# Assignement_1

March 22, 2023

The following modules define the RK4 process and solves the LEE

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
import sympy
import pdb

#constants
rsol = 6.957e10 #CGS
msol = 1.989e33 #CGS
lsol = 3.8e33   #cgs

G = 6.67e-8 #CGS
e_charge = 4.8e-10 #CGS electron charge
kb = 1.38e-16 #CGS boltzman constant
Na = 6.02e23 #advagadros number
amu = 1.66e-24 #g atomic mass unit
h = 6.63e-27  #planks constant cgs
me = 9.1e-28 #electron mass grams
c = 3e10 # speed of light CGS
a = 7.56e-15 #CGS radiation constant
R = 8.31e7 #gas constant CGS
```

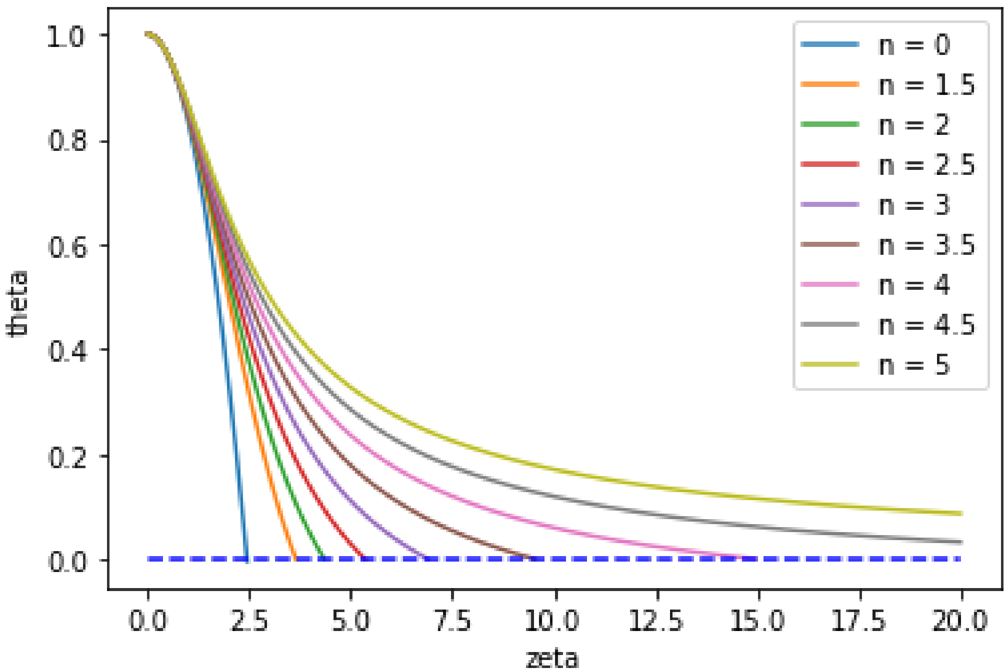
[2]:
```
# dU/dZeta function
def f_u(zeta, u, theta):
    if zeta == 0:
        return -1/3 #initial condition
    return -2*u/zeta - theta**n
# dTheta/dZeta == u
def f_theta(zeta, u, theta):
    return u
# simple test function
def y_func(x,y,th):
    return y
```

[3]:
```
def RK4(f, zeta0, u0, theta0, var):
    zeta = zeta0 + h
```

```python
    if var == 'u':
        k1 = f(zeta0, u0, theta0)
        k2 = f((zeta0+h/2), (u0+h*k1/2), theta0)
        k3 = f((zeta0+h/2), (u0+h*k2/2), theta0)
        k4 = f((zeta0+h), (u0+h*k3), theta0)
        k = (k1+2*k2+2*k3+k4)*h/6
        u = u0 + k
        return zeta, u
    elif var == 'theta':
        k1 = f(zeta0, u0, theta0)
        k2 = f((zeta0+h/2), (u0), theta0 +h*k1/2)
        k3 = f((zeta0+h/2), (u0), theta0 +h*k2/2)
        k4 = f((zeta0+h), (u0), theta0+h*k3)
        k = (k1+2*k2+2*k3+k4)*h/6
        theta = theta0 + k
        return zeta, theta
    else:
        print("WARNING")
```

[50]:
```python
#step size
h = 0.01
#polytrope index
n = 1.5
#number of steps
n_steps = 2000

#n_list = [0,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
#looping over all polytropes can make a list.


for n in [0,1.5,2,2.5,3,3.5,4,4.5,5]:
# initial conditions
    zeta = 0.0
    theta = 1.0
    u = 0.0
    theta_sol = []
    zeta_sol = []
    u_sol = []
    for i in range(n_steps):
        if theta < 0: #zeta_1 values
            break # we are only interesting in realistic solutions
        zeta_tmp, u = RK4(f_u,zeta, u, theta, 'u')
        zeta, theta = RK4(f_theta,zeta, u, theta, 'theta')
        zeta = zeta_tmp
        u_sol.append(u)
        theta_sol.append(theta)
        zeta_sol.append(zeta)
```

```
#     plt.plot(zeta_sol,u_sol, label = 'u')
    plt.plot(zeta_sol,theta_sol, label = 'n = '+ str(n))
# print('zeta1',zeta_sol[-1], 'u1', u_sol[-1])


plt.plot(zeta_sol, [0 for n in range(len(zeta_sol))], linestyle = '--', color =
 ↪'b')
plt.xlabel('zeta')
plt.ylabel('theta')
plt.legend()
plt.show()
```




The following code is for question 2a)

[ ]:
```
get the temp funciton
rho_c = sympy.Symbol('rho_c')#g/cm3
T = sympy.Symbol('T')
kb = sympy.Symbol('kb')
rho = sympy.Symbol('rho')
rad_constant = sympy.Symbol('rad_constant')
mu = sympy.Symbol('mu')
amu = sympy.Symbol('amu')
P = sympy.Symbol('P')
T_sol = sympy.solve(kb*rho*T/(mu*amu) + (a/3)*T**4 - P, T)
print(T_sol)
```

```
# it was the second peicewise
```

[5]:

```
def temperature(P, rho):
    return -np.sqrt(-2*P/(a*(9*R**2*rho**2/(16*a**2*mu**2) \
    + np.sqrt(P**3/a**3 + 81*R**4*rho**4/(256*a**4*mu**4)))**(1/3)) \
    + 2*(9*R**2*rho**2/(16*a**2*mu**2) + np.sqrt(P**3/a**3 + 81*R**4*rho**4/
 ↪(256*a**4*mu**4)))**(1/3))/2 \
    + np.sqrt(2*P/(a*(9*R**2*rho**2/(16*a**2*mu**2) + np.sqrt(P**3/a**3 +␣
 ↪81*R**4*rho**4/(256*a**4*mu**4)))**(1/3)) \
    + 6*R*rho/(a*mu*np.sqrt(-2*P/(a*(9*R**2*rho**2/(16*a**2*mu**2) + np.
 ↪sqrt(P**3/a**3 + 81*R**4*rho**4/(256*a**4*mu**4)))**(1/3)) \
    + 2*(9*R**2*rho**2/(16*a**2*mu**2) + np.sqrt(P**3/a**3 + 81*R**4*rho**4/
 ↪(256*a**4*mu**4)))**(1/3))) \
    - 2*(9*R**2*rho**2/(16*a**2*mu**2) + np.sqrt(P**3/a**3 + 81*R**4*rho**4/
 ↪(256*a**4*mu**4)))**(1/3))/2
```

[57]:

```
X1 = 0.715 #mass fraction of H
mu = 0.61 #mean molecular weight

#step size
h = 0.01
#polytrope index
n = 1.5
#number of steps
n_steps = 2000

zeta = 0.0
theta = 1.0
u = 0.0
theta_sol = []
zeta_sol = []
u_sol = []
for i in range(n_steps):
    if theta < 0: #zeta_1 values
        break # we are only interesting in realistic solutions
    zeta_tmp, u = RK4(f_u,zeta, u, theta, 'u')
    zeta, theta = RK4(f_theta,zeta, u, theta, 'theta')
    zeta = zeta_tmp
    u_sol.append(u)
    theta_sol.append(theta)
    zeta_sol.append(zeta)

#discard negative solution
print(theta_sol[-1])
zeta_sol = np.array(zeta_sol[:-1])
theta_sol = np.array(theta_sol[:-1])
u_sol = np.array(u_sol[:-1])
```

```
zeta1 = zeta_sol[-1]
u1 = u_sol[-1]
print(theta_sol[-1])
```

```
-0.00025690791719469586
0.0017802719863788162
```

[52]:
```
print('T = ',temperature(4.4e17,160)*10**-7, '10^7 K')
#check the funtion works
```

```
T =  2.016739694039279 10^7 K
```

[93]:
```
#loop over possible central densities to find reasonable answer
for rho_c in np.linspace(45,55,100): #reasonable guess
    print(rho_c)
    alpha = (msol/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
    K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
    rho = rho_c*theta_sol**n
    P = K*rho**(1+1/n)
    M = -4 * np.pi * alpha**3 * rho_c * np.power(zeta_sol,2) * u_sol
    luminosity = 0
    T_9 = temperature(P,rho)*10**(-9)
#     print('T_9_c =',T_9[0])
#     print('P =', P[0])
#     print('rho =', rho[0])
#     print('M =', M[-1])
#     print('M_tot = ', np.sum(M))
#     print('K =', K)
#     print('alpha = ', alpha)
    for i in range(len(rho)):
        if i ==0:
            dm = M[i]
        else:
            dm = M[i]-M[i-1]
        T_9 = temperature(P[i],rho[i])*10**(-9)
        g11 = 1 + 3.82*T_9 +1.51*T_9**2 + 0.144*T_9**3 -0.0144*T_9**4
        epp = 5.14e4 * g11 * rho[i] * X1**2 * T_9**(-2/3) *np.exp(-3.381/
 ↪(T_9**(1/3)))
        luminosity += epp*dm
    print('luminocity = ', luminosity)
    if (lsol-lsol/100 < luminosity < lsol + lsol/100):
        print('this value of rho_c gives is accurate within +/-1%', rho_c)
        print('luminosity = ', luminosity)

#values of rho_c ~ 49.5 gives the required luminosity
```

```
4.18e+33
45.0
```

```
luminocity =  3.010286885855511e+33
45.101010101010104
luminocity =  3.0271414341133815e+33
45.2020202020202
luminocity =  3.0440496304298757e+33
45.303030303030305
luminocity =  3.0610115103938167e+33
45.4040404040404
luminocity =  3.078027109491548e+33
45.505050505050505
luminocity =  3.0950964631074303e+33
45.60606060606061
luminocity =  3.112219606524262e+33
45.707070707070706
luminocity =  3.129396574923734e+33
45.80808080808081
luminocity =  3.146627403386861e+33
45.90909090909091
luminocity =  3.163912126894408e+33
46.01010101010101
luminocity =  3.181250780327312e+33
46.111111111111114
luminocity =  3.1986433984671265e+33
46.21212121212121
luminocity =  3.216090015996466e+33
46.313131313131315
luminocity =  3.233590667499366e+33
46.41414141414141
luminocity =  3.2511453874617816e+33
46.515151515151516
luminocity =  3.268754210271942e+33
46.61616161616162
luminocity =  3.286417170220797e+33
46.717171717171716
luminocity =  3.3041343015024295e+33
46.81818181818182
luminocity =  3.321905638214439e+33
46.91919191919192
luminocity =  3.339731214358384e+33
47.02020202020202
luminocity =  3.357611063840174e+33
47.121212121212125
luminocity =  3.3755452204704594e+33
47.22222222222222
luminocity =  3.3935337179650516e+33
47.323232323232325
luminocity =  3.4115765899453293e+33
47.42424242424242
```

```
luminocity =  3.4296738699385746e+33
47.525252525252526
luminocity =  3.447825591378456e+33
47.62626262626263
luminocity =  3.4660317876053436e+33
47.72727272727273
luminocity =  3.484292491866752e+33
47.82828282828283
luminocity =  3.50260773731767e+33
47.92929292929293
luminocity =  3.52097755702101e+33
48.03030303030303
luminocity =  3.539401983947929e+33
48.13131313131313
luminocity =  3.5578810509782665e+33
48.23232323232323
luminocity =  3.576414790900865e+33
48.333333333333336
luminocity =  3.595003236414001e+33
48.43434343434343
luminocity =  3.613646420125701e+33
48.535353535353536
luminocity =  3.632344374554147e+33
48.63636363636363
luminocity =  3.651097132128047e+33
48.73737373737374
luminocity =  3.6699047251870077e+33
48.83838383838384
luminocity =  3.6887671859818615e+33
48.93939393939394
luminocity =  3.7076845466750684e+33
49.04040404040404
luminocity =  3.7266568393410507e+33
49.14141414141414
luminocity =  3.7456840959665587e+33
49.24242424242424
luminocity =  3.7647663484510536e+33
this value of rho_c gives is accurate within +/-1% 49.24242424242424
luminosity =  3.7647663484510536e+33
49.343434343434346
luminocity =  3.7839036286070165e+33
this value of rho_c gives is accurate within +/-1% 49.343434343434346
luminosity =  3.7839036286070165e+33
49.44444444444444
luminocity =  3.8030959681603126e+33
this value of rho_c gives is accurate within +/-1% 49.44444444444444
luminosity =  3.8030959681603126e+33
49.54545454545455
```

