Trevor Picard

Grey Stellar Atmosphere Model

```
(*********************************************)
(*INITIAL VARIABLE/LIST DECLARATIONS*)

(*physical constants & known values*)
k = 1.38065 × 10^-16; (*erg K^-1*)
h = 6.6261 × 10^-27; (*cm^2 g s^-1*)
ma = 1.66053878 × 10^-24; (*g*)
me = 9.109383 × 10^-28; (*g*)
a = 7.565767 × 10^-15; (*erg cm^-3 K^-4*)
CΦ = 2 (2 π me k / h^2)^(3/2); (*cm^-3 K^-3/2*)
Tsun = 5 777; (*K*)

(*properties of a B0I star*)
Teff = 26 000; (*K*)
logg = 2.84;
g = 10^logg; (*cm s^-2*)

(*chosen mass fractions*)
X = 0.5;
Y = 0.4;
Z = 0.1;

(*atomic mass numbers of H, He, & Si, respectively*)
AH = 1.00790;
AHe = 4.00260;
ASi = 28.08550;

(*calculates a list of abundance fractions for each element*)
α = {X/AH, Y/AHe, Z/ASi} / (X/AH + Y/AHe + Z/ASi);

(*ionization potentials from NIST, converted to erg*)
eV2erg = 1.60217733 × 10^-12;
χH = eV2erg {13.5984340};
χHe = eV2erg {24.5873876, 54.4177630};
χSi = eV2erg {8.151683, 16.345845};

(*********************************************)
(*DATA IMPORT FUNCTION*)

(*imports raw data files as lists of strings delimited by whitespace*)
PvTdata = Import["./data/PvT.txt", "Words"];
```

```
qdata = Import["./data/hopf.txt", "Words"];
Udata = Import["./data/partf.txt", "Words"];
χdata = Import["./data/OPAL_opacities.txt", "Words"];

(*function that takes an empty list & a raw
 data file and puts numeric data into the list*)
datagrab[empty_, rawdata_] := Module[{i}, empty = {};
  i = 1;
  While[i < Length[rawdata],

   (*checks for & stores numeric data and discards the rest*)
   If[NumberQ[ToExpression[rawdata[[i]]]] &&
     DigitQ[Characters[rawdata[[i]]][[1]]],
    AppendTo[empty, {rawdata[[i]], rawdata[[i + 1]]}]];
   i += 2];

  (*converts the list of strings to a list of numbers*)
  empty = ToExpression[empty]]


(************************************************)
(*OPTICAL DEPTH & TEMPERATURE OF EACH LAYER*)

(*generates a range of optical depths from 10^-3-
 10^2 with 10 τ samples dex^-1*)
τ = Table[10^(j-3), {j, 0, 5, 0.1}];

(*imports Hopf function data and interpolates to generate a list of q(τ)*)
datagrab[hopf, qdata];
q = Table[Interpolation[hopf, τ[[j]]], {j, 1, Length[τ]}];

(*calculates a list of temperatures for each layer T(τ,q(τ))*)
T = Teff (3/4 (τ + q))^(1/4);

(************************************************)
(*PRESSURE OF THE OUTERMOST LAYER*)

(*imports the P vs. T data for log(g)=4*)
datagrab[PvT, PvTdata];
loggref = 4;

(*interpolates the P vs. T data to find P_g for log(g)=4 & τ=10^-3*)
logP0ref = Interpolation[PvT, T[[1]]/Tsun];

(*finds P_g(τ=10^-3) for log(g)=2.84 given that P_g ∝ g^(2/3)*)
```

```
logP0 = logP0ref + 2/3 (logg - loggref);
P0 = 10^logP0;

(*************************************************)
(*OPACITY INTERPOLATION*)

