blue).

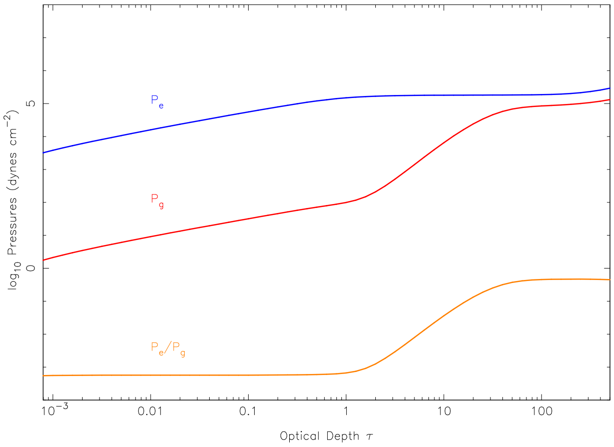

FIG. 3.— $\log P_e, P_g$, & P_e/P_g as a function of optical depth τ.

Also in Figure 1 a density enhancement around $\tau = 1$, but declines sharply afterwards. This is again due to an ionization effect, since both Calcium and Hydrogen begin to be ionized and release their electrons, dragging the density down.

Figure 2 shows a comparison between the physical parameters of a solar type star with that of a main sequence A star. Immediately one can see the immediate temperature difference, as well as the lack of a significant bump in the opacity or density. This is due to the fact that from the start, Hydrogen and Calcium are partially ionized which makes the opacity higher overall. The density difference is to be expected, since the A star has a radius almost 2.5 times that of the solar type star.

Figure 3 shows the contributions to the total gas pressure P_g, which shows an overall increase in gas pressure past $\tau = 1$, which corresponds to the peak density as seen in Figure 1. The gas pressure also levels off at around $\tau = 40$, since the thermal energy at this depth is going into ionizing Hydrogen and not into increasing the pressure any more until it is ionized completely.

Figure 4 shows the complete breakdown of the different ionization species for the solar type star, as well as the ionization states of the A star in the adjacent panel. Looking at the solar type star, you can see the ionization crossover at around $\tau = 40$ for H I and H II, as well as a crossover for Ca II and Ca III. For the A star you can see as well how much earlier He II begins to be appreciable, as well as how quickly Hydrogen begins to be completely ionized. You can also see where the partition function for Ca I begins to break down, especially for the case of the A star where the temperature reaches past $25,000$ K much quicker at around $\tau = 20$. The A star has two partial ionization zones, one visible at around $\tau = 10$ and another just past our range at $\tau > 300$.

Figure 5 shows the different contributions to the total electron density n_e. Around $\tau = 1$, n_e goes from being dominated by electrons from Ca II to those coming from H II, fully ionizing Hydrogen around $\tau = 40$ being exactly consistent with the discussion thus far. Contributions from Ca II continue to tail off, but the temperature rises enough to get contributions from He II and He III keeping n_e roughly constant.

Figure 6 shows the populations of the excited states for Hydrogen. Note how at high temperatures the population of the ground state slowly begins to decline. Limitations in the partition function again skew our results here at high temperatures, as it should be declining more rapidly.