```
luminocity =  3.8223433987505996e+33
this value of rho_c gives is accurate within +/-1% 49.54545454545455
luminosity =  3.8223433987505996e+33
49.64646464646464
luminocity =  3.8416459519315717e+33
49.74747474747475
luminocity =  3.861003659171372e+33
49.84848484848485
luminocity =  3.8804165518529716e+33
49.94949494949495
luminocity =  3.899884661274405e+33
50.05050505050505
luminocity =  3.9194080186492066e+33
50.15151515151515
luminocity =  3.938986655106719e+33
50.25252525252525
luminocity =  3.958620601692399e+33
50.35353535353536
luminocity =  3.978309889368188e+33
50.45454545454545
luminocity =  3.998054549012842e+33
50.55555555555556
luminocity =  4.0178546114222626e+33
50.656565656565654
luminocity =  4.0377101073097977e+33
50.75757575757576
luminocity =  4.05762106730662e+33
50.858585858585855
luminocity =  4.077587521961968e+33
50.95959595959596
luminocity =  4.0976095017435796e+33
51.06060606060606
luminocity =  4.117687037037947e+33
51.16161616161616
luminocity =  4.1378201581506043e+33
51.26262626262626
luminocity =  4.158008895306551e+33
51.36363636363636
luminocity =  4.178253278650464e+33
51.464646464646464
luminocity =  4.1985533382470727e+33
51.56565656565657
luminocity =  4.218909104081431e+33
51.666666666666664
luminocity =  4.2393206060592894e+33
51.76767676767677
luminocity =  4.2597878740073325e+33
51.868686868686865
```

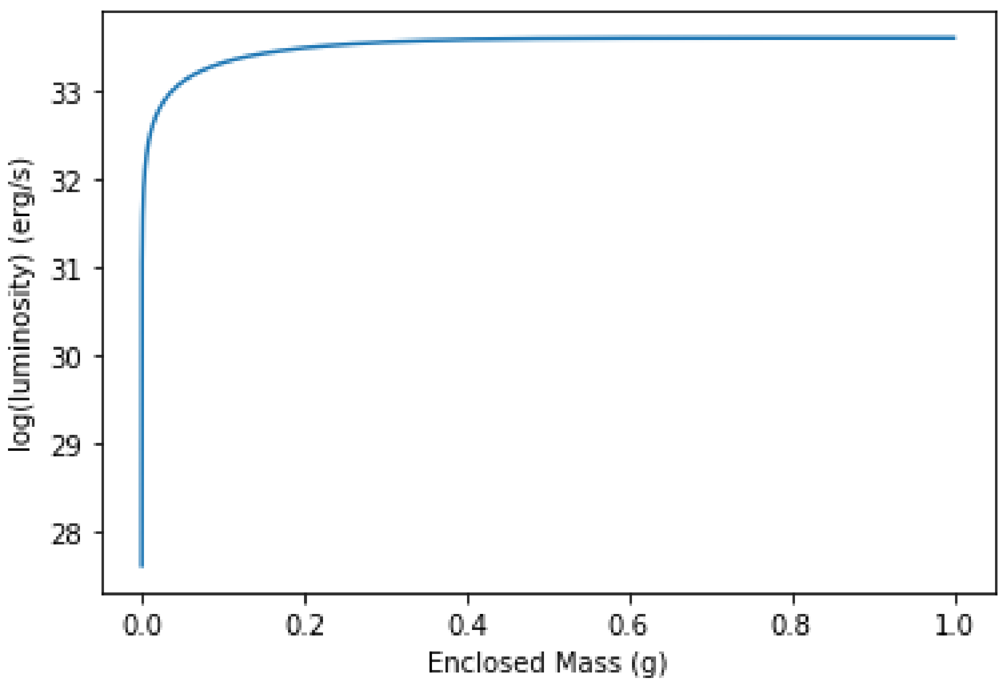
```
luminocity =  4.280310937673545e+33
51.96969696969697
luminocity =  4.300889826727485e+33
52.07070707070707
luminocity =  4.3215245707605785e+33
52.17171717171717
luminocity =  4.3422151992864646e+33
52.27272727272727
luminocity =  4.362961741741264e+33
52.37373737373737
luminocity =  4.3837642274838864e+33
52.474747474747474
luminocity =  4.404622685796344e+33
52.57575757575758
luminocity =  4.425537145884006e+33
52.676767676767675
luminocity =  4.44650763687595e+33
52.77777777777778
luminocity =  4.4675341878252053e+33
52.878787878787875
luminocity =  4.4886168277090735e+33
52.97979797979798
luminocity =  4.5097555854293675e+33
53.08080808080808
luminocity =  4.5309504898128276e+33
53.18181818181818
luminocity =  4.5522015696112495e+33
53.282828282828284
luminocity =  4.573508853501894e+33
53.38383838383838
luminocity =  4.594872370087653e+33
53.484848484848484
luminocity =  4.6162921478974654e+33
53.58585858585859
luminocity =  4.637768215386492e+33
53.686868686868685
luminocity =  4.659300600936436e+33
53.78787878787879
luminocity =  4.6808893328558536e+33
53.888888888888886
luminocity =  4.7025344393803433e+33
53.98989898989899
luminocity =  4.724235948672912e+33
54.09090909090909
luminocity =  4.745993888824175e+33
54.19191919191919
luminocity =  4.7678082878526885e+33
54.292929292929294
```

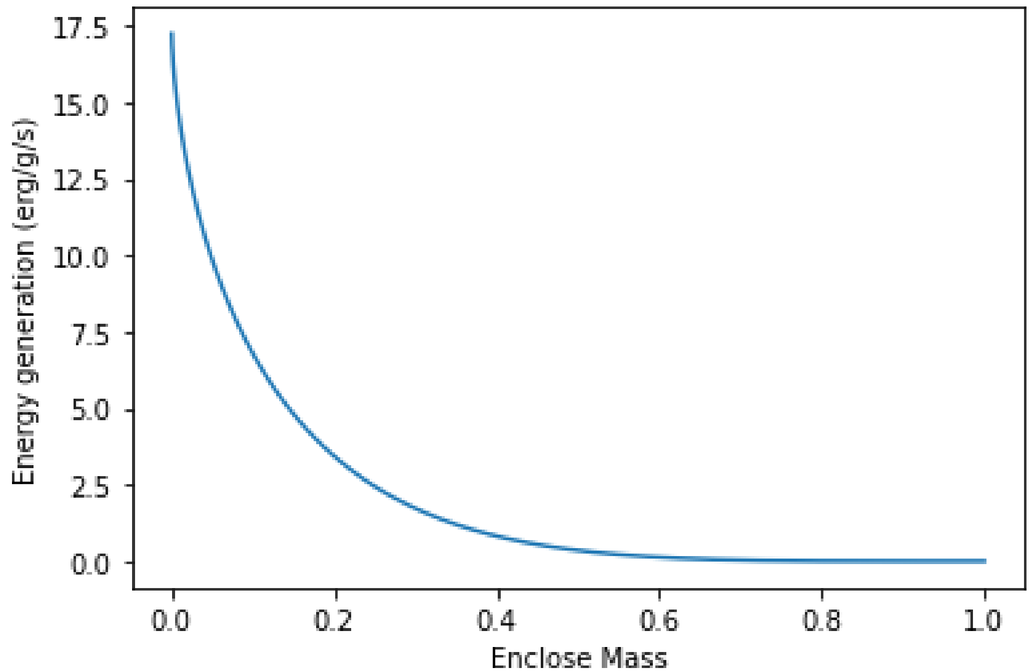
```
luminocity =  4.789679173705166e+33
54.39393939393939
luminocity =  4.8116065742568167e+33
54.494949494949495
luminocity =  4.833590517311544e+33
54.5959595959596
luminocity =  4.8556310306022054e+33
54.696969696969695
luminocity =  4.8777281417909863e+33
54.7979797979798
luminocity =  4.899881878469517e+33
54.898989898989896
luminocity =  4.9220922681592427e+33
55.0
luminocity =  4.9443593383116296e+33
```

[67]:

```
rho_c = 50 #that was determined really 49.5 but who is counting?

l_list = []
epp_list = []
mass_coor_list = []

alpha = (msol/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
rho = rho_c*theta_sol**n
P = K*rho**(1+1/n)
M = -4 * np.pi * alpha**3 * rho_c * np.power(zeta_sol,2) * u_sol
luminosity = 0
T_9 = temperature(P,rho)*10**(-9)
#     print('T_9_c =',T_9[0])
#     print('P =', P[0])
#     print('rho =', rho[0])
#     print('M =', M[-1])
#     print('M_tot = ', np.sum(M))
#     print('K =', K)
#     print('alpha = ', alpha)
for i in range(len(rho)):
    if i ==0:
        dm = M[i]
    else:
        dm = M[i]-M[i-1]
    T_9 = temperature(P[i],rho[i])*10**(-9)
    g11 = 1 + 3.82*T_9 +1.51*T_9**2 + 0.144*T_9**3 -0.0144*T_9**4
    epp = 5.14e4 * g11 * rho[i] * X1**2 * T_9**(-2/3) *np.exp(-3.381/(T_9**(1/
↪3)))
    luminosity += epp*dm
    epp_list.append(epp)
```

```
    l_list.append(luminosity)


epp_list = np.array(epp_list)
l_list = np.array(l_list)
plt.plot(M/msol,np.log10(l_list), label = 'luminosity')
plt.xlabel('Enclosed Mass (g)')
plt.ylabel('log(luminosity) (erg/s)')
plt.show()
plt.plot(M/msol, epp_list, label = 'energy generation')
plt.xlabel('Enclose Mass')
plt.ylabel('Energy generation (erg/g/s)')
plt.show()

print('Radius of this star is ', alpha*zeta1/rsol, 'rsol')
```






```
Radius of this star is  0.5507567731973254 rsol
```

Code below is for question 2b)

[38]:
```
#step size
h = 0.01
#polytrope index
n = 3.0
#number of steps
n_steps = 2000

#n_list = [0,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
#looping over all polytropes can make a list.


