AST 231: Problem Set 6

Gil Garcia

April 17, 2020

Problem 1. Pre-main sequence stars in their T Tauri phase are believed to be fully convective and, therefore, they satisfy the equation of state appropriate to an adiabatic zone through their entire volume. Also, their central densities are not high enough to make degeneracy effects important, so we can assume that the ideal gas law applies. Finally, they are still chemically homogeneous, so we can assume that they have a Population I chemical composition (say, X = 0.73, Y = 0.25 and Z = 0.02) throughout. With these assumptions, create a stellar model for a pre-main sequence star with characteristics similar to a typical T Tauri star, namely mass = 0.5 solar masses, effective temperature = 4000 K and luminosity = 1 solar luminosity. Plot the density, pressure and temperature of this star as a function of distance from the center. [Note: While it is true that pressure depends on density to the 5/3 power in this star, just like in the low mass white dwarf, the equation of state is not identical to that case. This star is NOT supported by electron degeneracy and you cannot use the value of K given in Problem Set 6. This question is different, and we have not specified K. Instead we have specified the mass, luminosity and effective temperature that your model must match. It will be acceptable to have a model that comes close to the specified values of mass, temperature and luminosity even if it does not match them precisely.]

Answer 1. To create a stellar model of a PMS star that fits the parameters ($M = 0.5\ M_\odot$, $L = 1 L_\odot$, and T = 4000 K), we integrate the hydrostatic equilibrium and mass conservation equations (as we did in problem set # 5). In our integrator, we also add a temperature calculation using the ideal gas law:

$$T = \frac{P \mu m_H}{k \rho} \tag{1}$$

where P is pressure, ρ is density, m_H is the mass of hydrogen, $\mu = 0.606$ (calculated in problem set # 4 using the chemical composition specific above), and k is the Boltzmann constant. We can then express the pressure of a T Tauri star using the following equation:

$$P = K \rho^{\frac{5}{3}} \tag{2}$$

To compute the K constant, we have to guess the initial density and initial pressure such that it will produce our desired mass. We construct a semi-automated program that guesses the best fit value of initial density and initial pressure. This program and the integrator is implemented in Python (code attached at the end). Using the this, we arrive at the following values:

$\boxed{\text{initial density} = 421 \text{ kg m}^{-3}}$ and $\boxed{\text{initial pressure} = 9.9918 \times 10^{12} \text{ kg m}^{-1}\text{s}^{-2}}$

Using these initial conditions, we integrate the the equations, which results in the the final properties of a star:

$$\boxed{\text{final mass} = 9.89 \times 10^{29}\text{kg} = 0.4999 \text{ M}_\odot}$$

$$\boxed{\text{final radius} = 1.498 \times 10^{9}\text{m} = 2.15 \text{ R}_\odot}$$

