CO31: Structure of White Dwarf Stars

Guo-Zheng Theodore Yoong
University College, Oxford
univ4499

8 February 2019

Abstract

This computing report investigates the equations of state of white dwarf stars. The equations of state are mainly governed by two coupled first-order ordinary differential equations on $\mathrm{d}m/\mathrm{d}r$ and $\mathrm{d}\rho/\mathrm{d}r$. The equations of state of white dwarf stars for both non-relativistic and relativistic cases are numerically solved via the fourth-order Runge-Kutta method in Python. The results are also compared and discussed. The critical mass of white dwarf stars, i.e. the Chandrasekhar mass M_C, is also computed.

1 Introduction

In astrophysics, white dwarf stars are one possible evolutionary state at the end of a stellar evolution cycle. These celestial objects generally comprise heavy nuclei, mainly ^{56}Fe, although ^{12}C may dominate if the nucleosynthesis stops prematurely [1].

The compression forces on the star due to its own gravitational field balance the forces due to the electron degeneracy pressure. The equations of state thereby lead to two coupled first-order differential equations for both the non-relativistic and relativistic treatments of the star. We therefore perform the *fourth-order Runge-Kutta method* to solve our differential equations and obtain the physical parameters of the star, such as the star's mass and radius.

2 Physical Background

2.1 Stellar Evolution

The stellar birth of low-mass stars begins in protostellar clouds [3]. At this stage, the gravitational force of the dust cloud pulls the dust particles inwards while releasing gravitational potential energy in the form of heat. The temperature of the dust core increases until a critical temperature is reached, in which nuclear fusion of hydrogen atoms begins, which drives stellar evolution.

The dust cloud takes the form of a sphere and a protostar is formed. Further accretion of dust to the core increases the mass and size of the protostar and it becomes a main sequence star. Protostars with insufficient mass will form brown dwarf stars instead of main sequence stars. Different main sequence stars will experience nuclear burning at different rates, and hence different lifetimes. Once the hydrogen fusion at the core of stars is complete, the thermal pressure resisting compression will subside, leading to contraction of the core.

For a low-mass star ($M \lesssim 0.5M_\odot$), the temperature at its outer hydrogen shell cannot sustain fusion, and it thus gradually collapses into a white dwarf as it expels planetary nebulae. For a middle-mass star ($0.5M_\odot \lesssim M \lesssim 8M_\odot$), hydrogen at the outer core will fuse, generating thermal pressure to expand the star into a red giant with a helium core. Eventually, its core will collapse until the electron degeneracy pressure is sufficient to resist further compression. It will then radiate its remnant heat as a white dwarf.

For a high-mass star ($M \gtrsim 8M_\odot$), the hydrogen outer shell fuses quickly. Once exhausted, the core must be supported by electron degeneracy pressure alone. Eventually, the contraction stops as the density becomes too large, causing a supernovae explosion. The remnant core is a neutron star, which can collapse further into a black hole if the mass of the neutron star $M \gtrsim 3M_\odot$.

In this practical, we focus on white dwarf stars, i.e. the final stage of stars with $M \lesssim 8M_\odot$.

2.2 Equations of Equilibrium

In mechanical equilibrium, assuming spherical symmetry, the gravitational force on an element of the star is balanced by its interior pressure [1].

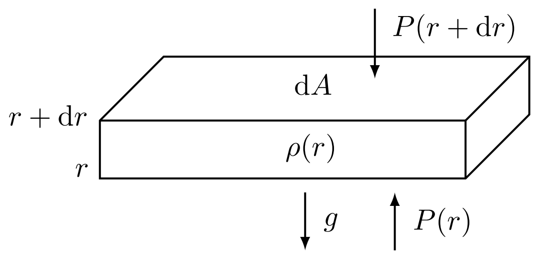

Figure 1: The balance of forces on an element of the star

If the area of element is $\mathrm{d}A$ and its density $\rho(r)$ is assumed to be uniform, then its mass $m(r)$ is $\rho\ \mathrm{d}A$ and its weight is $Gm(r)\rho(r)\ \mathrm{d}A\ \mathrm{d}r/r^2$. The difference in pressure between r and $r + \mathrm{d}r$ is equal to the weight per unit area of the element. We can write this as

$$\frac{\mathrm{d}P(r)}{\mathrm{d}r} = \frac{P(r + \mathrm{d}r) - P(r)}{\mathrm{d}r} = -\frac{G\rho(r)m(r)}{r^2}, \tag{1}$$

where G is the gravitational constant, $m(r)$ is the mass of star interior to r, and that the negative sign indicates the pressure increases as r decreases. Applying the chain rule, we rewrite equation (1) as

$$\frac{\mathrm{d}\rho}{\mathrm{d}r} = -\left(\frac{\mathrm{d}P}{\mathrm{d}\rho}\right)^{-1} \frac{G\rho(r)m(r)}{r^2}. \tag{2}$$

We can also express the derivative of $m(r)$ with respect to r as

$$\frac{\mathrm{d}m}{\mathrm{d}r} = 4\pi r^2 \rho(r). \tag{3}$$

The goal of this practical is to solve the two coupled first-order ordinary differential equations (2) and (3). However, $\mathrm{d}P/\mathrm{d}\rho$ will be different for relativistic and non-relativistic electrons. We explore the derivations and physical consequences of both regimes.

