Computational Astrophysics - Final Project Report

Modelling Stellar Energy Transport

Kaya Han Taş – 15064735

(University of Amsterdam - Astronomy & Astrophysics)

(Code: https://github.com/Computational-Astrophysics-API/set-up-github-Kaya-Han-Tas/blob/main/Mesa_code_final_version.py)

Introduction

- In this project we plan to Model the central parts of a Sun-Like Star, including both Radiative and Convective Energy Transport.
- **Assumptions:**
  - 1-) **There is no Conduction in our Star Model!**
  - 2-) **The Star is Spherically Symmetric!**
  - 3-) **We assume that all quantities are One-Dimensional!**
  - 4-) **Ideal Gas is assumed for all of the star!**
  - 5-) $\boldsymbol{X = 0.7, Y = 0.29 + 10^{-10}, Z = 1 - (X + Y)}$.
- **The Goal of the Project:**
  - **Getting a One-Dimensional Profile of the Internal Structure of the Star, only at one certain time in their life.** *(We are not looking for the structure change in time)*
  - This includes the following:
    - 1-) **Pressure Profile**
    - 2-) **Density Profile**
    - 3-) **Temperature Profile**
    - 4-) **Method of Energy Transport**
- The Equations for solving the **Internal Structure of the Radiative Zone** of the Sun are as follows:

$$\boxed{\frac{\partial r}{\partial m} = \frac{1}{4\pi r^2 \rho}} \quad \boxed{\frac{\partial P}{\partial m} = -\frac{Gm}{4\pi r^4}} \quad \boxed{\frac{\partial L}{\partial m} = \epsilon}$$

Change in Radial Coordinate, Pressure and Luminosity with Respect to the Mass Coordinate

- For the Change in Temperature there are two possibilities.

$$\boxed{\frac{\partial T}{\partial m} \begin{cases} \nabla^* \dfrac{T}{P}\dfrac{\partial P}{\partial m} \text{ if } \nabla_{adiabatic} < \nabla_{stable} \\ -\dfrac{3\kappa L}{256\pi^2 \sigma r^4 T^3} \text{ if otherwise} \end{cases}}$$

Change in Temperature with Respect to the Mass Coordinate

- We can also calculate the Pressure and Density as follows.

$$\boxed{P = \frac{4\sigma}{3c}T^4 + \frac{\rho k_B}{\mu m_u}T} \quad \boxed{\rho = \left(P - \frac{4\sigma T^4}{3c}\right) \times \frac{\mu m_u}{k_B T}}$$

Equations for conversion between Pressure and Density

- **<u>In the equations we have listed, the parameters are as follows:</u>**
  - $\boldsymbol{m}$: Mass Coordinate (Mass at Spherical Shells throughout the Star)
  - $\boldsymbol{r}$: Radial Coordinate (From Core to the Surface of the Star)
  - $\boldsymbol{\rho}$: Density
  - $\boldsymbol{P}$: Pressure
  - $\boldsymbol{L}$: Luminosity
  - $\boldsymbol{\epsilon}$: Total Energy Released from the Fusion Reactions in the Core of the Star.
  - $\boldsymbol{T}$: Temperature
  - $\boldsymbol{\nabla^*}$: Convective Temperature Gradient
  - $\boldsymbol{\nabla_{adiabatic}}$: Adiabatic Temperature Gradient
  - $\boldsymbol{\nabla_{stable}}$: Stable Temperature Gradient
  - $\boldsymbol{\nabla}$: THE Temperature Gradient
  - $\boldsymbol{\kappa}$: Opacity
  - $\boldsymbol{k_B}$: Boltzmann Constant
  - $\boldsymbol{\sigma}$: Stefan-Boltzmann Constant
  - $\boldsymbol{c}$: Speed of Light
  - $\boldsymbol{m_u}$: Atomic Mass Unit
  - $\boldsymbol{\mu}$: Mean Atomic Weight
    - $\boldsymbol{\mu = (2X + 3Y/4 + Z/2)^{-1}}$
    - $\boldsymbol{X}$: Fraction of Hydrogen
    - $\boldsymbol{Y}$: Fraction of Helium
    - $\boldsymbol{Z}$: Fraction of Heavier Elements

