Final Project Write Up: C++ Code to Model the Interior Structure of an Exoplanet

Jonah Walker, 400316001

DATASCI2G03

Dec 5, 2024

Introduction

Exoplanet research has been an extremely important part of modern astrophysics, having discovered over 5,000 planets orbiting in extrasolar systems. Exoplanets are extremely difficult to study because of their proximity to their host star. Being so close to the star, even the most ingenious methods can extract only a small amount of information regarding the planet. When it comes to studying the processes on the planet itself methods are slim, with transit spectroscopy giving a snapshot of the atmospheric gases, almost no information is available from the internal structure of a planet. In the search for extraterrestrial life, modeling the internal structure could be paramount given how life on earth is believed to have originated from hydrothermal vents at the ocean floor. The model presented in this report as my final project aims to aid researchers understand the internal structure of exoplanets using equations and values from the foundational paper Seager et al. (2007).

Methods

The internal interactions of planets are modeled by system of differential equations (Equations 1 and 2) and an equation of state (Equation 3). Equations 1, 2 and 3 will henceforth be referred to as the Key Equations. Equation one solves for the mass M contained in radius r, which is a function of density ρ which also has radius dependency. Equation two is the equation for hydrostatic equilibrium for a planet, where the pressure force outwards should be balanced by the gravitational force inwards. Equation 3 is the equation of state, more specifically a modified polytropic equation of state, and was derived from section 5 of Seager et al. (2007). This is a general version of the density-pressure relationship. More advanced approaches might use a different equation of state, such as the Vinet or BME, and might also incorporate temperature (Seager et al. 2007). However, this simplified version produced adequate values when testing with earth-analogue figures, so I decided it was appropriate for our purposes.

(1) $\frac{dm(r)}{dr} = 4\pi r^2 \rho(r)$

(2) $\frac{dP(r)}{dr} = \frac{-Gm(r)\rho(r)}{r^2}$

(3) $\rho(P) = \rho_0 + cP^n$

ρ_0, c and n are all properties of certain materials, as discussed in Table 3 of Seager et al. (2007). In my model, I assumed an iron core with values consistent with the first row. I assumed a perovskite mantle with values consistent with the second row. Finally, I assumed an SiC crust with values consistent with the third row.

Material	ρ_0 [kg m^{-3}]	c [kg m^{-3} Pa^{-n}]	n
Fe(α)	8300.00	0.00349	0.528
$MgSiO_3$ (perovskite)	4100.00	0.00161	0.541
(Mg,Fe)SiO_3	4260.00	0.00127	0.549
H_2O	1460.00	0.00311	0.513
C (graphite)	2250.00	0.00350	0.514
SiC	3220.00	0.00172	0.537

TABLE 3

FITS TO THE MERGED VINET/BME AND TFD EOS OF THE FORM $\rho(P) = \rho_0 + cP^n$. THESE FITS ARE VALID FOR THE PRESSURE RANGE $P < 10^{16}$ PA.

The boundary conditions in this code define how the planet is modeled in the plots. At a high level, the program integrates the equations of state (EOS) from the planet's center outward, calculating key properties like radius, mass, pressure, and density at each step. These boundary conditions are initialized in the integrate_and_plot function in planet_visualization.cpp. At the planet's center, the radius (r) and mass (M) are set to zero, while pressure (P) is at its maximum, requiring an iterative method to determine its value (discussed in the following paragraph). The integrate_and_plot function are the core of the program, performing numerical integration using user-specified step sizes for radius.

The function calls the density function, which serves two key purposes:

1. It determines the appropriate values for ρ_0, c, and n in the polytropic EOS (Equation 3) based on the current pressure and its corresponding planetary layer (core, mantle, or crust).
2. It outputs -1 when pressure reaches zero, signaling that the surface of the planet has been reached, at which point the integration loop stops. The function will also terminate if the integration reaches a user-specified maximum radius.

Estimating the central pressure of the planet required the use of the bisection iterative method, as the central pressure is not directly observable from exoplanet surveys. Instead, planetary radius and mass, properties observed in exoplanet transit studies, were used to constrain this estimate. The bisection method iteratively narrows the range of possible solutions for central pressure by halving the interval between bounds. This method is implemented in central_pressure_est.cpp. The algorithm starts with upper and lower bounds for central pressure and calculates the corresponding radius and mass for the midpoint. If the computed radius and mass are too high, the midpoint is too high, and the program adjusts it as the new upper bound. Conversely, if the computed values are too low, the midpoint becomes the new lower bound. This process repeats until the difference between the bounds is within a tolerance of 100 Pa. The user is then prompted to approve the derived central pressure before it is passed as an input to planet_visualization.cpp.