2.3 Equations of State

Assume that the star is made up of large ^{56}Fe nuclei, which contribute almost all the mass but no pressure, and their electrons, which contribute almost all the pressure but no mass. We therefore model the star as a sphere with a density much larger than normal matter, and assume that the electrons are unbound and act like a free Fermi gas. Using this model for the electron gas at 0 K [1], the number density of electrons can be expressed as

$$n = \int_0^{p_F} \frac{8\pi}{h^3} p^2 \, \mathrm{d}p = \frac{8\pi}{3h^3} p_F^3,$$

where h Planck's constant. The pressure is then given by

$$P = \frac{8\pi}{3h^3} \int_0^{p_F} p^3 v_p \, \mathrm{d}p \,, \tag{4}$$

where v_p is velocity of electron at a given momentum.

2.3.1 Non-relativistic Treatment

For $v_p \ll c$, we use $v_p = p/m_e$. The electron degeneracy pressure is then

$$P = \frac{8\pi}{15h^3 m_e} p_F^5 = \frac{1}{20} \left(\frac{3}{\pi}\right)^{\frac{2}{3}} \frac{h^2}{m_e} n^{\frac{5}{3}}. \tag{5}$$

Except for hydrogen (which is not present in white dwarves), each electron is accompanied by a proton and very nearly one neutron in a electrically neutral atom. The density in the electrically neutral atom is therefore $\rho = n_e(m_e + m_p + m_n)$. Assuming that $m_e \ll m_p, m_n$, and $m_p \approx m_n$, we have $n_e \sim \rho/2m_p$. Substituting our approximation into equation (5) for n_e, we obtain

$$P = \frac{1}{20} \left(\frac{3}{\pi}\right)^{\frac{2}{3}} \frac{h^2}{m_e} \left(\frac{\rho}{2m_p}\right)^{\frac{5}{3}},$$

and hence

$$\frac{\mathrm{d}P}{\mathrm{d}\rho} = \frac{1}{48} \frac{h^2}{m_e} \frac{2^{\frac{1}{3}}}{m_p^{\frac{5}{3}}} \left(\frac{3\rho}{\pi}\right)^{\frac{2}{3}}. \tag{6}$$

Substituting equation (6) into equation (2), we get

$$\frac{\mathrm{d}\rho}{\mathrm{d}r} = -k \frac{\rho^{\frac{1}{3}} m}{r^2}, \quad \text{where } k = \frac{48\pi^{\frac{2}{3}}}{2^{\frac{1}{3}} 3^{\frac{2}{3}}} \frac{G m_e m_p^{\frac{5}{3}}}{h^2}. \tag{7}$$

2.3.2 Relativistic Treatment

In this regime, we use $v_p = pc^2/\sqrt{p^2c^2 + m_e^2 c^4}$ instead. Substituting this expression into equation (4) yields

$$P = \frac{8\pi c^2}{3h^3} \int_0^{p_F} \frac{p^4}{\sqrt{p^2c^2 + m_e^2 c^4}} \, \mathrm{d}p \,. \tag{8}$$

Using the substitution $p = m_e c \sinh\theta$, we have $\mathrm{d}p = m_e c \cosh\theta \, \mathrm{d}\theta$. Equation (8) becomes

$$P = \frac{8\pi m_e^4 c^5}{3h^3} \int_0^{\theta_F} \sinh^4\theta \, \mathrm{d}\theta \,, \quad \text{where } \theta_F = \sinh^{-1} \frac{p_F}{m_e c}. \tag{9}$$

Substituting the relation $\sinh 4\theta = \dfrac{1}{8}\cosh 4\theta - \dfrac{1}{2}\cosh 2\theta + \dfrac{3}{8}$ into equation (9), we can easily perform the integral to obtain

$$P = \frac{8\pi m_e^4 c^5}{3h^3} \left(\frac{1}{32}\sinh 4\theta - \frac{1}{4}\sinh 2\theta + \frac{3}{8}\theta_F\right). \tag{10}$$

The chain rule for $\mathrm{d}P/\mathrm{d}\rho$ gives

$$\frac{\mathrm{d}P}{\mathrm{d}\rho} = \frac{\mathrm{d}P}{\mathrm{d}\theta_F}\frac{\mathrm{d}\theta_F}{\mathrm{d}p_F}\frac{\mathrm{d}p_F}{\mathrm{d}\rho}.$$

Noting that $p_F = (3h^3 n_e/8\pi)^{\frac{1}{3}}$ and using the same approximation of $n_e \sim \rho/2m_p$, we have

$$\begin{aligned}
\frac{\mathrm{d}P}{\mathrm{d}\theta_F} &= \frac{8\pi c}{3h^3}p_F^4,\\
\frac{\mathrm{d}\theta_F}{\mathrm{d}p_F} &= \frac{1}{\sqrt{p_F^2 + m_e^2 c^2}},\\
\frac{\mathrm{d}p_F}{\mathrm{d}\rho} &= \frac{1}{2^{\frac{4}{3}}3^{\frac{2}{3}}\pi^{\frac{1}{3}}}\frac{h\rho^{\frac{2}{3}}}{m_p^{\frac{1}{3}}}.
\end{aligned}$$

Putting these all together and substituting into equation (2), we obtain

$$\frac{\mathrm{d}\rho}{\mathrm{d}r} = -\frac{k\sqrt{a + b\rho^{\frac{2}{3}}}}{r^2}\rho^{\frac{1}{3}}m, \tag{11}$$

where

$$\begin{aligned}
k &= 2^{\frac{7}{3}}3^{\frac{1}{3}}\pi^{\frac{1}{3}}\frac{Gm_p^{\frac{4}{3}}}{h^2 c},\\
a &= 2^{\frac{8}{3}}\pi^{\frac{2}{3}}m_e^2 m_p^{\frac{2}{3}}c^2,\\
b &= 3^{\frac{2}{3}}h^2.
\end{aligned}$$

We have now obtained the forms of the coupled differential equations for both the non-relativistic and relativistic regimes. The Runge-Kutta method will be used to solve both cases, as outlined in the subsequent sections.