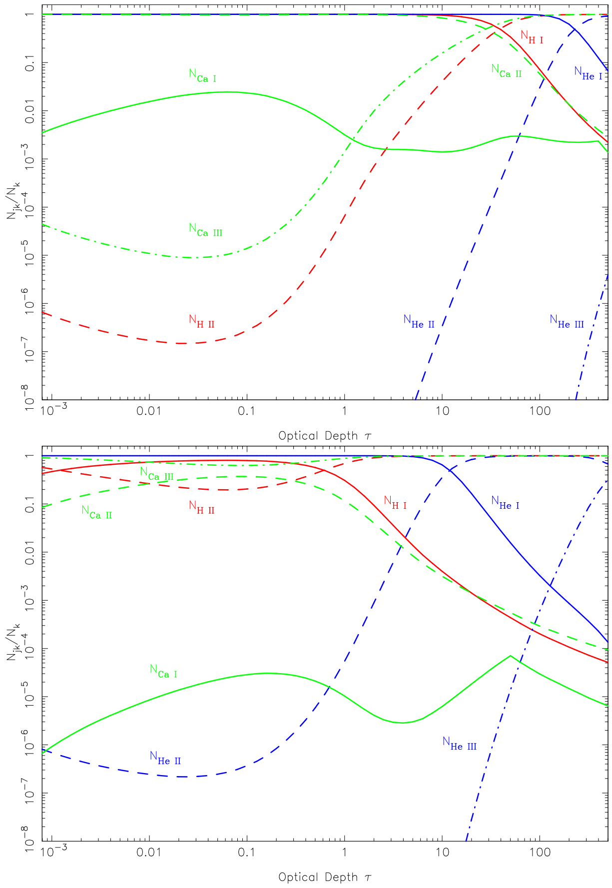

FIG. 4.— Ionization fractions as a function of optical depth τ for H, He, and Ca. A solar type star is shown in the upper panel, and a main sequence A star in the lower. Up to two ionizations have been included. Neutral is denoted by a solid line, singly ionized by a dashed line, and doubly ionized by dot-dashed line.

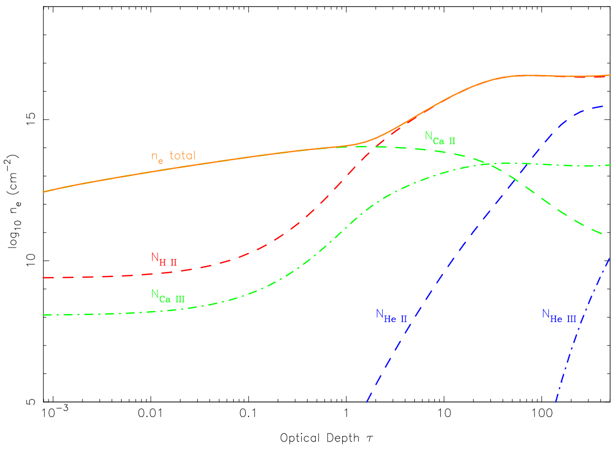

FIG. 5.— Electron density n_e contributions from the different ionization species.

4. SUMMARY & FUTURE WORK

In summary, our model presented allows examination of the effects of ionization and excitation on the physical properties of the star such as density, opacity, and pressure. It is currently limited by restrictions in the partition functions used, and most notably lacks a calcu-

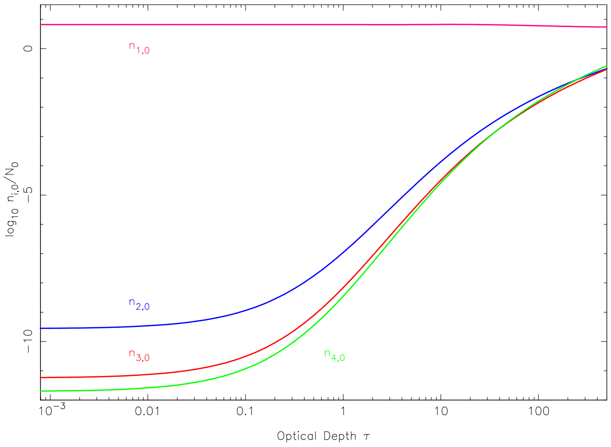

FIG. 6.— Population of excited states as a function of optical depth τ.

lation of the flux. Time restrictions made this calculation unfeasable, due to calculational complexity and problems in debugging the model presented in this paper. Future work by the author in the near future should allow this calculation to be included in the model.

I would like to thank Chas Miller for his generous contribution of the Rossland Mean Opacity table, and his debugging help to get past the hydrostatic equilibrium. I would also like to thank Sean Lindsay for the same. Finally, I would like to thank my senior undergraduate Astrophysics class, without which I would have never thought that I was looking straight at a partial ionization zone in the output of the model.

