Computational Astrophysics: Modelling Stellar Energy Transport

John Lopez

University of Amsterdam Spring 2024

1 Introduction

The Modules for Experiments in Stellar Astrophysics (MESA) software is an open-source 1D stellar evolution code regularly used in stellar evolution research. This software is able to model different types of stars in different stages of their lifetime, depending on the initial parameters Paxton et al. (2011). Similar to MESA, the goal of this project is to model the structure of a sun-like star, including both radiative and convective energy transport. However, unlike MESA, this model only builds the star in a single moment of time. My model can be found here: My GitHub Submission.

In this project, the star is assumed to be spherically symmetric, and in hydrostatic and thermal equilibrium. Since there is no time dependence, the structure will depend on mass and radius. To generate the internal structure of the star, this involves retrieving the pressure, density, temperature, and method of energy transport. The governing equations to do so are as follows:

$$\frac{\partial r}{\partial m} = \frac{1}{4\pi r^2 \rho} \tag{1}$$

$$\frac{\partial P}{\partial m} = -\frac{Gm}{4\pi r^2} \tag{2}$$

$$\frac{\partial L}{\partial m} = \epsilon \tag{3}$$

$$\frac{\partial T}{\partial m} = \begin{cases} \nabla^* \frac{T}{P}\frac{\partial P}{\partial m} & \text{if } \nabla_{\text{stable}} > \nabla_{\text{ad}} \\ \frac{3\kappa L}{256\pi^2 \sigma r^4 T^3} & \text{otherwise} \end{cases} \tag{4}$$

$$P = \frac{4\sigma}{3c}T^4 + \frac{\rho k_B}{\mu m_u}T \tag{5}$$

$$\rho = (P - \frac{4\sigma T^4}{3c})\frac{\mu m_u}{k_B T} \tag{6}$$

where **r** is the radial coordinate from the core to the surface of the star, $\boldsymbol{\rho}$ is the density, **P** is the pressure, **L** is the luminosity, $\boldsymbol{\epsilon}$ is the total energy released from the fusion reactions in the core of the star, **T** is the temperature, $\boldsymbol{\kappa}$ is the opacity, $\boldsymbol{k_B}$ is the Boltzmann constant, $\boldsymbol{\sigma}$ is the Stefan-Boltzmann constant, **c** is the speed of light, and $\boldsymbol{m_u}$ is the atomic mass unit. The $\boldsymbol{\nabla}$ refers to the specific temperature gradients: $\boldsymbol{\nabla_{ad}}$, $\boldsymbol{\nabla_{stable}}$, and $\boldsymbol{\nabla_*}$ which are adiabatic, stable, and convective temperature gradients respectively. Finally, $\boldsymbol{\mu}$ is the atomic weight, defined as $(2X + 3Y/4 + Z/2)^{-1}$ where X, Y, and Z are the fractions of hydrogen, helium, and metals respectively.

Equations (1), (2), (3), and (4) are ordinary differential equations (ODEs) describing the evolution of radius, pressure, luminosity, and temperature, respectively, with respect to the mass coordinate. At the stellar center, the mass $M_r = 0$, radius $r = 0$, and luminosity $L_r = 0$. These are therefore the **inner boundary conditions**. On the surface, the **outer boundary conditions** are density $\rho = 0$, temperature $T = T_0$, and mass $M_r = M$. These boundary conditions will be used to solve the ODEs. Finally, there are therefore five initial values to adjust to find the best model: r_0, T_0, ρ_0, P_0, and T_0. As for method of energy transport, this is determined by comparison of temperature gradients. When $\nabla_{stable} > \nabla_{ad}$ convection occurs, and the temperature ODE uses the equation containing the convective temperature gradient ∇_*. Otherwise, the transfer of energy becomes radiative and the other equation is used. This is known as the Schwarzschild criterion.

