University of Oslo

AST3310

Project 1

Authors
Erik Levén

March 12, 2017

Contents

I Intruduction

When stars are observed from space there is very little information we can actually gather. Since most stars only take up one pixel in our telescope images we have very little to work with in terms of mass or size of the star. What we can obtain is the elemental compositions thanks to spectroscopy and the luminosity of the star. We will therefore make some assumptions (stated in the project) and chose some parameters ourselves to manage to produce our numerical model.

The purpose of this project is to successfully model the effect of nuclear reaction of a star. We will start from a chosen radius away from the core and through a set of differential equations work our way inwards towards the core of the star. In the end we should have produced a star which has both mass, luminosity and radius going towards zero as we approach the core.

The projects starts with finding the energy produced in the nuclear reaction and then continues to solve the differential equation governing the behaviour of the star.

II Calculating Power

The first step in modelling the effects of the nuclear reaction is to calculate the energy produced in each of the reactions. We have limited ourselves to reaction in the PP1 and PP2 chains and made the following assumptions to the problem:

- The mass fractions of each atomic species are independent of radius, in other words the distribution of different elements is constant throughout the sun.
- Metals contribute the "usual number" of electrons ($\frac{Z}{2}$) and all other elements are fully ionized.
- The two first reactions in the PP1 and PP2 chains (both reaction are common to both chains) occur simultaneously.
- Since where looking at a "snap shot" of a star the mass fractions does not change over time but are constant.
- There is no stored amount of elements available which means we can not use more elements in the different reactions than we are able to produce (not included Hydrogen).
- Due to the very long process of having a proton decaying to a neutron we can combine the first two steps of the PP1 and the PP2 chain.

II.1 Finding the reaction rates

The reaction rates are given by the equation

$$r_{ik} = \frac{n_i n_k}{\rho(1+\delta_{ik})}\lambda_{ik} \tag{1}$$

where n_i and n_k is the number densities of elements i and k present in the reaction. δik is the Kronecker delta function, ρ is the mass density and λik is the proportionality function which is a function of temperature.

The number densities can be calculated with the equation

$$n_i = \frac{\rho X_i}{A_i m_\mu} \tag{2}$$

where X_i and A_i are the mass fraction and number of nucleons of that element i respectively. When it comes to the number density of electrons it depends on to what degree the elements are ionized. If for example an element is ionized once then the electron contribution from this element is the same as the number density of that element. If the element is ionized twice then the electron contribution is two times the number density of that element.

The proportionality functions λ_i are defined in table 3.1 in *AST3310: Astrophysical plasma and stellar interiors*[1]. These functions are given in units $[cm^3 s^{-1}]$ and will have to be adjusted to SI units since these are the units I have chosen to work with throughout the programme. This is done by multiplying the functions with 10^6. In addition the electron capture for $^7_4$Be has a upper limit at temperatures below 10^6 K of $\lambda_{e7} \leq 1,57 * 10^{-7}/n_e NA$ which can be implemented by the following code snippet:

```
if (T < 1e6):
    if (NA*lambda_e7 > 1.57e-7/n_e):
        lambda_e7 = 1.57e-7/(n_e*NA)
```

Since all the rates have units $[kg^{-1}s^{-1}]$ the power pr. second (ϵ) can be calculated by

$$\epsilon = \sum r_{ik} Qik \tag{3}$$

where Q_{ik} is the energy output corresponding to reaction ik. The energy output was given in MeV and since the SI unit is joule this has to be converted. This can be done by multiplying the value in MeV with $1.602176621 * 10^{-13}$

The last thing to consider before calculating are the actual rates. Since we are not allowed to use more elements than created we need to set up some premises for which reaction rates decide the amount of energy produced in the different reactions. For example, if $r_{34} \geq r_{e7}$ then we can be sure that enough $^7_4$Be is produced for the r_{e7} reaction. However, if $r_{34} \leq r_{e7}$ there is not enough $^7_4$Be produced for r_{e7} to determine the energy produced in that reaction. Instead the energy is determined by the rate in the previous reaction. The following code snippet show an example on how this can be controlled in the last 2 steps in the PP2 chain:

```
def PP2_chain(r34, r17, re7, Q34, Q17, Qe7):
    """Last three reactions in the PP2 chain. The first 2 reaction are calculated
    outside this function. Takes arguments reaction rates [kg s^-1] for all three
    reaction and energy produced in units [MeV] for all three reactions."""
    return r34*Q34 + r17*Q17 + re7*Qe7

# Do we produce more Be7 than we consume
if (r34 > re7):
# Do we produce more Be7 than we consume
    if (re7 > r17):
        # Do we produce more Li7 than we consume
        E += self.PP2_chain(r34, r17, re7, Q34, Q17, Qe7)*MeVToJoule
    else:
        # Production of He4 is goverened by the rate of production for Li7
        E+= self.PP2_chain(r34, re7, re7, Q34, Q17, Qe7)*MeVToJoule
else:
    # Production of Li7 is goverend by the rate of production for Be7
    E+= self.PP2_chain(r34, r34, r34, Q34, Q17, Qe7)*MeVToJoule
```

As a "sanity check" we were provided for a given temperature and density with the energy pr. volume pr. second for the six reactions included in the PP1 and PP2 chain which we can compare to our own results. The "sanity checks" were:

- $r_{pp}(Q_{pp} + Q_{dp})\rho = 4.04 * 10^2 [Jm^{-3}s^{-1}]$
- $r_{33}Q_{33}\rho = 1.94 * 10^{-6} [Jm^{-3}s^{-1}]$
- $r_{34}Q_{34}\rho = 4.88 * 10^{-5} [Jm^{-3}s^{-1}]$
- $r_{e7}Q_{e7}\rho = 1.53 * 10^{-6} [Jm^{-3}s^{-1}]$
- $r_{17}Q_{17}\rho = 5.32 * 10^{-4} [Jm^{-3}s^{-1}]$

The actual results after running the code with the same temperature and density were:

- $r_{pp}(Q_{pp} + Q_{dp})\rho = 4.04 * 10^2 [Jm^{-3}s^{-1}$
- $r_{33}Q_{33}\rho = 1.94 * 10^{-6} [Jm^{-3}s^{-1}]$

- $r_{34}Q_{34}\rho = 4.88 * 10^{-5}[Jm^{-3}s^{-1}]$
- $r_{e7}Q_{e7}\rho = 1.53 * 10^{-6}[Jm^{-3}s^{-1}]$
- $r_{17}Q_{17}\rho = 5.32 * 10^{-4}[Jm^{-3}s^{-1}]$

In other words, so far the code looks like it is working as it should.