REFERENCES

Burden, R.L., & Faires, J.D., 2001, "Numerical Analysis", 7th ed., Brooks/Cole

Cox, A. N., & Tabor, J. E., ApJS, 31, 271

Grey, D., 2005, "Observation and Analysis of Stellar Photospheres", 3rd ed., Cambridge University Press

Press, W.H. et. al., 1996, "Numerical Recipes in Fortran 77: The Art of Scientific Computing", 2nd ed., Cambridge University Press

APPENDIX

A. THE CODE

```
c common block variables

      double precision alpha(0:4), ca_u(0:3), temp, theta
      double precision ca_ion(0:2), h_ion(0:2), he_ion(0:2), Ntot
      double precision final_ratio(0:5), he_u(0:2), h_u(0:2)
      common /massalphas/ alpha, temp, theta,
     &  ca_ion, he_ion, h_ion, Ntot,
     &  ca_u, he_u, h_u
```

```
!********************************************!
!        Stellar Atmosphere                  !
!        modeler.f                           !
!        Ryan Hamilton                       !
!        ASTR545, Due 12/8/2006              !
!                                            !
!********************************************!

      PROGRAM modeler

      implicit none
      external myfunc
      include 'crazy.com'

      double precision nemin,nemax,tol,zbrent,myfunc, ne, nn
      double precision A_H, A_He, A_Z, G, lilg, pi, c, Mh, Me
      double precision mX, mY, mZ, Pe, Pn, boltz, junker, ebolt
      double precision ca_ratios(0:5), he_ratios(0:5), h_ratios(0:5)
      double precision nk(0:5), logtau, teff, opacity
      double precision rho, rhoe, rhon, dx, dtau, Rsun, Msun
      double precision Pg, Pgold, Pguess, dp, tau, oldtau, newtau
      double precision n(0:5), boltzrt(5)
      integer d, timer
      character*30 h_part_files(0:2)
      character*30 he_part_files(0:2)
      character*30 ca_part_files(0:2)

      parameter (mX=0.70)
      parameter (mY=0.28)
      parameter (mZ=0.02)
      parameter (A_H=1.00794)
      parameter (A_He=4.002802)
      parameter (A_Z=40.078)
      parameter (boltz=8.617343e-5) ! in eV/K
      parameter (ebolt=1.380658e-16) ! in erg/K
      parameter (c=2.99792458e10) ! in cm/s
      parameter (G=6.6742e-8) ! cm^3/g/s^2
      parameter (pi=3.1415926536)
      parameter (Mh=1.67262171e-24) ! in g
      parameter (Me=9.1093825e-28) ! in g

      ! The initial conditions
      teff= 5770.0
      !teff = 10000.0           ! in K
      Pg = 2000.0              ! in dynes/cm^2, my guess after a few tries
      ! Gstar
      Rsun = 6.96e10           ! in cm
      Msun = 1.9892e33         ! in g
      ! Astar
      !Msun = (3.0)*1.9892e33   ! in g
      !Rsun = (2.4)*6.96e10     ! in cm
      lilg = G*Msun/(Rsun**2.0) ! Plane parallel approximation
      dtau = 0.10              ! in log(tau)
      tol  = 1.0e-6
      nemin = 1.0e04
      nemax = 1.0e25
      dp = 0.0                 ! sanity check
      oldtau = 0.0             ! sanity check

      alpha(0)=(1.0+(mY/mX)*(A_H/A_He)+(mZ/mX)*(A_H/A_Z))**-1.0
      alpha(1)=(1.0+(mX/mY)*(A_He/A_H)+(mZ/mY)*(A_He/A_Z))**-1.0
```

```
      alpha(2)=(1.0+(mX/mZ)*(A_Z/A_H)+(mY/mZ)*(A_Z/A_He))**-1.0

      OPEN(UNIT=44,file='ionizations.dat')
      OPEN(UNIT=55,file='gasses.dat')
      OPEN(UNIT=66,file='pressures.dat')
      OPEN(UNIT=77,file='necontrib.dat')
      OPEN(UNIT=88,file='excitation.dat')
!     My collection of format statements for the various outputs...
!     Damn this is a lot of output
235   FORMAT(1pe11.4,2x,
     &      1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,
     &      1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3)
236   FORMAT(1pe11.3,2x,1pe11.3,2x,1pe11.3,2x,1pe11.3)
237   FORMAT(1pe11.4,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,