# for n in [0,1.5,2,2.5,3,3.5,4,4.5,5]:
#initial conditions
zeta = 0.0
theta = 1.0
u = 0.0
theta_sol = []
zeta_sol = []
u_sol = []
for i in range(n_steps):
    if theta < 0: #zeta_1 values
```

```
        break # we are only interesting in realistic solutions
    zeta_tmp, u = RK4(f_u,zeta, u, theta, 'u')
    zeta, theta = RK4(f_theta,zeta, u, theta, 'theta')
    zeta = zeta_tmp
    u_sol.append(u)
    theta_sol.append(theta)
    zeta_sol.append(zeta)

print(theta_sol[-1])
zeta_sol = np.array(zeta_sol[:-1],dtype = float)
theta_sol = np.array(theta_sol[:-1],dtype = float)
u_sol = np.array(u_sol[:-1],dtype = float)
zeta1 = zeta_sol[-1]
u1 = u_sol[-1]
mu = 0.61
print(theta_sol[-1])
print(zeta1)
print(u1)
```

```
-0.00034481156450529536
7.909340565897925e-05
6.8899999999998975
-0.04251363565024241
```

[39]:

```
## Solove for rho_c for a given T_c using the EOS
# T_c (has values to use)
rho_c = sympy.symbols('rho_c', real = True)
mu = 0.61
m_star = 10**5 * msol
alpha = (m_star/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
P_c = K*rho_c**(1+1/n)
# expr = kb*rho_c*T_c/(mu*amu) + (a/3)*T_c**4 - P_c
print(a)

T_c = [2e7,2.5e7,3e7,3.5e7]
rho_c_list = []
for T_c in T_c:
    expr = sympy.Eq(kb*rho_c*T_c/(mu*amu) + (a/3)*T_c**4 - P_c, 0)
    print('T_c=',T_c)
    print('expr=',expr)
    rho_c_sol = float(sympy.solve(expr, rho_c)[0])
    rho_c_list.append(rho_c_sol)
    print('rho_c =',rho_c_sol)
rho_c_list = np.array(rho_c_list)
```

```
7.56e-15
T_c= 20000000.0
```

```
expr= Eq(2.7256567252617e+15*rho_c -
8.2714413116921e+17*rho_c**2.0*(1/rho_c)**0.666666666666667 + 403200000000000.0,
0)
rho_c = 0.0033359404000243937
T_c= 25000000.0
expr= Eq(3.40707090657713e+15*rho_c -
8.2714413116921e+17*rho_c**2.0*(1/rho_c)**0.666666666666667 + 984375000000000.0,
0)
rho_c = 0.0065155085937976445
T_c= 30000000.0
expr= Eq(4.08848508789255e+15*rho_c -
8.2714413116921e+17*rho_c**2.0*(1/rho_c)**0.666666666666667 + 2.0412e+15, 0)
rho_c = 0.011258798850082333
T_c= 35000000.0
expr= Eq(4.76989926920798e+15*rho_c -
8.2714413116921e+17*rho_c**2.0*(1/rho_c)**0.666666666666667 + 3.781575e+15, 0)
rho_c = 0.017878555581380736
```

[32]:
```
X1 = 0.715
Xcno = 0.014
mu = 0.61
m_star = 10**5 * msol


radius = []
l_list = []
for rho_c in rho_c_list:
    enclosed_mass = []
    rho = rho_c * theta_sol**n
    alpha = (m_star/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
    K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
    P = K*rho**(1+1/n)
    M = -4*np.pi*alpha**3*rho_c*np.power(zeta_sol,2)*u_sol
    luminosity = 0
    radius.append(alpha*zeta1/rsol)
    for i in range(len(theta_sol)):
        if i ==0:
            dm = M[i]
            enclosed_mass.append(M[i])
        else:
            dm = M[i]-M[i-1]
            enclosed_mass.append(enclosed_mass[i-1]+dm)
        T_9 = temperature(P[i], rho[i])*10**(-9)
        g14 = 1 - 2.0*T_9 + 3.41*T_9**2 - 2.43*T_9**3
        ecno = 8.24e25 *g14 * Xcno*X1 * rho[i] * T_9**(-2/3) * np.exp(-15.
 ↪231*T_9**(-1/3)-(T_9/0.8)**2)
        luminosity += ecno*dm
```

```
    l_list.append(luminosity)
    print('luminosity = ', luminosity/lsol, 'Lsol')
    print('Radius = ', alpha*zeta1/rsol, 'Rsol')