(*generates lists of logR & logT equivalent to the OPAL opacity table*)
logℝ = Range[-8, 1, 0.5];
logϒ = Range[4, 5, 0.05];

(*finds the position of the first
 relevant logχ value in the OPAL opacity table*)
pos = First[Flatten[Position[χdata, "-0.619"]]];

(*generates a list of logχ values,
each grouped with its corresponding logR & logT*)
χtable = Flatten[Table[{{logϒ[[a]], logℝ[[b]]}, 0},
    {a, 1, Length[logϒ]}, {b, 1, Length[logℝ]}], 1];
For[
 i =
  1;
 j = 1, i ≤ Length[logϒ], ++i;
 ++pos, While[j ≤ i Length[logℝ], χtable[[j, 2]] = χdata[[pos]];
  ++pos; ++j]]

(*changes logχ (string format) to χ (number format)*)
χtable = ToExpression[χtable];
For[i = 1, i ≤ Length[χtable], ++i, χtable[[i, 2]] = 10^χtable[[i,2]]]

(*creates an opacity function for interpolation*)
χf = Interpolation[χtable];


(*************************************************)
(*PARTITION FUNCTIONS*)

(*Θ from Gray's Table D.2,
one for each T(τ) + range of values for interpolation*)
Θ = 5040/T;
Θ = Range[0.2, 2, 0.2];
AppendTo[Θ, 9999];

(*takes the logU_jk data and finds the first values of H & Si*)
datagrab[𝒰, Udata];
𝒰 = Quiet[Flatten[𝒰]];
```

```
Hpos = First[Flatten[Position[𝒰, 0.368]]];
Sipos = First[Flatten[Position[𝒰, 1.521]]];

(*stores logU_jk values in a list for each j*)
HI = Table[{Θ[[j - Hpos + 1]], 𝒰[[j]]}, {j, Hpos, Hpos + 10}];
SiI = Table[{Θ[[j - Sipos + 1]], 𝒰[[j]]}, {j, Sipos, Sipos + 10}];
SiII = Table[{Θ[[j - Sipos - 13]], 𝒰[[j]]}, {j, Sipos + 14, Sipos + 24}];

(*generates a list of U_jk for each element,
where the value for HI, SiI, & SiII are interpolated*)
UH = Table[10^{Interpolation[HI, Θ[[j]]], 0}, {j, 1, Length[Θ]}];
UHe = Table[10^{0, 0.301, 0}, {j, 1, Length[Θ]}];
USi = Table[10^{Interpolation[SiI, Θ[[j]]], Interpolation[SiII, Θ[[j]]], 0}, {j, 1, Length[Θ]}];

(************************************************)
(*DETAILED BALANCING FUNCTIONS*)

(*function that calculates the relative number densities in terms of
 electron density between adjacent ionization states given a layer (m),
ionization stage (j), ionization potential (χ), & partition function (U)*)
Yj[m_, j_, χ_, U_] := (CΦ T[[m]]^(3/2))/ne (U[[m, j + 1]])/(U[[m, j]]) e^(-χ[[j]]/(kT[[m]]))

(*calculates the first ionization fraction for an element given a layer (m),
number of possible ionizations (n),
ionization potential (χ), & partition function (U)*)
f1[m_, n_, χ_, U_] := If[n == 1, (1 + Yj[m, 1, χ, U])^-1,
  (1 + Yj[m, 1, χ, U] + Yj[m, 1, χ, U] * Yj[m, 2, χ, U])^-1]

(************************************************)
(*HI EXCITATION FRACTION FUNCTIONS*)

(*calculates the excitation energy & statistical weight of state i*)
χi[i_] := χH[[1]] (1 - 1/i^2) (*erg*)
gi[i_] := 2 i^2

(*calculates the partition function for HI using i=
 1-3 [U21 = 1 according to Table D.2 from Gray]*)