     &       1pe9.3,2x,1pe9.3,2x,1pe9.3)
238   FORMAT(1pe11.4,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3,2x,1pe9.3)

8888   FORMAT(1pe11.4,2x,1pe11.4,2x,1pe11.4,2x,1pe11.4,2x,
     &  1pe11.4,2x,1pe11.4,2x,1pe11.4)

      oldtau=0.0
      DO 6767 logtau=-3.2,3.5,dtau
       tau=10**logtau
       call hopf(tau,teff,temp)
       theta=5040.0/temp
!      Now begins the exceptionally boring interpolating parts
       h_part_files(0)='hI_part_func.dat'
       h_ion(0)=13.6
       CALL natural_spline(h_part_files(0), theta, h_u(0))
       h_u(1)=0.0
       he_part_files(0)='heI_part_func.dat'
       he_ion(0)=24.587
       CALL natural_spline(he_part_files(0), theta, he_u(0))
       he_part_files(1)='heII_part_func.dat'
       he_ion(1)=54.418
       CALL natural_spline(he_part_files(1), theta, he_u(1))
       ca_part_files(0)='caI_part_func.dat'
       ca_ion(0)=6.113
       CALL natural_spline(ca_part_files(0), theta, ca_u(0))
       ca_part_files(1)='caII_part_func.dat'
       ca_ion(1)=11.872
       CALL natural_spline(ca_part_files(1), theta, ca_u(1))
       ca_part_files(2)='caIII_part_func.dat'
       ca_ion(2)=50.913
       CALL natural_spline(ca_part_files(2), theta, ca_u(2))
!      Now ends the exceptionally boring interpolating parts
!      Allow 150 for hydro. equilib. tries; if you don't converge then,
!      program will bail and yell to avoid infinite loops and craziness
       DO 666 timer=1,150
         Ntot = Pg/(temp*ebolt)
         ne = zbrent(myfunc,nemin,nemax,tol)
!        Electron Pressure and density
         Pe = ne*ebolt*temp
         rhoe = ne*Me
!        Neutral Pressure and density
         nn = Ntot-ne
         Pn = nn*ebolt*temp
         rhon = Mh*nn*(A_H*alpha(0)+A_He*alpha(1)+A_Z*alpha(2))
         rho = rhoe+rhon
!        Getting the opacity, calculating hydrostatic equilibrium pressure
         call bilinear_interp(temp,rho,opacity)
!        Hydrostatic Equilibrium
         dp = (lilg)*(tau-oldtau)/opacity
         Pguess = Pgold + dp
         IF (abs(Pguess - Pg) .le. 1.0e-6) THEN
           dx = dx + (tau-oldtau)/(rho*opacity)
           oldtau = tau
           Pgold = Pg
           Pg = Pg + dp
           ! Success, you get to leave jail early
           go to 777
         ELSE
           Pg = (Pg + Pguess)/2.0
           IF (timer .eq. 125) THEN
             WRITE(6,*)"Warning: Possible non-convergence"
           ENDIF
```

```
        ENDIF
       IF (timer .eq. 149) THEN
         WRITE(6,*) "FUBAR; Pressure divergence, bailing..."
         STOP
       ENDIF
666   CONTINUE

!     Calculating Ionization Fractions
777   nk(0)=alpha(0)*nn
      nk(1)=alpha(1)*nn
      nk(2)=alpha(2)*nn
      CALL fjk_T(1, h_u, h_ion, ne, h_ratios)
      CALL fjk_T(2, he_u, he_ion, ne, he_ratios)
      CALL fjk_T(2, ca_u, ca_ion, ne, ca_ratios)

      ! Calculating Boltzman ratios
      boltzrt(1)=((2.0*1**2.0)/h_u(0))*
     &  exp(-13.6*(1.0-1.0/(1.0**2))/(boltz*temp))
      boltzrt(2)=((2.0*2**2.0)/h_u(0))*
     &  exp(-13.6*(1.0-1.0/(2.0**2))/(boltz*temp))
      boltzrt(3)=((2.0*3**2.0)/h_u(0))*
     &  exp(-13.6*(1.0-1.0/(3.0**2))/(boltz*temp))
      boltzrt(4)=((2.0*4**2.0)/h_u(0))*
     &  exp(-13.6*(1.0-1.0/(4.0**2))/(boltz*temp))

      ! For Figure 1a,1b,1c,1d
      WRITE(55,8888) logtau,dx,temp,log10(opacity),rho,Pg
      ! For Figure 2
      WRITE(66,236) logtau, Pg, Pe, Pe/Pg
      ! For Figure 3
      WRITE(44,235) logtau,temp,Ntot,ne,
     & h_ratios(0),h_ratios(1),he_ratios(0),he_ratios(1),he_ratios(2),
     & ca_ratios(0),ca_ratios(1),ca_ratios(2)
      ! For Figure 4
      WRITE(77,237) logtau,
     & h_ratios(1)*(nk(0)),he_ratios(1)*(nk(1)),he_ratios(2)*(nk(1)),