2 Methodology

For this project, I created an object-oriented python script built around an Euler-method ODE solver. Essentially, this program consists of a single class that possesses the entirety of the constants, equations, calculations, solvers, and analysis needed to model the 1-D stellar structure of a sun-like star. To recap, the Euler method is a first-order accurate numerical procedure for solving ODEs when provided with an initial value. The solution is approximated by numerous small steps, along the derivative of the function in question.

My model begins by initializing all of the required constants and initial variables. Since the goal is to generate a sun-like star, the initial variables are based on values of the sun ($R_\odot$, $M_\odot$, $T_\odot$, $\rho_\odot$, and $P_\odot$). Next, all equations required to solve the aforementioned ODEs are defined, this includes equation (5) which calculates the total pressure as the sum of radiation and gas pressure, and equation (6) which calculates the density of the system. Other equations include gravity **g**, scale height $\boldsymbol{H_p}$, heat capacity $\boldsymbol{C_p}$, and total flux **F**. Other key equations used are the temperature gradients ∇_{stable} and ∇_{ad} as follows:

$$\nabla_{stable} = \frac{3\kappa H_p L\rho}{64\pi r^2 \sigma T^4} \tag{7}$$

$$\nabla_{ad} = \frac{P}{T\rho C_p} \tag{8}$$

To calculate the convective temperature gradient ∇_*, a root finder function was employed to find the roots of the following 3rd degree polynomial:

$$\xi^3 + \frac{U}{l_m^2}\xi^2 + \frac{U^2\Omega}{l_m^3}\xi + \frac{U}{l_m^2}\left(\nabla_{\text{ad}} - \nabla_{\text{stable}}\right) = 0 \tag{9}$$

where

$$U = \frac{64\sigma T^3}{3\kappa\rho^2 C_p}\sqrt{\frac{H_p}{g}} \tag{10}$$

and

$$\nabla_* = \xi^2 + \frac{U\Omega}{l_m}\xi + \nabla_{ad} \tag{11}$$

After finding all roots for each step, the root finder function then selected the root with the minimum "imaginary" part, or the most "real" root, to calculate the convective temperature gradient. Finally, the last equations defined were the four ODEs mentioned earlier, with the temperature ODE being split into two different equations.

To calculate κ and ϵ, linear 2D interpolation was incorporated into the model. To feed the interpolator, two files, **opacity.txt** and **epsilon.txt**, were used. The **opacity.txt** file contained multiple sets of values for $log_{10}(R)$, density, temperature, and $log_{10}(\kappa)$. The **epsilon.txt** contained multiple sets of values for $log_{10}(R)$, density, temperature, and $log_{10}(\epsilon)$. For the purpose of the model, everytime either κ or ϵ were to be calculated, the respective function would take an input temperature and density, and use these to interpolate a value for either variable.

In order to verify that these interpolator functions are calculating accurate values, a "sanity check" for κ and ϵ were used. This involved using the interpolators on a respective reference value table for each variable. The κ reference table contained multiple rows of values for $log_{10}(T)$, $log_{10}(R)$ [cgs], $log_{10}(\kappa)$ [cgs], and $\kappa \times 10^3$ [SI]. The ϵ reference table contained multiple rows of values for $log_{10}(T)$, $log_{10}(R)$ [cgs], $log_{10}(\epsilon)$ [cgs], and $\epsilon \times 10^3$ [SI]. Essentially, this sanity check worked by inputting each pair of $log_{10}(T)$ and $log_{10}(R)$ values into the respective interpolator, and comparing the output κ or ϵ value to the κ or ϵ reference value (cgs and SI) to see if the interpolated value is within 5%. If so, the interpolator is therefore functioning properly and would output a **PASS**, or **FAIL** if otherwise.

