Nikhil Patten
27 October 2022
Dr. Moe
ASTR5420

95%

Please show all work. If you collaborate with other students, write their names at the top of your homework. Please also write the total number of hours you spent on this. If you use any code or plotting routines, print those out and attach them to your hand-written solutions.
This assignment took me approximately 6 hours to complete.
I collaborated with Alex and Chase on this assignment.

1. **(15%): Fusion Efficiencies and Stellar Lifetimes:**

(a) **Show that the fractional mass change while fusing four protons into one He nucleus during the proton-proton chain is $\Delta m/m \approx 0.7\%$. Assuming the Sun fuses 10% of its hydrogen into helium, demonstrate that the solar main-sequence lifetime is 10 Gyr.** ?

+12/15

$$m_H = 1.007276466621 \text{ Da}$$
$$m_{He} = 4.002603254 \text{ Da}$$
$$1 \text{ Da} = 1.66053906660 \text{ kg}$$
$$\frac{\Delta m}{m} = \frac{m_f - m_i}{m_i}$$
$$\frac{\Delta m}{m} = \frac{4.002603254\,(1.66053906660) - 4\,(1.007276466621)\,(1.66053906660)}{4\,(1.007276466621)\,(1.66053906660)}$$

what is this? doesn't really matter since it cancels

$$\boxed{\frac{\Delta m}{m} = -0.658\%}$$

Code:

```
In [19]: m_p = 1.007276466621 *(1.66053906660)
         # NP Mass of proton in kg
         m_he = 4.002603254 *(1.66053906660)
         # NP Mass of 4He in kg
         print('delta m/m: ' +format((m_he -4 *m_p)\
             *100 /(4 *m_p), '.2E') +' % (proton-proton chain)')
         # NP Printing change in mass percent

delta m/m: -6.58E-01 % (proton-proton chain)
```

(b) **Demonstrate that the triple a reaction (fusing three He nuclei into one C nucleus) releases only 10% the energy of the proton-proton chain, i.e., $\Delta m/m \approx 0.07\%$. This should demonstrate why the horizontal branch (He core burning) lifetime is 1 Gyr for the Sun, or in general 10% of the MS lifetime relatively independent of stellar mass.**

$$m_C = 12 \text{ Da}$$
$$\frac{\Delta m}{m} = \frac{m_f - m_i}{m_i}$$
$$\frac{\Delta m}{m} = \frac{12\,(1.66053906660) - 3\,(4.002603254)\,(1.66053906660)}{3\,(4.002603254)\,(1.66053906660)}$$
$$\boxed{\frac{\Delta m}{m} = -0.0650\%}$$

Code:

```
In [20]: m_c = 12 *(1.66053906660)
         # NP Mass of 12C in kg
         print('delta m/m: ' +format((m_c -3 *m_he)\
             *100 /(3 *m_he), '.2E') +' % (triple alpha)')
         # NP Printing change in mass percent

         delta m/m: -6.50E-02 % (triple alpha)
```

The triple-alpha process produces one-tenth the energy as the proton-proton chain!

+20/20

2. **(20%) Compute the distance a proton must quantum tunnel through the Coulomb barrier to achieve nuclear fusion in the center of our Sun. Assume the nuclear binding energy dominates below $r < 3$ fermi, the central temperature of our sun is 15 million K, and the kinetic energy of the proton is $E_{kin} = \frac{1}{2}m_p v_p^2$. Compute the distance for two cases: (a) the proton is travelling at the upper 10th percentile of the Maxwell-Boltzmann velocity distribution, and (b) the fastest 0.1%. Express your answers in fermi units.**
For $r > 3$ fm, assume Coulomb potential.

$$U = \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2}{r^2}$$
$$K = \frac{1}{2}m_p v_p^2$$

Find distance corresponding to coulomb potential from kinetic energy.

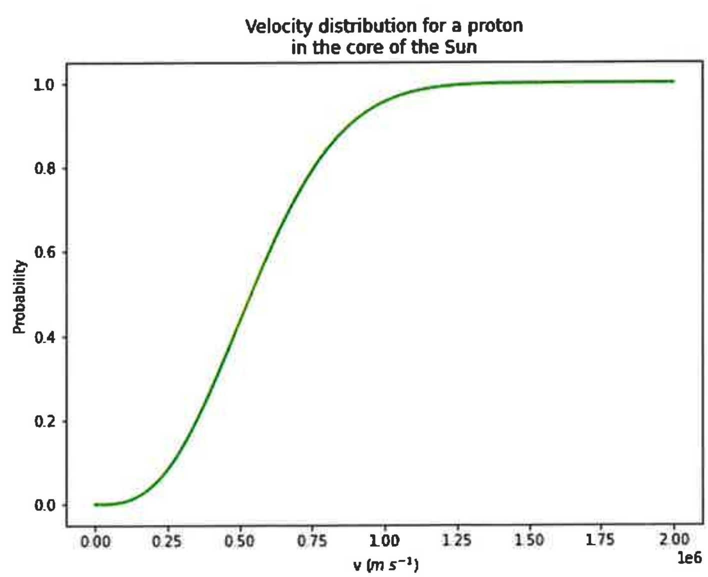
$$\frac{1}{2}m_p v_e^2 = \frac{1}{4\pi\epsilon_0}\frac{e^2}{r}$$
$$r = \frac{1}{2\pi\epsilon_0}\frac{e^2}{m_p v_e^2}\ \mathrm{m} \times \frac{10^{15}\ \mathrm{fm}}{1\ \mathrm{m}}$$
$$r = \frac{(10^{15})\, e^2}{2\pi\epsilon_0 m_p v_e^3}\ \mathrm{fm}$$