- Now let's talk about **THE Temperature Gradient (∇)!**
- It is required to determine **whether a specific zone in the star is Radiative or Convective.**
- This also determines the **equation we use for the Change in Temperature!**
- The following are the formulas we need to use:

$$\boxed{\nabla_{stable} = \frac{3\kappa H_p L\rho}{64\pi r^2\sigma T^4} = \frac{3\kappa LP}{64\pi\sigma GmT^4}} \quad \boxed{H_p = \frac{k_B T}{\mu m_u g}} \quad \boxed{g = G\frac{m(r)}{r^2}}$$

Equations of Stable Temperature Gradient, Scale Height, and Gravitational Acceleration

$$\boxed{\nabla_{adiabatic} = \frac{P}{T\rho C_p}} \quad \boxed{C_p = \frac{5}{2}\frac{k_B}{\mu m_u}}$$

Equations of Adiabatic Temperature Gradient (Assuming Ideal Gas), and Heat Capacity (As Constant Pressure)

- Convection takes over when $\nabla_{adiabatic} < \nabla_{stable}$!
- **THE Temperature Gradient can be two things:**
  - 1-) **If Mass Shell is Convectively Stable:** $\boxed{\nabla = \nabla_{stable}}$
  - 2-) **If Mass Shell is Convective:** $\boxed{\nabla = \nabla^*}$

- To find the **Convective Temperature Gradient** ∇^* we need to find Roots of the following Equation!

$$\boxed{\xi^3 + \frac{U}{l_m^2}\xi^2 + \frac{U^2}{l_m^3}\Omega + \frac{U}{l_m^2}(\nabla_{adiabatic} - \nabla_{stable}) = 0} \quad \boxed{U = \frac{64\sigma T^3}{3\kappa\rho^2 C_p}\sqrt{\frac{H_p}{g}}} \quad \boxed{l_m = H_p} \quad \boxed{\Omega = \frac{4}{l_m}}$$

3rd Degree Polynomial that needs to be solved with at least 1 Real Root!

(The other listed equations are used as variables in the 3rd Degree Polynomial)

- Once we find ξ, we can calculate Convective Temperature Gradient ∇^* as follows.

$$\boxed{\nabla^* = \xi^2 + \frac{U}{l_m}\xi + \nabla_{adiabatic}}$$

Equation for Convective Temperature Gradient

- In the guide for the project, it is given to us that to find which zone is Radiative or Convective, we need to calculate the Total Flux, Radiative Flux and Convective Flux!
- For these, we can use the following equations. (Assuming no Conduction!)

$$\boxed{F = \frac{L}{4\pi r^2}} \quad \boxed{F_R = \frac{16\sigma T^4}{3\kappa\rho H_p}\nabla} \quad \boxed{F_C = F - F_R}$$

Equations of Total Flux, Radiative Flux and Convective Flux

(Radiative Flux found from Onno Pols Lecture Notes – Equation 5.15)

Methods & Code Description

- In the code, we define a called `Star_model` that contains all of our variables and functions.
- Then we define `__init__(self)` function that has the Constants and Initial Values such as:
  - k_B: Boltzmann Constant
  - σ: Stefan-Boltzmann Constant
  - c: Speed of Light
  - m_u: Atomic Mass Unit
  - μ: Mean Atomic Weight
  - ρ_0: Initial Density
  - M_0: Initial Mass
  - R_0: Initial Radius
  - L_0: Initial Luminosity
  - T_0: Initial Temperature
  - C_p: Heat Capacity

- We also define a variable (`end_step`) to store the final step of our Integration.
- We finally define another variable (`adaptive_timestep`) that helps the user to change if they want adaptive timestepping or not during the Integration.

- We first define a function (`get_opacity`) to get the Opacity values from "**opacity.txt**".
- In the function, the vital point is the conversion of units.
- Since our initial values and constants are in SI units, we need the output to be in SI units too.
- But the "**opacity.txt**" file uses CGS units.
- For that reason, we need to convert the Density value from SI to CGS, then take the logarithm of it to use it in the file! (For temperature, only the logarithm part applies)
- After finding the Opacity value from the "opacity.txt" file, since it is in logarithm, we need to convert it back to a normal number, then turn that into SI units as well!
- Using Unit Analysis, we can find that the conversion from CGS to SI for Opacity is **multiplying it with 10^{-1}**.