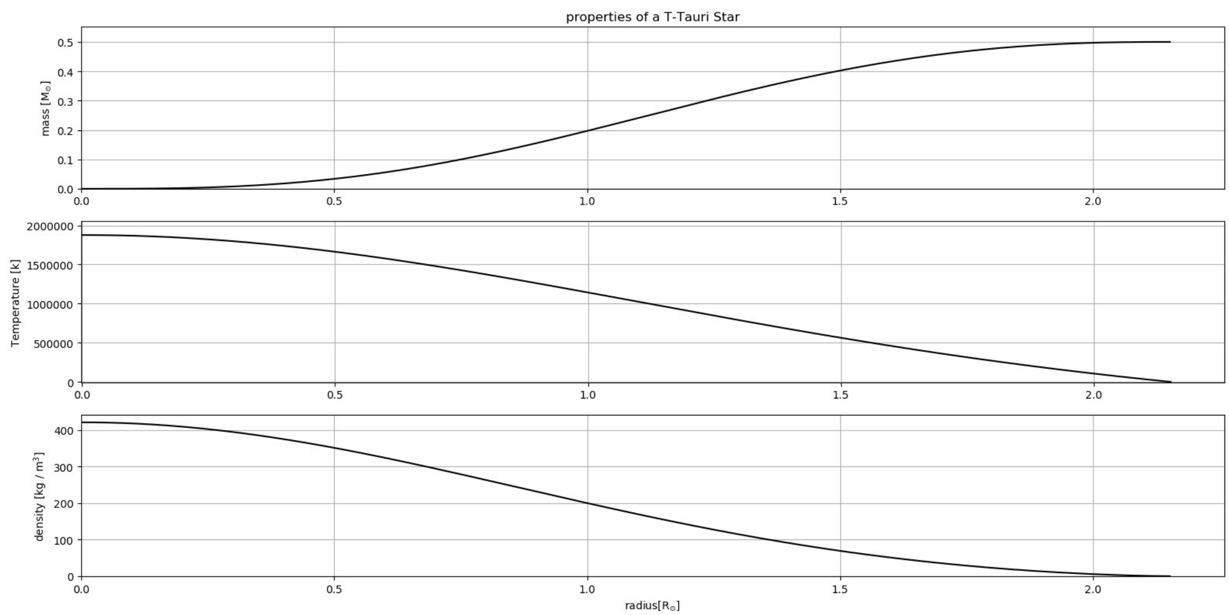
Using these properties, we can use the following equation to solve for luminosity:

$$\mathrm{L} = 4\pi \mathrm{R}^2 \sigma_{\mathrm{SB}} \mathrm{T}_{\mathrm{eff}}^4 \tag{3}$$

where σ_{SB} is the Steffan-Boltzmann constant.
With T= 4000 K and and R = 1.498×10^9 m, we get a luminosity of

$$\boxed{\mathrm{L} = 4.09 \times 10^{26} \text{ W} = 1.069 \text{ L}_\odot}$$

We can take a closer look at how the properties of the T Tauri star evolve as a function of radius by constructing the plot below:



We see that as a function of radius, the mass increases until 0.5 $\mathrm{M}_\odot$, temperature decreases to 0, and density decreases to 0 as well.

Code

```python
import numpy as np
import matplotlib.pyplot as plt


DR = 1e3 # our step size in the integrations


####################
####################          #1
####################


#some constants that will be needed
g = 6.67e-11 #m^3 kg^-1 s^-2
solar_mass = 1.98e30 #kg
solar_radius = 6.963e8 #meters
solar_lum = 3.828e26 #watts

h = 6.626e-34 #J s (planck constant)
m_e = 9.109e-31 #kg (mass of electron)
m_H = 1.67e-27 #kg (mass of hydrogen)
c = 2.99e8 # m/s (speed of light)
k_boltz = 1.28e-23 # m^2 kg s^-2 K^-1 (boltzmannn constant)
mu = 0.606 # (mean molecular weight for X=0.73, Y=0.25, Z=0.02)
sigma_sb = 5.67e-8 # W m^-2 K-4 (steffan boltzmann constant)
t_eff = 4000 #K (effective temp of T-tauri star)



#defining our routines

def temperature(pressure,density):
    return pressure *mu*m_H   /  ( k_boltz * density )

def density(pressure,k_const): #density for non-rel case
    return (pressure /k_const )**(3/5)

def mass_step(density,r,dr): #using mass conservation diff eq to calculate one step in mass
    return 4*np.pi*r**2*density*dr

def pressure_step(mass,radius,density,dr): #using hydrostatic diff eq to calculate one step in pressure
    return ((g*mass) / radius**2 ) * density * dr