III Solving the differential equations

III.1 Finding the opacity

The temperature inside a star is amongst other variables proportional to the opacity (κ), the resistance to have energy transported throughout the gas. The opacity of a gas depends on the ionization degree and the atomic species present in the gas. We need this value to be able to continue with solving the differential equations. In our case the κ-value is not something we have to calculate. We were given a .txt document[2] with opacity values corresponding to different temperatures and densities. The provided values of κ were however not continuous, so one of our tasks was to perform a linear 2D interpolation to find the best κ-fit to our T and ρ values. I actually did this in three different ways before I decided on which technique served best. One of the reasons I started this project in C++ was because I wanted a fast programme. The assignment of finding the κ-values however slowed the programme down considerably. At first I added a python script [4] inside the C++ scrip which used several least squares methods to extract the four closest log10(R)- and log10(T)-values with their corresponding κ-values. I then created a NxN matrix in which I expanded the κ-values in (self made interpolation) and retrieved the closest values. This worked fine, and was fairly quick even though it took some time to actually code. I also tried compared it with the python function "interp2d" which took a considerable longer time (about 100% longer). This however became a big problem when performing is once for every iteration in the for-loop later on. After some discussion with a fellow AST3310 student we found out that if we code the whole programme in Python, we could create a Python-class for the whole project. The reason was we could now perform the "inter2d"-function once in the innit-function and use the results later on in the programme. In this way we only needed to perform the actual interpolation once, which considerably lowered the run-time of the programme. The following code snippet shows how we performed interpolation:

```
def __init__(self, L_0, M_0, R_0, rho_0, T_0, X, Y, Y3, Y4, Z, Z7Li, Z7Be):
    """ Performing the linear 2D interpolation o6f the opacity values. These will be
    used later on but performing the interpolation in the init function allows us to
    just call the ready function later on. We therefore only have to interpolate once
    which greatly improves the time performance of the programme"""

    data = np.genfromtxt("location")                          # Collecting data in one array
    logT = data[1:,0]                                         # logT-values
    logR = data[0,1:]                                         # logR-values
    n_T = len(logT)                                           # Number of logT-values
    n_R = len(logR)                                           # Number of logR-values
    self.f = inter.interp2d(logR, logT, data[1:, 1:], bounds_error = False)# interpolate

def opacity(self, T, rho):
    """ Function which takes emperature and density in SI-units and returns the
    opacity value corresponding to these values from the interpolation-function
    in the innit-function"""

    rho = rho/1000.0                               # Converting the density to cgs
    R = np.log10(rho/((T/1e6)**3))                 # As defined by Appendix D
    T = np.log10(T)

    return float(10**(self.f(R, T))/10.0)         # Returning the opacity in SI-units
```

This function "self.f" can now be used later on in the programme without having to establish it more then once. The extrapolation of values in this case is done automatically inside the inter2d-function. This is not a problem for the accuracy since I in my code do not come outside this range.

As with the energy values we were given a "sanity check" for the κ-values. The values provided were The values achieved by the interpolation function were which as we can see match relatively well

Table 1: Table of "sanity checks" for the κ-values

log_{10}T	log_{10}R (cgs)	$log_{10}\kappa$ (cgs)	κ (SI)
3.750	-6.00	-1.546	$2.84 * 10^{-3}$
3.755	-5.95	-1.508	$3.10 * 10^{-3}$
3.755	-5.80	-1.572	$2.68 * 10^{-3}$
3.755	-5.70	-1.610	$2.46 * 10^{-3}$
3.755	-5.55	-1.674	$2.12 * 10^{-3}$
3.770	-5.95	-1.328	$4.70 * 10^{-3}$
3.780	-5.95	-1.204	$6.25 * 10^{-3}$
3.795	-5.95	-1.024	$9.45 * 10^{-3}$
3.770	-5.80	-1.392	$4.05 * 10^{-3}$
3.775	-5.75	-1.354	$4.43 * 10^{-3}$
3.780	-5.70	-1.306	$4.94 * 10^{-3}$
3.795	-5.55	-1.162	$6.89 * 10^{-3}$
3.800	-5.50	-1.114	$7.69 * 10^{-3}$

with the "sanity check" values in table 1. The reasons for the differences could be many and since we don't know which simplification were or were not made to achieve these "sanity checks" it would seem almost impossible to get the exact same values. The number of valid digits in constants (maybe not in these examples but surely later on) would also effect the results. In addition, it seems unlikely that there's a linear relation between each κ-value, and even though we assumed this it's not necessary that the "sanity check" assumed the same. All in all I feel comfortable that these values are correct enough to continue with the rest of the project. One just has to bare this in mind when analyzing results in the end.

