Modeling Stars:
Two Method Comparison and Analysis

Najla Alahmadi, Michael Cock, Rebecca Halloran, Brady Metherall, Hector Robinson

March 7, 2017

In this project, we modelled a spherically symmetric, static star by numerically integrating the four stellar equations. We started by doing this for a star that had the same core conditions as the sun, and then did many other stars. Using the results, we can find a relation between luminosity and temperature of the stars, and create an HR diagram. We were to create code that would be able to take a stars composition and mass to determine mainly temperature, luminosity and surface radius and build an HR diagram. Before we get there, lets talk about our program that will numerically integrate one star.

To perform the calculations we started from the core and integrated outward using the fourth order Runge-Kutta method. We specified a core pressure, core temperature, core density, and composition of the star. From there, the outward integration calculates total mass, luminosity, pressure, and temperature at every other radius. The integration stops when the boundary conditions are met, that is, the pressure goes to zero. Once this point is reached it means we are at the surface, we can take the temperature here, and use it for the Hertzpring-Russell Diagram (further refered to as HR diagram), Figure 1 shows samples of our code in greater detail.

Figure 2 shows our results using the initial conditions of the core of the sun. These plots are as we would expect for a 1 solar mass star, but they do not get the exact values as expected. One thing that differs from our star to the sun is that it is only radiative for two radius steps, which is about 2×10^5 meters, and convective for the rest. This is in contrast to the sun, where the radiative zone is about 70% of the whole star. We were also unable to incorporate the optical depth so we used the surface temperature instead. The surface temperature should go to zero but our pressure hit zero first triggering the integration to stop. If we were able to use the temperature at a specific optical depth, we would have expected temperatures that produced a better HR diagram. We expect the final values for mass, luminosity, temperature, and pressure to be approximately $1M_\odot$, $1L_\odot$, 5770 K, and 0 Pa, respectively.

Knowing our model works properly but does not get proper values we can go further and get a general form of an HR diagram. By running this program for a total number of **number of stars** times we obtained the HR diagram shown in Figure 3.

We can see that it **looks similar to/does not really resemble** the known HR diagram. We can explore a well known program to get a sense of what the HR diagram should look like and compare it to our own model. We will use MESA to do the comparison. This might not allow for a proper comparison because our model is static and has no rotation but it should give a general shape of the HR diagram.

The main goal was to create an HR diagram by simulating a star manually and plotting the data. To do this we needed to complete many sub-parts along the way. The first objective of our team was to get a profile on github and ensure everyone knows how to pull, add, commit and push properly. In the first few tries there were of course a couple troubles where someone forgot to pull right before pushing to ensure their virsion was up to date with the original version. Eventually, everyone was able to add, commit then push a change with no problems. Splitting up work in a fair and convenient was a small challenge. Working in a big group makes it easier to write the code because we are able to all bring our ideas together and come up with a

```python
#Defines all global variables
import scipy.constants

def init(x = 0.73, y = 0.25, z = 0.02):
    global a, gamma, m_H, k, G, c, f_pp, f_3a, g_ff, mu, X, Y, Z
    a = 7.565767 * 10 **(-16)
    gamma = 5./3.
    m_H = scipy.constants.m_p
    k = scipy.constants.k
    G = scipy.constants.G
    c = scipy.constants.c
    f_pp = 1.
    f_3a = 1.
    g_ff = 1.
    mu = (2.*x + 3./4. * y + 1./2. * z)**-1
    X = x
    Y = y
    Z = z
```

(a) All constants are declared in *initialize.py*.

```python
#Define all our constants. This is where we pass X, Y, Z
initialize.init(0.7,0.25,0.05)
val = 2.
while val < 10:

    print val, "solar initial conditions"
    # Initial core conditions of the star
    P0=val*2.477 *10**16. #in units of pascal
    M0=10**-4 #units of kg
    L0=10**-4 #units of J/s or W
    T0=val*1.571*10**7 #units of K
```

(b) First *main.py* calls *initialize.init* to define our constants, then the core conditions are specified. We then open our data file, run our Runge-Kutta method, and then print the results to the data file.