The distance tunneled will therefore be the difference between 3 fm and the calculated result above based on the speed of the particles in plasma. To find speed at different percentiles, we need to integrate the Maxwell velocity distribution and find the speed associated with the given percentile.
Code:

```
In [4]: def f_V(m, T, V):
            '''Function to return the velocity distribution
            at a given temperature for a particle.
            Inputs:
            ----------------------------------------------
            m: Mass of the particle in kg. -float.
            T: Temperaure in Kelvin. -float
            V: Velcoities to run distribution over. -np.array
            Returns:
            ----------------------------------------------
            f_V: Velocity distribution for inputted parameters
            -np.array'''
            k = 1.38e-23
            # NP Boltzmann constant in kgs units
            f_V = V **2 *np.exp(-1 *(m *V **2) /(2 *k *T))\
                *4 *np.pi *((m) /(2 *np.pi *k *T)) **1.5
            return f_V
```

```
In [40]: V = np.linspace(1, 2000000, 10000)
         # NP Defining velocity grid to evaluate velocity distribution on
         T_c = 15e6
         # NP Central temperature of the Sun
         v_distrib = f_V(m_p, 15 *10 **6, V)
         # NP Calculating velocity distribution for the Sun
         integrate_dis = np.array([np.trapz(v_distrib[V < i], V[V < i])\
             for i in V])
         # NP Inegrating distribution to find probability distribution
         f = plt.figure(figsize = [8, 6])
         # NP Making figure larger
         plt.plot(V, integrate_dis)
         # NP Plotting velocity distribution
         plt.xlabel(r'v ($m$ $s^{-1}$)')
         plt.ylabel(r'Probability')
         # NP Labeling axes
         plt.title('Velocity distribution for a proton\n'
             'in the core of the Sun')
         # NP Labeling plot
         plt.savefig('/d/www/nikhil/public_html/ASTR5420/images/'\
             'sunvdistrib.png')
         # NP Saving plot
```

Plot:



good idea!

To find speeds at the different percentiles, find the index of of the minimum value of the absolute difference between the integrated velocity distribution and the percentile. In other words, find the velocity when the probability is closest to a given percentile.

Code:

```
In [45]: percentiles = [0.9, 0.999]
         # NP Defining percentiles to find speed for
         args = np.array([np.argmin(np.abs(integrate_dis -i))\
             for i in percentiles])
         # NP Finding index of speeds corresponding to percentiles
         vs = V[args]
         # NP Finding speeds corresponding to percentiles
         eps = 8.85e-12
         # NP Defining permetivitty of free space in mks units
         rs = (10**15 *(1.6e-19)**2) /(2 *np.pi *eps *m_p\
             *vs**2)-3
         # NP Calculating tunneling distance for different speeds
         print('Tunneling distance for upper 10 percentile: '\
             +format(rs[0], '.2E') +' fm')
         print('Tunneling distance for upper 0.1 percentile: '\
             +format(rs[1], '.2E') +' fm')
         # NP Printing results

         Tunneling distance for upper 10 percentile: 3.53E+02 fm
         Tunneling distance for upper 0.1 percentile: 1.34E+02 fm
```

$$\boxed{r_{10\%} = 353 \text{ fm}}$$

$$\boxed{r_{0.1\%} = 134 \text{ fm}}$$

3. **(25%) Critical Masses for Fusion - use the relation we derived in class for a star's central temperature $T_c = 0.65GMm_H/kR$ in the following:**

x 23/25

(a) **Energy generation of the CNO cycle exceeds the proton-proton chain when the central temperature exceeds 17 million K. For what MS masses (in $M_\odot$) does this occur?**

$$T_c = \frac{0.65GMm_H}{kR}$$

$$T_c = \frac{0.65GMm_H}{kR_\odot\left(\frac{M}{M_\odot}\right)^{0.8}}$$

$$T_c = \frac{0.65GM^{0.2}m_H}{kR_\odot}M_\odot^{0.8}$$

$$M^{0.2} = \frac{kR_\odot T_c}{0.65Gm_H}M_\odot^{-0.8}$$

$$M = \left(\frac{kR_\odot T_c}{0.65Gm_H}\right)^5 M_\odot^{-4}$$

$$M\,(M_\odot) = \left(\frac{kR_\odot T_c}{0.65Gm_H}\right)^5 M_\odot^{-4}\frac{1}{M_\odot}$$

$$M\,(M_\odot) = \left(\frac{kR_\odot T_c}{0.65Gm_H M_\odot}\right)^5$$

$$\boxed{M\,(M_\odot) = 1.80M_\odot}$$

Code:

```
In [35]: R_o = 6.957e8
         # NP Radius of Sun
         M_o = 2e30
         # NP Mass of Sun
         G = 6.67e-11
         # NP Gravatational constant
         k = 1.38e-23
         # NP Boltzmann constant
         T_ci = np.array([17e6, 11e6])
         # NP Temperatures
         M_cs = (k *R_o *T_ci /(0.65 *G *m_p *M_o))**5
         # NP Calculating requires MS masses to achieve central temperatures
         print('Required mass for CNO cycle: ' +format(M_cs[0],\
             '.2E') +' Solar masses')
         # NP Printing result

Required mass for CNO cycle: 1.80E+00 Solar masses
```

$R \propto M^{0.8}$ valid only across $0.2-20\,M_\odot$

(b) Estimate the minimum mass of a MS star (in $M_\odot$) to fuse hydrogen via the proton-proton chain, which requires a central temperature above 11 million K.