3 Runge-Kutta Method

3.1 Single Differential Equation

Consider the differential equation $\mathrm{d}y/\mathrm{d}t = f(y,t)$. Taking the integral with respect to t from y_n to y_{n+1}, we obtain

$$\int_{y_n}^{y_{n+1}} \frac{\mathrm{d}y}{\mathrm{d}t}\,\mathrm{d}t = \int_{y_n}^{y_{n+1}} f(y,t)\,\mathrm{d}t\,. \tag{12}$$

We can convert (12) to a second-order Runge-Kutta method via Simpson's method with step size h, as outlined in [2], which gives

$$\int_{y_n}^{y_{n+1}} f(y,t)\,\mathrm{d}t = \frac{h}{6}[f(y_n,t_n) + 4f(y_{n+\frac{1}{2}},t_{n+\frac{1}{2}}) + f(y_{n+1},t_{n+1}) + ...] + \mathcal{O}(h^5). \tag{13}$$

To improve this to fourth-order, we require four calculations of $f(y,t)$ per iteration, which are

$$\begin{aligned}
k_1 &= f(y_n,t_n),\\
k_2 &= f(y_n + \frac{h}{2}k_1, t_n + \frac{h}{2}),\\
k_3 &= f(y_n + \frac{h}{2}k_2, t_n + \frac{h}{2}),\\
k_4 &= f(y_n + hk_3, t_n + h),
\end{aligned}$$

and therefore

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5). \tag{14}$$

We then let $t_n \leftarrow t_n + h$ and $y_n \leftarrow y_{n+1}$, and continue iterating.

3.2 Coupled Differential Equations

We can generalise the fourth-order Runge-Kutta method to two coupled first-order ordinary differential equations. Let $\mathrm{d}x/\mathrm{d}t = f(x, y, t)$ and $\mathrm{d}y/\mathrm{d}t = g(x, y, t)$. Similar to what we did in the previous section, we have

$$\begin{aligned}
k_1 &= f(x_n, y_n, t_n),\\
l_1 &= g(x_n, y_n, t_n),\\
k_2 &= f(x_n + \frac{h}{2}k_1, y_n + \frac{h}{2}l_1, t_n + \frac{h}{2}),\\
l_2 &= g(x_n + \frac{h}{2}k_1, y_n + \frac{h}{2}l_1, t_n + \frac{h}{2}),\\
k_2 &= f(x_n + \frac{h}{2}k_2, y_n + \frac{h}{2}l_2, t_n + \frac{h}{2}),\\
l_3 &= g(x_n + \frac{h}{2}k_2, y_n + \frac{h}{2}l_2, t_n + \frac{h}{2}),\\
k_4 &= f(x_n + hk_3, y_n + hl_3, t_n + h),\\
l_4 &= f(x_n + hk_3, y_n + hl_3, t_n + h).
\end{aligned}$$

Similar to what we had in equation (14),

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5), \tag{15}$$

$$y_{n+1} = y_n + \frac{h}{6}(l_1 + 2l_2 + 2l_3 + l_4) + \mathcal{O}(h^5). \tag{16}$$

Once again, we let $t_n \leftarrow t_n + h$, $x_n \leftarrow x_{n+1}$, and $y_n \leftarrow y_{n+1}$ and continue iterating.

The Runge-Kutta can be applied to any number of coupled first-order ordinary differential equations. For example, if we have three coupled first-order differential equation, we let $\mathrm{d}x/\mathrm{d}t = f(x, y, z, t)$, $\mathrm{d}y/\mathrm{d}t = g(x, y, z, t)$, $\mathrm{d}z/\mathrm{d}t = h(x, y, z, t)$ and solve these them in a similar fashion by letting $k_1 = f(x_n, y_n, z_n, t_n)$, $l_1 = g(x_n, y_n, z_n, t_n)$ and $m_1 = h(x_n, y_n, z_n, t_n)$, iterate towards k_4, l_4 and m_4, update x_n, y_n and z_n, and repeat.

3.3 Error Considerations

We have used Simpson's Rule throughout the derivation, resulting in a local truncation error of order $\mathcal{O}(h^5)$. This results in a global accumulated error of $\mathcal{O}(h^4)$. We will observe later in our computational analysis that $\rho(r)$ and $m(r)$ vary slowly with r, and hence the fourth derivative is very small. The Runge-Kutta method therefore provides a good approximation to the coupled first-order ordinary differential equations.

It is also worth noting that the time complexity of the Runge-Kutta method is $\mathcal{O}(n)$. For astrophysical problems which require many iterations, the linear time can be computationally taxing.