Figure 1: Important sections of the code.

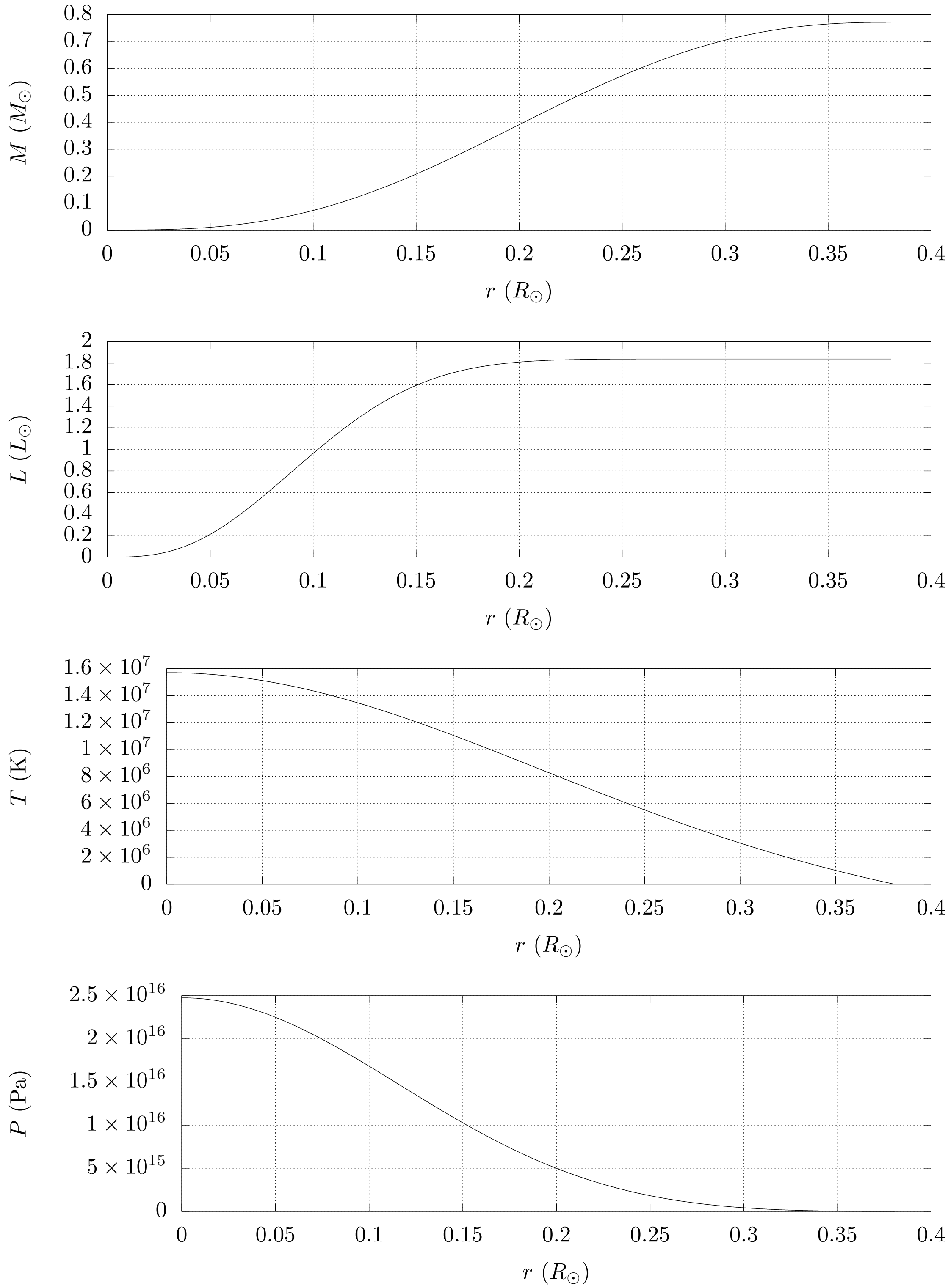

Figure 2: Simulation results of the Sun.