$R = 0.1R_\odot$ for $M \lesssim 0.2 M_\odot$

For fully convective stars: $R = 0.1 R_\odot = 1 R_J$ ~~$R = 0.1 R_\odot$~~ like in c

$$M\,(M_\odot) = \left(\frac{kR_\odot T_c}{0.65 G m_H M_\odot}\right)^5$$

$$\boxed{M\,(M_\odot) = 0.205 M_\odot}$$

should be $0.08 M_\odot$

Code:

```
In [36]: print('Required mass for proton-proton chain: ' +format\
             (M_cs[1], '.2E') +' Solar masses')
         # NP Printing result

Required mass for proton-proton chain: 2.05E-01 Solar masses
```

(c) Estimate the minimum mass of a brown dwarf (in M_J) to fuse deuterium, which requires a central temperature above 2 million K.

Main sequence mass-radius relation no longer valid. For brown dwarfs, radius nearly constant ($\approx R_J$).

$$T_c = \frac{0.65 G M m_H}{k R_J}$$

$$M = \frac{k R_J T_c}{0.65 G m_H}$$

$$M\,(M_J) = \frac{k R_J T_c}{(0.65)\,G m_H M_J}$$

$$\boxed{M\,(M_J) = 14.4 M_J}$$

Code:

```
In [40]: R_J = 7.15e7
         # NP Radius of Jupiter
         T_d = 2e6
         # NP Temperature for deuterium fusion
         M_J = 1.90e27
         # NP Mass of Jupiter
         M_bd = (k *R_J *T_d) /(G *0.65 *m_p *M_J)
         # NP Calulating mass required for deuterium fusion
         print('Brown dwarf minimum mass: ' +format(M_bd, '.2E')\
             +' Jupiter masses')
         # NP Printing result

Brown dwarf minimum mass: 1.43E+01 Jupiter masses
```

(d) **Estimate the minimum mass (in $M_\odot$) of a degenerate core of a giant to achieve triple α reactions, thereby lifting the degeneracy. Triple α reactions require central temperatures above 500 million K. What is the name for this short-lived phase in stellar evolution?**
For White Dwarf (applicable because core is degenerate):

$$R = 0.01 R_\odot \left(\frac{M}{0.8 M_\odot}\right)^{-1/3}$$

Apply this relation to find the mass of a degenerate helium core that achieves helium fusion.

$$T_c = \frac{0.65 G M m_H}{kR}$$

$$T_c = \frac{0.65 G M m_H}{(0.01)\, k R_\odot \left(\frac{M}{0.8 M_\odot}\right)^{-1/3}}$$

$$T_c = \frac{65 G M m_H M^{1/3}}{k R_\odot (0.8 M_\odot)^{1/3}}$$

$$T_c = \frac{65 G m_H M^{4/3}}{k R_\odot (0.8 M_\odot)^{1/3}}$$

$$M^{4/3} = \frac{k R_\odot (0.8)^{1/3}}{65 G m_H} (M_\odot)^{1/3}$$

$$M = \left(\frac{k R_\odot T_c (0.8)^{1/3}}{65 G m_H}\right)^{3/4} M_\odot^{1/4}$$

$$M\,(M_\odot) = \left(\frac{k R_\odot T_c (0.8)^{1/3}}{65 G m_H}\right)^{3/4} M_\odot^{-3/4}$$

$$M\,(M_\odot) = \left(\frac{k R_\odot T_c (0.8)^{1/3}}{65 G m_H M_\odot}\right)^{3/4}$$

$$\boxed{M_c = 0.413 M_\odot}$$

Code:

```
In [66]: T_he = 500e6
         # NP Temperature for triple alpha process
         M_he = ((k *R_o *T_he *(0.8) **(1/3)) /(65 *G *m_p *M_o))\
             **(.75)
         # NP Calculating mass for deuterium fusion
         print('Mass of core needed for deuterium fusion: '\
             +format(M_he, '.2E') +' Solar masses')
         # NP Printing result

         Mass of core needed for deuterium fusion: 4.13E-01 Solar masses
```

x40/40

4. **(40%) Modeling the Structure of a Fully Convective M-dwarf**

(a) i. **You will model the structure of an M-dwarf just below the boundary that separates fully convective stars from those with small radiative cores. Initially assume a central temperature of T_c = 11 million K (just enough to achieve proton-proton chain nuclear fusion) and central density of ρ_c = 25 g cm^{-3}. The initial core temperature and density determines the overall stellar radii R_*, mass M_*, and luminosity L_* (see part d).**

ii. **Assume solar-like abundance with X = 0.74 and Y = 0.26, where we can neglect metals Z since we are modeling fully convective stars and therefore don't need Rosseland mean** opacities. Compute the mean molecular weight μ for fully neutral atoms (Eqn. 5.127), which adequately describes most of the gas in our cool M-dwarf. Then use the ideal gas law (Eqn. 5.107) to compute the central pressure P_c. At the center of the star, the radius is $r = 0$ and the enclosed masses $M(r) = 0$ and luminosities $L(r)$ are also zero.

