Stars Final Project: Gravity

Introduction

The goal of this project was to create code for numerically solving the stellar structure equations to solve for the structure of individual stars, and generate the main sequence. The code for this project can be found on github at https://github.com/amaar-quadri/StarsModifiedGravity.

Algorithmic Choices

There were many algorithmic choices made to solve the stellar structure equations effectively and efficiently.

The stellar structure equations themselves are stored in the StellarConfiguration python class. This class can calculate the 5 derivatives of interest for density, temperature, mass, luminosity and optical depth. The main function where this is done is called get_state_derivative which takes the radius and stellar state at that radius and returns the corresponding derivatives. This is all done in vector form. This structure allows all the code for the stellar structure equations and any modifications to them to be centralized in one place. Additionally, when creating the StellarConfiguration, the composition of the star as well as any gravity modifications can be specified.

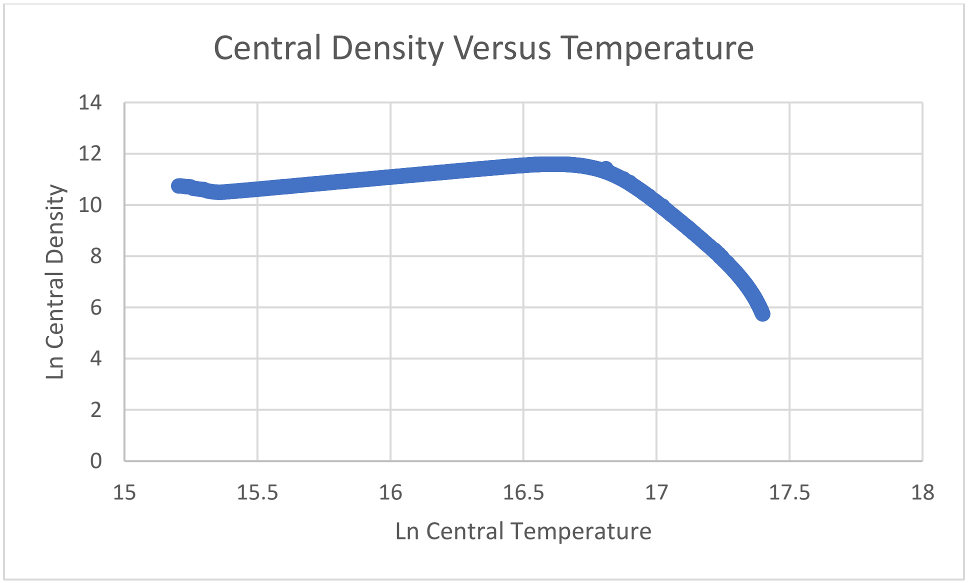
The numerical integration is done in NumericalIntegration.py. The integration itself is done using adaptive step RK45 in scipy's solve_ivp function in the trial_solution function. An event function is setup which detects when the estimated remaining optical depth (as specified in the project description) drops below the specified threshold or when the mass exceeds 1000 solar masses. This event function is then passed as one of the parameters into solve_ivp, which causes it to terminate once either of the desired conditions are met. The stellar data is returned in a 5xN matrix, where the 5 rows correspond to density, temperature, mass, luminosity and optical depth, and the N columns correspond to the radius values which are also returned in a 1xN vector. The fractional surface luminosity is also calculated as specified in the project description. Additionally, the stellar data is truncated at the point where the optical depth remaining is 2/3, which is defined to be the surface of the star. One final column of data is appended at the exact surface (via interpolation of the two surrounding data points) and the temperature at this point is manually set such that the surface boundary condition is met exactly.

The point and shoot code is located in the solve_bvp function in NumericalIntegration.py. It implements a simple bisection algorithm with a few modifications. Firstly, a guess for the central density is provided which will be used as one of the 2 endpoints for the search. As long as this guess remains to be one of the endpoints, the next tested density will be closer to the guessed density than to the other endpoint. This means that convergence will be faster than for simple bisection. The factor by which the interval was divided was empirically chosen to be 0.9, which means that the interval will decrease by a factor of 10 each time, instead of by a factor of 2. This means that the convergence will speed up by a factor of $\log_2 10 \approx 3.32$. The effectiveness of this depends on the accuracy of predictions that can be made which will be discussed later.

The second optimization to the point and shoot code was to vary the integration accuracy as the solution converged. The solve_ivp function has a parameter called rtol (relative tolerance) which specifies the required precision for each RK45 step. In the point and shoot code, when the range of densities being considered is very large at the start, a lower value of rtol was used to get very fast but coarse results. As the solution converged and the range of densities decreased, rtol is decreased to get more accurate results. This resulted in significant speed improvements.