Table 2: Table of achieved κ-values

log_{10}T	log_{10}R (cgs)	$log_{10}\kappa$ (cgs)	κ (SI)
3.750	-6.00	-1.546	$2.84 * 10^{-3}$
3.755	-5.95	-1.507	$3.11 * 10^{-3}$
3.755	-5.80	-1.570	$2.69 * 10^{-3}$
3.755	-5.70	-1.612	$2.44 * 10^{-3}$
3.755	-5.55	-1.674	$2.12 * 10^{-3}$
3.770	-5.95	-1.326	$4.72 * 10^{-3}$
3.780	-5.95	-1.206	$6.23 * 10^{-3}$
3.795	-5.95	-1.025	$9.44 * 10^{-3}$
3.770	-5.80	-1.385	$4.12 * 10^{-3}$
3.775	-5.75	-1.342	$4.55 * 10^{-3}$
3.780	-5.70	-1.298	$5.03 * 10^{-3}$
3.795	-5.55	-1.162	$6.89 * 10^{-3}$
3.800	-5.50	-1.114	$7.69 * 10^{-3}$

III.2 The equations

In the assignment we were given five equation of which four were differential equations to solve.

$$\begin{aligned}
\frac{\partial r}{\partial m} &= \frac{1}{4\pi r^2 \rho} \\
\frac{\partial P}{\partial m} &= -\frac{Gm}{4\pi r^4} \\
\frac{\partial L}{\partial m} &= \epsilon \\
P &= P_G + P_{rad} \\
\frac{\partial T}{\partial m} &= -\frac{3\kappa L}{256\pi^2 \sigma r^4 T^3}
\end{aligned}$$

The last equation, $\frac{\partial T}{\partial m}$, is based on the premise of radiation being the only energy transport possibility in the star. If we would have taken into account convection or conduction in the energy transportation this equation would not longer be valid.

As we can see the differential equation are all differentiated with respect to m instead of r. The reason for this is a more stable solution when performed numerically with respect to m than with respect to r. Since the density becomes larger as we approach the core of the star the the mass within a radius R decreases less and less. R on the other hand goes rather quickly towards the core, hence the more stable solution when differentiating with respect to m.

These equation contains seven unknowns, $(r, \rho, P, L, P_G, P_{rad}, T)$ so the next assignment was to find two more equations which would let us solve them. When assuming an ideal gas inside the star we can use the equation of state for an ideal gas to give us an equation connecting ρ, P and T. This equation states

$$PV = Nk_BT \tag{4}$$

where $N = \frac{m}{\mu m_\mu}$. μ is the average atomic weight and m_μ is the atomic mass unit. To find the average atomic weight we must first find the total number of particles in the star. The number densities of each element was given by equation 2 so the mean molecular weight must therefore be

$$\mu_{unit} = \rho \sum_i n_i$$

This number has the unit mass, but since we want it as a dimensionless variable we divide it with the atomic mass unit m_μ ending up with the following expression for the average atomic weight.

$$\begin{aligned}\mu_0 &= \sum_i \frac{\frac{\rho}{n_i}}{m_\mu} \\ &= \sum_i \frac{\rho m_\mu}{\rho(\frac{X_i}{A_i})m\mu}\end{aligned}$$

which in our case with fully ionized Helium and Hydrogen and the "usual" contribution of electrons from metals sums up to

$$\mu_0 = \frac{1}{2X + \frac{3}{4}Y + \frac{Z}{2}} \tag{5}$$

The second equation we need is related to the radiative pressure P_{rad}. This pressure is simply given by

$$P_G = \frac{4\sigma}{3c}T^4 \tag{6}$$

where σ is the Stefan-Boltzmann constant and c is the speed of light.

We now have seven equation and seven unknowns and can begin to solve them. For this project I chose a simple Euler method to performing the differential calculations. It could be argued that a Runge-Kutta method could be more precise, but considering the relative error margin already mentioned in section III.1 I felt comfortable with choosing a possible less accurate method for the advantage of running-time of the programme. In addition, the Euler method works relatively fine as long as long as we do not have oscillating or circular motion in our variables. The numeric differential equation now take the form

```
M[i+1] = M[i] + dm                                                    # mass
r[i+1] = r[i] + dm*1.0/(4*np.pi*r[i]**2*rho[i])                       # distance
P[i+1] = P[i] + dm*((-self.G*M[i])/(4*np.pi*r[i]**4))                 # pressure
L[i+1] = L[i] + dm*E[i]                                               # luminosity
T[i+1] = T[i] + dm*((-3*K*L[i])/(256*np.pi**2*self.sigma*r[i]**4*T[i]**3))# temperature

rho[i+1] = self.find_rho(P[i+1], T[i+1])                              # density
E[i+1] = self.energy_produced(T[i+1], rho[i+1])/rho[i+1]              # energy
```

III.3 Variable step length

The last thing do to before running the code and see if the differential equations work as they should is to implement variable step length[3]. The Euler solver states that $V[i+1] = V[i] + dV$, but how can we be sure dV is not too big. A big dV could give us imprecise or even unphysical results. This is where variable step length come into the picture. When implemented it should make sure that the dV values are not bigger than a preset value which makes sure we maintain the wanted level of precision.

A differential equation in a general form could look like

$$\frac{dV}{dm} = f$$

We normally require that

$$\frac{|dV|}{V} < p$$

where p is normally a number less than 1. If however this is not true then we need to lower the value of dV. Since we know we can express $dV = dm * f$ we can by lowering the value of dm lower dV as well such that is takes on a more acceptable value. We can calculate this new value of dm as

$$dm = \frac{dV}{f} ? \frac{pV}{d} \tag{7}$$

If we have many different values of f corresponding to several differential equations we simply chose the dm which gives us the smallest absolute value. An example on how this could be done is shown in the code below:

```
def DSS(self, dm, dm1, r, rho, M, E, L, T):
    """ Method to calculate the variable step length. Takes the old and
     the initial dm (dm and dm1) as input together with the variable values
    and returns the new dm """

    # Variable step length constant
    p = 1e-4;

    # Establishing variables to perform the variable step length
    dm_test = np.zeros(4)
    dV = np.zeros(4)
    V_test = np.zeros(4)

    # Setting values for dV
    dV[0] = dm*1.0/(4*np.pi*r**2*rho)
    dV[1] = dm*(-self.G*M/(4*np.pi*r**4))
    dV[2] = dm*E
    dV[3] = dm*(-(3*K*L)/(256*np.pi**2*self.sigma*r**4*T**3))

    #setting value for V
    V_test[0] = r[i]
    V_test[1] = P[i]
    V_test[2] = L[i]
    V_test[3] = T[i]

    # setting values for the dm-tests
    dm_test[0] = abs(p*r/(1.0/(4*np.pi*r**2*rho)))
    dm_test[1] = abs(p*P/((-self.G*M)/(4*np.pi*r**4)))
    dm_test[2] = abs(p*L/E)
    dm_test[3] = abs(p*T/(-(3*K*L)/(256*np.pi**2*self.sigma*r**4*T**3)))

    # checking if we need to change the step length
    for j in range(len(dV)):
        if (abs(dV[j])/V_test[j] > p):
            dm = -min(dm_test)
            break
        else:
            # reseting to initial dm
            dm = dm1

# Beginning of the for-loop in which we integrate
for i in range(N-1):
    if dynamic_step_size == True:
        dm = self.DSS(dm, dm1, r[i], rho[i], M[i], E[i], L[i], T[i])
```