The Fourth-order Runge-Kutta method is employed in the integrate_and_plot function to solve the differential equations governing planetary structure. This method calculates rates of change for mass and pressure at four different points within each step, computes the corresponding midpoints for radius, mass, pressure, and density, and combines these values into a weighted average. Runge-Kutta offers high precision without requiring extremely small step sizes, making it efficient for modeling planetary interiors across thousands of kilometers. This method ensures the accuracy and computational feasibility of the model.

For my project's expansion, I implemented the central pressure parameter, which serves as an analogue for the core/mantle pressure ratio. This reflects the proportion of the planet's pressure attributed to the core relative to the total central pressure. Specifically, the core fraction of central pressure determines the point at which the pressure transitions from the core's EOS to the mantle's EOS. For example, a core fraction of 40% means we switch from a core-density regime to a mantle-density regime when central pressure has decreased to 40% of the initial central pressure. By increasing this value, you are promoting the influence of the mantle on the overall model of the planet, and by decreasing it, you are promoting the influence of the core on the model of the planet. If you wanted to model a mantle-only planet, you would set this value to 100%. By varying this input, the model can simulate planets with different core/mantle ratios, offering flexibility for exploring a variety of planetary compositions. This parameterization enables the user to study how variations in core size and pressure distribution impact the overall structure and internal dynamics of the planet.

For Earth-like planets, a core fraction of approximately 40% aligns with the observed size and density of Earth's core relative to its mantle. This value corresponds to the core-mantle transition when defined by radius rather than pressure. Using a hard-coded core radius of 3470 km, the pressure at the boundary was approximately 130 GPa. Assuming Earth's central pressure is around 360 GPa, this transition represents roughly 40% of the total pressure. This is shown in the following output in an early version of the code, where the boundary occurred just after the pressure dipped under 130 GPa.

```
[walkej37@phys-ugrad proj_final]$ planet_model
Enter the central pressure (GPa) (P Earth ~360 GPa): 360
Enter the step size (km): 100
Enter the maximum radius to integrate (km) (R Earth ~6378 km): 7000
r (km)          M (Earth masses)        P (GPa)         rho (kg/m^3)
Core 100                8.91413e-06             359.718         12711.2
Core 200                7.1302e-05              359.037         12709.3
Core 300                0.000240564             357.908         12704.9
Core 400                0.000569952             356.33          12697.6
Core 500                0.0011125               354.305         12687.4
Core 600                0.00192093              351.835         12674.2
Core 700                0.00304762              348.922         12658.1
Core 800                0.00454447              345.569         12639
Core 900                0.00646288              341.781         12616.9
Core 1000               0.00885363              337.562         12591.8
Core 1100               0.0117668               332.915         12563.8
Core 1200               0.0152518               327.848         12532.7
Core 1300               0.0193572               322.364         12498.6
Core 1400               0.0241303               316.472         12461.3
Core 1500               0.029618                310.176         12421
Core 1600               0.0358655               303.486         12377.5
Core 1700               0.0429172               296.407         12330.8
Core 1800               0.0508161               288.95          12280.9
Core 1900               0.0596037               281.122         12227.7
Core 2000               0.0693202               272.933         12171.2
Core 2100               0.080004                264.394         12111.2
Core 2200               0.0916918               255.514         12047.8
Core 2300               0.104419                246.304         11980.8
Core 2400               0.118217                236.776         11910.1
Core 2500               0.133119                226.943         11835.7
Core 2600               0.149152                216.818         11757.4
Core 2700               0.166342                206.413         11675.1
Core 2800               0.184712                195.743         11588.6
Core 2900               0.204283                184.824         11497.7
Core 3000               0.225073                173.669         11402.2
Core 3100               0.247094                162.298         11301.9
Core 3200               0.270358                150.726         11196.5
Core 3300               0.294868                138.972         11085.5
Core 3400               0.318314                128.107         10968.6
Core 3500               0.331181                122.522         5144.26
Mantle 3600             0.344766                117.046         5130.06
Mantle 3700             0.359089                111.668         5115.82
Mantle 3800             0.374164                106.376         5101.5
Mantle 3900             0.390008                101.157         5087.07
Mantle 4000             0.406638                96.0039         5072.49
Mantle 4100             0.424069                90.9075         5057.72
Mantle 4200             0.442317                85.8604         5042.72
Mantle 4300             0.461396                80.8561         5027.44
Mantle 4400             0.481321                75.8887         5011.83
Mantle 4500             0.502104                70.9531         4995.86
```