4 Computational Analysis

For this practical, we use Python as our programming language of choice. We create three functions outlined by [1]. The first is a generic fourth-order Runge-Kutta method solver, which has been customised slightly for the case of our coupled differential equations. The remaining two define the relevant differential equations for both the non-relativistic and relativistic cases. The step size we choose is $\delta r = 5000$ m.

4.1 Non-relativistic case

For the non-relativistic case, we choose an input central density $\rho_c = 10^{14}$ kg m^{-3}, and solve for $\rho(r)$ and $m(r)$. To avoid any singularities, we let ρ_c be measured at 10 m $\ll 5 \times 10^7$ m. The plots of $\rho(r)$ and $m(r)$ are shown in Figure 2.

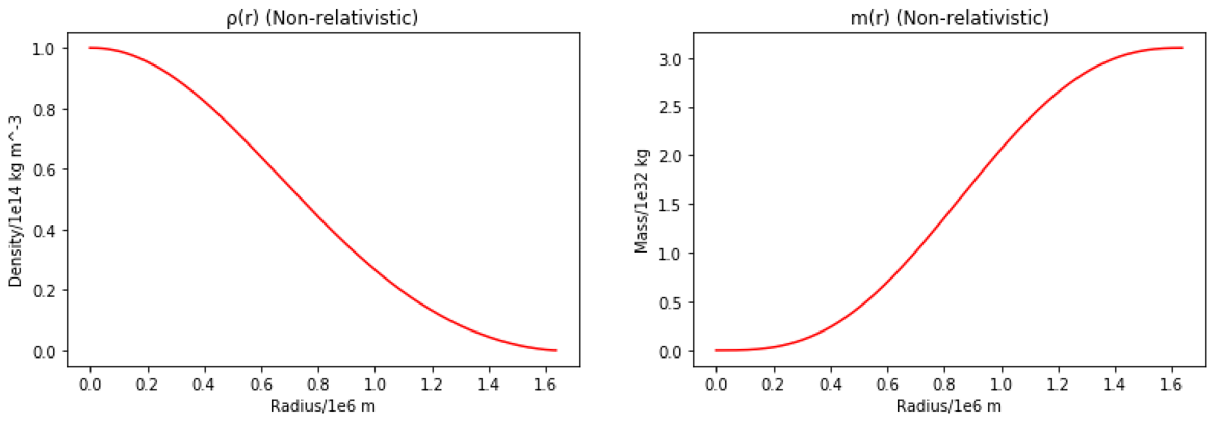

Figure 2: Plots of $\rho(r)$ and $m(r)$ for the non-relativistic case

We notice that using `numpy` is to our advantage, as an input r of order 5×10^7 m for the non-relativistic case will result in a `RuntimeWarning` error being raised due to an underflow with `double_scalars`. This is because the density goes to zero at a point long before 5×10^7 m. The values beyond that are `NaN`, and are not present in the plots. Overall, we can conclude that for a star with $\rho_c = 10^{14}$ kg m^{-3}, it has overall radius $R \sim 1.6 \times 10^6$ m and total mass $M \sim 3.2 \times 10^{32}$ kg.

We are also interested in the total mass M and overall radius R as a function of ρ_c for $10^6 < \rho_c < 10^{14}$ kg m^{-3}. We sample 50 points evenly spaced along the $\log \rho_c$ axis and obtain their corresponding masses and radii. This time, to filter out the `NaN` values, we perform the list comprehension `[val for val in nonrel_mass if val == val]` as `NaN` has the property `NaN != NaN`. Maintaining our log-scale, the corresponding plots are shown in Figure 3.

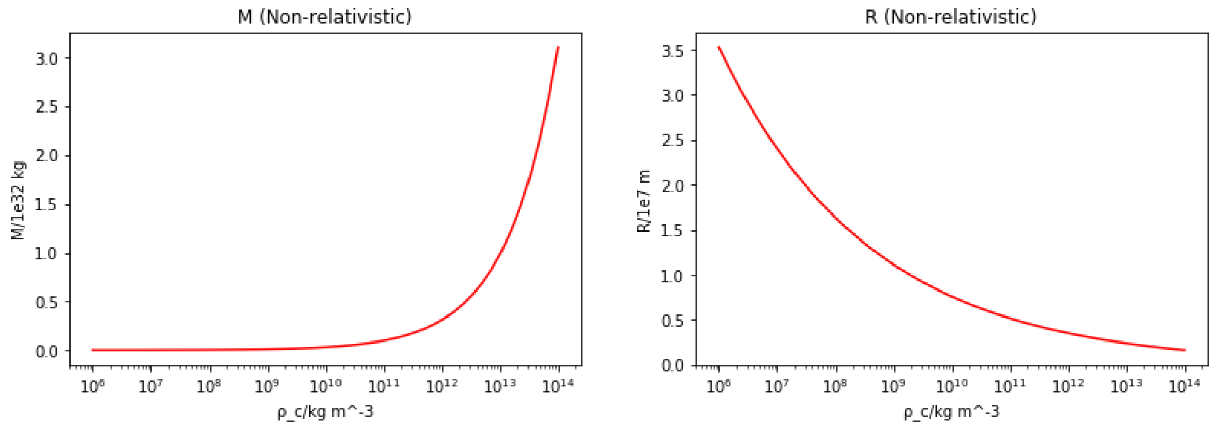

Figure 3: Plots of M and R for the non-relativistic case

As we can see in the plots, with increasing central density, the total mass and radius of the white dwarf increases and decreases respectively. However, there is no upper limit to M, which is not consistent with astronomical data and the Chandrasekhar limit, which we discuss in the next section. This is where the relativistic considerations come into play.