```

```python

#we define a fxn that will be doing the intergrating
def newton_intergration(initial_density,initial_pressure):
    # now we initialize our variables and set them equal to the initial conditions
    k_const = initial_pressure / (initial_density**(5/3) )
    print('k_const:',format(k_const,'E'))
    #print(k_const)
    r = 0 #start at the core, radius is 0
    m = 0 #at the core, w a radius of 0, there is no mass enclosed
    d = initial_density #initial density in kg / m^3. this is the variable we will pass the newton_intergration fcn.
    p = k_const * (d)**(5/3) #initial pressure at the core
    t = temperature(p,d) #intial temperature

    #intializing lists to store all values at each intigration step

    r_lst = []
    m_lst =[]
    d_lst = []
    p_lst=[]
    t_lst = []

    #now we write the integrator using the fact that at the outer boundar, pressure will be 0:

    iter=0
    while p > 0:
        #keep track of num of steps we take
        iter+=1

        #updating our lsts at each step
        r_lst += [r]
        m_lst += [m]
        d_lst += [d]
        p_lst += [p]
        t_lst += [t]


        #now we update the values:
        r = r + DR #updating our radius value by our radius step size
        d = density(p,k_const) #calculating the new pressure value
        dm = mass_step(d,r,DR) #calculating the change in mass
        m = dm + m #updating our mass value
        dp = pressure_step(m,r,d,DR) #calculating the change in pressure
        p = p - dp #updating our pressure value. pressure decreases as we increase radius.
        t = temperature(p,d)
```

```
        #print(m,r,p,t)
    return r_lst,m_lst,d_lst,p_lst,t_lst




t_tauri_mass = 0.5 * solar_mass
t_tauri_temp = 4000
t_tauri_lum = 1 * solar_lum


mass,temp,lum = 0,0,0
threshold = 1e-6



'''
pressure_guess = 9.99e12
while abs( (mass/solar_mass) - 0.5) > threshold:
    r_lst,m_lst,d_lst,p_lst,t_lst = newton_intergration(421,pressure_guess)
    mass = m_lst[-1]
    t = t_lst[-1]
    print('mass in m_sun:\t',mass / solar_mass)
    print('temp',t)
    print('p_guess', format(pressure_guess,'E'))
    pressure_guess+=1e8
'''


r_lst,m_lst,d_lst,p_lst,t_lst = newton_intergration(421,9.9918e12)

print( 'final mass', m_lst[-1] )
print('final radius', format(r_lst[-1],'E') )

lum = 4 * np.pi * (r_lst[-1])**2 * sigma_sb * t_eff**4
print('lum',lum )

print()
print( 'final mass [m_sun]', m_lst[-1] /solar_mass )
print('final radius [r_sun]' ,r_lst[-1] /solar_radius )
print('lum [l_sun]',lum / solar_lum)



r_arr = np.array(r_lst)
m_arr = np.array(m_lst)
d_arr = np.array(d_lst)
```

```
p_arr = np.array(p_lst)
t_arr = np.array(t_lst)



plt.subplot(311)
plt.title(r'properties of a T-Tauri Star')
plt.plot( r_arr / solar_radius ,m_arr / solar_mass ,color='k',label=r'M$_r$')
plt.ylabel(r'mass [M$_{\odot}$]')
#plt.axhline(1.44,ls='--',color='k',label=r'Chandrasekhar Limit: 1.44 M$_{\odot}$')
#plt.ylim(-0.01,1.6)
left,right = plt.xlim()
plt.xlim(0,right)
plt.xlim(0,right)
bttm,top = plt.ylim()
plt.grid()
#plt.legend()


plt.subplot(312)
plt.plot(  r_arr /solar_radius , (t_arr) ,color='k',label='temperature')
plt.ylabel(r'Temperature [k]')
left,right = plt.xlim()
plt.xlim(0,right)
plt.xlim(0,right)
bttm,top = plt.ylim()
plt.grid()



plt.subplot(313)
plt.plot(  r_arr / solar_radius ,(d_arr),color='k',label='density')
plt.ylabel(r'density [kg / m$^{3}$]')
left,right = plt.xlim()
plt.xlim(0,right)
bttm,top = plt.ylim()
plt.ylim(0,top)


plt.grid()



plt.xlabel(r'radius[R$_{\odot}$]')
#plt.tight_layout()
plt.show()
```