     & ca_ratios(1)*(nk(2)),ca_ratios(2)*(nk(2)),ne,Ntot
      ! For Figure 5
      WRITE(88,238) logtau,boltzrt(1),boltzrt(2),boltzrt(3),boltzrt(4)
6767 CONTINUE
     CLOSE(UNIT=44)
     CLOSE(UNIT=55)
     CLOSE(UNIT=66)
     CLOSE(UNIT=77)
     CLOSE(UNIT=88)
     END

!**********************************************!
!              Ne Solver                !
!           RTH 10/03/2006              !
!     Solves for the electron density given !
!     temperature, ionization species, etc. !
!                                       !
!**********************************************!

     DOUBLE PRECISION FUNCTION myfunc(ne)

     implicit none
     include 'crazy.com'

     integer j
     double precision boltz, C_I, ne
     double precision N1_N0h,N1_N0he,N2_N1he,N1_N0ca,N2_N1ca,N3_N2ca
     double precision N1_Nh, N1_Nhe, N2_Nhe, N1_Nca, N2_Nca, N3_Nca
     double precision den

     parameter (boltz=8.617343e-5)
     parameter (C_I=2.07e-16)

     ! Hydrogen can be once ionized
     N1_N0h=(1.0/ne)*(1.0/((10**h_u(0)/10**h_u(1))*C_I
    &   *exp(h_ion(0)/(boltz*temp))*temp**(-3.0/2.0)))
     den=1.0+N1_N0h
     N1_Nh=N1_N0h/den

     ! Helium can be once or doubly ionized
     N1_N0he=(1.0/ne)*(1.0/((10**he_u(0)/10**he_u(1))*C_I
    &   *exp(he_ion(0)/(boltz*temp))*temp**(-3.0/2.0)))
     N2_N1he=(1.0/ne)*(1.0/((10**he_u(1)/10**he_u(2))*C_I
    &   *exp(he_ion(1)/(boltz*temp))*temp**(-3.0/2.0)))
```

```
      den=1.0+N1_N0he+N1_N0he*N2_N1he
      N1_Nhe=N1_N0he/den
      N2_Nhe=N1_N0he*N2_N1he/den

      ! Calcium can be once or doubly ionized
      N1_N0ca=(1.0/ne)*(1.0/((10**ca_u(0)/10**ca_u(1))*C_I
     &   *exp(ca_ion(0)/(boltz*temp))*temp**(-3.0/2.0)))
      N2_N1ca=(1.0/ne)*(1.0/((10**ca_u(1)/10**ca_u(2))*C_I
     &   *exp(ca_ion(1)/(boltz*temp))*temp**(-3.0/2.0)))
      den=1.0+N1_N0ca+N1_N0ca*N2_N1ca
      N1_Nca=N1_N0ca/den
      N2_Nca=N1_N0ca*N2_N1ca/den

      myfunc=-ne+(Ntot-ne)*( alpha(0)*N1_Nh + alpha(1)*(N1_Nhe+2*N2_Nhe)
     &  + alpha(2)*(N1_Nca+2*N2_Nca))
      RETURN
      END

!**********************************************!
!       Linear Interpolation         !
!          RTH 9/27/2006             !
!                                    !
! Requires:                          !
!   x1, x2, y1, y2, newx             !
! Returns:                           !
!   newy                             !
!                                    !
!**********************************************!

      SUBROUTINE lin_interp(oldx1, oldx2, oldy1, oldy2,
     &   newx, newy)

      double precision oldx1, oldx2, oldy1, oldy2
      double precision newx, newy, slope, y_int

      slope=(oldy2-oldy1)/(oldx2-oldx1)
      y_int=oldy1-(slope*oldx1)
      newy=slope*newx+y_int

      RETURN
      END SUBROUTINE lin_interp

!**********************************************!
!      Hopf Function and Temperatures  !
!          RTH 12/05/2006            !
!                                    !
!**********************************************!

      SUBROUTINE hopf(mytau,teff,newtemp)

      double precision tau(100),qt(100),teff,newtemp,mytau
      double precision newqt
      integer trk, rownum
      trk=1
      open(unit=9,file="hopf")
789   read(9,*,END=987) tau(trk), qt(trk)
      IF (mytau .ge. tau(trk-1) .and. mytau .lt. tau(trk)) rownum=trk
      trk=trk+1
      go to 789
987   continue
      trk=trk-1
      close(unit=9)
      call lin_interp(tau(rownum-1),tau(rownum),qt(rownum-1),qt(rownum),
     & mytau,newqt)
      newtemp=teff*((3.0/4.0)*(mytau+newqt))**(1.0/4.0)
      RETURN
      END SUBROUTINE hopf

!**********************************************!
!       Bilinear Interpolation       !