- For getting the Total Energy Generation, we again define a function (`get_total_energy`) to get values from "**epsilon.txt**".
- The same thing we did to get values from "**opacity.txt**" applies for this one as well!
- The only difference is the conversion from CGS to SI for Total Energy, which from Unit Analysis can be found as **multiplying it with 10^{-4}**.
- <u>For both functions the following are implemented:</u>
  - 1-) Checking whether the Interpolation works well or not by using the Sanity Check table given in the guide for the project.
  - 2-) Printing a Warning if the value is extrapolated due to the given Temperature and/or Radius value being out of bounds.

- Next, we define the functions `density_calculator` and `pressure_calculator` using the *Equations for conversion between Pressure and Density*.
- Note that to calculate the Initial Pressure during the integration, `pressure_calculator` will be used.
- **<u>Then we define the following functions which will be used during integration:</u>**
  - **`radial_coordinate_change`**: Using the *Change in Radial Coordinate with respect to the Mass Coordinate.*
  - **`pressure_change`**: Using the *Change in Pressure with respect to the Mass Coordinate.*
  - **`luminosity_change`**: Using the *Change in Luminosity with respect to the Mass Coordinate.* (uses the `get_total_energy` function)
  - **`temperature_change`**: Using the *Change in Temperature with respect to the Mass Coordinate's "if otherwise" equation.* (The other equation will be used if the $\nabla_{adiabatic} < \nabla_{stable}$ condition is satisfied during Integration)
  - **`nabla_stable_calculate`**: Using the *Equations of Stable Temperature Gradient, Scale Height, and Gravitational Acceleration.*
  - **`nabla_ad_calculate`**: Using the *Equations of Adiabatic Temperature Gradient (Assuming Ideal Gas), and Heat Capacity (As Constant Pressure).*
  - **`nabla_star_calculate`**: Using the *3rd Degree Polynomial* and *Equation for Convective Temperature Gradient.*

- Note that to find the real root(s) of the *3rd Degree Polynomial* we need to solve to find Convective Temperature Gradient ∇^*, we use `np.root` and `np.real` in the code. *(Refer to the comments in the code for details on how the process works.)*
- We also need to define functions for the Fluxes to determine whether a zone is Radiative or Convective.
- **For this we define the following functions which again will be used during integration:**
  - **`total_flux`**: Using the *Equation of Total Flux.*
  - **`radiative_flux`**: Using the *Equation of Radiative Flux.*
  - **`convective_flux`**: Using the *Equation of Convective Flux.*

- Now we need to solve the Stellar Structure Equations i.e. do the Integration.
- For this we define the `ODE_solver` that uses the **Euler Method**.
- **The `ODE_solver` Function does the following step by step:**
  - 1-) Starts a timer that will count how much time has passed while integrating.
  - 2-) Defines the Number of Steps we will take during the Integration.
  - 3-) Defines `np.array`s for all of the parameters we have: *Mass, Radius, Luminosity, Pressure, Temperature, Density, Total Energy, THE Temperature Gradient, Convective Temperature Gradient, Stable Temperature Gradient, Adiabatic Temperature Gradient, Total Flux, Radiative Flux, Convective Flux.*
  - 4-) Sets the Initial Values of the *Mass, Radius, Luminosity, Temperature, Density* arrays as the Initial Values inside the `__init__(self)` function.
  - 5-) Calculates the Initial Values of *Pressure, Total Energy, Scale Height, Gravitational Acceleration, THE Temperature Gradient, Convective Temperature Gradient, Stable Temperature Gradient, Adiabatic Temperature Gradient, Total Flux, Radiative Flux, Convective Flux* by using those.
  - 6-) Starts iterating over the Number of Steps we have defined.
  - 7-) Defines a Progress Bar that checks the progress by using the current Mass value that the step has i.e. `m[i]`.
  - 8-) Checks if *Radius, Luminosity, Pressure, Temperature, Density* has hit zero or not, if it has, it stops the Integration and prints out the results.
  - 9-) Checks if *Mass* has hit zero or not, if it has, it stops the Integration and prints out the results. (Which means that we have finished the Integration successfully from Surface to the Core)
  - 10-) Calculates *Opacity, Gravitational Acceleration and Scale Height* using the current values of the Parameters in the iteration.
  - 11-) Calculates *Convective Temperature Gradient, Stable Temperature Gradient, Adiabatic Temperature Gradient* and *Total Flux.*
  - 12-) Checks if $\nabla_{adiabatic} < \nabla_{stable}$ condition is satisfied or not.
    - If satisfied: Takes *THE Temperature Gradient as Convective Temperature Gradient i.e.* $\nabla = \nabla^*$ and calculates *Radiative Flux, Convective Flux, Change in Temperature with respect to the Mass Coordinate (uses "if* $\nabla_{adiabatic} < \nabla_{stable}$*" equation)*
    - If not satisfied: Takes *THE Temperature Gradient as Stable Temperature Gradient i.e.* $\nabla = \nabla_{stable}$ and takes *Radiative Flux as Total Flux (since the Mass Shell is Convectively Stable, Convective Flux is zero),* then calculates *Change in Temperature with respect to the Mass Coordinate* by using `temperature_change` function that uses the *"if otherwise" equation.*