IV Checking initial parameters

As a starting point we chose to initial parameters to be the actual values at the bottom of the solar convection points (except P). The values chosen were:

$$L_0 = 1.0L_\odot$$
$$R_0 = 0.72R_\odot$$
$$M_0 = 0.8M_\odot$$
$$\rho_0 = 5.1\bar{\rho}_\odot$$

When running the programme the following plots of the results was achieved: First of all we can establish that the code seems to be working as it should. When comparing plot 2 to the "first sanity check" plot shown in the project text[1] pg. 106 (non updated version). The values differ a little bit to the "sanity check", but as stated when analyzing table 2 minor deviation are to be expected.

As we can see, the results in figure 2 does not seem to be very physical results. We would expect

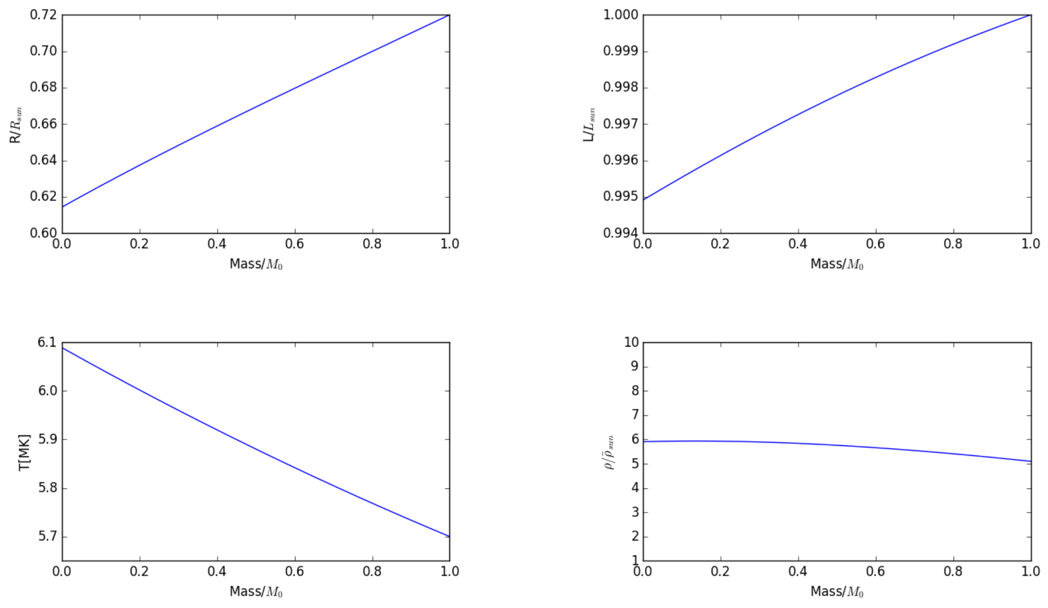

Figure 1: Plot of the final results as achieved by the initial conditions as stated above

that then the mass goes towards zero so would the luminosity and the radius. The next part of this project will be to establish an algorithm that can initial parameters that results in physical results.

V Analyzing step length

How big or small the step length dm is supposed to be can often be hard to know before you start. This section will analyze the behaviour of the $\frac{\partial r}{\partial m}$ plot for different values of dm. Since the initial parameters from the bottom of the sun's convection zone result in a more or less linear $\frac{\partial r}{\partial m}$, different sizes in dm would not effect the precision at any noticeable degree. I therefore chose new initial parameters which resulted in a more curved $\frac{\partial r}{\partial m}$. The values chosen were in fact a result of a least square method described later in this project were the luminosity, mass and radius all went within 5% of the initial values. These parameters did not however result in a very physical solution, which I will describe later. For now though, they serve the purpose. The initial values chosen were:

- $L_0 = L_\odot$
- $M_0 = 0.8_\odot$
- $T_0 = 1.9494 * 10^7$
- $R_0 = 0.54R_\odot$
- $\rho_0 = 3.672\bar{\rho}_\odot$

The dm-values started at -10^{25} to -10^{29} with a factor of $\frac{1}{10}$ between each value. The following plot shows the results: As seen above it seems like $dm = -10^{29}$ gives a significant error in the calculations. The different graphs seems to converge towards the values from $dm = -10^{25}$.

To check these results we activate the variable step length and run the programme again: As seen in figure 3 the variable step length graph follows the $dm = -10^{25}$ closely(almost indistinguishable). We can therefor establish that if the variable step length is not activated, $dm = -10^{25}$ would give accurate results.

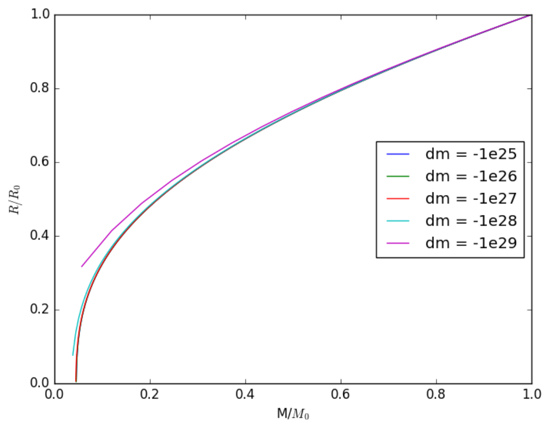

Figure 2: Plot of the final results for different values of dm

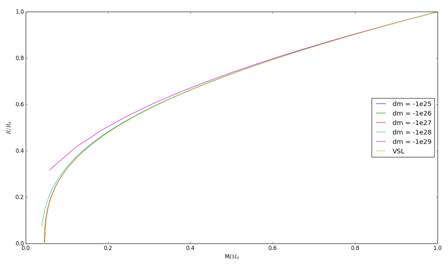

Figure 3: M as a function of R for different dm values including the variable step length results.