!          RTH 12/4/2006             !
!                                    !
!**********************************************!

      subroutine bilinear_interp(newtemp,newrho,newopac)

      double precision newtemp, newrho, slope, y_int
      double precision temps(50), rho(50), opacity(20)
      double precision newopac, opac1, opac2
      integer N, M, O, P, colnum, rownum
```

```
      open(unit=3,file="chi_rho.dat")
      M=1
60    READ(3,*,END=600) rho(M)
      if(newrho .ge. rho(M-1) .and. newrho .lt. rho(M)) colnum=M
      M=M+1
      go to 60
600   CONTINUE
      M=M-1
      CLOSE(UNIT=3)

      open(unit=2,file="chi_t.dat")
      N=1
50    READ(2,*,END=500) temps(N)
      if(newtemp .ge. temps(N-1) .and. newtemp .lt. temps(N)) rownum=N
      N=N+1
      go to 50
500   CONTINUE
      N=N-1
      CLOSE(UNIT=2)

      open(unit=1,file="chi.dat")
      O=1
70    READ(1,*,END=700) opacity(1),opacity(2),opacity(3),opacity(4),
     &      opacity(5),opacity(6),opacity(7),opacity(8),opacity(9),
     &     opacity(10),opacity(11),opacity(12),opacity(13),opacity(14),
     &     opacity(15),opacity(16),opacity(17),opacity(18)
      if (O .eq. rownum-1 .or. O .eq. rownum) THEN
        P=colnum
        slope=(opacity(P)-opacity(P-1))/(rho(P)-rho(P-1))
        y_int=opacity(P-1)-(slope*rho(P-1))
        newopac=slope*newrho+y_int
        if (O .eq. rownum-1) opac1=newopac
        if (O .eq. rownum) opac2=newopac
      endif
      O=O+1
      go to 70
700   CONTINUE
      O=O-1
      CLOSE(UNIT=1)

      slope=(opac2-opac1)/(temps(rownum)-temps(rownum-1))
      y_int=opac1-(slope*temps(rownum-1))
      newopac=slope*newtemp+y_int
      return
      end subroutine

!**********************************************!
!     num5.f - Homework 2, Problem #5  !
!        RTH   ASTR545, 9/19/06        !
!   Adapted to use in this beast of a model !
!                                      !
!**********************************************!

      SUBROUTINE fjk_T(l, u, ion, ne, final_ratio)

      implicit none
      include 'crazy.com'

      integer l, ll, j
      double precision denominator, u(0:5), ion(0:5)
      double precision saha_ratio(0:5), k, C_I, total, ne
      parameter (k=8.617343e-5)
      parameter (C_I=2.07e-16)

!     Nj+1/Nj == 1/ne * 1/saha == 1/ne * 1/phihat
      DO 22 ll=0,l-1
        saha_ratio(ll)=(1.0/ne)*(1.0/((10**u(ll)/10**u(ll+1))*C_I
     &   *exp(ion(ll)/(k*temp))*temp**(-3.0/2.0)))
22    CONTINUE

599   FORMAT(' n_e = ',1pe8.2)
600   FORMAT(4x,'j',3x,'Temp.',6x,'Theta',8x,'u(j)',8x,'u(j+1)',7x,
     &    'Ion_pot',6x,'Nj/Nj+1')
601   FORMAT(2x,i3,2x,f6.0,2x,f9.3,4x,f9.3,4x,f9.3,4x,1pe12.3,
     &    2x,1pe11.3)
      j=0
      if (l .EQ. 1) then
        denominator=1.0+saha_ratio(0)
        final_ratio(0)=1.0/denominator
```

```
      final_ratio(1)=saha_ratio(0)/denominator
    else
      denominator=1
      denominator=1+saha_ratio(j)+(saha_ratio(j+1)*saha_ratio(j))
      final_ratio(0)=1/denominator
      final_ratio(1)=saha_ratio(0)/denominator
      final_ratio(2)=saha_ratio(0)*saha_ratio(1)/denominator
    endif
    total=0
    DO 33 j=0,l
      total=total+final_ratio(j)
33  CONTINUE
    RETURN
    END SUBROUTINE fjk_T

!**********************************************!
!     zbrent.f                                !
!     Numerical Recipies Magic Code           !
!     ASTR545, Added 10/03/06                 !
!   Adapted to use in this beast of a model   !
!                                             !
! This is the generic version on the web page !
! with the following changes: I have declared !
! ZBRENT as double precision and I have       !