l_list = np.array(l_list)
enclosed_mass = np.array(enclosed_mass)
```

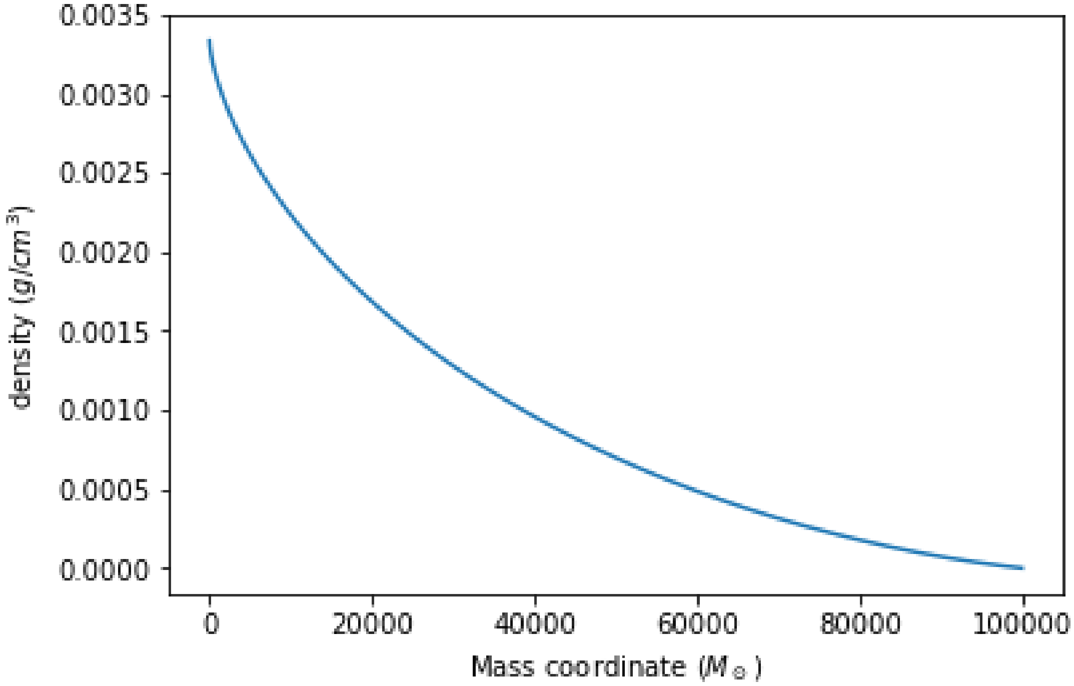
```
luminosity =  22.097494995921718 Lsol
radius =  1316.8729208343386 Rsol
luminosity =  2251.6097727531023 Lsol
radius =  1053.498336667471 Rsol
luminosity =  79202.3195090114 Lsol
radius =  877.9152805562258 Rsol
luminosity =  1391839.6328846605 Lsol
radius =  752.4988119053363 Rsol
```

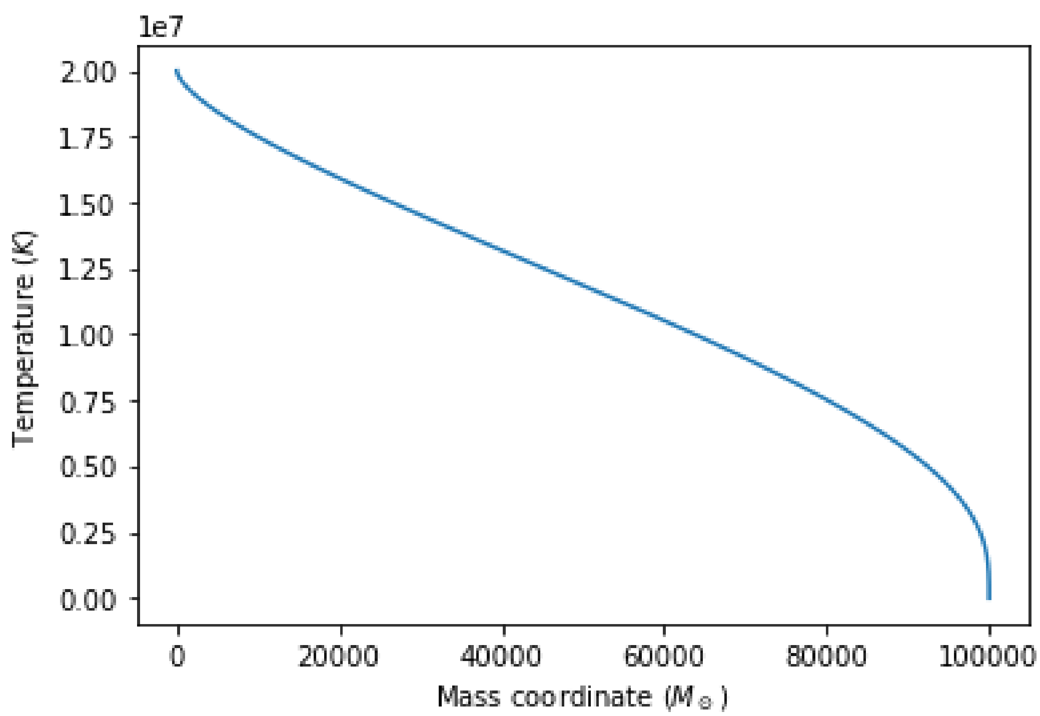
[27]:
```
print(enclosed_mass[-1])
# looks good
```

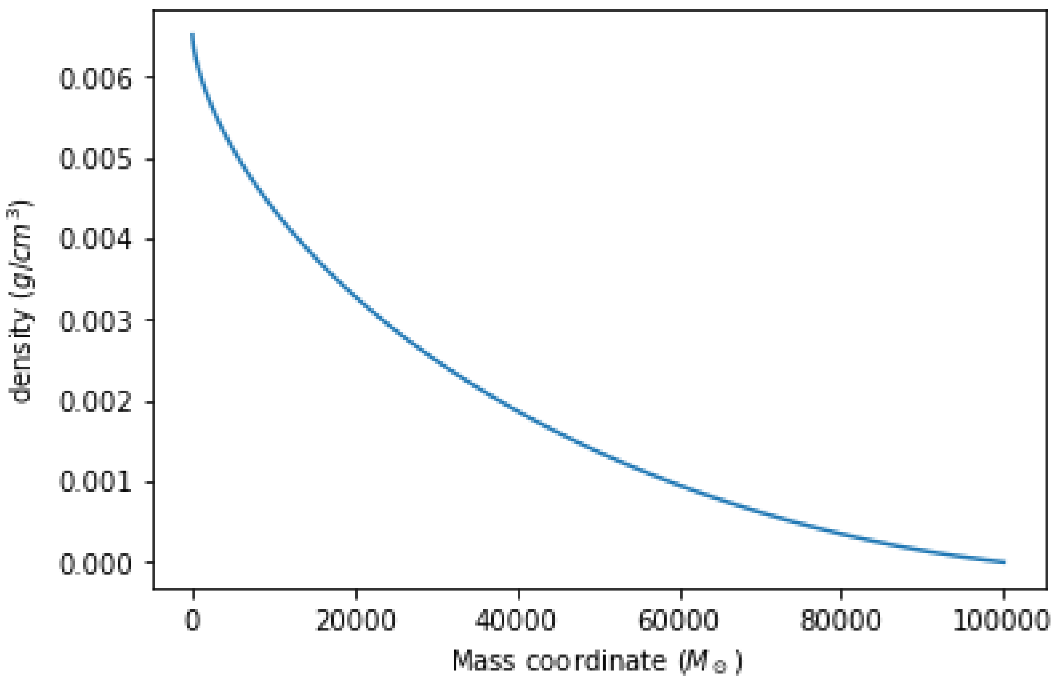
```
1.9889999999999907e+38
```

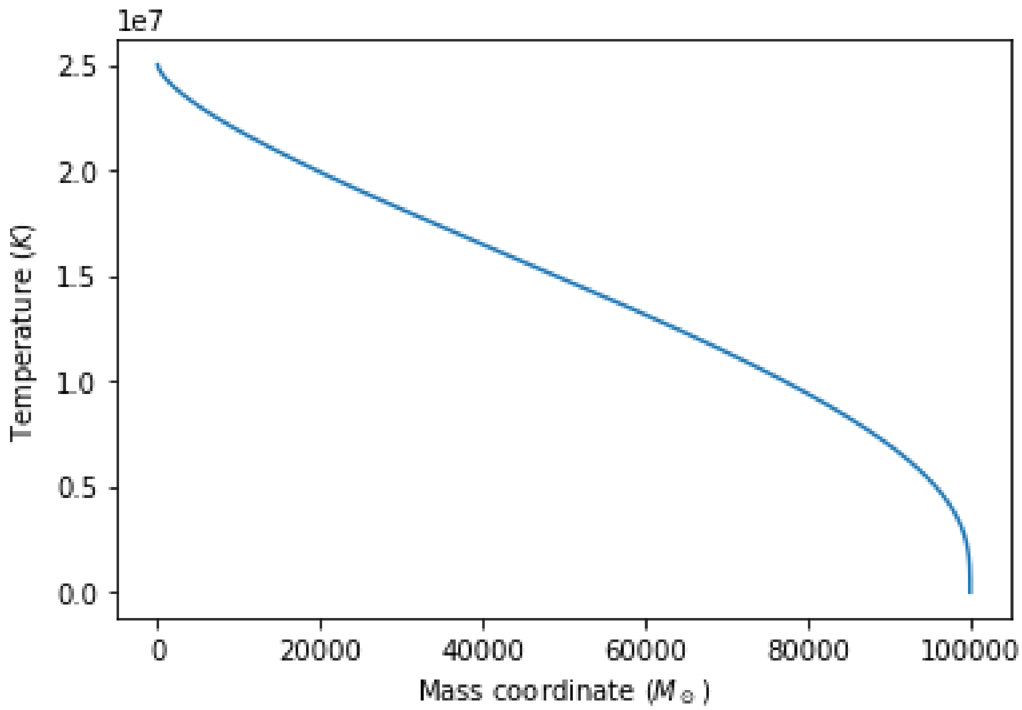
[33]:
```
print('The radius of the stellar models are:',radius, 'R_sol')
print('THe luminosity of the stellar models are:', l_list/lsol, 'L_sol')
print(rho_c_list)
for rho_c in rho_c_list:
    rho = rho_c * theta_sol**n
    alpha = (m_star/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
    K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