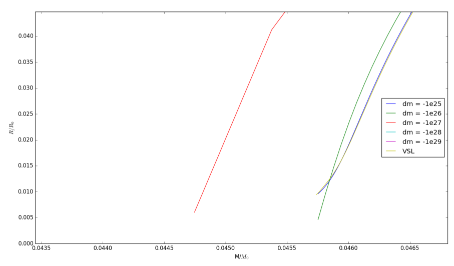

Figure 4: Same as figure 2 but zoomed in on variable step length graph (VSL) at $R/R_0 \to 0$.

VI Analyzing the parameters

I this section I have chosen the same initial parameters as described in section **??** with a step length of $dm = -1e26$. This choice of parameters might not yield as physical solution (see conclusion and discussion for explanation) as can be seen in the different plots of for example temperature and density. They are relatively constant in the beginning and rapidly increasing close to the core. I am aware of this problem and will describe and handle it later in section VII. For now, these parameters are just example parameters to show the behaviour and connection between the different parameters chosen.

The The reason for this relatively high step length (as described in section IV, this step length might yield some inaccuracies) is because the code runs 10 times faster with it. Since this section is not interested as much in precision but behaviour of the graphs which should be the same, I deemed this an acceptable sacrifice. I just have to bare in mind that the results from this section might not be as accurate as needed when trying to find the final initial parameters.

Since what we have basically done is to solve an equation set with seven unknowns and seven equations we would expect that the change in the value of one parameter would effect the results of the other parameters. This section therefore take on the task of analyzing how the different parameters effect each other. What we would expect are parameters where the four plots visualized in figure 2 and figure 17 goes towards physical values. In other words:

- The radius should go towards zero when the mass goes towards zero
- The luminosity should go towards zero when the mass goes towards zero
- The temperature should increase when the mass goes towards zero
- The density should increase when the mass goes towards zero

I would like to add that finding the relation was a bit hard without going into the equations governing them. When doing this however it became hard keep track on what depended on what. For that reason I have focused my analysis on a physical interpretation.

VI.1 Changing R_0

The following four plot are the results of changing the initial R_0 parameter from $0.3 * R_\odot$ to $1.5 * R_\odot$ with a step size of 0.3. The remaining initial parameters are

- $L_0 = L_\odot$
- $M_0 = 0.8_\odot$
- $\rho_0 = 3.672\bar{\rho}_\odot$
- $T_0 = 1.9494 * 10^7$

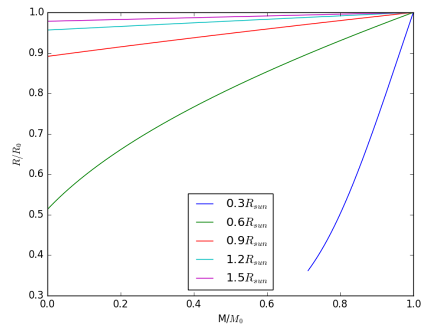

Figure 5: M as a function of R for the chosen R_0-values.

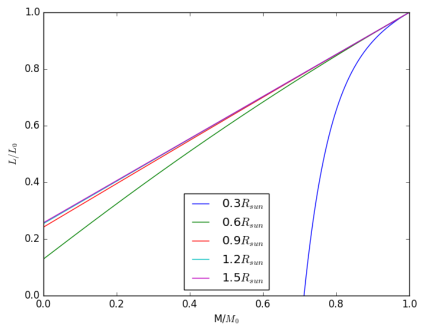

Figure 6: M as a function of L for the chosen R_0-values.

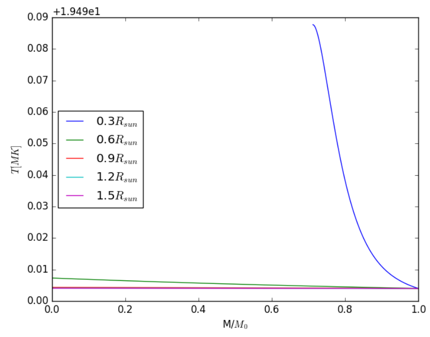

Figure 7: M as a function of T for the chosen R_0-values.

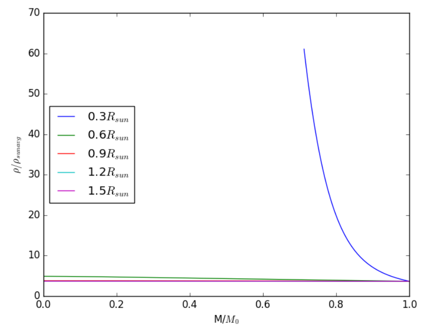

Figure 8: M as a function of ρ for the chosen R_0-values.

When comparing these plots to the expectations listed in section VI we can conclude that when the initial radius gets close to or higher than $R_\odot$ the values of the temperature, luminosity and density does not take physical values. This can be explained by the mass of the star being constant. Since the total mass in this case is constant, a large star would have to have a small increase in density (or even constant) to make up for the total mass. However, we expect our star to have a larger density as we get closer to the core. Stars with smaller radius (not too small though) would therefore fit better with our model. Higher density at the core means higher temperature at the core, which we also would expect in our star. You could also say that a big star would take away relatively big "chunks" of the mass for each step inwards. If the density does not match the radius we would run out of mass to fast. An other way of explaining it could be to consider the luminosity plot. As R_0 grows larger the code "expects" a large star. If however T_0 and ρ_0 are relatively small compared to R_0 then the energy production, ϵ, is lower than expected. Since $\frac{\partial L}{\partial m} = \epsilon$ this means that the luminosity does not change as much as expected, which we can see in figure 6. Had the luminosity changed more for each time step it could have reached zero for big values of R_0.