! commented out the line "call dummy_call (eps)!
!          - CWC                              !
! Further changed so it bails on error instead !
! of pausing                                  !
!          - RTH                              !
!                                             !
!**********************************************!

    DOUBLE PRECISION FUNCTION ZBRENT(FUNC,X1,X2,TOL)
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    PARAMETER (ITMAX=100,EPS=3.E-16)
    A=X1
    B=X2
    FA=FUNC(A)
    FB=FUNC(B)
    IF(FB*FA.GT.0.) STOP 'Root must be bracketed for ZBRENT.'
    FC=FB
    DO 11 ITER=1,ITMAX
      IF(FB*FC.GT.0.) THEN
        C=A
        FC=FA
        D=B-A
        E=D
      ENDIF
      IF(ABS(FC).LT.ABS(FB)) THEN
        A=B
        B=C
        C=A
        FA=FB
        FB=FC
        FC=FA
      ENDIF
C      call dummy_call( eps )
      TOL1=2.*EPS*ABS(B)+0.5*TOL
      XM=.5*(C-B)
      IF(ABS(XM).LE.TOL1 .OR. FB.EQ.0.)THEN
        ZBRENT=B
        RETURN
      ENDIF
      IF(ABS(E).GE.TOL1 .AND. ABS(FA).GT.ABS(FB)) THEN
        S=FB/FA
        IF(A.EQ.C) THEN
          P=2.*XM*S
          Q=1.-S
        ELSE
          Q=FA/FC
          R=FB/FC
          P=S*(2.*XM*Q*(Q-R)-(B-A)*(R-1.))
          Q=(Q-1.)*(R-1.)*(S-1.)
        ENDIF
        IF(P.GT.0.) Q=-Q
        P=ABS(P)
        IF(2.*P .LT. MIN(3.*XM*Q-ABS(TOL1*Q),ABS(E*Q))) THEN
          E=D
          D=P/Q
        ELSE
```

```
          D=XM
          E=D
        ENDIF
      ELSE
        D=XM
        E=D
      ENDIF
      A=B
      FA=FB
      IF(ABS(D) .GT. TOL1) THEN
        B=B+D
      ELSE
        B=B+SIGN(TOL1,XM)
      ENDIF
      FB=FUNC(B)
11    CONTINUE
      PAUSE 'ZBRENT exceeding maximum iterations.'
      ZBRENT=B
      RETURN
      END

!**********************************************!
!        NATURAL CUBIC SPLINE ALGORITHM 3.4 !
!         Numerical Analysis, R. Burden & !
!             J.D. Faires, 2001        !
!         Adapted by RTH 09/19/2006    !
!                                      !
! NOTE: S(X) = A(J) + B(J)*(X-X(J)) +  !
!             C(J)*(X-X(J))**2 +       !
!             D(J)*(X-X(J))**3         !
! FOR X(J) < X < X(J+1)                !
!                                      !
!**********************************************!

      SUBROUTINE natural_spline(FILENAME, point, splinepoint)

      DIMENSION X(25),A(25),B(25),C(25),D(25),H(25),XA(25),XL(25)
      DIMENSION XU(25),XZ(25)
      double precision point, splinepoint
      INTEGER N, I, J, M
      CHARACTER FILENAME*30,AA*1
      INTEGER INP,OUP,FLAG

      OPEN(UNIT=4,FILE=FILENAME)
      N=1
50    READ(4,*,END=500) X(N),A(N)
      N=N+1
      go to 50
500   CONTINUE
      N=N-1
      CLOSE(UNIT=4)

      M=N-1
      DO 11 I=1,M
11    H(I) = X(I+1)-X(I)
      DO 21 I=2,M
21    XA(I) = 3*(A(I+1)*H(I-1)-A(I)*(X(I+1)-X(I-1))+A(I-1)*H(I))/
     *(H(I)*H(I-1))
      XL(1) = 1.0
      XU(1) = 0.0
      XZ(1) = 0.0
      DO 31 I=2,M
            XL(I) = 2*(X(I+1)-X(I-1))-H(I-1)*XU(I-1)
            XU(I) = H(I)/XL(I)
31    XZ(I) = (XA(I)-H(I-1)*XZ(I-1))/XL(I)
      XL(N) = 1.0
      XZ(N) = 0.0
      C(N) = XZ(N)
      DO 41 I=1,M
            J = N-I
            C(J) = XZ(J)-XU(J)*C(J+1)
            B(J) = (A(J+1)-A(J))/H(J)-H(J)*(C(J+1)+2*C(J))/3
41    D(J) = (C(J+1)-C(J))/(3*H(J))
      DO 51 I=1,M
        !Getting to the correct ranges to use the correct coefficients...
        IF (point .GE. X(I)) then
         IF (point .LE. X(I+1)) then
           splinepoint=A(I)+B(I)*(point-X(I))+C(I)*(point-X(I))**2
     &              +D(I)*(point-X(I))**3
         ENDIF
```

```
      ENDIF
51   CONTINUE
     RETURN
     END SUBROUTINE natural_spline
```