4.2 Relativistic case

Our other function adjusts for our relativistic $\mathrm{d}\rho/\mathrm{d}r$. Our input ρ_c is the same as the non-relativistic case. The new plots of $\rho(r)$ and $m(r)$ are shown in Figure 4.

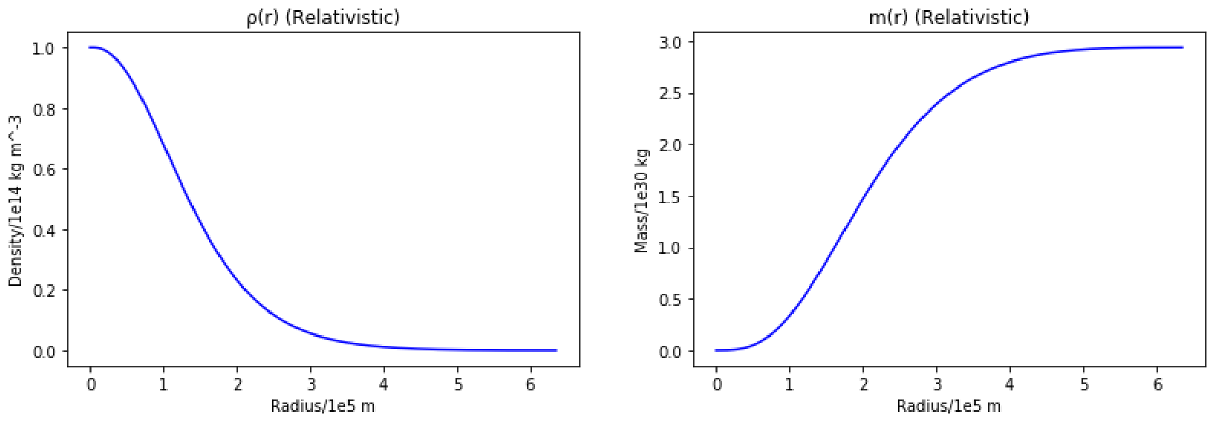

Figure 4: Plots of $\rho(r)$ and $m(r)$ for the relativistic case

This time, our plots tell us that for a star with $\rho_c = 10^{14}$ kg m^{-3}, it has overall radius $R \sim 6.4 \times 10^5$ m and total mass $M \sim 3.0 \times 10^{30}$ kg. These values are a few orders of magnitude smaller than what we obtained in the previous section. We also perform an identical analysis for M and R as a function of ρ_c in the same limits for the relativistic case. The new plots of M and R are shown in Figure 5.

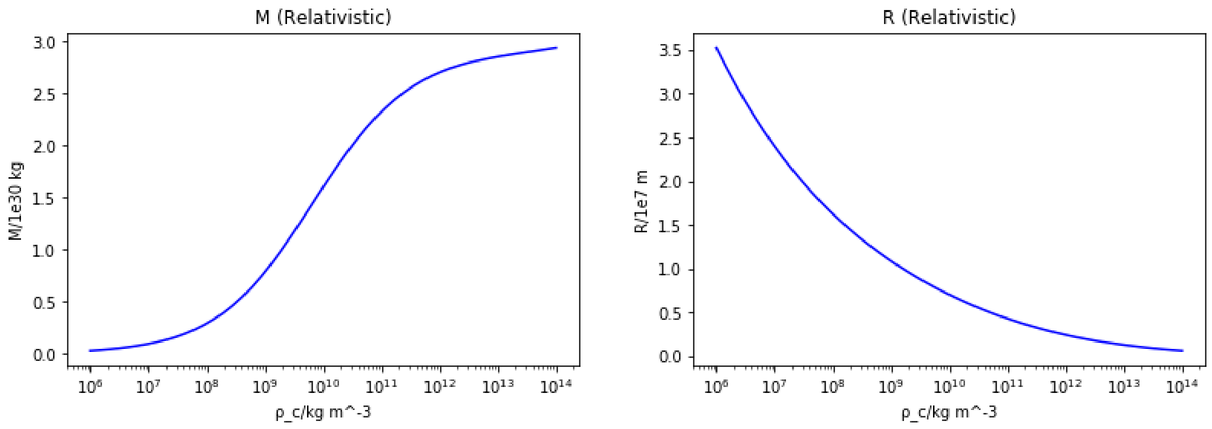

Figure 5: Plots of M and R for the relativistic case

This time, we observe a clear upper limit for M as ρ_c increases. This is consistent with the Chandrasekhar limit. If the core density is large enough such that the electron degeneracy pressure is insufficient to maintain hydrostatic equilibrium, the star collapses. This implies the existence of black holes and that no white dwarf can exist with a mass higher than this limit, which is the Chandrasekhar limit M_C. To observe this, we superpose and compare our plots.

4.3 Comparisons

We compare the plots of M and R against ρ_c for both cases as shown in Figure 6. Note that we use a log scale for M instead, due to the difference in order of magnitude.