Results

The program estimated a value for Earth's central pressure with these inputs:

- Radius: 6300 km
- Mass: 1 Earth mass
- Outputted central pressure: 363.229 GPa

The program successfully modeled an Earth-like planet with the following inputs:

- Central pressure: 363.229 GPa
- Core fraction: 0.4

Output Summary, corroborated by Hales and Roberts (1970):

- Derived Radius: Just over 6300 km, aligning closely with Earth's observed radius.
- Core Radius: Approximately 3400 km, consistent with Earth's core dimensions.
- Mantle Density: Averaged around 5000 kg/m^3, a realistic estimate for silicate materials.
- Core Density: In the range of 12,000 to 10,000 Kg/m^3, a realistic estimate for Earth's core

These results closely match the expected values for an Earth analogue, validating the accuracy of the implemented methods. They are also comparable to results from Figure 6 of Seager et al. (2007) in terms of the shape of the density vs. radius plot, which further corroborates the reliability of the polytropic equation of state used in this model.

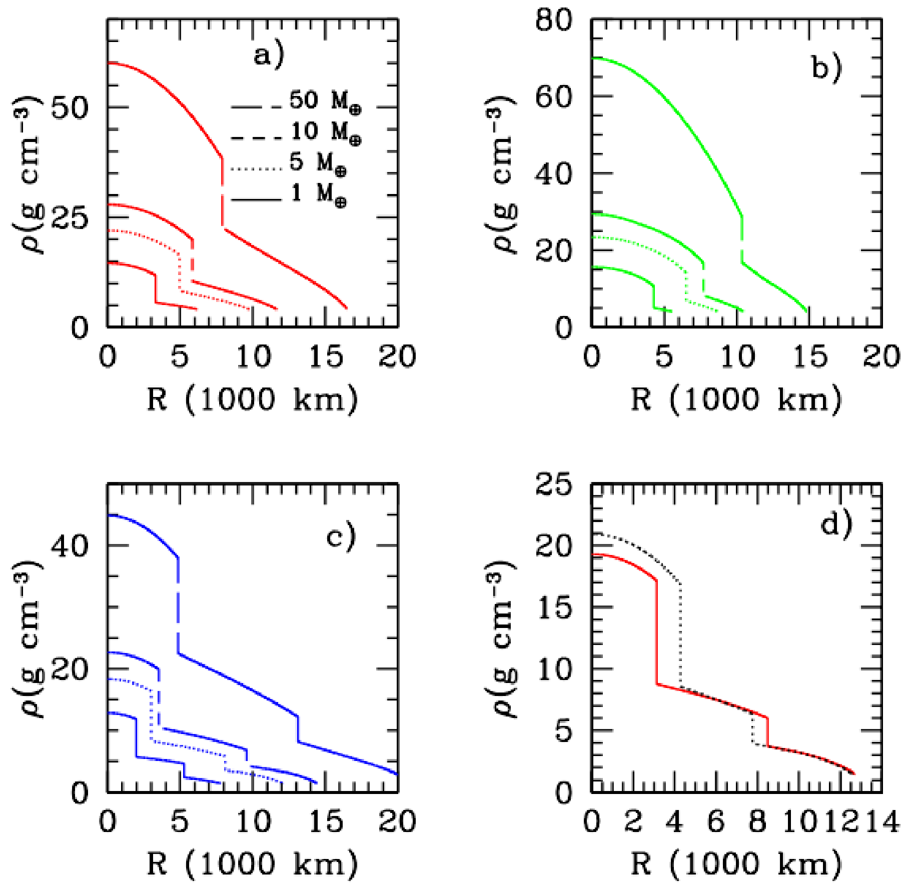


FIG. 6.— Interior structure of solid exoplanets. From top to bottom the curves in panels a, b, and c are for planets with $M_p = 50$, 10, 5, and 1 $M_\oplus$ respectively. Panel a: silicate planets with a 32.5% by mass Fe core and a 67.5% $MgSiO_3$ mantle. Panel b: as in panel a but for planets with a 70% Fe core and 30% silicate mantle. Panel c: interior structure for water planets with 6.5% Fe core, 48.5% $MgSiO_3$ shell, and 45% outer water ice layer. Panel d: interior model for two different water exoplanets with the same planet mass and radius: $M_p = 6.0 M_\oplus$ and $R_p = 2.0 M_\oplus$. The solid curve is for a model with layers in percentages by mass of Fe/$MgSiO_3$/H_2O of: 17/33/50 (similar to the composition of the water planet in (Léger et al. 2004)) and the dotted line for 6.5/48.5/45 (similar to the composition of Ganymede).