- 13-) Checks if the Adaptive Timestepping is wanted by the user or not.
- 14-) If wanted, does Adaptive Timestepping by first *taking the absolute values of the changes in Radius, Pressure, Luminosity and Temperature*, then taking the *current values of Radius, Pressure, Luminosity and Temperature*, *divides them,* then *multiplies it with a percentage p,* then takes the *smallest difference we get from the array we obtained as the next timestep* ***which is dm!*** *(Since we move from Mass Shell to Mass Shell)*
- 15-) Updates the *Radius, Pressure, Luminosity, Temperature, Mass, Density and Total Energy* values by subtracting the change and multiplying with the timestep *dm*.
- 16-) If the Integration is finished, checks the elapsed time, and prints it.
- 17-) Returns the arrays containing the values of *Radius, Pressure, Luminosity, Temperature, Density, Total Energy, Mass, THE Temperature Gradient, Convective Temperature Gradient, Stable Temperature Gradient, Adiabatic Temperature Gradient, Convective Flux, Radiative Flux, Total Flux* for each Iteration/Integration step we have taken.

- **Then we define function `plot_parameters` to plot our results which are the following:**
  - 1-) Mass vs. Radius
  - 2-) Radius vs. Integration Step
  - 3-) Density vs. Radius (log-scaled)
  - 4-) Pressure vs. Radius (log-scaled)
  - 5-) Temperature vs. Radius
  - 6-) Luminosity vs. Radius
  - 7-) Temperature Gradient Comparison (Convective vs. Stable vs. Adiabatic)

- Note that all of the **plots above are normalized by their Initial Values!**
- Finally, we define **`plot_cross_section`** function to plot the Cross-Section of our Modelled Star.
- **It plots the zones of the Stars by the following conditions:**
  - 1-) **If Luminosity is larger than $0.995\ L_{\odot}$:** We are at the Outside of the Core.
    - **If Convective Flux is larger than zero:** The Zone is Convective.
    - **If Convective Flux is zero:** The Zone is Radiative.

  - 2-) **If Luminosity is smaller than $0.995\ L_{\odot}$:** We are at the Inside of the Core.
    - **If Convective Flux is larger than zero:** The Zone is Convective.
    - **If Convective Flux is zero:** The Zone is Radiative.

- We are done with defining our `Star_model()` class.
- We then define an Object called `star` that calls our class.
- We do the Sanity checks for Opacity and Total Energy values by using the values on the Tables given in the guide for the project.
- We then use `ODE_solver` to get the Parameter Values per Integration Step.
- We do the plotting by using `plot_parameters` function and then plot the cross section of the star with zones by using `plot_cross_section` function.

Code Results & Discussion

- Now let's talk about the plots we obtain from our code.
- We start with the *Mass vs. Radius, Density vs. Radius, Pressure vs. Radius, Temperature vs. Radius and Luminosity vs. Radius* plots.