B. INPUT TABLES

TABLE 1
Partition functions for the species used in the model. Grey (2005)

θ	H I	He I	He II	Ca I	Ca II	Ca III
0.2	0.368	0.000	0.301	5.238	0.825	0.000
0.4	0.303	0.000	0.301	1.332	0.658	0.000
0.6	0.301	0.000	0.301	0.465	0.483	0.000
0.8	0.301	0.000	0.301	0.181	0.391	0.000
1.0	0.301	0.000	0.301	0.073	0.344	0.000
1.2	0.301	0.000	0.301	0.028	0.320	0.000
1.4	0.301	0.000	0.301	0.010	0.309	0.000
1.6	0.301	0.000	0.301	0.003	0.304	0.000
1.8	0.301	0.000	0.301	0.001	0.302	0.000
2.0	0.301	0.000	0.301	0.000	0.301	0.000

TABLE 2
Hopf function

τ	$q(\tau)$
0.00	0.577351
0.01	0.588236
0.03	0.601242
0.05	0.610758
0.10	0.627919
0.20	0.649550
0.30	0.663365
0.40	0.673090
0.50	0.680240
0.60	0.685801
0.80	0.693534
1.00	0.698540
1.50	0.705130
2.00	0.707916
2.50	0.709191
3.00	0.709806
4.00	0.710120
5.00	0.710398
∞	0.710446