U11[m_] := Sum[gi[i] e^(-χi[i]/(kT[[m]])), {i, 1, 3}];
U21 = 1;

(*calculates the excitation fraction for HI given i, n_e, & T*)
```

```
Φ11[m_] := CΦ T[[m]]^(3/2) U21/U11[m] e^(-χH[[1]]/(k T[[m]]));

ni11n1[i_, edensity_, m_] := 1/(1 + edensity^-1 Φ11[m]) gi[i]/U11[m] e^(-χi[i]/(k T[[m]]))

(*************************************************)
(*PRESSURE CONVERGENCE LOOP FUNCTIONS*)

(*calculates n given a P_gas & T*)
nP[P_, T_] := P/(k T)

(*calculates radiation pressure for a given T*)
Prad[i_] := a/3 T[[i]]^4

(*calculates a P_i estimate from the opacity,
optical depth, and pressure of the layer above*)
Pprime[i_] := P[[i - 1]] + g (τ[[i]] - τ[[i - 1]])/(χρT[[i - 1]])

(*calculates a refined pressure estimate from the opacity,
optical depth, and pressure of the layer above*)
P2prime[i_, χ_] := P[[i - 1]] + 2 g (τ[[i]] - τ[[i - 1]])/(χρT[[i - 1]] + χ)

(*************************************************)
(*SOLUTION LISTS WITH INITAL VALUES*)

(*declarations of parameters that will contain solutions for each layer*)
P = {P0 + Prad[1]};(*dynes cm^-2*)
Pg = {P0};(*dynes cm^-2*)
PN = {};(*dynes cm^-2*)
Pe = {};(*dynes cm^-2*)
Pr = {};(*dynes cm^-2*)
PePg = {};
ne = {};(*cm^-3*)
fH = {};
fHe = {};
fSi = {};
nefH = {};(*cm^-3*)
nefHe = {};(*cm^-3*)
nefSi = {};(*cm^-3*)
ρ = {};(*g cm^-3*)
x = {0};(*cm*)
```

```
χρT = {};(*cm^2 g^-1*)
HIex = {};

(************************************************)
(************************************************)
(*BEGINNING OF LAYER SOLVING FUNCTION*)

layer[m_] := Module[
  (*local variables that are reset after each function call*)
  {first = True, last = False, start, fHv, fHev, fSiv, nev, n,
   nN, nHv, nHev, nSiv, ρN, ρe, Pp, logT, logR, equilibrium},

  (*lists of ionization fractions as a function of n_e,
  which is yet to be determined*)
  fHv = {f1[m, 1, χH, UH], f1[m, 1, χH, UH] * Yj[m, 1, χH, UH]};
  fHev = {f1[m, 2, χHe, UHe], f1[m, 2, χHe, UHe] * Yj[m, 1, χHe, UHe],
    f1[m, 2, χHe, UHe] * Yj[m, 1, χHe, UHe] * Yj[m, 2, χHe, UHe]};
  fSiv = {f1[m, 2, χSi, USi], f1[m, 2, χSi, USi] * Yj[m, 1, χSi, USi],
    f1[m, 2, χSi, USi] * Yj[m, 1, χSi, USi] * Yj[m, 2, χSi, USi]};

  (*calculation of logT_i for this layer*)
  logT = Log10[T[[m]]];

  (************************************************)
  (*STARTS THE PRESSURE CONVERGENCE LOOP*)
  Label[start];

  (*calculates n using the pressure of the (solved) layer above if it's
   the first time through the loop, otherwise P_i'' is used instead*)
  If[m == 1, n = nP[Pg[[m]], T[[m]]],
   If[first, Pp = Pprime[m], Pp = P2prime[m, χf[logT, logR]]];
   n = nP[Pp - Prad[m], T[[m]]]];

  (*solves the appropriate equation numerically
   for a positive electron density, then assigns the value
   of the solution to a variable*)nev = n_e /. Flatten[Solve[