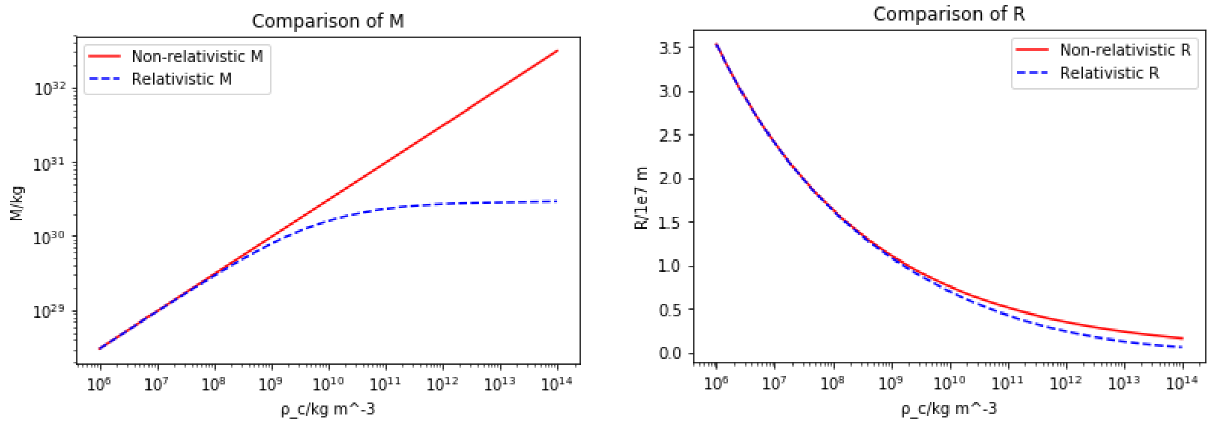

Figure 6: Comparisons of M and R

The theoretical Chandrasekhar mass is $M_C = 2.765 \times 10^{30}$ kg. By taking the last value of our `rel_total_mass` array, our computationally determined $M_C = 2.935 \times 10^{30}$ kg, which deviates from the theoretical value by roughly 6%.

We also plot M as a function of R for both comparisons, which also allow us to see the Chandrasekhar limit clearly.

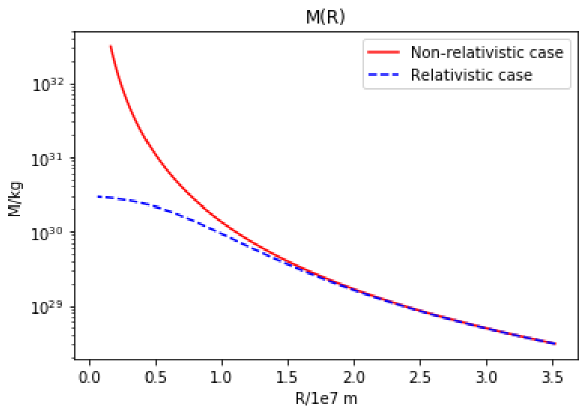

Figure 7: Comparison of $M(R)$

We first notice that for both cases, M is larger for smaller R, which represents a stronger gravitational pull due to the denser core and smaller electron degeneracy pressure. More importantly, we once again observe the Chandrasekhar limit for the relativistic case.

5 Conclusion

We have shown that the strucutre of white dwarf stars can be modelled using hydrostatic equilibrium and electron degeneracy pressure considerations. The two coupled first-order differential equations can be solved with the fourth order Runge-Kutta method. We then use our Runge-Kutta method solver for both the non-relativistic and relativistic cases, and note that the relativistic case enforces the Chandrasekhar limit. We obtain this limit to be $M_C = 2.935 \times 10^{30}$ kg, with an error of around 6%. However, this error is large from an empirical perspective. We have introduced many sources of error, such as capping the order of the Runge-Kutta method, capping the step size, and allowing underflow. Nevertheless, the fourth order Runge-Kutta method we have employed is remarkably accurate, and our function can be modified to solve any number of coupled first-order ODEs.

References

[1] University of Oxford, Physics Practical Course Laboratory Scripts, CO31, 2018

[2] Numerical Methods for Physicists, A. OHare, Oxford Physics, 2005.