Finally, the code to generate a whole sequence of stars is in StarSequenceGenerator.py. This section of the code essentially just amounts to calling the solve_bvp function for a variety of specified central temperatures. However, there were 2 interesting optimizations to speed up this process. Firstly, the set of desired central temperatures was split into 4 groups, each of which was computed in parallel using python's multiprocessing library. This allows computers with multiple cores to divide up the work and significantly speed up the process. The number of cores used can easily be increased on computers with more cores or decreased so that the computer can continue to be responsive for other tasks (like watching YouTube) while the code is running.

The second optimization has to do with making accurate predictions of the central density given past data. The following graph shows the relationship between central density and temperature on a log-log plot.

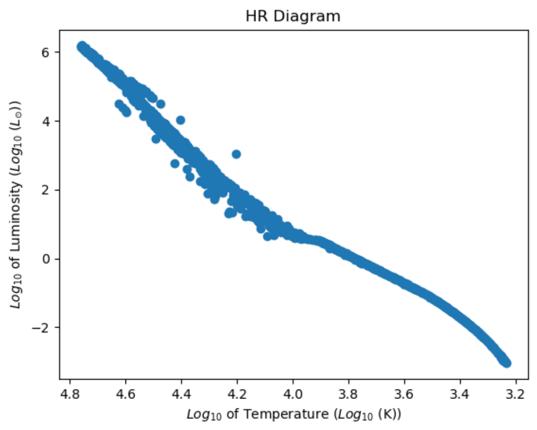


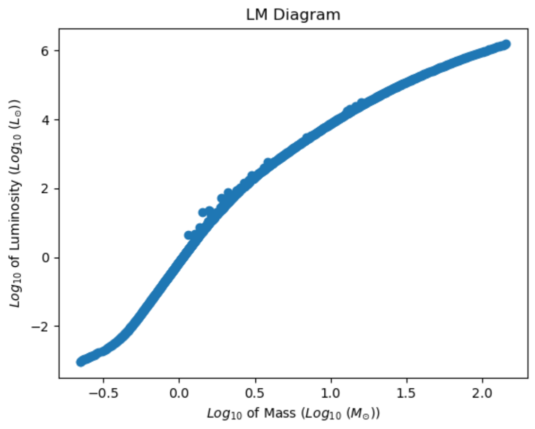
As you can see, the graph has significant portions that can accurately be predicted by a low degree polynomial, or even a straight line. The predict_rho_c function in StarSequenceGenerator uses the existing data to perform a polynomial fit on the data in log-log form using numpy's polyfit function. Since there are long term variations in the slope of the data, using a polynomial fit over the whole dataset would not be robust. Instead, a polynomial fit was done over just the last few data points. From experimenting with different number of data points and different degree polynomials, it was determined that a simple linear fit using just the last 2 data points was the simplest and most robust solution. Performance was evaluated using a large dataset of stellar densities and temperatures using the test_rho_c_predictions function. The resulting predictions had a mean absolute error of $0.00457\,\frac{\mathrm{g}}{\mathrm{cm}^3}$ and a mean percentage error of just -0.012%! This level of accuracy combined with the improvements that rely on it in the point and shoot code was one of the main performance improvements to the code.

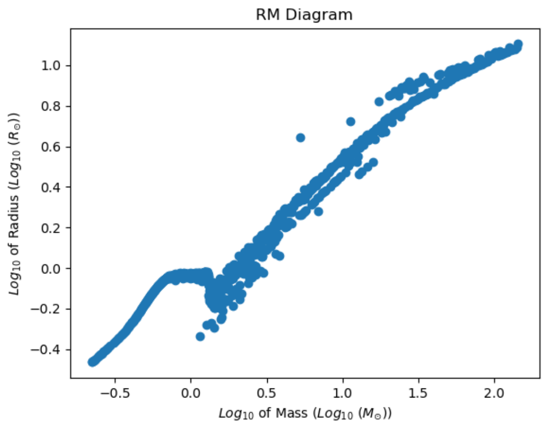
The rest of the code consists of functions for graphing, saving and loading data, as well as constants and some utility functions that are used throughout.

Main Sequence Replication

The code was used to create the following HR, LM and RM plots, with central temperatures ranging from 4 to 40 million kelvin.