Output Informing Results

[walkej37@phys-ugrad proj_final]$ final_project

We will begin by calculating the central pressure of the planet using the bisection method.

Please provide the total radius, total mass, and core fraction to perform this calculation.

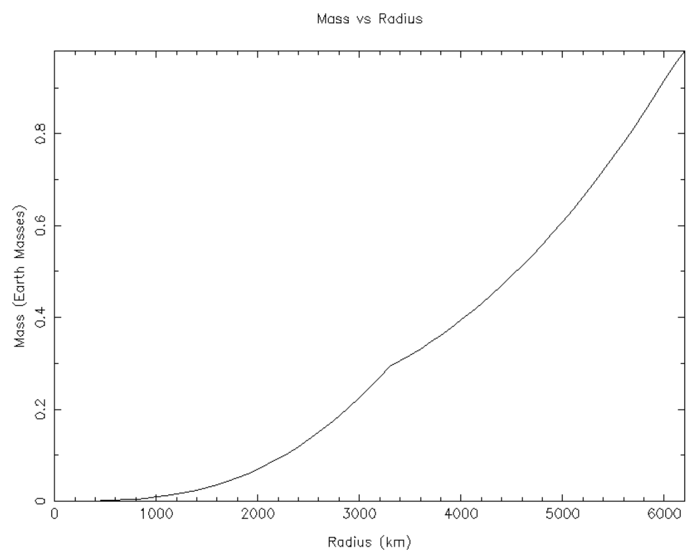
Enter the total radius of the planet (km): 6300

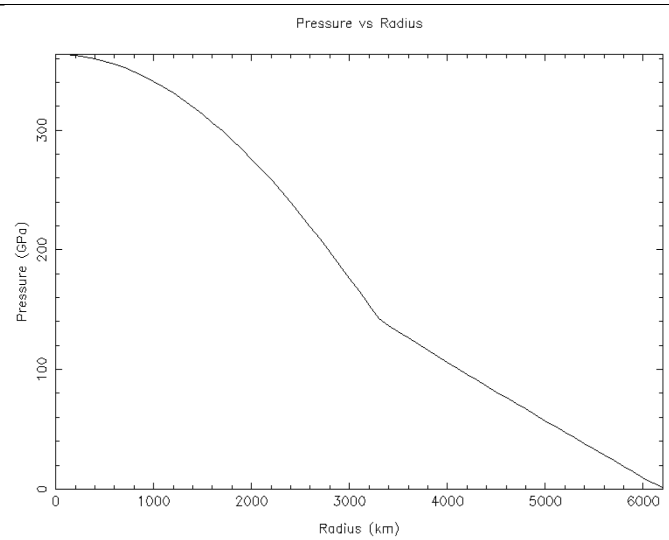
Enter the total mass of the planet (Earth masses): 1

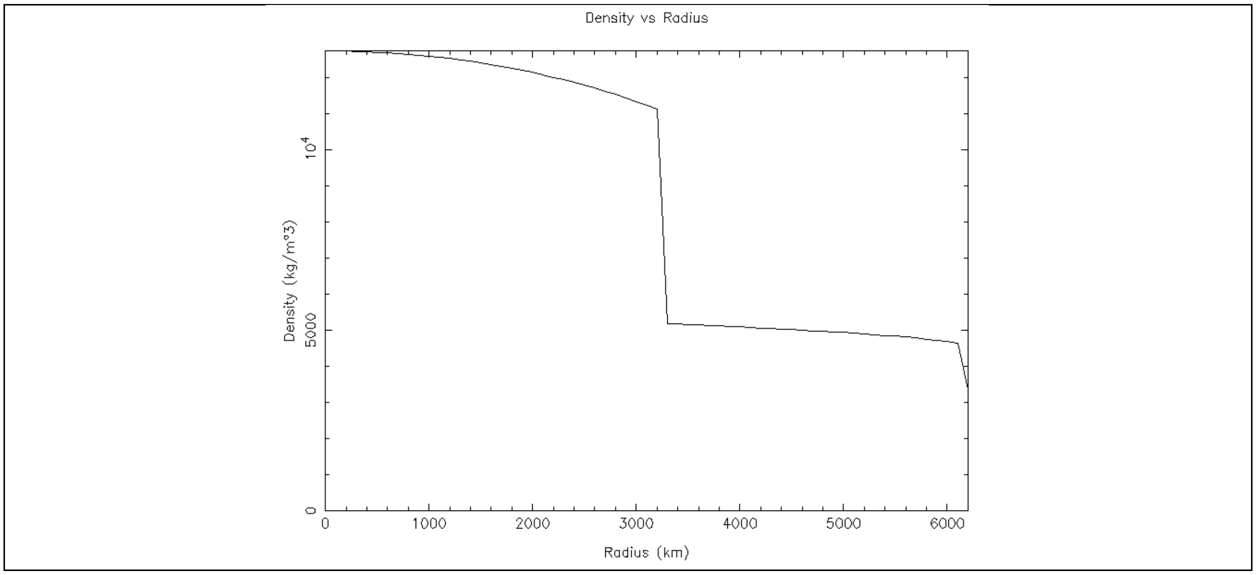
Enter the core fraction of central pressure (e.g., 0.4 for 40%): 0.4