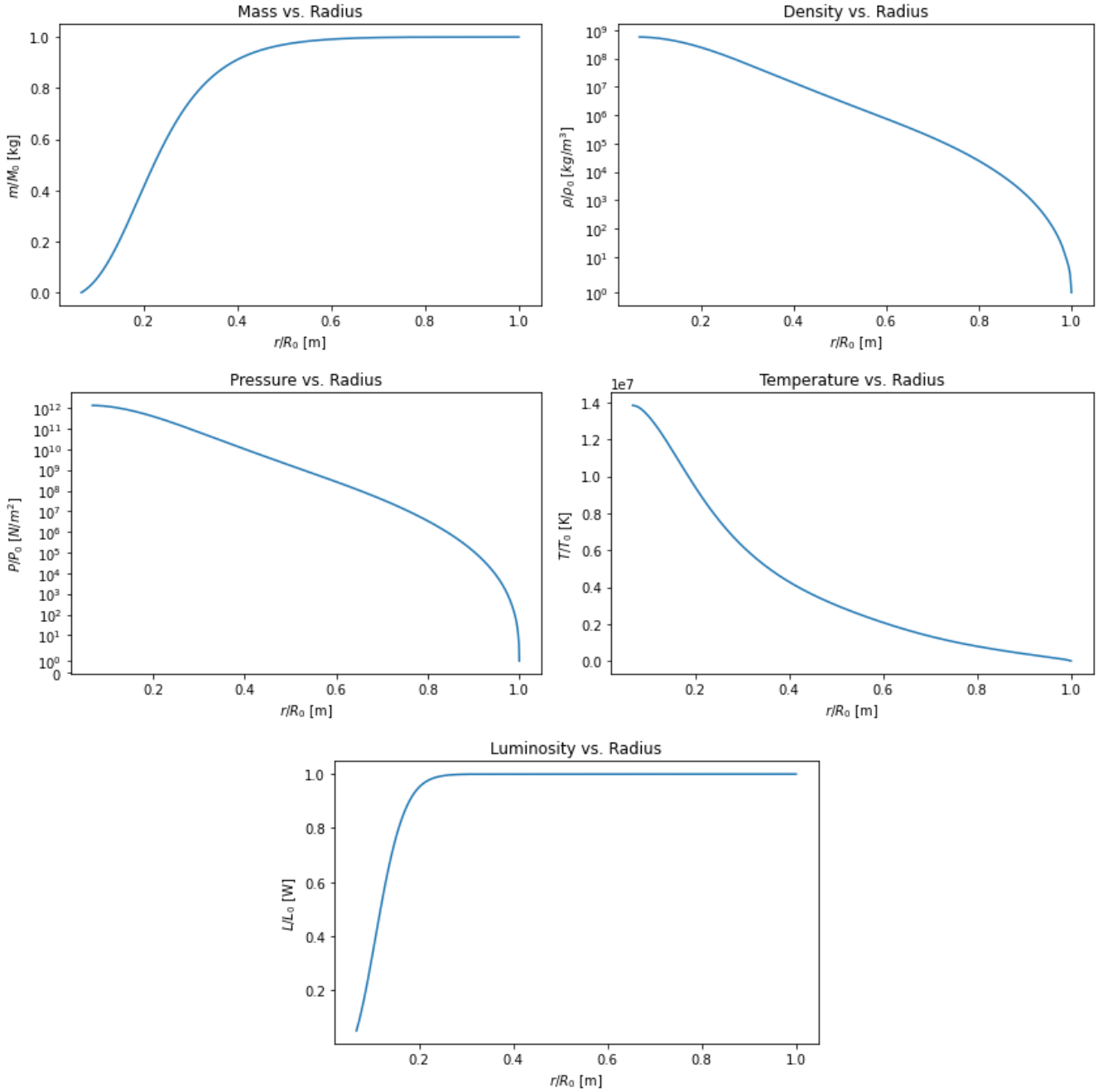

Figure 1: Plots showing *Mass vs. Radius, Density vs. Radius (log-scaled), Pressure vs. Radius (log-scaled), Temperature vs. Radius and Luminosity vs. Radius*

- Note that in these figures the left side is **the core,** and the right side is **the surface.**
- **Mass vs. Radius:** As expected, Mass of the "Mass Shell" goes up as we go to the surface.
- **Density vs. Radius:** As expected, Density goes down as we go to the surface. ($\boldsymbol{\rho \propto M/R^3}$)
- **Pressure vs. Radius:** As expected, Pressure goes down as we go to the surface. ($\boldsymbol{P \propto T^4 + \rho T}$)
- **Temperature vs. Radius:** As expected, Temperature goes down as we go to the surface since the core is significantly hotter than the Surface.
- **Luminosity vs. Radius:** As expected, Luminosity goes up as we go to the surface. ($\boldsymbol{L \propto R^2}$)

- Now, let's look at the *Radius vs. Integration Step* plot.