In order to make the solver more accurate and efficient, adaptive mass stepping was implemented into the model as another function. Mass was selected as the changing value since mass is the variable of integration in each ODE. The ODEs take the general form of:

$$\frac{dV}{dm} = f \tag{12}$$

where **V** is the changing variable with each **dm**, and **f** is the function that describes how **V** changes with **m**. To control how much the variable changes with each step, a condition is imposed:

$$|\frac{dV}{V}| < p \tag{13}$$

where **p** is a small fraction that keeps **V** from changing too much. In my code, the optimal **p** was found to be 0.01. Putting these two equations, the mass step can be found:

$$dm = \frac{pV}{f} \tag{14}$$

Since this project has multiple changing variables, the mass step function calculates **dm** for each variable, and the smallest **dm** is selected to maintain stability. I reference the course page AST3310 by B. V. Gudiksen, Institute for Theoretical Astrophyics, University of Oslo for help with the mass step: AST3310.

All of these steps lead to the main event of the code, the ODE solver. The "ODE Euler" function utilizes the Euler method to approximate the solution of the ODEs that describe the stellar interior. These approximations happen in under a fixed amount of steps, which my code sets the limit to 10^5. The solver works by starting at the stellar surface, and incrementally working its way towards the center of the star, essentially performing an approximation at each layer of enclosed mass. With each step, the variables that populate the ODEs are updated and recorded in output arrays. These essentially document the parameters of each layer of enclosed mass from outside-in. These parameters include radius, mass, density, pressure, flux, luminosity, temperature, epsilon, the overall temperature gradient, the stable temperature gradient, the adiabatic temperature gradient, and the convective temperature gradient.

Starting on the stellar surface, the function calculates and records all initial variables before beginning the first approximation. A key part in how this function operates is determining which temperature gradient is dominating at the current step as this decides whether convection or radiation is the form of energy transfer occurring, which thereby determines which temperature ODE to solve. Referring back to equation (4), convection is occurring when $\nabla_{stable} > \nabla_{ad}$, and radiative energy transfer is occurring otherwise. With each step, the solver checks this criterion through comparisons of the different temperature gradients. If convection is occurring, ∇_* is calculated and recorded as the dominating temperature gradient, and the proper temperature ODE is calculated. Otherwise, if radiative energy transfer is occurring, ∇_{stable} is calculated and recorded as the dominating temperature gradient, and the proper temperature ODE is calculated. Outside of this criteria, the other ODEs are calculated normally each step.

With all of the required variables and ODEs calculated, the mass step can then be approximated using the aforementioned mass step function. This mass step is then used to update every variable to be used in the next step. To determine when to stop the approximations, each step performs a check to see if mass has reached approximately zero. If so, the solver was successful and the variables are printed out. Otherwise, if radius, density, pressure, luminosity, or temperature become less than or equal to zero, something in the solver is malfunctioning and the operation is canceled.

Now that the variables have all been approximated throughout the interior of the star, the data can be visualized and analyzed, which is done by two final functions. To visualize how the variables evolved with each step, one function generates six subplots as functions of normalized radius: mass, density, pressure, luminosity, temperature, and the three temperature gradients. The other function generates a figure of depicting the cross section of the star, showing where convective and radiative energy transfer occurs in the core and the shell of the star, again as a function of normalized radius, where the shell is $L \geq 0.995L_\odot$ and the core is otherwise. Additionally, this function calculates the radius of the total core, alongside the width of the convective layer in the shell, both as a percentage of the total stellar radius.

3 Results

To evaluate this model, two tests were performed. For the first test, the model was ran using initial variables equal to that of the sun: $L_0 = L_\odot$, $R_0 = R_\odot$, $M_0 = M_\odot$, $\rho_0 = 1.42 \times e^{-7} \times \rho_\odot$ (for unit conversions), and $T_0 = T_\odot$. The resulting evolution of variables is visualized in figure 1. and the corresponding stellar cross section can be seen in figure 2. In the output of the model, after 3066 steps it is reported that the final mass is within 0% of M_0, radius is within 6.69% of R_0, and luminosity is within 3.31% of L_0. While luminosity is within acceptable range, the radius is a tad out of range, leading me to believe something in my code is not accurately calculating radius. When looking at figure 1. All parameters appear to have acceptable behaviors, with mass and luminosity increasing as radius increases, while density, pressure, and temperature all decrease with increasing radius. The temperature gradient plot also proves accurate when compared to figure 2. It can be seen that where convection occurs in the cross section, the stable temperature gradient is dominating, and where radiation occurs, the adiabatic temperature gradient is domination. However, another issue with radius can be seen in figure 2. as the reported surface convection area is extremely small at only 0.25% of R_0. Regardless, the methods of energy transfer occur in the proper locations, and the core is of reasonable size.