For the luminosity-mass diagram, a linear trendline was fitted to the data. The result was y = 3.620 * x - 0.084. This implies a mass luminosity relation of

$$L \propto M^{3.62}$$

Which is quite close to the textbook relation of

$$L \propto M^{3.5}$$

For the mass-radius diagram, the trendline gives an equation of y = 0.577 * x - 17.818. This implies a radius mass relation of

$$R \propto M^{0.577}$$

Which is reasonably close to the textbook relation for high mass stars of

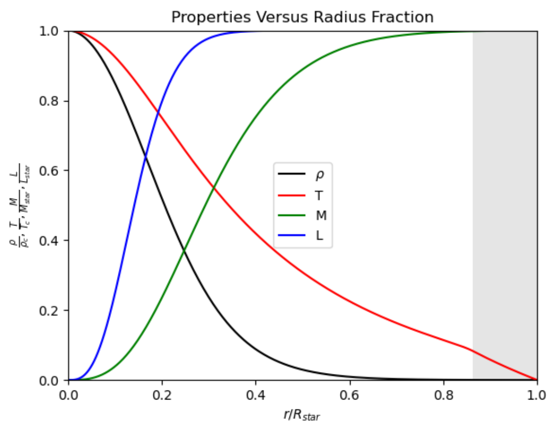
$$R \propto M^{0.555}$$

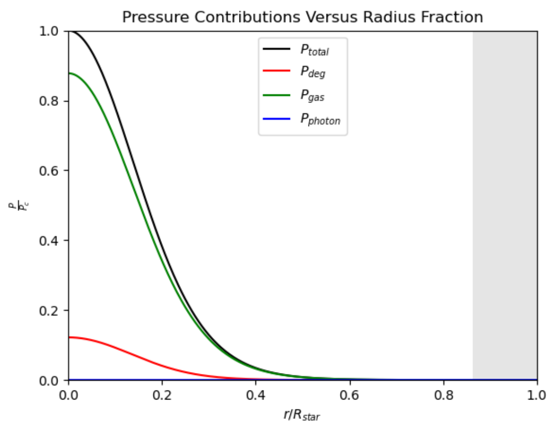
The deviations are likely due to a discrepancy in the amount of heavy metals in the star. For the generation of all the plots in this report, the following composition was used:

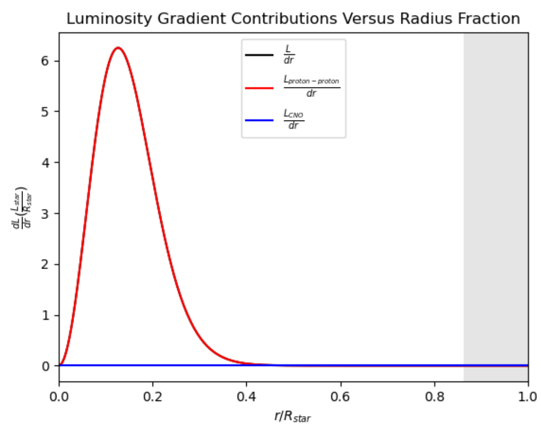
$$X = 0.7, Y = 0.269, Z = 0.031$$

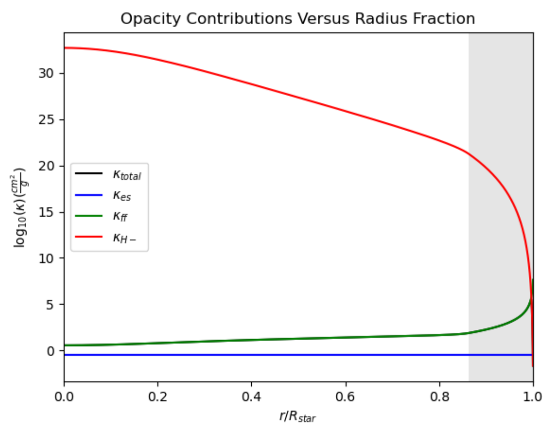
<u>Analyzing 2 Stars in Depth</u>

The code was used to produce the following plots for a star with a central temperature of 8.23544 million Kelvin. The convective regions are shown in gray.