VI.2 Changing T_0

The following four plot are the results of changing the initial T_0 parameter from $0.2 * T_{innit}$ to $0.36 * T_{innit}$ with a step size of 0.04 where $T_{innit} = 5.7 * 10^6 [K]$. The remaining initial parameters are:

- $L_0 = L_\odot$
- $M_0 = 0.8_\odot$
- $\rho_0 = 3.672\bar{\rho}_\odot$
- $R_0 = 0.54R_\odot$

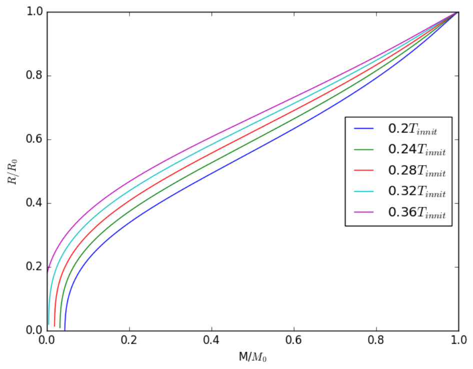

Figure 9: M as a function of R for the chosen T_0-values.

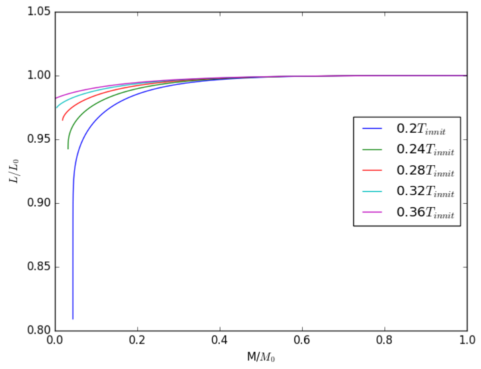

Figure 10: M as a function of L for the chosen T_0-values.

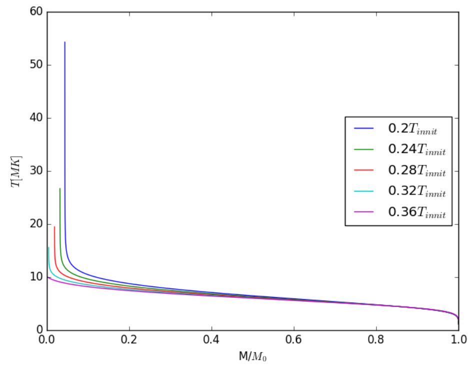

Figure 11: M as a function of T for the chosen T_0-values.

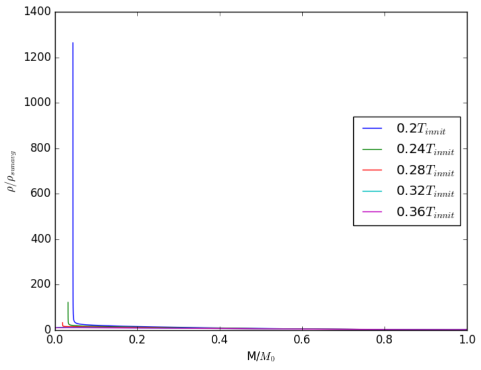

Figure 12: M as a function of ρ for the chosen T_0-values.

As seen in the plots all chosen values of T_0 seems to be in the area of a physical value for the other chosen parameters. What we can conclude is that values much lower or much higher than the chosen ones will not lead to physical solutions. For example, if $T_0 = 10T_{innit}$ we would from the graph expect the mass to go towards zero faster than the radius. A hot star is in general a big star. The reactions inside the star should reflect the reactions of a big star. If R_0 is not big enough R will never reach $R = 0$. We would therefore, following the argumentation for the R_0-values, expect that warm stars in this model would not reach zero radius before the mass reaches zero unless R_0 is big enough. The same but reversed argument can be done for the case where T_0 is much smaller than the chosen values.

VI.3 Changing ρ_0

The following four plot are the results of changing the initial ρ_0 parameter from $0.2 * \rho_{innit}$ to $0.36 * \rho_{innit}$ with a step size of 0.04 where $\rho_{innit} = 5.1\bar{\rho}_{\odot}$. The remaining initial parameters are:

- $L_0 = L_{\odot}$
- $M_0 = 0.8_{\odot}$
- $T_0 = 1.9494 * 10^7$
- $R_0 = 0.54R_{\odot}$

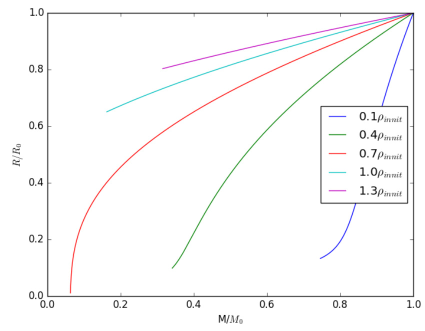

Figure 13: M as a function of R for the chosen ρ_0-values.

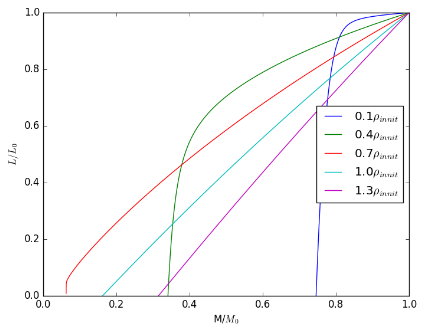

Figure 14: M as a function of L for the chosen ρ_0-values.

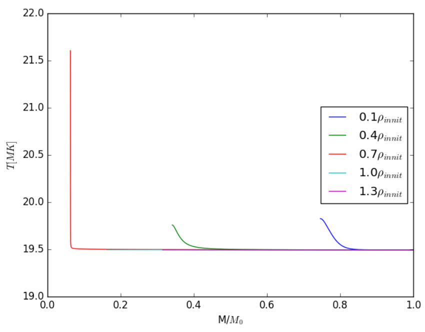

Figure 15: M as a function of T for the chosen ρ_0-values.