For the second test, I had to narrow down the best values that generated star that met certain benchmarks. The first benchmark required that the final mass, radius, and temperature (final meaning at the very center of the core) were within 0-5% of M_0, R_0, and L_0 respectively. The second benchmark required that the radius of the core (where $L < 0.995 L_\odot$) reached out to at least 10% of R_0. Finally, the third benchmark required that the width of the continuous convection zone near the surface of the star should be at least 15% of R_0.

The best values I could find were as follows: $L_0 = 1.0 L_\odot$, $R_0 = 1.0 R_\odot$, $M_0 = 0.97 M_\odot$, $\rho_0 = 2000.0 \times 1.42 \times e^{-7} \times \rho_\odot$ (for unit conversions), and $T_0 = 10.0 T_\odot$. Noticeably, ρ_0 is a few magnitudes larger than the other values. I noticed the model responded strongest with density and therefore pressure. In the output of the model, after 2046 steps it is reported that the final mass is within 0% of M_0, radius is within 1.93% of R_0, and luminosity is within 5.05% of L_0. While luminosity is a tad high, the values are all within or roughly on par with the first benchmark. This means the model is calculating values all the way through to the center of the star. For the second benchmark, figure 4. demonstrates that the core radius is 25.27% of R_0, well above the 10% mark. However, the third benchmark is where most problems came up. With these values, I could only achieve a surface convection width of 2.26% of R_0. I tried countless different combinations and scaling of the initial variables, however this was the best I could achieve without the model failing to approximate, primarily because luminosity kept going to zero before mass. I believe this is due in part to a bug somewhere in my code related to how either radius is calculated, or how one of the temperature gradients is calculated. It looks like the surface convection layer is not being influenced as strongly by the initial variables as it should. When looking at figure 3, all of the curves seem to be within reason, however the convection temperature gradient cannot be seen in the temperature gradient figure. This could mean the bug is related to how the convective temperature gradient is calculated.