$$\boxed{\mu = 1.24}$$

$$\boxed{P_c = 1.83 \times 10^{15} \text{N m}^{-2}}$$

Code:

```
In [70]: X = 0.74
         # NP Hydrogen fraction
         Y = 0.26
         # NP Helium fraction
         mu = 1 /(X +Y /4)
         # Equation 5.127
         print('mu: ' +format(mu, '.2E'))
         # NP Printing result
         P_c = rho_c *k *T_c /(mu *m_p)
         print('P_c: ' +format(P_c, '.2E') +' Nm^-2')

         mu: 1.24E+00
         P_c: 1.83E+15 Nm^-2
```

iii. Take a small step $\Delta r = 10^{-5} R_\odot$ outward. Make sure to keep track of $T(r)$, $\rho(r)$, $P(r)$, $M(r)$, and $L(r)$ at each step in radius. You will use this fixed step and Euler's method to numerically integrate the differential equations of stellar structure. In practice, astronomers typically use adaptive radial steps and a Runge-Kutta method, which requires numerical evaluations of the second derivatives of the stellar structure equations. But for sufficiently small Dr, Euler's linear method and the first derivatives are sufficient. If you want, you can test convergence of your solutions by adopting different step sizes.

iv. Compute the mass ΔM in that shell with radius r and width Δr using the equation of mass conservation (Eqn. 5.4) and your previously determined density ρ. Add this shell mass ΔM to your previously determined enclosed mass $M(r)$, and update $M(r)$ accordingly.

v. Convert your previously determined temperature into units of $T_9 = T/10^9$ K. Then compute the energy production rate per unit mass for the proton-proton chain according to Eqn. 6.25 (which is in units of erg s^{-1} g^{-1}) and your previously evaluated T_9 and ρ. Then compute the luminosity ΔL in that shell with radius r and width Δr using the energy conservation equation (Eqn. 5.22). Add this shell luminosity ΔL to your previously determined enclosed luminosity $L(r)$, and update $L(r)$ accordingly.

vi. Compute the temperature change ΔT assuming energy transport is fully convective (Eqn. 5.81) and the equation of state is an ideal monatomic gas ($\gamma = 5/3$). As before, use your previously determined ρ, T, P and $g = GM(r)/r^2$. Add this temperature change ΔT to your previously determined temperature to update $T(r)$.

vii. Compute the pressure change DP according to the equation for hydrostatic equilibrium (Eq. 5.1), again assuming your previously determined values for ρ and $M(r)$. Add this pressure change ΔP to your previously determined pressure to update $P(r)$.

viii. Finally, update your density $\rho(r)$ using the same ideal gas law and mean molecular weight as in part ii, now using your updated values for $P(r)$ and $T(r)$.

ix. Repeat steps iii – viii until the temperature falls below $T < 3{,}000$ K (just above the photosphere of an M-dwarf).

Code:

In [76]:
```
T_c = 11e6
# NP Central temperature in K
rho_c = 25 *1000
# NP Central density in kg m^-3
dr = 10 **-5 *R_o
# NP Radius step
T = [T_c]
# NP Initial temperature array
M = [0]
# NP Inital mass array
rho = [rho_c]
# NP Initial density array
L = [0]
# NP Initial luminosity array
R = [0]
# NP Inital radius array
g = [0]
# NP Inital surface gravity array
P = [P_c]
# NP Inital pressure array
i = 0
# NP Iterator value
L_o = 3.828e26
# NP Solar Luminosity
```

In [51]:
```
dr = 10 **-5 *R_o
# NP Radius step
T = [T_c]
# NP Initial temperature array
M = [0]
# NP Inital mass array
rho = [rho_c]
# NP Initial density array
L = [0]
# NP Initial luminosity array
R = [0]
# NP Inital radius array
g = [0]
# NP Inital surface gravity array
P = [P_c]
# NP Inital pressure array
i = 0
# NP Iterator value
L_o = 3.828e26
# NP Solar Luminosity
```

```
In [52]: while(T[i] > 3000):
         # NP Iterating until temperature drops below 3000 K
             R.append(R[i]+dr)
             # NP updating radius
             M.append(M[i] +rho[i] *4 *np.pi *R[i+1]**2 *dr)
             # NP Updating mass
             T.append(T[i] -((2/5) *(rho[i] *G *M[i+1]\
                 *T[i] *dr)/(P[i] *R[i+1]**2)))
             # NP Updating temperature
             en = 2.4 *(rho[i] /1000) *X**2 /((T[i+1] /\
                 (10 **9)) **(2/3)) *np.exp(-3.38/\
                 (T[i+1]/(10**9)) **(1/3))
             # NP Calculating energy per unit mass for a shell
             L.append(L[i] +4*np.pi*R[i+1]**2*dr *rho[i] *en)
             # NP Updating Luminosity
             g.append(G *M[i+1]/(R[i+1]**2))
             # NP Updating surface gravity
             P.append(P[i] -rho[i] *g[i+1]*dr)
             # NP Updating pressure
             rho.append(mu *m_p *P[i+1] /(k *T[i+1]))
             # NP Updating density
             i += 1
             # NP Increasing step
         R = np.array(R)
         M = np.array(M)
         T = np.array(T)
         rho = np.array(rho)
         P = np.array(P)
         g = np.array(g)
         L = np.array(L)
         # NP Converting lists to arrays at the end
```