Derived central pressure: 363.229 GPa.

Do you approve this central pressure? (y/n): y







Conclusion

This project demonstrated the feasibility of modeling planetary interiors using numerical methods and simplified equations of state. By incorporating user-defined parameters such as core fraction and central pressure, the program offers flexibility for simulating a wide range of terrestrial planets. The model successfully recreated Earth-like properties, making it a useful tool for exploring exoplanetary structures in future studies.

One limitation of the current model is its reliance on a simplified equation of state. While the modified polytropic EOS provided reasonable results for Earth-like planets, more complex EOS models (e.g., Vinet or Birch-Murnaghan) could improve accuracy for planets with significantly different compositions or internal pressures.

For further improvement, the model could be extended to include temperature-dependent equations of state and to simulate the effects of planetary evolution, such as mass loss or core crystallization, on internal structure.

References

Hales, A. L., and J. L. Roberts. 1970. "Shear Velocities in the Lower Mantle and the Radius of the Core." *Bulletin of the Seismological Society of America* 60 (5): 1427–36. https://doi.org/10.1785/BSSA0600051427.

Seager, S., M. Kuchner, C. A. Hier-Majumder, and B. Militzer. 2007. "Mass-Radius Relationships for Solid Exoplanets." *The Astrophysical Journal* 669 (2): 1279. https://doi.org/10.1086/521346.