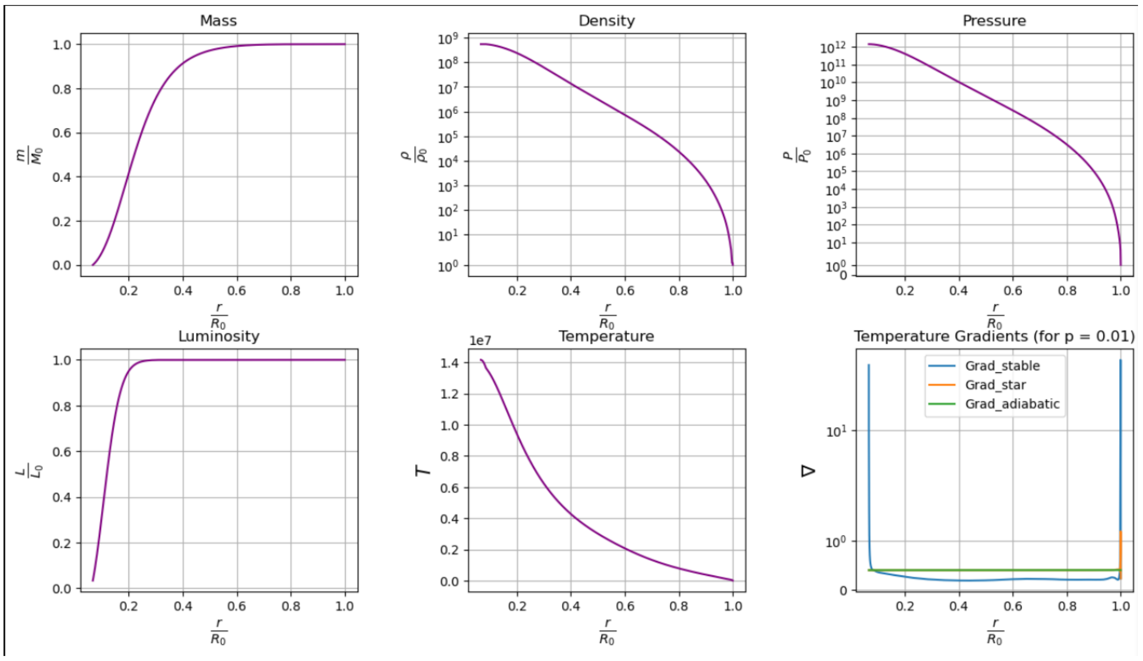

Figure 1: Plots of key variables from model using solar initial parameters. In order of plots: mass, density, pressure, luminosity, temperature, and temperature gradients as functions of radius. All variables are normalized by their solar counterparts.

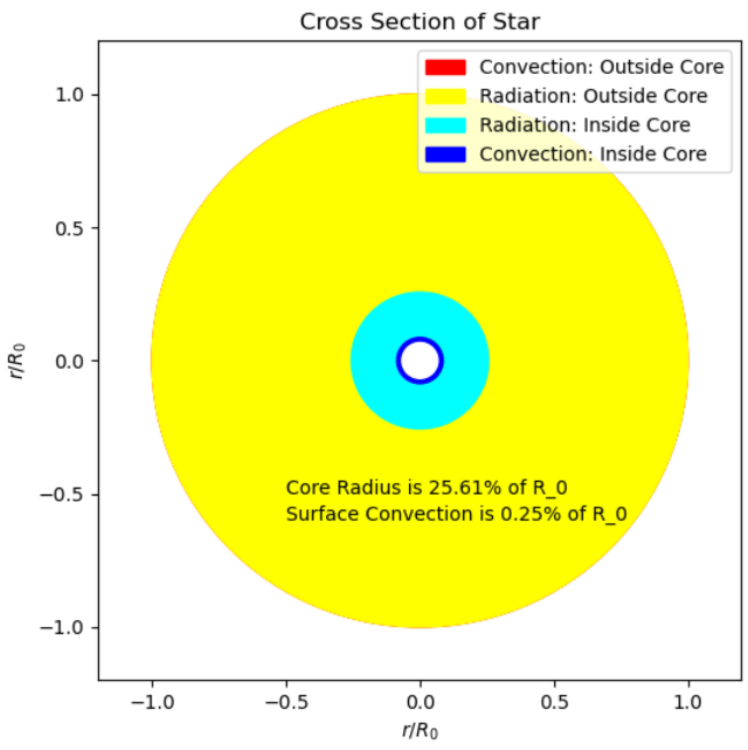
Cross Section of Star
🟥 Convection: Outside Core
🟨 Radiation: Outside Core
🟦 Radiation: Inside Core
🟦 Convection: Inside Core
Core Radius is 25.61% of R_0
Surface Convection is 0.25% of R_0
r/R_0
r/R_0

Figure 2: Figure depicting the cross section of the stellar model using solar initial parameters. The axis' are both radius normalized by solar radius.