(b) **What are your final values for stellar radius R_*, mass M_*, and luminosity L_* (all in solar units). Given your computed M_*, what values for R_* and L_* would you have expected from the standard main-sequence relations? Are your solutions close?**
Code:

```
In [59]: print('Final radius: ' +format(R[len(R)-1] /R_o, '.2E') +' R_o')
         print('Final mass: ' +format(M[len(M)-1] /M_o, '.2E') +' M_o')
         print('Final luminosity: ' +format(L[len(L)-1] /L_o, '.2E') +' L_o')

         Final radius: 4.90E-01 R_o
         Final mass: 3.47E-01 M_o
         Final luminosity: 3.31E-02 L_o
```

$$\boxed{R_* = 0.490 R_\odot}$$
$$\boxed{M_* = 0.347 M_\odot}$$
$$\boxed{L_* = 0.0331 L_\odot}$$

From MS relations and calculated mass, compute expected radius and luminosity.

$$M_* = 0.347 M_\odot$$
$$R_* \approx \left(\frac{M_*}{M_\odot}\right)^{0.8}$$
$$L_* \approx \left(\frac{M_*}{M_\odot}\right)^{3.5}$$

$$\boxed{R_* \approx 0.429 R_\odot}$$
$$\boxed{L_* \approx 0.0246 L_\odot}$$

Code:

```
In [69]: R_p = (M[len(M)-1] /M_o) **0.8
         # NP Calculating predicted radius from MS relation
         print('Predicted R: ' +format(R_p, '.2E') +' R_o')
         # NP Printing result
         L_p = (M[len(M)-1] /M_o) **3.5
         # NP Calculating predicted luminosity from MS relation
         print('Predicted L: ' +format(L_p, '.2E') +' L_o')
         # NP Printing result

Predicted R: 4.29E-01 R_o
Predicted L: 2.46E-02 L_o
```

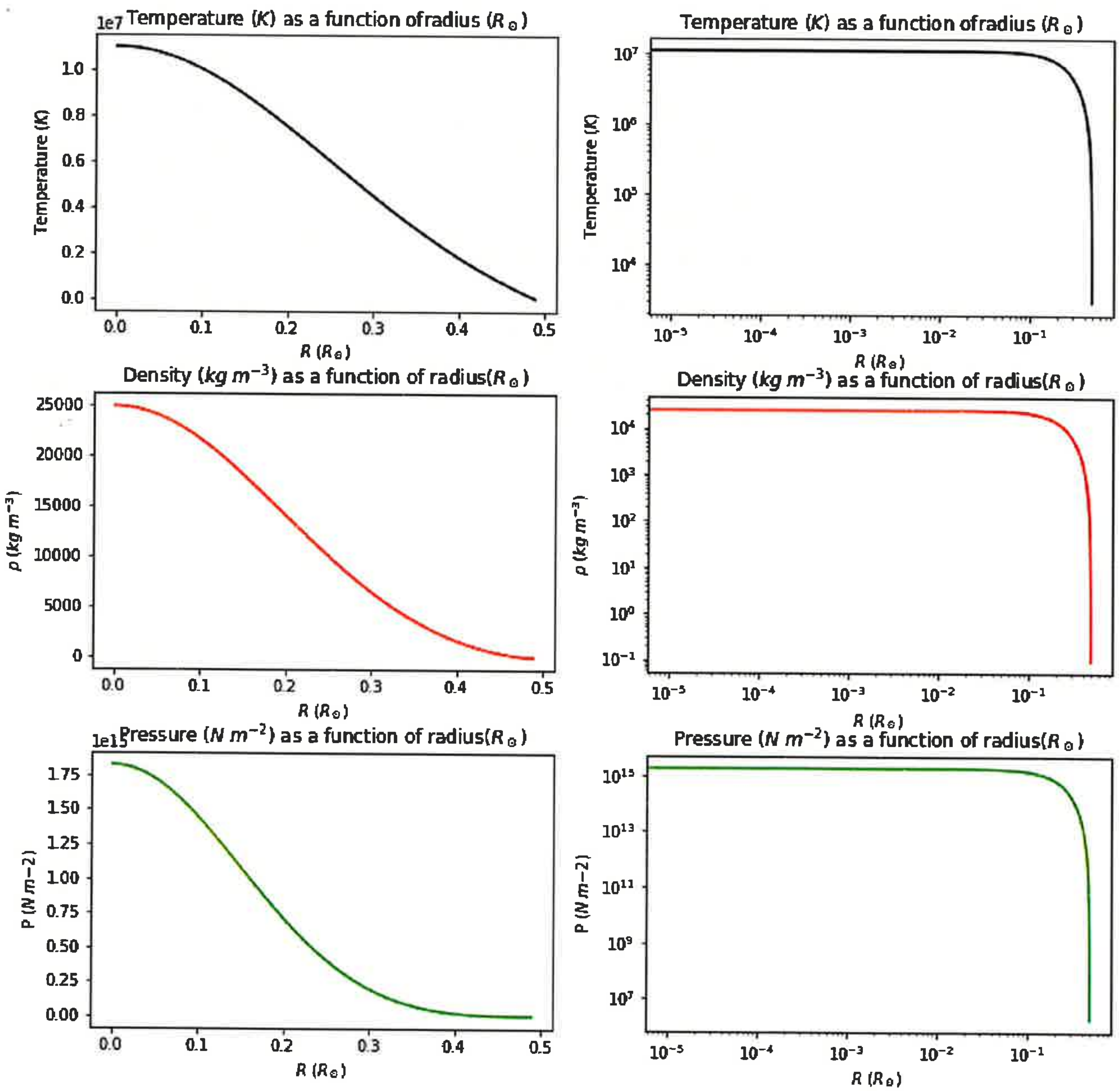
Given my calculated M_*, my other calculated values for R_* and L_* roughly match up with predicted values for R_* and L_* from MS relations.