       Reduce[n_e == (n - n_e) (α[[1]] * fHv[[2]] + α[[2]] (fHev[[2]] + 2 fHev[[3]]) +
             α[[3]] (fSiv[[2]] + 2 fSiv[[3]])) && n_e > 0, n_e]][[1]]];

  (*calculations leading to logR(n_e),*)
  nN = n - nev;
  ρe = me nev;
  ρN = nN ma / (X/AH + Y/AHe + Z/ASi);
  logR = Log10[(ρe + ρN) / (T[[m]] × 10^-6)];
```

```
(*prevents the pressure convergence loop from
 being applied if the layer is ready to be fully solved
 (i.e., if it's the first layer or if the pressure loop has converged)*)
If[(m == 1) || last, Goto[equilibrium]];

(*checks if pressure convergence meets the tolerance level. if YES,
Pi = Pi'' and detailed balance is solved. if NO, the loop repeats*)
If[Abs[(P2prime[m, χf[logT, logR]] - Pp)/Pp] ≤ 10^-5,
 AppendTo[P, P2prime[m, χf[logT, logR]]];
 last = True];

(*restarts the pressure convergence loop with Pi' =
 Pi'' if hydrostatic equilibrium wasn't achieved*)
first = False;
Goto[start];

(**********************************************)

(*starting point once hydrostatic equilibrium is achieved*)
Label[equilibrium];


(*calculates njk and fjk(ne)*)
nHv = nN α[[1]];
nHev = nN α[[2]];
nSiv = nN α[[3]];
fHv = fHv /. ne → nev;
fHev = fHev /. ne → nev;
fSiv = fSiv /. ne → nev;

(*adds the parameters of the solved layer to their arrays*)
AppendTo[fH, fHv];
AppendTo[fHe, fHev];
AppendTo[fSi, fSiv];
AppendTo[nefH, fHv[[2]] nHv];
AppendTo[nefHe, {fHev[[2]] nHev, 2 fHev[[3]] nHev}];
AppendTo[nefSi, {fSiv[[2]] nSiv, 2 fSiv[[3]] nSiv}];
AppendTo[ne, nev];
AppendTo[Pe, nev k T[[m]]];
AppendTo[PN, nN k T[[m]]];
AppendTo[Pr, Prad[m]];
If[m > 1, AppendTo[x, (τ[[m]] - τ[[m - 1]])/(χf[logT, logR] (ρN + ρe)) + x[[m - 1]]];
 AppendTo[Pg, Pe[[m]] + PN[[m]]]];
```

```
  AppendTo[PePg, Pe[[m]]/Pg[[m]]];
  AppendTo[ρ, ρN + ρe];
  AppendTo[χρT, χf[logT, logR]];
  AppendTo[HIex, Table[ni11n1[i, nev, m], {i, 1, 3}]]
  (*END OF LAYER SOLVING FUNCTION*)]

(************************************************)

(************************************************)
(*SOLVING AND SAVING ALL ATMOSPHERIC LAYERS*)

(*loops through the layer solving function*)
For[i = 1, i ≤ Length[τ], ++i, layer[i]]

(*saves all of the data to a txt file,
where each column corresponds to a different quantity,
and each row to a different layer of the atmosphere*)
data = Table[{τ[[j]], x[[j]], Log10[T[[j]]], χρT[[j]], Log10[ρ[[j]]],
    P[[j]], Pg[[j]], PN[[j]], Pe[[j]], Pr[[j]], PePg[[j]],
    Log10[fH[[j, 1]]], Log10[fH[[j, 2]]], Log10[fHe[[j, 1]]],
    Log10[fHe[[j, 2]]], Log10[fHe[[j, 3]]], Log10[fSi[[j, 1]]],
    Log10[fSi[[j, 2]]], Log10[fSi[[j, 3]]], Log10[ne[[j]]],
    Log10[nefH[[j]]], Log10[nefHe[[j, 1]]], Log10[nefHe[[j, 2]]],
    Log10[nefSi[[j, 1]]], Log10[nefSi[[j, 2]]], Log10[HIex[[j, 1]]],
    Log10[HIex[[j, 2]]], Log10[HIex[[j, 3]]]}, {j, 1, Length[τ]}];
Export["atm_data.txt", data, "Table"];
```