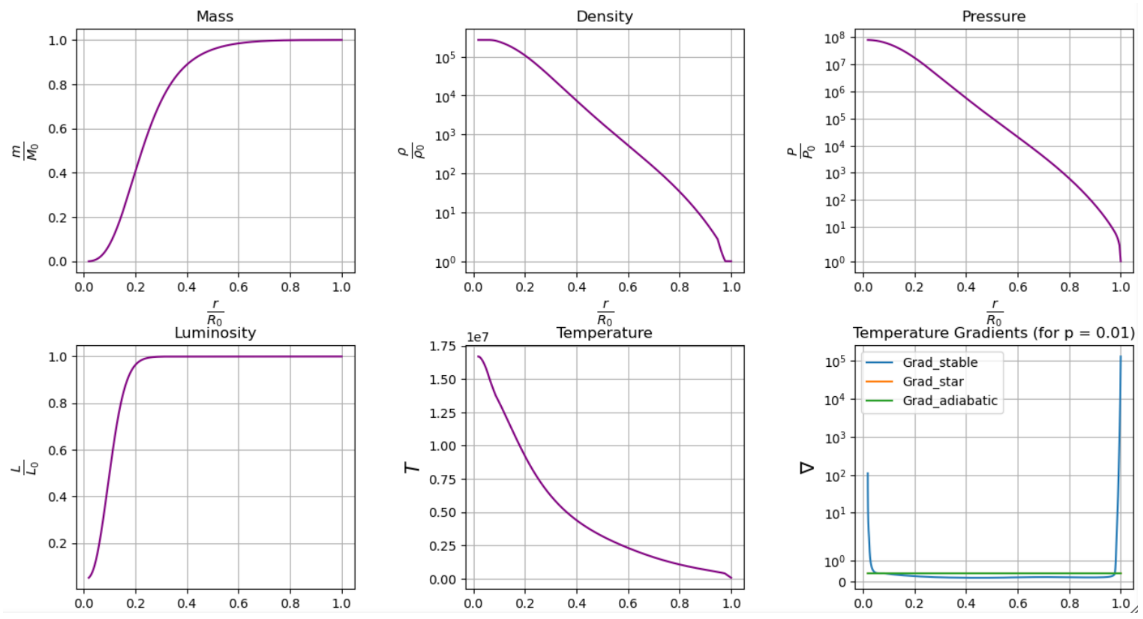

Figure 3: Plots of key variables from model using best found initial parameters. In order of plots: mass, density, pressure, luminosity, temperature, and temperature gradients as functions of radius. All variables are normalized by their solar counterparts.

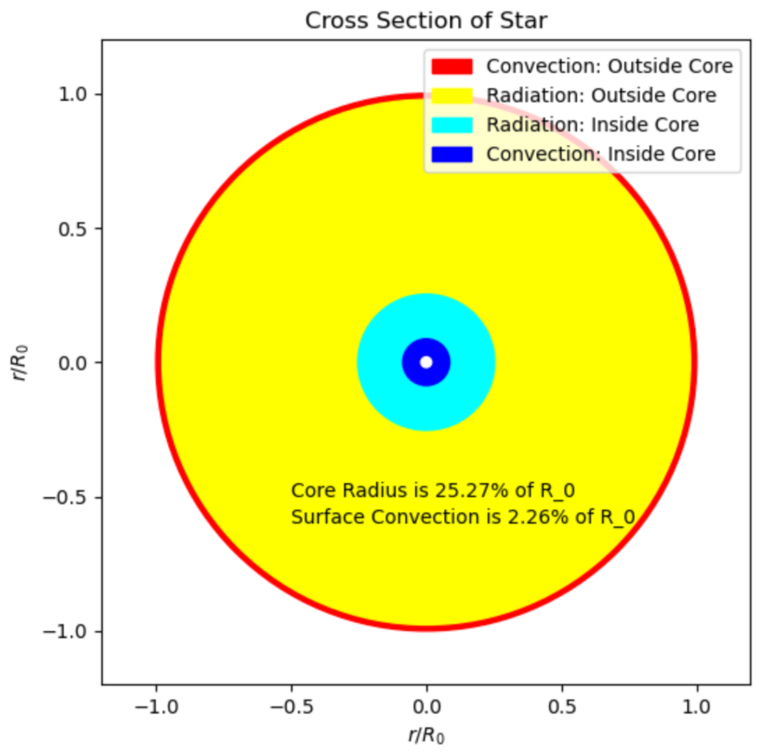

Figure 4: Figure depicting the cross section of the stellar model using the best found initial parameters. The axis' are both radius normalized by solar radius.

References

Paxton B., Bildsten L., Dotter A., Herwig F., Lesaffre P., Timmes F., 2011, , 192, 3