[3] B3 Astrophysics Lecture Series, P.Podsiadlowski, Oxford Physics, 2006

[4] Asian Physics Olympiad 2012, Theory Question 2, 2012

Appendix A Python Code

This appendix outlines the Python code used for this practical. The original code was compiled as an `ipynb` in Jupyter Notebook.

```python
# Author: Guo-Zheng Theodore Yoong, Date: 8/2/2019

import numpy as np
import matplotlib.pyplot as plt
from astropy import constants as const

def ode_solve_rk(f, g, y0, t):
    """
    Solves the coupled ODEs, i.e. dx/dt = f(x,y,t) and dy/dt = g(x,y,t),
       using Runge Kutta algorithm.

    Input:
    f, g: functions that receives the current state (x and y) and the
       current position/time (t), and returns the derivative values of the
       state, dx/dt and dy/dt.
    y0: the initial state of the system, given in a column matrix of
       dimensions 2 x 1 (in this case mass and density).
    t: vector of positions/time steps with length N where the values of y
       will be returned.

    Output:
    y: 2 x N matrix that contains the values of y at every position/time
       step. Columns correspond to position/time and rows to the element of
       y.
    """
    N = len(t) # Number of iterations
    h = (t[-1]-t[0])/(N-1) # Step size
    x_curr, y_curr = y0[0][0], y0[1][0] # Initialise current x and y
    y = y0 # Creates a copy of y0
    for i in range(0, N-1): # Python is zero-indexed
        # This loop performs the Runge-Kutta algorithm
        k1 = f(x_curr,y_curr,t[i])
        l1 = g(x_curr,y_curr,t[i])
        k2 = f(x_curr+0.5*h*k1, y_curr+0.5*h*k1, t[i]+0.5*h)
        l2 = g(x_curr+0.5*h*k1, y_curr+0.5*h*k1, t[i]+0.5*h)
        k3 = f(x_curr+0.5*h*k2, y_curr+0.5*h*k2, t[i]+0.5*h)
        l3 = g(x_curr+0.5*h*k2, y_curr+0.5*h*k2, t[i]+0.5*h)
        k4 = f(x_curr+h*k2, y_curr+h*k2, t[i]+h)
        l4 = g(x_curr+h*k2, y_curr+h*k2, t[i]+h)

        # Update next values of x_curr and y_curr
        x_curr, y_curr = x_curr + h*(k1+2*k2+2*k3+k4)/6, y_curr + h*(l1+2*l2+2*l3+l4)/6
        y[0].append(x_curr)
        y[1].append(y_curr)
    return y

def get_nonrel_density(rho0, radius):
    """
    Obtains the density, rho, as function of the radial distance, r, using the implemented ODE solver using the non relativistic equation.

    Input:
    rho0: the central density at r = 0.
    radius: the grid points of radial distance where the density is calculated in form of vector with N elements. Variable name changed for consistency.

    Output:
    density: a vector with N elements that contains the density at radial distance given in r. Variable name changed for consistency.
    mass: a vector with N elements containing the cumulative mass of the white dwarf from r=0 to the given radial distance in r.
    """
    # Initialising constants, all in S.I. units
    h = const.h.value # Planck's constant
    m_e = const.m_e.value # Mass of Electron
    m_p = const.m_p.value # Mass of Proton
    G = const.G.value # Gravitational constant

    k = (48*(np.pi**(2/3))*G*m_e*(m_p**(5/3)))/((h**2)*(2**(1/3))*(3**(2/3))) # For calculation of drho_dr

    r0 = 10 # An initial radius close to 0 with the central density, but not 0 so that 1/r^2 is defined
    m0 = (4/3)*rho0*np.pi*(r0**3) # The corresponding initial mass

    drho_dr = lambda rho, m, r: -k*m*(rho**(1/3))/(r**2) # Lambda function for drho_dr
    dm_dr = lambda rho, m, r: 4*np.pi*rho*(r**2) # Lambda function for dm_dr

    init_rho_and_mass = [[rho0], [m0]]
    return ode_solve_rk(drho_dr, dm_dr, init_rho_and_mass, radius) # Here,
[[rho],[mass]] is y, and radius is t.

def get_rel_density(rho0, radius):
    """
    Obtains the density, rho, as function of the radial distance, r, using the implemented ODE solver using the relativistic equation.

    Input and output is the same as the non-relativistic function.
    """
    # Initialising constants, all in S.I. units
    h = const.h.value # Planck's constant
    m_e = const.m_e.value # Mass of Electron
    m_p = const.m_p.value # Mass of Proton
    G = const.G.value # Gravitational constant
    c = const.c.value # Speed of light

    # For calculation of drho_dr
    k = ((2**(7/3))*(3**(1/3))*(np.pi**(1/3))*G*(m_p**(4/3)))/((h**2)*c)
    a = (2**(8/3))*(np.pi**(2/3))*(m_e**2)*(m_p**(2/3))*(c**2)
    b = (3**(2/3))*(h**2)

    r0 = 10 # An initial radius close to 0 with the central density, but not 0 so that 1/r^2 is defined
    m0 = (4/3)*rho0*np.pi*(r0**3) # The corresponding initial mass

    drho_dr = lambda rho, m, r: -k*m*(rho**(1/3))*np.sqrt(a+b*(rho**(2/3)))/(r**2) # Lambda function for drho_dr
    dm_dr = lambda rho, m, r: 4*np.pi*rho*(r**2) # Lambda function for dm_dr

    init_rho_and_mass = [[rho0], [m0]]
    return ode_solve_rk(drho_dr, dm_dr, init_rho_and_mass, radius) # Here, [[rho],[mass]] is y, and radius is t.

# As stated in the script, 5    10^7
# We choose an initial rho0 of 10^14
radius = np.linspace(10, 50000000, 10000) # We can filter off NaN Values later