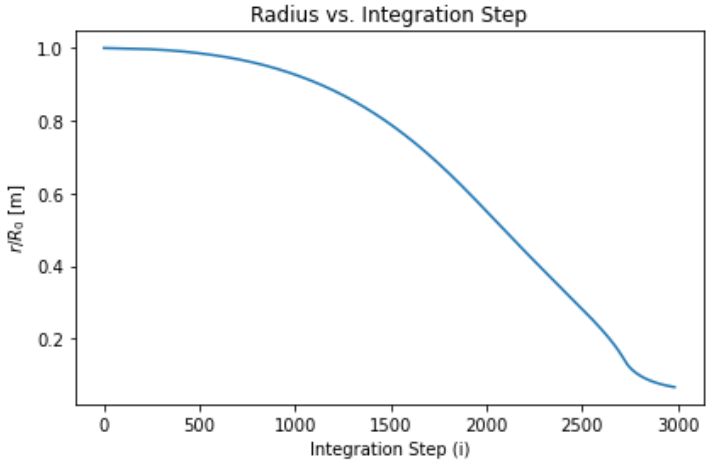

Figure 2: Plot showing *Radius vs. Integration Step*

- Compared to other plots, this is reversed!
- Now the left side is **the surface,** and the right side is **the core!**
- We see that our Integration is from **the surface to the core!** (Since as integration step goes up, we go to the core…)

- Now let's check the Temperature Gradient Comparison plot.

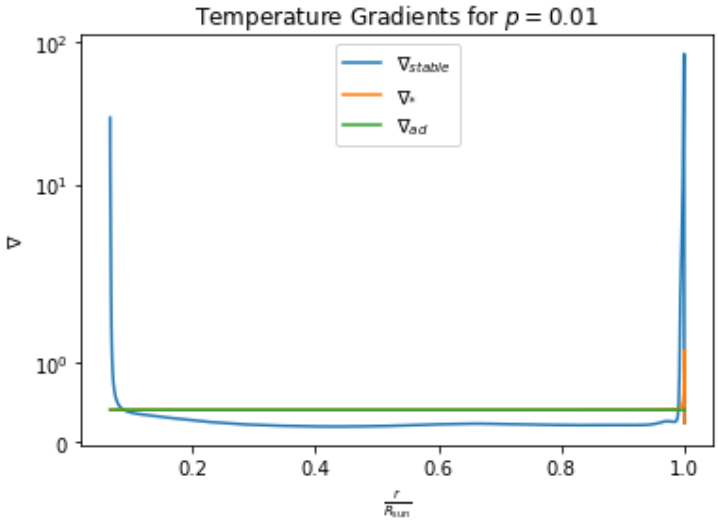

Figure 3: Plot comparing *Stable, Convective and Adiabatic Temperature Gradients*
(This plot matches the sanity check given in the guide for the project!)

- In this plot, again the left side is **the core,** and the right side is **the surface.**
- We see that the **Stable Temperature Gradient goes up** on the Core and the Surface, which is expected since the most outer layer and most inner layer are convective! ($\nabla_{adiabatic} < \nabla_{stable}$ satisfied i.e. Convective Zone!)

- Finally, we check the Cross-Section of our Star Model which is given on the next page.

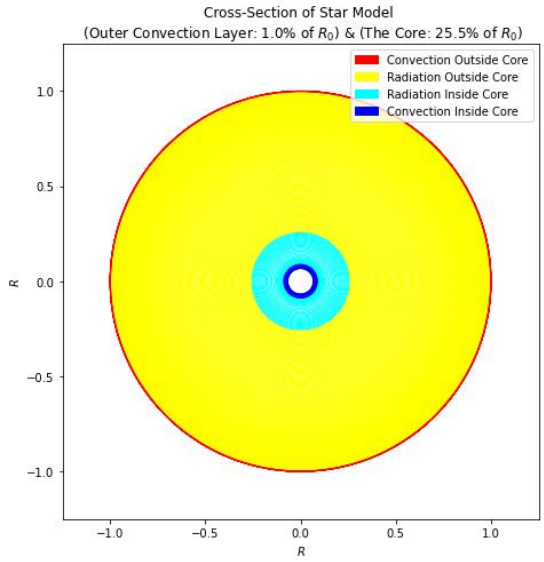

Figure 4: *The Cross-Section of our Star Model*
(This plot matches the sanity check given in the guide for the project!)