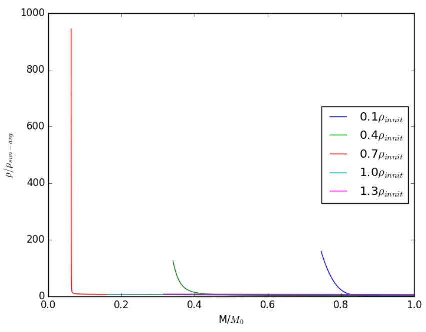

Figure 16: M as a function of ρ for the chosen ρ_0-values.

As seen from the plots, $\rho_0 = 0.7\rho_{innit}$ seems to be the best fit from these values. These results can be explained with the same argumentation as above. A large initial density should correspond to a large star. If the star is large then we would again expect the mass to go towards zero faster than the radius, as seen in the plot.

VI.4 Changing P_0

I have chosen not to add any plots for different P_0 because the value of P_0 is set by ρ_0 and T_0.

$$P = \frac{\rho k_B T}{\mu m_\mu} + \frac{4\sigma T^4}{3c}$$

Changing P_0 would therefore be the same as changing the other two (or any combination of the two) and thus lead to similar results. If we wanted to lower the pressure for example we could instead lower ρ or T. If we had chosen to change both the pressure, density and temperature at the same time the equation of state would not be valid and I assume we would get some pretty funky results.

What we can state though is when finding appropriate initial values for our star, we do not have to have the initial value of P_0 in mind. We simply have to choose between different R_0, ρ_0 and T_0 values.

VII Finding physical parameters

One of our tasks was to find initial parameters that produced physical results. The physical result we are looking for is a star that has mass, density and radius all going to zero as we approach the core of the star. Our definition of "close" will in this project be constrained to luminosity, mass and radius all ending up as 5% of the initial parameters of the star. In other words:

- $\frac{L}{L_0} < 5\%$
- $\frac{M}{M_0} < 5\%$
- $\frac{R}{R_0} < 5\%$

To achieve this we were able to change the initial parameters T_0, R_0, ρ_0 and P_0. To find initial parameter that fulfilled these condition I used a modified least square method. I iterated over the T_0, R_0 and ρ_0 values since P is a function of ρ and T as described in section VI.4. The following code shows how I set up the logarithm:

```
def least_square(no_values, R_0_min, R_0_max, rho_0_min, rho_0_max, T_0_min, T_0_max,
N, dm):
    """This function performs a least square method. It takes the arguments for
    the min and max values of R_0, rho_0 and T_0 where no_values is the number of
    points in each list. It also takes the arguments number of integration points
    N and step length dm. The function then prints out the sum of the percentages
    (last_value/first_value) as they get closer and closer towards zero together
    with fhe factor in front"""

    R_0_list = np.linspace(R_0_min, R_0_max, no_values)      # list of R_0-values
    rho_0_list = np.linspace(rho_0_min, rho_0_max, no_values)# list of rho_0-values
    T_0_list = np.linspace(T_0_min, T_0_max, no_values)      # list of T_0-values

    # Variables to use in the least square method
    R_0_save = 0
    rho_0_save = 0
    T_0_save = 0
    least = 1e6

    for i in range(len(R_0_list)):
        R_01 = R_0_list[i]
        for j in range(len(rho_0_list)):
            rho_01 = rho_0_list[j]
            for k in range(len(T_0_list)):
                T_01 = T_0_list[k]
                # Integrating through the class
                A = project1(L_0, M_0, R_01, rho_01, T_01, X, Y, Y3, Y4, Z, Z7Li, Z7Be)
                r, M, L, rho, P, T, E = A.integrate(dm, N, dynamic_step_size = True)
                L_lim = L[-1]/L[0]
                M_lim = M[-1]/M[0]
                R_lim = r[-1]/r[0]
                if (L_lim + M_lim + R_lim < least):
                    # Checking if the new value is a better fit than the last
                    R_0_save = R_01
                    rho_0_save = rho_01
                    T_0_save = T_01
                    least = L_lim + M_lim + R_lim
                    print "Least seperate values = ", L_lim, M_lim, R_lim
                    print "Least total value = ", least
                    print "Parameters used = ", R_0_save, rho_0_save, T_0_save
                    print " "
```

The range of the parameters used in the least square method was determined from the results in section VI. However, I chose a rather long range for the intervals since it is hard to predict exactly how the variables change depending on the others. The best-fit value I achieved was:

- $T_0 = 4.96584 * 10^6$ [K]
- $R_0 = 0.4055 R_\odot$ [m]
- $\rho_0 = 2.0196 \bar{\rho}_\odot$ [kg m^{-3}]

These values would match those of a star smaller than our sun. With variable step length activated with $p = 10^{-4}$ for 100.000 steps the final results were:

- $\frac{L}{L_0} = 0.001$
- $\frac{M}{M_0} = 0.025$
- $\frac{R}{R_0} = 0.009$

As we can see these results are all well within the 5% constraint. These parameters also passes the test of the core ($L/L_0 < 0.995$) stretching out to 10% of R_0. When R = 10% of R_0 L/L_0 is approximately 0.27. The parameters as also within the $\frac{1}{5} \rightarrow 5$ times the initial given parameters

The plot for these values corresponding to plot 2 took the form of figure 17.