    P = K*rho**(1+1/n)
    T = temperature(P,rho)
    plt.plot(enclosed_mass/msol, rho)
    plt.ylabel('density ($g/cm^{3}$)')
    plt.xlabel('Mass coordinate ($M_\odot$)')
    plt.show()
    plt.plot(enclosed_mass/msol, T)
    plt.ylabel('Temperature ($K$)')
    plt.xlabel('Mass coordinate ($M_\odot$)')
    plt.show()
```

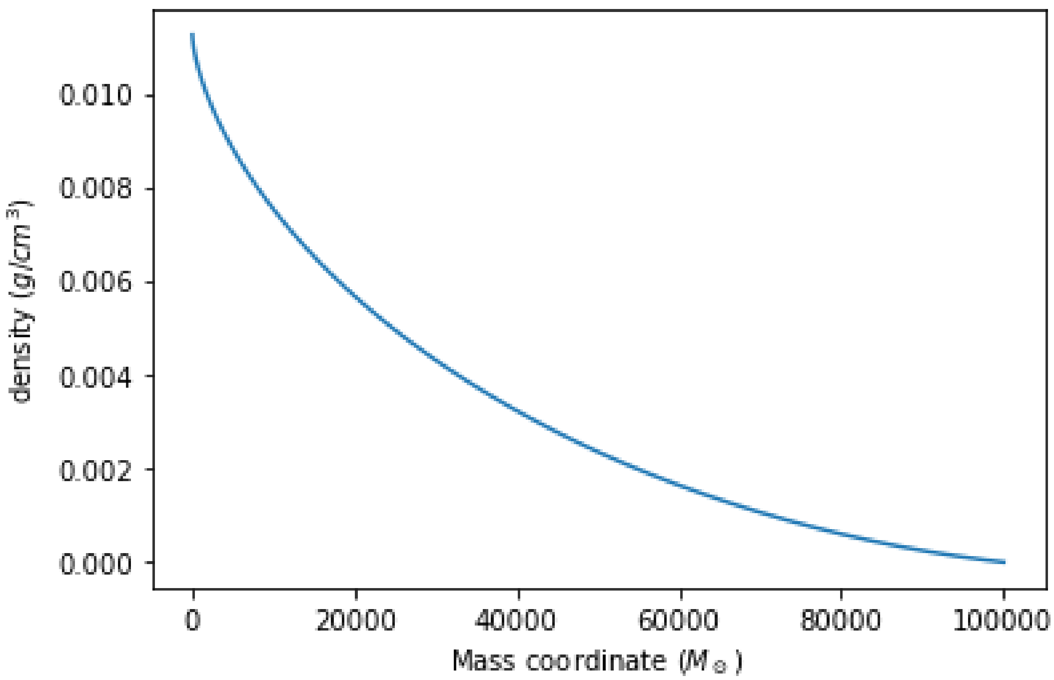
```
The radius of the stellar models are: [1316.8729208343386, 1053.498336667471,
877.9152805562258, 752.4988119053363] R_sol
THe luminosity of the stellar models are: [2.20974950e+01 2.25160977e+03
7.92023195e+04 1.39183963e+06] L_sol
[0.00333594 0.00651551 0.0112588  0.01787856]
```


0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
density ($g/cm^3$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)

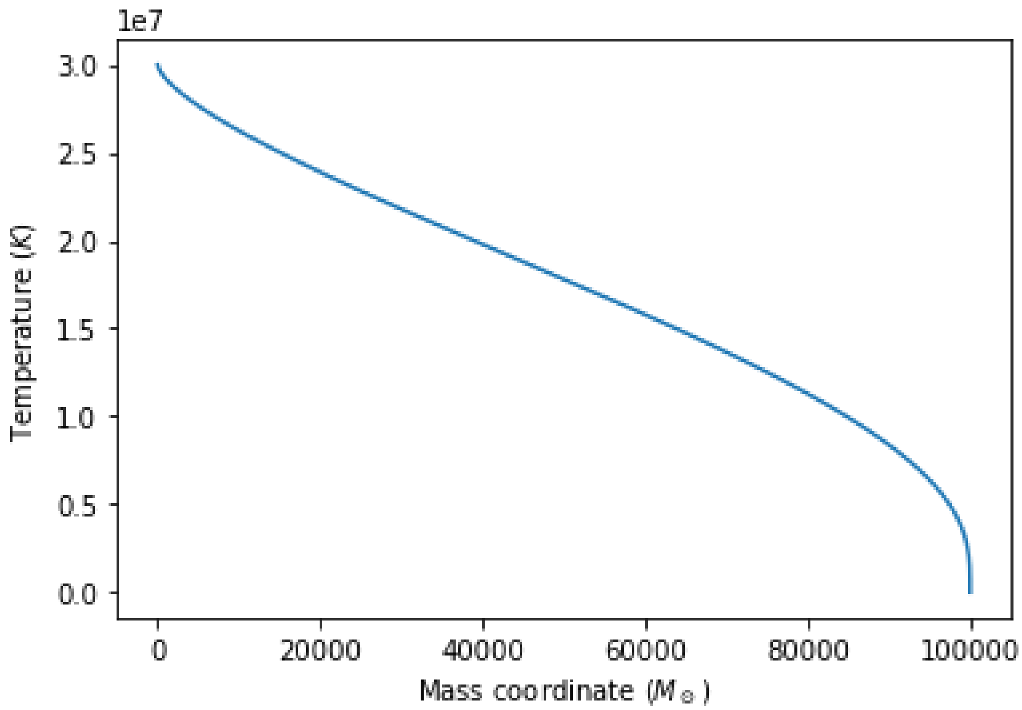

1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Temperature ($K$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)