Code Screenshots

Open | planet_visualization.cpp /1/home/walkej37/proj_final | Save

central_pressure_est.cpp | makefile | planet_visualization.cpp | final_project_header.h

```cpp
#include <iostream>
#include <cmath>
#include "cpgplot.h"
#include "final_project_header.h"

// Values for the polytropic equation of state based on material in each layer
const float rho_core = 8300.00; // Values for Fe (core)
const float c_core = 0.00349;
const float n_core = 0.528;

const float rho_mantleL = 4500.00; // Values for MgSiO3 (mantle)
const float c_mantleL = 0.0018;
const float n_mantleL = 0.5;

const float rho_crust = 3220.00; // Values for SiC (crust)
const float c_crust = 0.00172;
const float n_crust = 0.537;

// Equation of State
float density(float P, float P_core_threshold, float P_crust_threshold) {
    float rho_0, c, n;

    if (P > P_core_threshold) {  // Core
        rho_0 = rho_core;
        c = c_core;
        n = n_core;
    } else if (P > P_crust_threshold) {  // Mantle
        rho_0 = rho_mantleL;
        c = c_mantleL;
        n = n_mantleL;
    } else if (P > 0) {  // Crust
        rho_0 = rho_crust;
        c = c_crust;
        n = n_crust;
    } else {
        std::cerr << "Pressure = 0. Surface reached.\n";
        return -1;  // integrate_and_plot loop termination
    }

    return rho_0 + c * pow(P, n);
}
```

```cpp
void integrate_and_plot(float P_c, float dr, float r_max, float P_core_frac) {
    // Initial conditions
    float r = 1e-6; // Initial radius (small but not 0) in meters
    float M = 0.0; // Initial mass in kg
    float P = P_c; // Initial central pressure in Pa

    // Threshold pressures
    float P_core_threshold = P_c * P_core_frac; // Core-mantle boundary
    float P_crust_threshold = P_c * 0.01; // Crust threshold

    // Arrays for storing values
    const int MAX_POINTS = 10000;
    float r_values[MAX_POINTS], M_values[MAX_POINTS], P_values[MAX_POINTS], rho_values[MAX_POINTS];
    int point_count = 0;

    // Integration loop
    while (r < r_max && P > 0 && point_count < MAX_POINTS) {
        float rho = density(P, P_core_threshold, P_crust_threshold);

        if (rho == -1) break; // Stop the loop if surface is reached

        // Store values
        r_values[point_count] = r / 1000.0; // Convert to km
        M_values[point_count] = M / EARTH_MASS; // Convert to Earth masses
        P_values[point_count] = P / GPa_to_Pa; // Convert to GPa
        rho_values[point_count] = rho;
        point_count++;

        // Runge-Kutta (RK4) for integration
        float k1_M = 4.0 * M_PI * r * r * rho;
        float k1_P = -G * M * rho / (r * r);

        float r_k2 = r + 0.5 * dr;
        float M_k2 = M + 0.5 * dr * k1_M;
        float P_k2 = P + 0.5 * dr * k1_P;
        float rho_k2 = density(P_k2, P_core_threshold, P_crust_threshold);
        float k2_M = 4.0 * M_PI * r_k2 * r_k2 * rho_k2;
        float k2_P = -G * M_k2 * rho_k2 / (r_k2 * r_k2);

        float r_k3 = r + 0.5 * dr;
        float M_k3 = M + 0.5 * dr * k2_M;
        float P_k3 = P + 0.5 * dr * k2_P;
        float rho_k3 = density(P_k3, P_core_threshold, P_crust_threshold);
        float k3_M = 4.0 * M_PI * r_k3 * r_k3 * rho_k3;
        float k3_P = -G * M_k3 * rho_k3 / (r_k3 * r_k3);
```

```cpp
        float r_k4 = r + dr;
        float M_k4 = M + dr * k3_M;
        float P_k4 = P + dr * k3_P;
        float rho_k4 = density(P_k4, P_core_threshold, P_crust_threshold);
        float k4_M = 4.0 * M_PI * r_k4 * r_k4 * rho_k4;
        float k4_P = -G * M_k4 * rho_k4 / (r_k4 * r_k4);

        M += (k1_M + 2 * k2_M + 2 * k3_M + k4_M) * dr / 6.0;
        P += (k1_P + 2 * k2_P + 2 * k3_P + k4_P) * dr / 6.0;
        r += dr;
    }

    // PGPLOT Visualization
    if (cpgopen("/xwindow") > 0) {
        cpgscr(0, 1.0, 1.0, 1.0);
        cpgscr(1, 0.0, 0.0, 0.0);

        // Plot Mass vs Radius
        cpgenv(0, r_values[point_count - 1], 0, M_values[point_count - 1], 0, 0);
        cpglab("Radius (km)", "Mass (Earth Masses)", "Mass vs Radius");
        cpgline(point_count, r_values, M_values);


        // Plot Pressure vs Radius
        cpgenv(0, r_values[point_count - 1], 0, P_values[0], 0, 0);
        cpglab("Radius (km)", "Pressure (GPa)", "Pressure vs Radius");
        cpgline(point_count, r_values, P_values);


        // Plot Density vs Radius
        cpgenv(0, r_values[point_count - 1], 0, rho_values[0], 0, 0);
        cpglab("Radius (km)", "Density (kg/m^3)", "Density vs Radius");
        cpgline(point_count, r_values, rho_values);

        cpgclos();
    } else {
        std::cerr << "PGPLOT failed to open.\n";
    }
}
```

```cpp
int main() {
    // Inform the user about the process
    std::cout << "We will begin by calculating the central pressure of the planet using observed values of the exoplanet through the bisection method.\n";
    std::cout << "Please provide the total radius, total mass, and core fraction to perform this calculation.\n";

    // User inputs for central pressure bisection calculation
    float total_radius, total_mass, P_core_frac;

    std::cout << "Enter the total radius of the planet (km) (Earth ~6300km): ";
    std::cin >> total_radius;
    total_radius *= 1000;  // Convert to meters

    std::cout << "Enter the total mass of the planet (Earth masses): ";
    std::cin >> total_mass;
    total_mass *= EARTH_MASS;  // Convert to kg

    std::cout << "Enter the core fraction of central pressure (A core fraction of .40 means we switch from a core-density regime to a mantle-density regime when central pressure has decreased to .40 of the initial central pressure. Essentially, an analogue for the core/mantle pressure ratio. In earth models, this valus is ~0.40): ";
    std::cin >> P_core_frac;

    // Tolerance for bisection method in Pa
    float tolerance = 1e6;
    // Derive central pressure using bisection method
    float P_c = bisection_central_pressure(total_radius, total_mass, tolerance, P_core_frac);

    // Display the derived central pressure and get user approval
    std::cout << "Derived central pressure: " << P_c / 1e9 << " GPa.\n";
    std::cout << "Do you approve this central pressure? (y/n): ";
    char approval;
    std::cin >> approval;

    if (approval != 'y' && approval != 'Y') {
        std::cout << "Program terminated.\n";
        return 0;  // Exit the program
    }

    // Proceed with step size and maximum radius inputs
    std::cout << "Enter the step size (km): ";
    float dr;
    std::cin >> dr;
    dr *= 1000;
```

```cpp
    std::cout << "Enter the maximum radius to integrate (km): ";
    float r_max;
    std::cin >> r_max;
    r_max *= 1000;

    // Throw over to integrate_and_plot function
    integrate_and_plot(P_c, dr, r_max, P_core_frac);

    return 0;
}
```