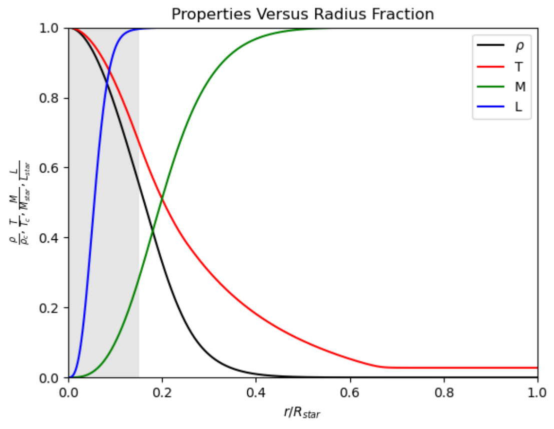


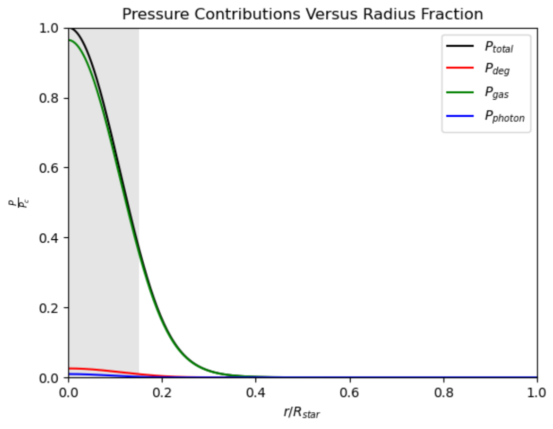


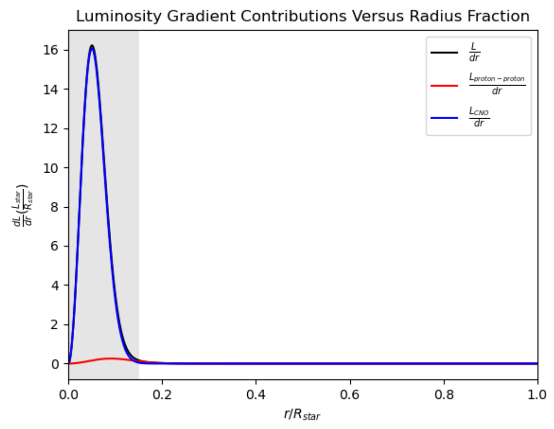


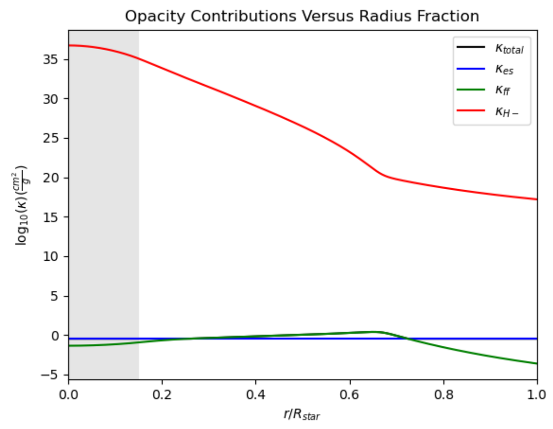


The code was also used to generate a star with a central temperature of 24 million Kelvin. The corresponding plots are shown below.









The overall properties of these stars are summarized in the following table.

Central Temperature (million K)	Central Density ($\frac{g}{cm^3}$)	Radius (R_{sun})	Mass (M_{sun})	Luminosity (L_{sun})	Surface Temperature (K)
8.23544	60.6296	0.8689	0.6600	0.05975	3060.17
24	25.4575	2.8611	3.6554	350.459	14757.70

In the lower mass star, the surface layers are convective, whereas in the higher mass star the core is convective. The convection in the core of the higher mass stars is explained by the large temperature gradients and the large energy generation rates due to the CNO cycle. In the lower mass star, the surface convection zone is explained by large temperature gradients caused by the high opacity of unionized hydrogen.

In the lower mass star, the proton-proton chain is the dominant energy generation source because the temperature is not high enough to start the CNO cycle. However, in the larger mass star the temperature is much higher, so the dominant energy generation source is the CNO cycle.

In both stars, the opacity is dominated by κ_{H-} throughout most of the interior of the star. This is contrary to what we would expect, since the opacity due to unionized H- atoms is only expected to have an effect near the surface where the temperature is low. Additionally, in the low mass star, the dominant opacity source near the surface is κ_{ff}. Given these two facts, it is likely that an error resulting in the two opacities being swapped.

Work Distribution

The entire codebase that was used to generate all the graphs in this report as well as for the presentation was written entirely by me (you can confirm this from the git history). My groupmates were slow to help and all but 2 of them didn't contribute a single line of code. The code that those 2 members did write was for graphing the HR and RM diagrams, but it was lacking functionality and as the time of the presentation approached, it was superseded by the plot_HR and plot_RM functions in StellarSequenceGraphing.py (both of which are less than 20 lines of code, whereas the entire project is over 1000 lines of code).

I didn't really have much of a problem with this uneven work distribution because I took this course as an elective out of interest, not necessity. I really enjoyed all the lectures and I enjoyed programming this myself, and I intend to continue to work on it and experiment with other modifications after this course. I hope that you take this into account while marking.

Thanks for teaching an awesome course, I really enjoyed it (even if my marks weren't the best 😜)! I am amazed just how much I learned from it!