density ($g/cm^3$)
Mass coordinate ($M_\odot$)
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
20000
40000
60000
80000
100000

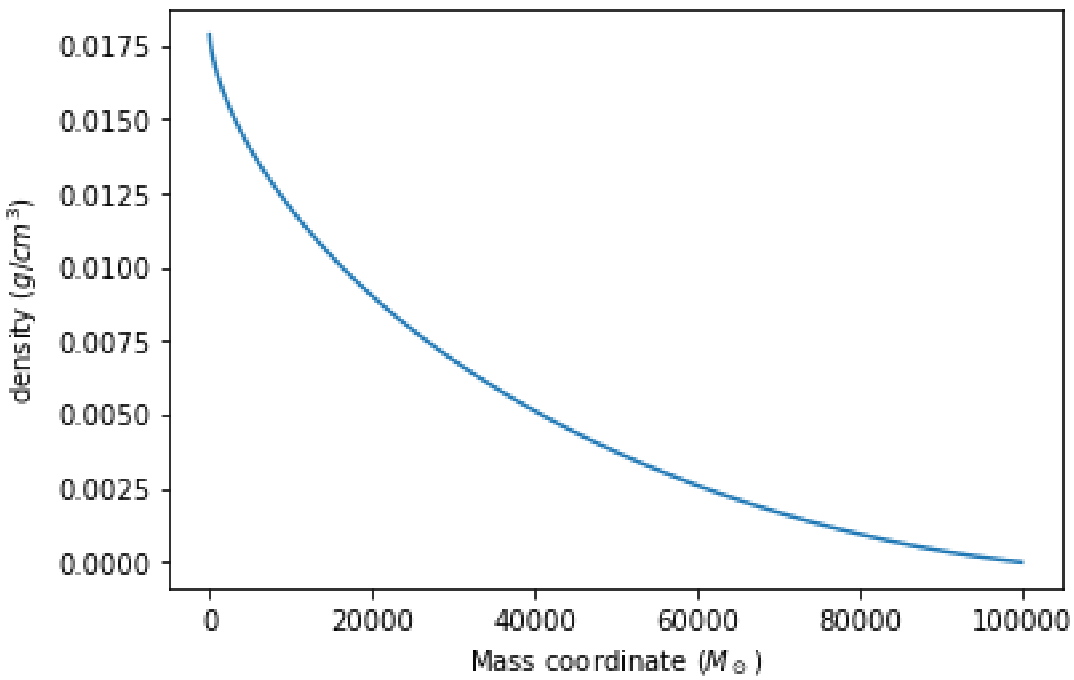

1e7
Temperature ($K$)
Mass coordinate ($M_\odot$)
2.5
2.0
1.5
1.0
0.5
0.0
0
20000
40000
60000
80000
100000

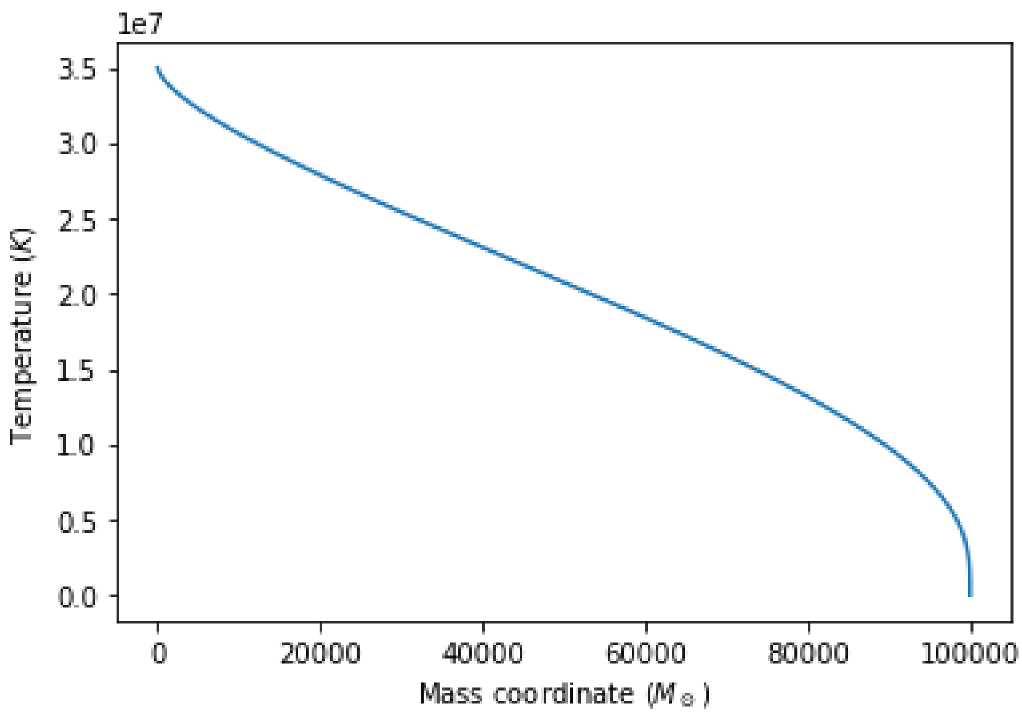


0.010
0.008
0.006
0.004
0.002
0.000
density ($g/cm^3$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)

1e7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Temperature ($K$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)


0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
density ($g/cm^3$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)


1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Temperature ($K$)
0
20000
40000
60000
80000
100000
Mass coordinate ($M_\odot$)

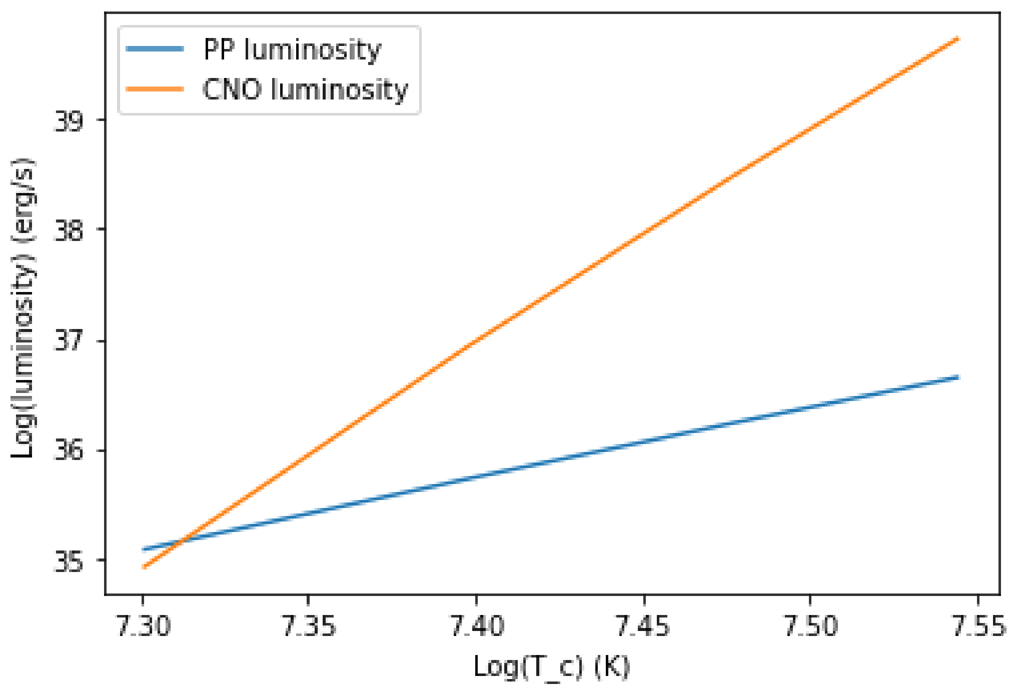
[37]:
```
P_g_center = R*rho[0]*T[0]/mu
central_P = R*rho[0]*T[0]/mu + a/3 * T[0]**4