C++ Tab Width: 8 Ln 160, Col 41 INS

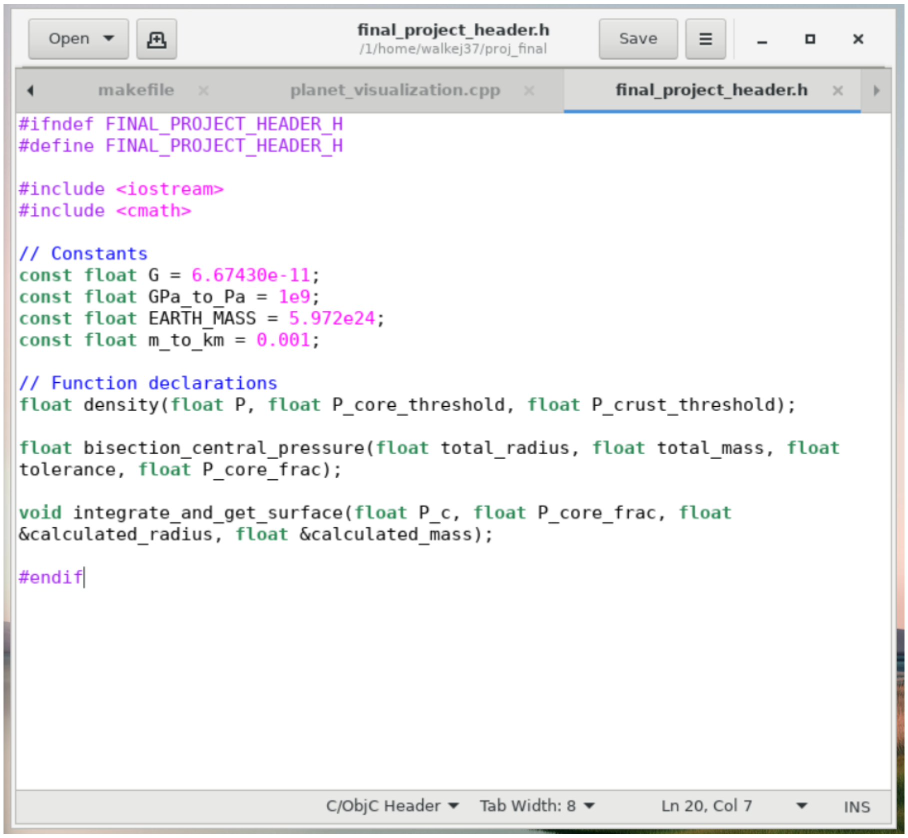
Open central_pressure_est.cpp /1/home/walkej37/proj_final Save

central_pressure_est.cpp makefile planet_visualization.cpp final_project_header.h