rho0 = 10**14

nonrel_rho_and_mass = get_nonrel_density(rho0, radius) # Error will be raised; ignore
nonrel_density = nonrel_rho_and_mass[0]
nonrel_mass = nonrel_rho_and_mass[1]

rel_rho_and_mass = get_rel_density(rho0, radius) # Error will be raised; ignore
rel_density = rel_rho_and_mass[0]
rel_mass = rel_rho_and_mass[1]

# Scaling for plotting purposes
nonrel_radius_scaled = list(map(lambda x: x/(10**6), radius))
```

```python
nonrel_density_scaled = list(map(lambda x: x/(10**14), nonrel_density))
nonrel_mass_scaled = list(map(lambda x: x/(10**32), nonrel_mass))

rel_radius_scaled = list(map(lambda x: x/(10**5), radius))
rel_density_scaled = list(map(lambda x: x/(10**14), rel_density))
rel_mass_scaled = list(map(lambda x: x/(10**30), rel_mass))

plt.plot(nonrel_radius_scaled, nonrel_density_scaled, c='red', label="
    Density")
plt.xlabel('Radius/1e6 m')
plt.ylabel('Density/1e14 kg m^-3')
plt.title('rho(r) (Non-relativistic)')
plt.show()

plt.plot(nonrel_radius_scaled, nonrel_mass_scaled, c='red', label="Mass")
plt.xlabel('Radius/1e6 m')
plt.ylabel('Mass/1e32 kg')
plt.title('m(r) (Non-relativistic)')
plt.show()

plt.plot(rel_radius_scaled, rel_density_scaled, c='blue', label="Density")
plt.xlabel('Radius/1e5 m')
plt.ylabel('Density/1e14 kg m^-3')
plt.title('rho(r) (Relativistic)')
plt.show()

plt.plot(rel_radius_scaled, rel_mass_scaled, c='blue', label="Mass")
plt.xlabel('Radius/1e5 m')
plt.ylabel('Mass/1e30 kg')
plt.title('m(r) (Relativistic)')
plt.show()

samples = np.logspace(6,14) # Default no. of samples is 50

nonrel_overall_radius = []
nonrel_total_mass = []
for sample in samples:
    nonrel_rho_and_mass = get_nonrel_density(sample, radius) # Error will be
        raised; ignore
    nonrel_mass = nonrel_rho_and_mass[1]
    nonrel_mass = [val for val in nonrel_mass if val == val]
    lastcount = len(nonrel_mass)
    nonrel_total_mass.append(nonrel_mass[-1])
    nonrel_overall_radius.append(radius[lastcount-1])

nonrel_overall_radius_scaled = list(map(lambda x: x/(10**7),
    nonrel_overall_radius))
nonrel_total_mass_scaled = list(map(lambda x: x/(10**32), nonrel_total_mass)
    )

plt.plot(samples, nonrel_overall_radius_scaled, c='red', label="R")
plt.xscale('log')
plt.xlabel('rho_c/kg m^-3')
plt.ylabel('R/1e7 m')
plt.title('R (Non-relativistic)')
plt.show()

plt.plot(samples, nonrel_total_mass_scaled, c='red', label="M")
plt.xscale('log')
plt.xlabel('rho_c/kg m^-3')
plt.ylabel('M/1e32 kg')
plt.title('M (Non-relativistic)')
plt.show()

rel_overall_radius = []
rel_total_mass = []
for sample in samples:
    rel_rho_and_mass = get_rel_density(sample, radius) # Error will be raised; ignore
    rel_mass = rel_rho_and_mass[1]
    rel_mass = [val for val in rel_mass if val == val]
    lastcount = len(rel_mass)
    rel_total_mass.append(rel_mass[-1])
    rel_overall_radius.append(radius[lastcount-1])

rel_overall_radius_scaled = list(map(lambda x: x/(10**7), rel_overall_radius))
rel_total_mass_scaled = list(map(lambda x: x/(10**30), rel_total_mass))

plt.plot(samples, rel_overall_radius_scaled, c='blue', label="R")
plt.xscale('log')
plt.xlabel(' _c/kg m^-3')
plt.ylabel('R/1e7 m')
plt.title('R (Relativistic)')
plt.show()

plt.plot(samples, rel_total_mass_scaled, c='blue', label="M")
plt.xscale('log')
plt.xlabel(' _c/kg m^-3')
plt.ylabel('M/1e30 kg')
plt.title('M (Relativistic)')
plt.show()

plt.plot(samples, nonrel_overall_radius_scaled, c='red', label="Non-relativistic R")
plt.plot(samples, rel_overall_radius_scaled, c='blue', label="Relativistic R", linestyle='dashed') # Dashed line for relativistic case
plt.xscale('log')
plt.xlabel(' _c/kg m^-3')
plt.ylabel('R/1e7 m')
plt.title('Comparison of R')
plt.legend(loc='upper right')
plt.show()

plt.plot(samples, nonrel_total_mass, c='red', label="Non-relativistic M")
plt.plot(samples, rel_total_mass, c='blue', label="Relativistic M", linestyle='dashed') # Dashed line for relativistic case
plt.xscale('log')
plt.yscale('log')
plt.xlabel(' _c/kg m^-3')
plt.ylabel('M/kg')
plt.title('Comparison of M')
plt.legend(loc='upper left')
plt.show()

# The script requires R as a function of M, so this is included. However, as M as a function of R is more intuitive, the report focuses more on log M(R) instead
plt.plot(nonrel_total_mass, nonrel_overall_radius_scaled, c='red', label="Non-relativistic case")
plt.plot(rel_total_mass, rel_overall_radius_scaled, c='blue', label="Relativistic case", linestyle='dashed')
plt.xscale('log')
plt.xlabel('M/kg')
plt.ylabel('R/1e7 m')
plt.title('R(M)')
plt.legend(loc='upper right')
plt.show()

plt.plot(nonrel_overall_radius_scaled, nonrel_total_mass, c='red', label="Non-relativistic case")
plt.plot(rel_overall_radius_scaled, rel_total_mass, c='blue', label="Relativistic case", linestyle='dashed')
plt.yscale('log')
plt.ylabel('M/kg')
plt.xlabel('R/1e7 m')
plt.title('M(R)')
plt.legend(loc='upper right')
plt.show()

M_C = rel_total_mass[-1]
print(M_C)
```