solution faster. We learned to work as a group in an efficient way, even though our ideas would not work sometimes we were able to listen to everyone and everyone was able to contribute. Different people have different styles of writing code so working in a group of five each person has to learn to read and interpret someone elses code. At the same time, we learned to comment our code so that the rest of our group would be able to understand what had been written and why.

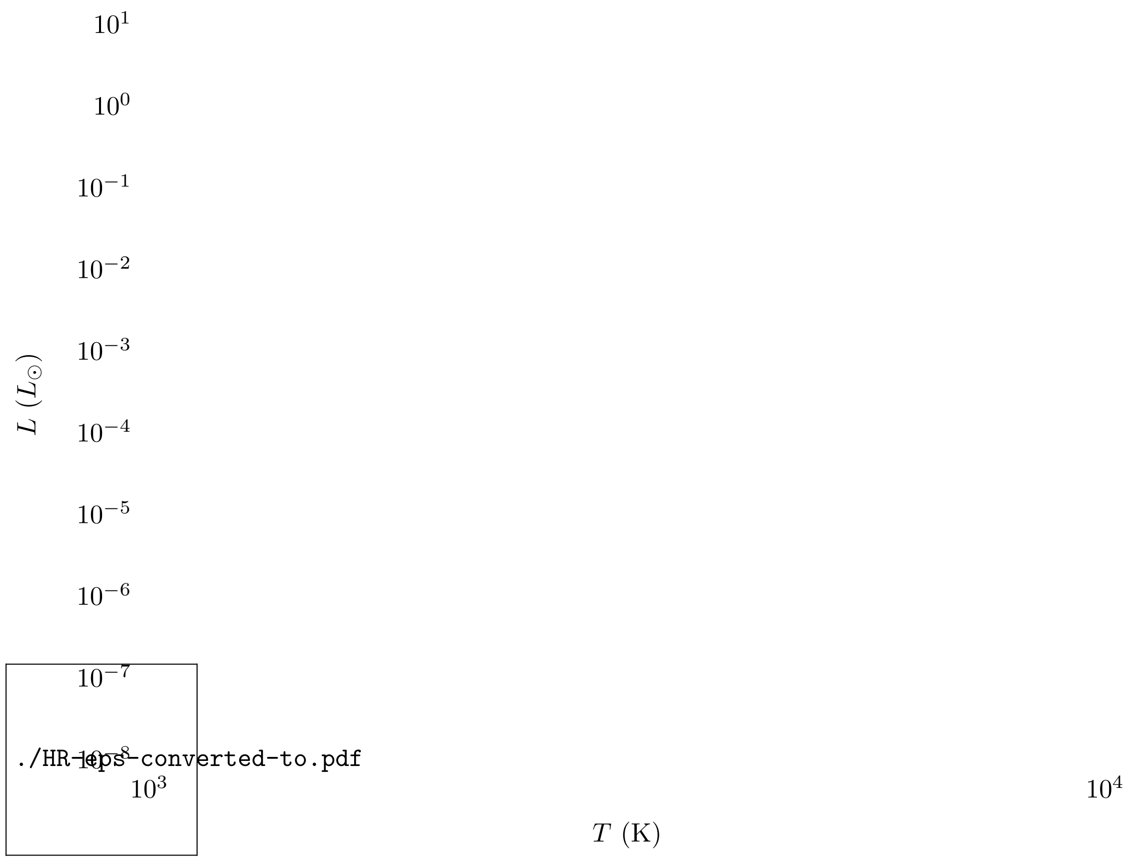

Figure 3: Our reproduction of the HR diagram. The radius of the circles represents the radius of the stars.