(c) **Plot $T\,(r/R_\odot)$, $\rho\,(r/R_\odot)$, $P\,(r/R_\odot)$, $M/M_\odot\,((r/R_\odot)$, and $L/L_\odot\,(r/R_\odot)$ in both linear space and log-space to get a feel for the different profiles of stellar structure.**
Code:

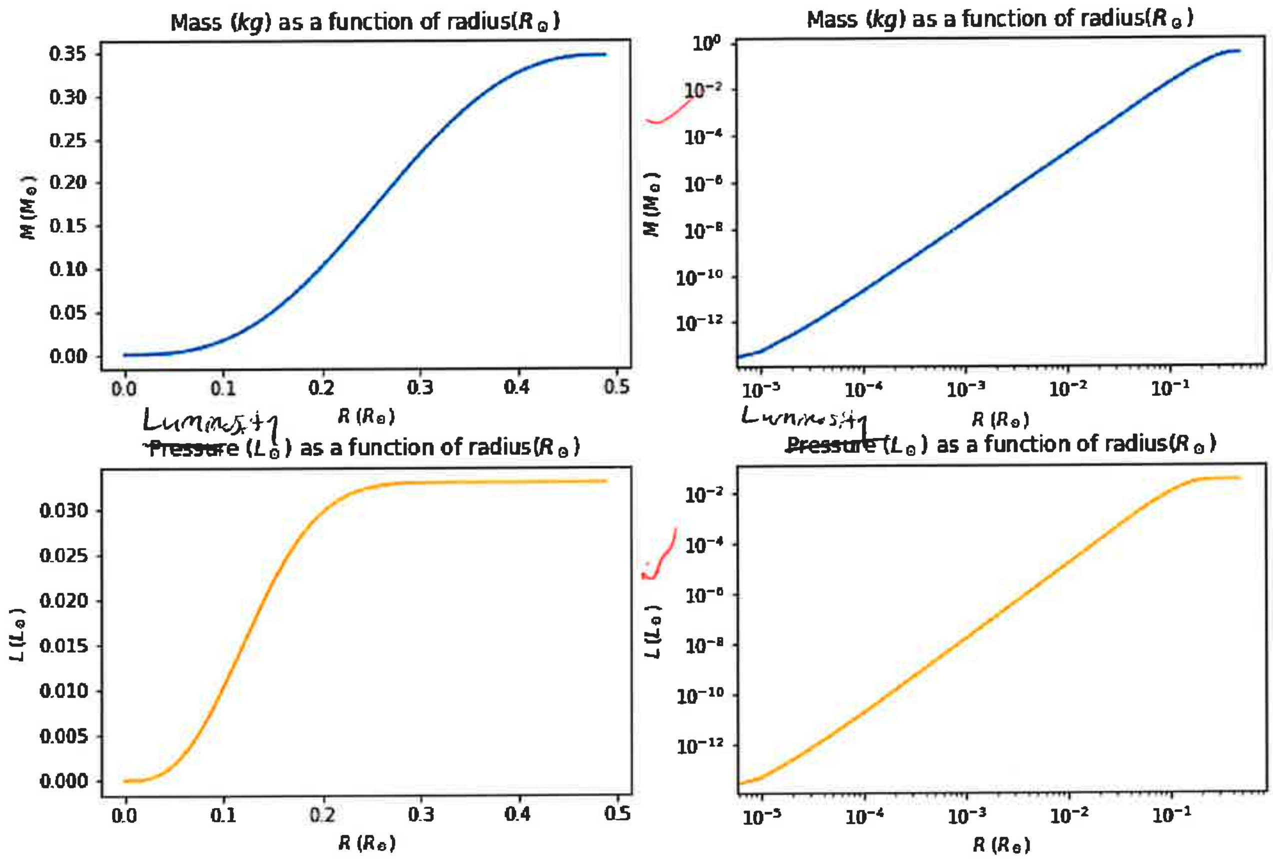
```
In [140]: plt.figure(figsize = [12, 12])
          # NP Making figure larger
          plt.subplots_adjust(hspace=.3)
          plt.subplot(3, 2, 1)
          # NP First plot
          plt.title(r'Temperature $(K)$ as a function of'
              'radius $(R_\odot)$')
          plt.plot(R/R_o, T, 'k')
          plt.xlabel(r'$R$ $(R_\odot)$')
          plt.ylabel('Temperature $(K)$')
          plt.subplot(3, 2, 2)
          # NP Second plot
          plt.title(r'Temperature $(K)$ as a function of'
              'radius $(R_\odot)$')
          plt.plot(R/R_o, T, 'k')
          plt.xlabel(r'$R$ $(R_\odot)$')
          plt.ylabel('Temperature $(K)$')
          plt.yscale('log')
          plt.xscale('log')
          plt.subplot(3, 2, 3)
          # NP Third plot
          plt.title(r'Density $(kg$ $m^{-3})$ as a function of radius'
              '$(R_\odot)$')
          plt.plot(R/R_o, rho, 'r')
          plt.xlabel(r'$R$ $(R_\odot)$')
          plt.ylabel(r'$\rho$ $(kg$ $m^{-3})$')
          plt.subplot(3, 2, 4)
          # NP Fourth plot
          plt.title(r'Density $(kg$ $m^{-3})$ as a function of radius'
              '$(R_\odot)$')
          plt.plot(R/R_o, rho, 'r')
          plt.xlabel(r'$R$ $(R_\odot)$')
          plt.ylabel(r'$\rho$ $(kg$ $m^{-3})$')
          plt.yscale('log')
          plt.xscale('log')
      plt.subplot(3, 2, 5)
      # NP Fifth plot
      plt.title(r'Pressure $(N$ $m^{-2})$''\n as a function of radius'
          '$(R_\odot)$')
      plt.plot(R/R_o, P, 'g')
      plt.xlabel(r'$R$ $(R_\odot)$')
      plt.ylabel('P $(N$ $m{-2})$')
      plt.subplot(3, 2, 6)
      # NP Sixth plot
      plt.title(r'Pressure $(N$ $m^{-2})$''\n as a function of radius'
          '$(R_\odot)$')
      plt.plot(R/R_o, P, 'g')
      plt.xlabel(r'$R$ $(R_\odot)$')
      plt.ylabel('P $(N$ $m{-2})$')
      plt.yscale('log')
      plt.xscale('log')
      plt.savefig('/d/www/nikhil/public_html/ASTR5420/images/'
          'tpPmlrelations1.png')
      # NP Saving figure
```