beta = P_g_center/central_P

print(beta)

P_g = R*rho*T/mu
P = R*rho*T/mu + a/3 * T**4
beta_av = np.average(P_g/P)
print(beta_av)
# (its mostly supported by radiation pressure)
```

```
0.022045240962935267
0.022045240963006613
```

Solution for 2c)

[48]:
```
X1 = 0.715
Xcno = 0.014
mu = 0.61
m_star = 10**5 * msol
T_c = [2e7,2.5e7,3e7,3.5e7]

radius = []
lpp_list = []
lcno_list = []

for rho_c in rho_c_list:
    enclosed_mass = []
    rho = rho_c * theta_sol**n
    alpha = (m_star/(-4*np.pi*rho_c*u1*zeta1**2))**(1/3)
    K = ((alpha**2)*4*np.pi*G*rho_c**((n-1)/n))/(n+1)
    P = K*rho**(1+1/n)
    M = -4*np.pi*alpha**3*rho_c*np.power(zeta_sol,2)*u_sol
    luminosity_cno = 0
    luminosity_pp = 0
    radius.append(alpha*zeta1/rsol)
    for i in range(len(theta_sol)):
        if i ==0:
            dm = M[i]
            enclosed_mass.append(M[i])
        else:
            dm = M[i]-M[i-1]
            enclosed_mass.append(enclosed_mass[i-1]+dm)
        T_9 = temperature(P[i], rho[i])*10**(-9)
        g11 = 1 + 3.82*T_9 +1.51*T_9**2 + 0.144*T_9**3 -0.0144*T_9**4
```

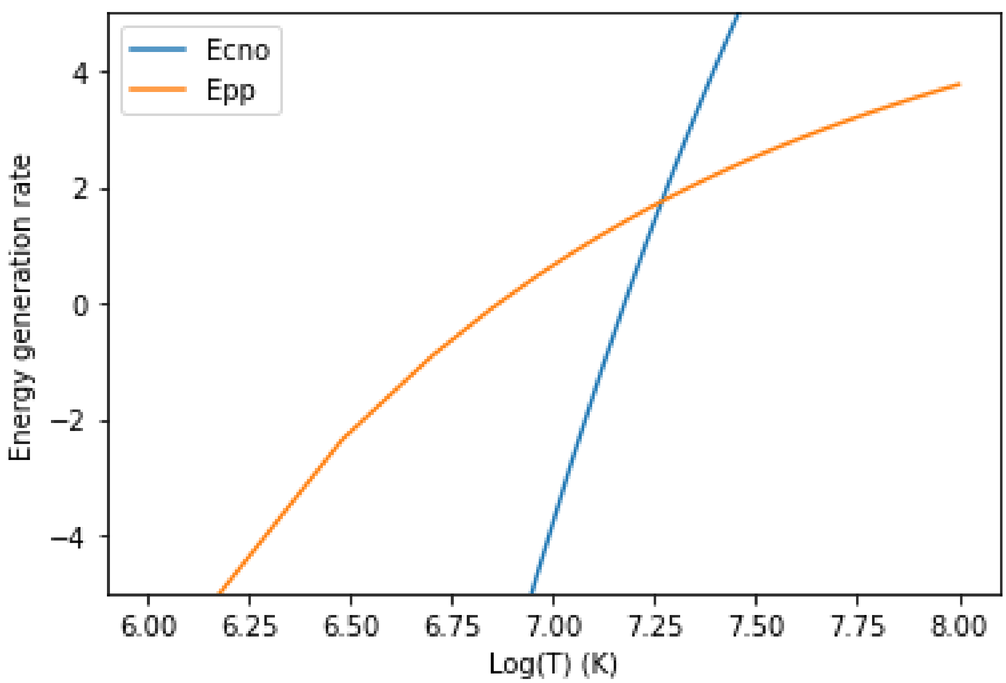
```
        epp = 5.14e4 * g11 * rho[i] * X1**2 * T_9**(-2/3) *np.exp(-3.381/
↪(T_9**(1/3)))
        g14 = 1 - 2.0*T_9 + 3.41*T_9**2 - 2.43*T_9**3
        ecno = 8.24e25 *g14 * Xcno*X1 * rho[i] * T_9**(-2/3) * np.exp(-15.
↪231*T_9**(-1/3)-(T_9/0.8)**2)
        luminosity_pp += epp*dm
        luminosity_cno += ecno*dm
    lpp_list.append(luminosity_pp)
    lcno_list.append(luminosity_cno)

lpp_list = np.array(lpp_list)
lcno_list = np.array(lcno_list)
enclosed_mass = np.array(enclosed_mass)

plt.plot(np.log10(T_c) , np.log10(lpp_list), label = 'PP luminosity')
plt.plot(np.log10(T_c) , np.log10(lcno_list), label = 'CNO luminosity')
plt.xlabel('Log(T_c) (K)')
plt.ylabel('Log(luminosity) (erg/s)')
plt.legend()
plt.show()
```

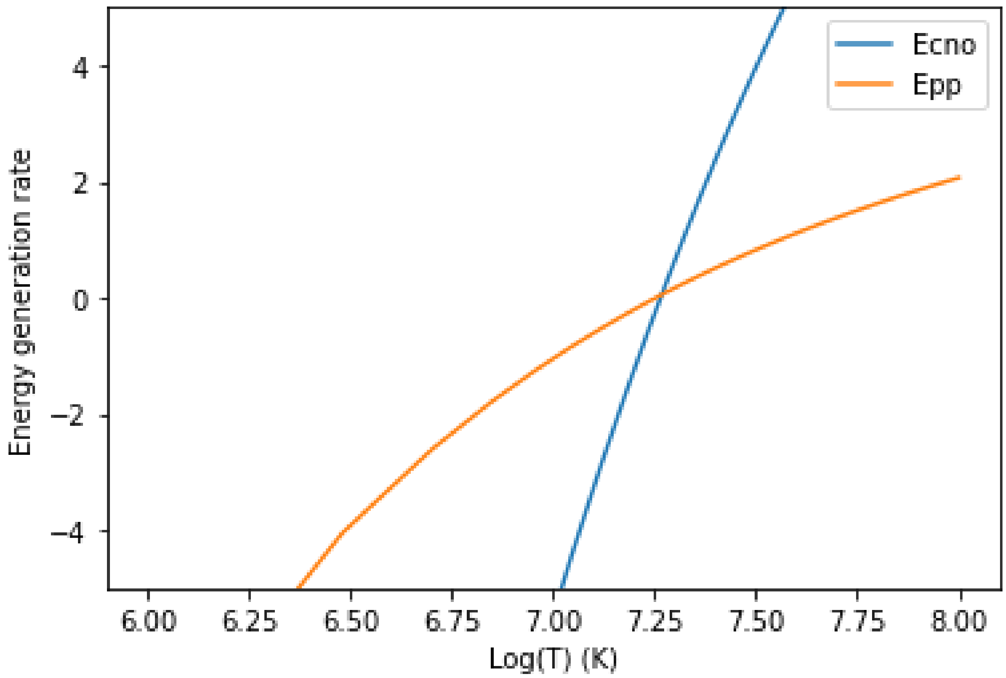



[20]:

```
#some left over code that I do not have the heart to delete

Xcno = 0.014
```

```
X1 = 0.715

rho_c = [50, 1]

T_array = np.linspace(1e6,1e8, 50) #reasonable range
for rho_c in rho_c:
    ecno_list = []
    epp_list = []
    for T in T_array:
        T_9 = T*10**-9
        g14 = 1 - 2.0*T_9 +3.41*T_9**2 - 2.43*T_9**3
        ecno = 8.24e25 *g14 * Xcno*X1 * rho_c * T_9**(-2/3) * np.exp(-15.
 ↪231*T_9**(-1/3)-(T_9/0.8)**2)
        ecno_list.append(ecno)
        g11 = 1 + 3.82*T_9 +1.51*T_9**2 + 0.144*T_9**3 -0.0144*T_9**4
        epp = 5.14e4 * g11 * rho_c * X1**2 * T_9**(-2/3) *np.exp(-3.381/
 ↪(T_9**(1/3)))
        epp_list.append(epp)
    ecno_list = np.array(ecno_list)
    epp_list = np.array(epp_list)
    plt.plot(np.log10(T_array),np.log10(ecno_list), label = 'Ecno')
    plt.plot(np.log10(T_array), np.log10(epp_list), label = 'Epp')
    plt.xlabel('Log(T) (K)')
    plt.ylabel('Energy generation rate')
    plt.legend()
    plt.ylim(-5,5)
    plt.show()
```




Ecno
Epp
Energy generation rate
Log(T) (K)
4
2
0
−2
−4
6.00
6.25
6.50
6.75
7.00
7.25
7.50
7.75
8.00

```
[ ]:
```