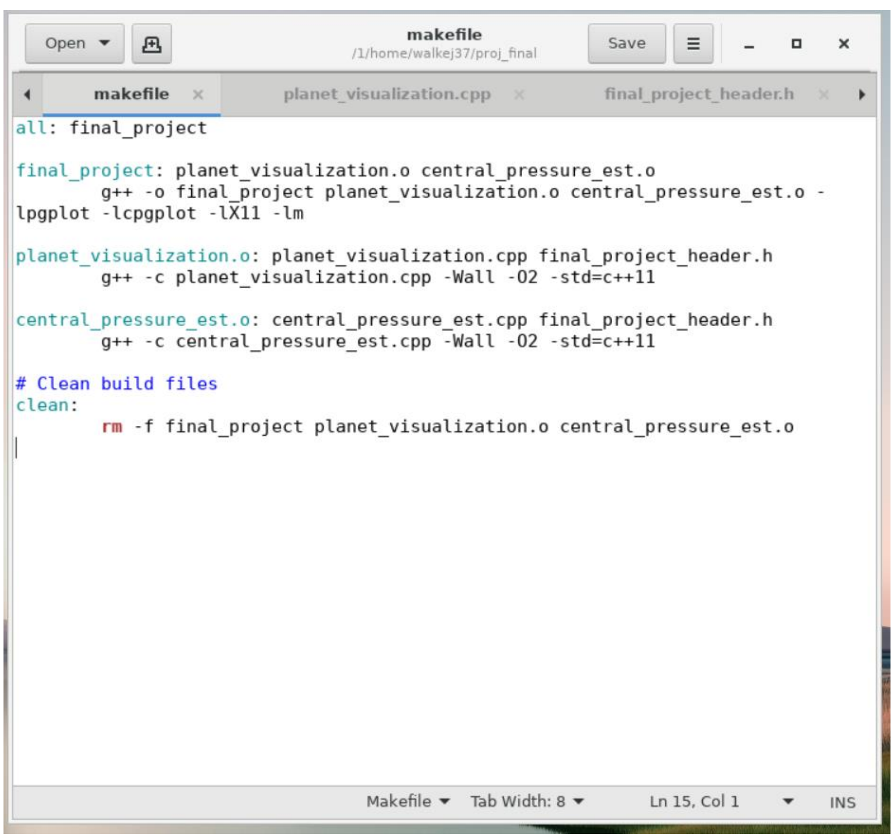
```cpp
#include <cmath>
#include "final_project_header.h"

float bisection_central_pressure(float total_radius, float total_mass, float tolerance, float
P_core_frac) {
    // Define bounds for central pressure (reasonable starting guesses)
    float P_low = 1e9; // Lower bound (Pa)
    float P_high = 1e12; // Upper bound (Pa)
    float P_mid = (P_low + P_high) / 2.0;

    // Variables to hold calculated radius and mass
    float calculated_radius, calculated_mass;

    // Perform bisection
    while ((P_high - P_low) > tolerance) {
        P_mid = (P_low + P_high) / 2.0;  // Midpoint

        // Integrate outward from the guessed central pressure
        integrate_and_get_surface(P_mid, P_core_frac, calculated_radius, calculated_mass);

        // Compare calculated values to target values
        if (calculated_radius > total_radius || calculated_mass > total_mass) {
            // Guessed pressure is too high
            P_high = P_mid;
        } else {
            // Guessed pressure is too low
            P_low = P_mid;
        }
    }

    // Return the midpoint as the estimated central pressure
    return P_mid;
}
```

```cpp
void integrate_and_get_surface(float P_c, float P_core_frac, float &calculated_radius, float
&calculated_mass) {
    // Initial conditions
    float r = 1e-6;
    float M = 0.0;
    float P = P_c;

    float P_core_threshold = P_c * P_core_frac; // Core-mantle boundary
    float P_crust_threshold = P_c * 0.01; // Crust threshold
    float dr = 1000;

    // Integrate outward until pressure reaches 0
    while (P > 0) {
        float rho = density(P, P_core_threshold, P_crust_threshold);
        float dMdr = 4.0 * M_PI * r * r * rho;
        float dPdr = -G * M * rho / (r * r);

        // Update values
        M += dMdr * dr;
        P += dPdr * dr;
        r += dr;
    }

    // Set the outputs
    calculated_radius = r; // Final radius where pressure reaches 0
    calculated_mass = M; // Total enclosed mass
}
```

C++ Tab Width: 8 Ln 1, Col 17 INS




```cpp
#ifndef FINAL_PROJECT_HEADER_H
#define FINAL_PROJECT_HEADER_H

#include <iostream>
#include <cmath>

// Constants
const float G = 6.67430e-11;
const float GPa_to_Pa = 1e9;
const float EARTH_MASS = 5.972e24;
const float m_to_km = 0.001;

// Function declarations
float density(float P, float P_core_threshold, float P_crust_threshold);

float bisection_central_pressure(float total_radius, float total_mass, float
tolerance, float P_core_frac);

void integrate_and_get_surface(float P_c, float P_core_frac, float
&calculated_radius, float &calculated_mass);

#endif
```




```makefile
all: final_project

final_project: planet_visualization.o central_pressure_est.o
	g++ -o final_project planet_visualization.o central_pressure_est.o -
lpgplot -lcpgplot -lX11 -lm

planet_visualization.o: planet_visualization.cpp final_project_header.h
	g++ -c planet_visualization.cpp -Wall -O2 -std=c++11

central_pressure_est.o: central_pressure_est.cpp final_project_header.h
	g++ -c central_pressure_est.cpp -Wall -O2 -std=c++11

# Clean build files
clean:
	rm -f final_project planet_visualization.o central_pressure_est.o
```