In [145]:
```
plt.figure(figsize = [12, 8])
# NP Setting figure size
plt.subplots_adjust(hspace=.3)
# NP Adjusting figure spacing
plt.subplot(2, 2, 1)
# NP First plot
plt.title(r'Mass $(kg)$ as a function of radius'
    '$(R_\odot)$')
plt.plot(R/R_o, M/M_o, 'b')
plt.xlabel(r'$R$ $(R_\odot)$')
plt.ylabel('$M$ $(M_\odot)$')
plt.subplot(2, 2, 2)
# NP Second plot
plt.title(r'Mass $(kg)$ as a function of radius'
    '$(R_\odot)$')
plt.plot(R/R_o, M/M_o, 'b')
plt.xlabel(r'$R$ $(R_\odot)$')
plt.ylabel('$M$ $(M_\odot)$')
plt.yscale('log')
plt.xscale('log')
plt.subplot(2, 2, 3)
# NP Third plot
plt.title(r'Pressure $(L_\odot)$ as a function of radius'
    '$(R_\odot)$')
plt.plot(R/R_o, L/L_o, color = 'orange')
plt.xlabel(r'$R$ $(R_\odot)$')
plt.ylabel('$L$ $(L_\odot)$')
plt.subplot(2, 2, 4)
# NP Fourth plot
plt.title(r'Pressure $(L_\odot)$ as a function of radius'
    '$(R_\odot)$')
plt.plot(R/R_o, L/L_o, color = 'orange')
plt.xlabel(r'$R$ $(R_\odot)$')
plt.ylabel('$L$ $(L_\odot)$')
plt.xscale('log')
plt.yscale('log')
plt.savefig('/d/www/nikhil/public_html/ASTR5420/images/'
    'tpPmlrelations2.png')
# NP Saving figure
```

Plot:



1e7 Temperature (K) as a function ofradius (R ⊙)
Temperature (K)
R (R⊙)
Temperature (K) as a function ofradius (R ⊙)
Temperature (K)
R (R⊙)
Density (kg m−3) as a function of radius(R ⊙)
ρ (kg m−3)
R (R⊙)
Density (kg m−3) as a function of radius(R ⊙)
ρ (kg m−3)
R (R⊙)
1e15 Pressure (N m−2) as a function of radius(R ⊙)
P (N m−2)
R (R⊙)
Pressure (N m−2) as a function of radius(R ⊙)
P (N m−2)
R (R⊙)



(d) **Increase the initial central temperature Tc by 10% (while fixing $\rho_c = 25$ g cm^{-3}), and report your final R_*, M_*, and L_* (all in solar units). Similarly, increase the central density by 10% (while fixing $T_c = 11$ million K) and report R_*, M_*, and L_*.**

Code:

In [147]:
```
T_c = 11e6 *1.1
# NP Central temperature in K
rho_c = 25 *1000
# NP Central density in kg m^-3
X = 0.74
# NP Hydrogen fraction
Y = 0.26
# NP Helium fraction
mu = 1 /(X +Y /4)
# Equation 5.127
# NP Printing result
P_c = rho_c *k *T_c /(mu *m_p)
dr = 10 **-5 *R_o
# NP Radius step
T = [T_c]
# NP Initial temperature array
M = [0]
# NP Inital mass array
rho = [rho_c]
# NP Initial density array
L = [0]
# NP Initial luminosity array
R = [0]
# NP Inital radius array
g = [0]
# NP Inital surface gravity array
P = [P_c]
# NP Inital pressure array
i = 0
# NP Iterator value
while(T[i] > 3000):
# NP Iterating until temperature drops below 3000 K
    R.append(R[i]+dr)
    # NP updating radius
    M.append(M[i] +rho[i] *4 *np.pi *R[i+1]**2 *dr)
    # NP Updating mass
    T.append(T[i] -((2/5) *(rho[i] *G *M[i+1]\
        *T[i] *dr)/(P[i] *R[i+1]**2)))
    # NP Updating temperature
    en = 2.4 *(rho[i] /1000) *X**2 /((T[i+1] /\
        (10 **9)) **(2/3)) *np.exp(-3.38/\
        (T[i+1]/(10**9)) **(1/3))
    # NP Calculating energy per unit mass for a shell
    L.append(L[i] +4*np.pi*R[i+1]**2*dr *rho[i] *en)
    # NP Updating Luminosity
    g.append(G *M[i+1]/(R[i+1]**2))
    # NP Updating surface gravity
    P.append(P[i] -rho[i] *g[i+1]*dr)
    # NP Updating pressure
    rho.append(mu *m_p *P[i+1] /(k *T[i+1]))
    # NP Updating density
    i += 1
    # NP Increasing step
R = np.array(R)
M = np.array(M)
T = np.array(T)
rho = np.array(rho)
P = np.array(P)
g = np.array(g)
L = np.array(L)
# NP Converting lists to arrays at the end
print('Final radius: ' +format(R[len(R)-1] /R_o, '.2E') +' R_o')
print('Final mass: ' +format(M[len(M)-1] /M_o, '.2E') +' M_o')
print('Final luminosity: ' +format(L[len(L)-1] /L_o, '.2E') +' L_o')
```

```
Final radius: 5.14E-01 R_o
Final mass: 4.00E-01 M_o
Final luminosity: 5.94E-02 L_o
```

$$\boxed{R_* = 0.514 R_\odot}$$
$$\boxed{M_* = 0.400 M_\odot}$$
$$\boxed{L_* = 0.0594 L_\odot}$$

```
In [148]: T_c = 11e6
          # NP Central temperature in K
          rho_c = 25 *1000 *1.1
          # NP Central density in kg m^-3
          X = 0.74
          # NP Hydrogen fraction
          Y = 0.26
          # NP Helium fraction
          mu = 1 /(X +Y /4)
          # Equation 5.127
          # NP Printing result
          P_c = rho_c *k *T_c /(mu *m_p)
          dr = 10 **-5 *R_o
          # NP Radius step
          T = [T_c]
          # NP Initial temperature array
          M = [0]
          # NP Inital mass array
          rho = [rho_c]
          # NP Initial density array
          L = [0]
          # NP Initial luminosity array
          R = [0]
          # NP Inital radius array
          g = [0]
          # NP Inital surface gravity array
          P = [P_c]
          # NP Inital pressure array
          i = 0
          # NP Iterator value
          while(T[i] > 3000):
          # NP Iterating until temperature drops below 3000 K
              R.append(R[i]+dr)
              # NP updating radius
              M.append(M[i] +rho[i] *4 *np.pi *R[i+1]**2 *dr)
              # NP Updating mass
              T.append(T[i] -((2/5) *(rho[i] *G *M[i+1]\
                  *T[i] *dr)/(P[i] *R[i+1]**2)))
              # NP Updating temperature
              en = 2.4 *(rho[i] /1000) *X**2 /((T[i+1] /\
                  (10 **9)) **(2/3)) *np.exp(-3.38/\
                  (T[i+1]/(10**9)) **(1/3))
              # NP Calculating energy per unit mass for a shell
              L.append(L[i] +4*np.pi*R[i+1]**2*dr *rho[i] *en)
              # NP Updating Luminosity
              g.append(G *M[i+1]/(R[i+1]**2))
              # NP Updating surface gravity
              P.append(P[i] -rho[i] *g[i+1]*dr)
              # NP Updating pressure
              rho.append(mu *m_p *P[i+1] /(k *T[i+1]))
              # NP Updating density
              i += 1
              # NP Increasing step
          R = np.array(R)
          M = np.array(M)
          T = np.array(T)
          rho = np.array(rho)
          P = np.array(P)
          g = np.array(g)
          L = np.array(L)
          # NP Converting lists to arrays at the end
          print('Final radius: ' +format(R[len(R)-1] /R_o, '.2E') +' R_o')
          print('Final mass: ' +format(M[len(M)-1] /M_o, '.2E') +' M_o')
          print('Final luminosity: ' +format(L[len(L)-1] /L_o, '.2E') +' L_o')

          Final radius: 4.67E-01 R_o
          Final mass: 3.31E-01 M_o
          Final luminosity: 3.47E-02 L_o
```

$$\boxed{R_* = 0.467R_\odot}$$
$$\boxed{M_* = 0.331M_\odot}$$
$$\boxed{L_* = 0.0347L_\odot}$$