- This is the plot we obtain **by using the Initial Values as the Sun's**.
- But for the Goals given in the guide to the project are different than what we have right now.
- For that reason, the code has been used to test different Initial Values.
- <u>As a result, the following Initial Values were found to be best fit:</u>
  - $\boldsymbol{L_{initial} = 1.1\, L_{\odot}}$
  - $\boldsymbol{R_{initial} = 0.95\, R_{\odot}}$
  - $\boldsymbol{M_{initial} = 1.0\, M_{\odot}}$
  - $\boldsymbol{T_{initial} = 0.7\, T_{\odot}}$
  - $\boldsymbol{\rho_{initial} = 49\, \rho_{\odot}}$
  - *Note that there could be better values as the best fit, but these are the values I've found by my own testing.*

- Using these Initial values, we get the following plot.

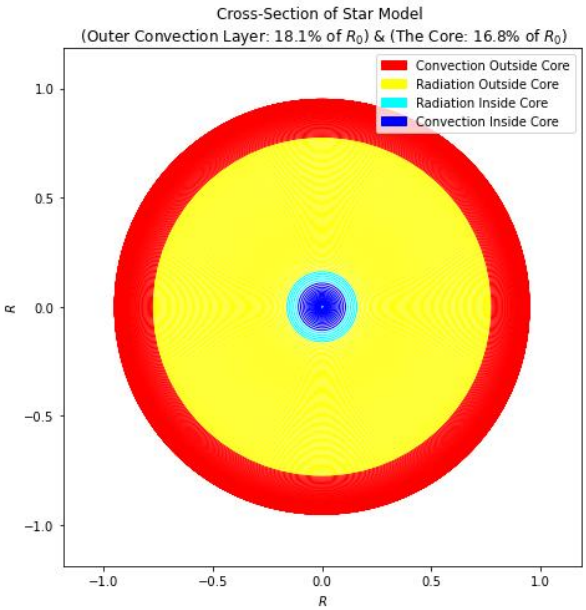

Figure 5: *The Cross-Section of our "Best-Fit" Star Model*

- **<u>In here we satisfy the goals given in the guide to the project:</u>**
  - 1-) Luminosity, Mass and Radius goes to zero within %5 of their Initial Values! (*To see this, check the Code's Output given below!*)
  - 2-) Core reaches to at least %10 of the Initial Radius. (*Shown on the plot title!*)
  - 3-) Has a continuous Convection Zone near surface of the star with a width of at least %15 of Initial Radius. (*Shown on the plot title!*)

- The code's output for the "Best-Fit" Star Model is given below.

```
                        Progress: 99.6677 %
------------------------------------------------------------------------
            Parameter other than Mass has reached zero!
------------------------------------------------------------------------
                             Step: 3767
------------------------------------------------------------------------
                  Mass: 6.607240774473459e+27
                Equal to: 0.33% of Initial Mass
------------------------------------------------------------------------
                  Radius: 53964.06441383071
                Equal to: 0.01% of Initial Radius
------------------------------------------------------------------------
               Luminosity: -1.4434827130870977e+23
             Equal to: -0.03% of Initial Luminosity
------------------------------------------------------------------------
                  Density: 40094044.6390479
------------------------------------------------------------------------
                  Pressure: 8.4461457232516e+19
------------------------------------------------------------------------
                 Temperature: 155881108.55728325
------------------------------------------------------------------------
                    Modelling is Finished.
                 Time elapsed: 38.29 seconds.
```

(Note: Luminosity is minus since we take the last value of the Array to print out the results, and the last value is already below zero hence the negative value.)

References

[1] Modelling Stellar Energy Transport, Computational Astrophysics, UvA, 2023/2024

[2] Variable Steplength by Boris Vilhelm Gudiksen
https://www.uio.no/studier/emner/matnat/astro/AST3310/v24/projects/project2/variablesteplength.pdf

[3] cross_section.py by Boris Vilhelm Gudiksen
https://www.uio.no/studier/emner/matnat/astro/AST3310/v24/projects/project2/cross_section.py