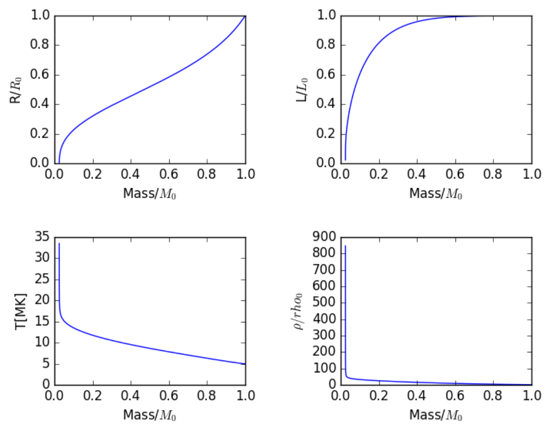

Figure 17: Plot of the final results as achieved by the initial parameters acquired by the least square method. We can clearly see both the luminosity and radius go towards zero when the mass goes towards zero

There are of course many different parameter variations that passes the 5%-limit test, the one chosen in this project was just the first combination that was found.

When comparing analyzing the values together as a function of radius figures 18 and 19 were produced.

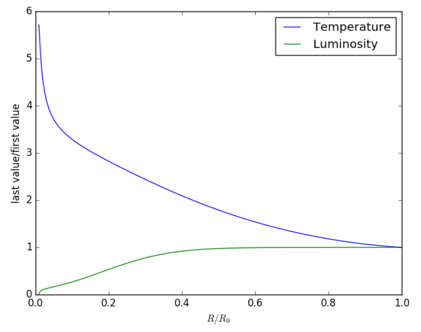

Figure 18: Temperature and luminosity and mass as a function of radius for the final initial parameters. As we can see, the luminosity and mass goes towards zero with the radius and the temperature the temperature increases as it we get closer to the core.

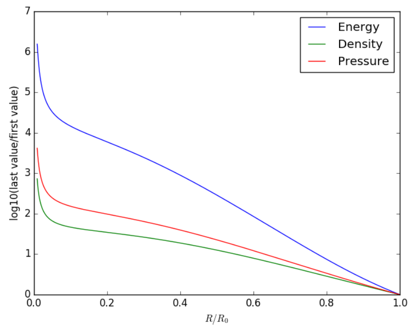

Figure 19: Energy produced, density and pressure as a function of radius for the final initial parameters. We can see that all three parameters behaves in the same way but the energy production seems to increase faster than the other two.

VIII Discussion and conclusion

It would seem as though the model works as it should. The model managed to reproduce all three "sanity checks" (energy production, kappa-values and initial plots) to an accepted level of accuracy. In addition the programme managed to through a least square model find initial values ($R_0 = 0.4055 R_{astrosun}[m], \rho_0 = 2.0196\bar{\rho}_\odot[kg/m^3], T_0 = 4.96584 * 10^6[K]$) which made the luminosity, mass and radius all go less than 5% of the initial values. Furthermore the core ($L/L_0 < 0.995$) reached out to more than 10% of R_0 as stated in the project assignment. The final results shown in figures 18 and 19 shows why the particular initial parameters I chose work. From figure 18 we can see that the temperature and luminosity behave as we would expect them to in a star (with the assumptions we have made in this project). Considering we have assumed a noble gas, as the density and pressure increases so would the energy production as shown in figure 19. Of coarse, my results are only one solution in this parameter space. There are many more solution that might give accurate and even better approximations. The reason I chose this solution was mainly due to the number of approximations we have done in this project. Even if you would have found a set of initial parameters that had luminosity, radius and mass all converged to zero, that might not have reflected a real physical solution. What I can say is that the initial parameters I ended up with all reflect a star slightly smaller than our sun. As shown in section VII the final values of the parameters are all a bit less than those of our sun at the bottom of the convection zone. I would have been more worried if say R_0 and ρ_0 were less than those of our sun's but T_0 was greater, which is a mathematical possibility in this project. In fact, that is the case for the set of initial parameters used in section VI. This lead to some unphysical results as seen in by example figure 12. The temperature as a function of mass is more or less constant before it starts to rapidly increase as we close in on the core. The final parameters chosen however are not all in scale with our sun. Given that we the initial parameters all reflect the properties at the bottom of the stars convection zone, the final R_0 value is about 56% of the sun's, the ρ_0 value is about 40% and the T_0 value is about 87%. I don't know if they are suppose to be completely in scale with the sun's, specially considering our approximation in this project, but I think I would have liked them to be a little bit closer. This could of coarse have been done by fine tuning the least square model to find parameters more in scale, but as we had no constrictions or guidelines regarding if they should be in scale or not, I simply chose to interpret this as a consequence of the approximations.

I would like to mention that in class we were told that the radiative pressure should be around 10% of the total pressure when we are close to the core (in the core). My code however had the radiative pressure as only a fraction of the total pressure. I did not know exactly how to explain this other than the fact that I might not have been close enough to the core to actually get the correct result. I ended the programme when $R/R_0 = 0.009$ which would mean that I am still 6264 km away from the centre. I could have continue the least square method to get even closer to the the core than presently, but unfortunately, time is not always on a students side. With that said, it was hard to know exactly how much to write in this report, and I fear I might have written a lot I might not have had to. If so, I apologise for that and refer you to the last reference as a token of good will.

IX Source code

The code together with pictures and other files can be found at:
`https://github.com/erikalev/AST3310`

References

[1] B.V. Gudiksen (2017), textitAST3310: Astrophysical plasma and stellar interiors `http://www.uio.no/studier/emner/matnat/astro/AST3310/v17/beskjeder/notes/notesv1.pdf`

[2] `http://www.uio.no/studier/emner/matnat/astro/AST3310/v16/noter/opacity.txt`

[3] B.V. Budiksen, VARIABLE STEPLENGTH `http://www.uio.no/studier/emner/matnat/astro/AST3310/v16/noter/variablesteplength.pdf`

[4] E.L. Leven (2017), opacity.py `https://github.com/erikalev/AST3310`

[5] Apologetic link `https://www.youtube.com/watch?v=k9TRFSEMDNA`

X Comments

For sanity checks I have in this project worked with the "old ones". In other words, not the new posted on the web page on the 9th of Mars, but the original ones posted on the web page on the 23rd of January.

For this project I worked on my own besides from the solution of the interpolation function created in the init-function. This was a collaboration with John Christian Skorgan, who also takes the coarse.