TOV Solver problem

January 15, 2021

1 Newtonian

Initially, I attempted to implement the Newtonian equations using dimensionless variables and geometric units. This version of the code was unsuccesful, so I decided to do the calculation in physical units, move to geometric units, then to dimensionless variables. The first step was to write the code in CGS and SI units ands compare results.

1.1 CGS units

The equations used below are from two different sets of notes. I did this to ensure that there were no type errors in the equations in one of the paper [2, 5].

To start I define the equation of state, which is in general:

$$p = K\rho^{\gamma} \tag{1.1}$$

where $\gamma = (n+1)/n$ is the polytropic index. I start with the relativistic case which is:

$$p = K\rho^{4/3} \tag{1.2}$$

This K is for relativistic limit of Fermi Gas. The value of K can be calculate explicitly. It is give in two different functional forms in the two papers [2, 5]. They are

$$K = \frac{\hbar c}{12\pi^2}\left(\frac{3\pi^2 Z}{Am_n}\right)^{4/3} \tag{1.3}$$

$$K = \frac{1}{8}\left(\frac{3}{\pi}\right)^{1/3}\frac{hc}{(m_H\mu_e)^{4/3}} \tag{1.4}$$

This corresponds to a cgs value of:

$$K = 1.23 \times 10^{15} \mu_e^{-4/3} \tag{1.5}$$

here μ_e is equal to A/Z and is the number of nucleons per electron in a neutron star and is 2. I calculated all of these values and confirmed they were the same:

```
g_cgs = 6.67259 * 10.**(-8.) #cm^3 g^1 s^-1
hbar_cgs = 1.05457266 * 10**(-27.) #erg s
c_cgs = 3.0 * 10.**10. # cm /s^1
mh_cgs = 1.6749 * 10.**(-24.) #g

#from Sanjay's Paper
K_cgs = hbar_cgs*c_cgs
K_cgs = K_cgs /(12*np.pi**2.)
K_cgs = K_cgs * (3 * np.pi **2.)**(4./3.)
K_cgs = K_cgs * (mh_cgs)**(-4./3.)
print K_cgs
K_cgs = K_cgs * 0.5**(4./3.)

#From Princeton Lecture
K = (h_cgs * c_cgs)/ 8.
K = K * (3/np.pi)**(1./3.)
K = K * (1./(mh_cgs))**(4./3.)
print K
K = K * 0.5**(4./3.)
```

After printing the output for both equations, I confirm that they are consistent with each other and with the literature.

Once I have calculated K and defined all of my constants, I need to write the code for the equation of state:

```
def EOS(p):
   rho = (p/K_cgs)
   rho = rho**(1./gamma)

   return rho
```

In order to do the integration, I need to define the TOV function. In the case of newtonian physics, this is a simple system of two equations. This can be found in numerous place [3–5]

$$\frac{dM}{dr} = 4\pi\rho r^2 \tag{1.6}$$

$$\frac{dp}{dr} = -\frac{G\rho M}{r^2} \tag{1.7}$$

I define this in a function called TOV, which will later be called by the python differential equation solver

```
def TOV(r,y):
   M = y[0]
   p = y[1]

   rho = EOS(p)

   dMdr = 4 * np.pi * rho * r**2.
   dpdr = - rho * g_cgs * M * r**(-2.)

   return dMdr, dpdr
```

To solve this system, we need initial conditions. For computational reasons, the initial conditions are defined at some small Δr away from the center instead of at r=0. The mass at the center is zero ($M(\Delta r) = 0$). **I tested various r_0 and found that $r_0 < 0.1$ the mass and radius results are the same up to 0.0001 km and 0.0001 M$_\odot$**

Different pressures will yield different radii. I look at a variety of initial pressures. According to [1] the average pressure $\approx 10^{34}$ dynes/cm^2. So for now, I start at 10^{34}. Additionally, we need to define the range of radii that the integrator will explore. I start with $r < 20$km. Since this calculation is done is cgs, we need to multiply by 10^3 to get to meters then 10^2 to get to cm. The code for this is:

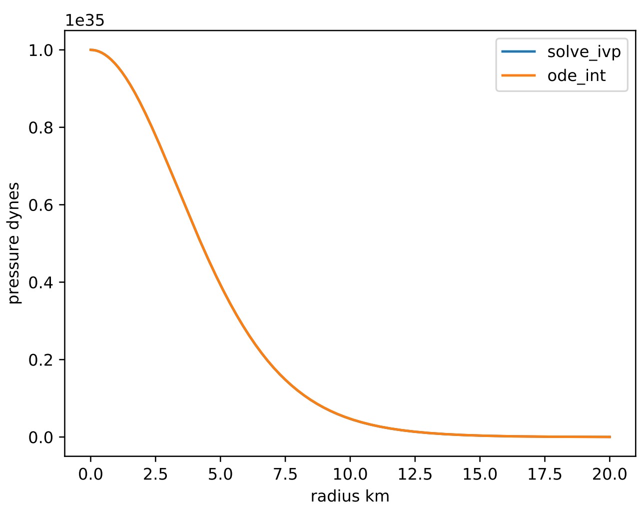
```
M0 = 0
p0 = 10.**34.
y0 = [M0,p0]

r_0 = 0.1
r_stop = 20. #km
r_stop = r_stop * 10**3. #to m
r_stop = r_stop * 100 #to cm
r = np.linspace(r_0,r_stop,num=4000)
```

The next step is to call the scipy integrator. The goal is to use scipy.solve_ivp with option RK45 as this is a 5th order Runge-Kutta algorithm. I also test the solution using the older odeint routine as a cross check. (In order to use ODEint, you need to define TOV(y,r) instead of TOV(r,y) so I define TOV_odeint with this change.)

```
soln = solve_ivp(TOV,(r_0,r_stop),[M0,p0])
rs = soln.t
m = soln.y[0]
p = soln.y[1]

rs = rs / 100
rs = rs / 10**(3.) #back to km


soln = odeint(TOV_odeint,y0,r)
mass = soln[:,0]
```

```
press = soln[:,1]
r = r/100
r = r/1000

#plot results
plt.plot(rs,p,label="solve_ivp")
plt.plot(r,press,label="ode_int")
plt.legend()
plt.xlabel("radius km")
plt.ylabel("pressure dynes")
plt.savefig("plots/pressure-radius_cgs.pdf")
plt.close()
```

I compare these two results.



The two methods are consistent with each other.

From here, I add a boundary condition for the edge of the star. (only in solve_ivp)

```
def star_boundary(r,y):
   return y[1]

#Set star boundary at pressure = 0
star_boundary.terminal = True
```

1.1.1 Non Relativistic Case

I want to test the Non relativstic case ($\gamma = 5/3$). This has a different K value.

The equations from

1.2 SI units

As mentioned above, I also wrote a version of the code for SI units. This code was written independently so that any errors in the CGS code would not be duplicated.

The constants defined in the SI system are:

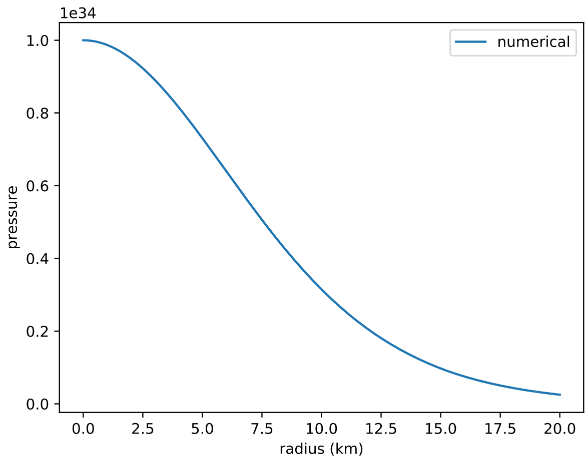
```
G = 6.67259 * 10.**(-11.) # m^3 kg^-1 s^-2
c = 3.00 * 10.**8. #m/s
hbar = 1.0546 * 10.**(-34.) # J * s
h = hbar*2.*np.pi
M_sun = 1.989 * 10.**(30.) # kg
n = 3./2.
gamma = (n+1.)/n

m_e = 9.1094 * 10.**(-31.) # kg
m_h = 1.6749 * 10.**(-27.) # kg
```

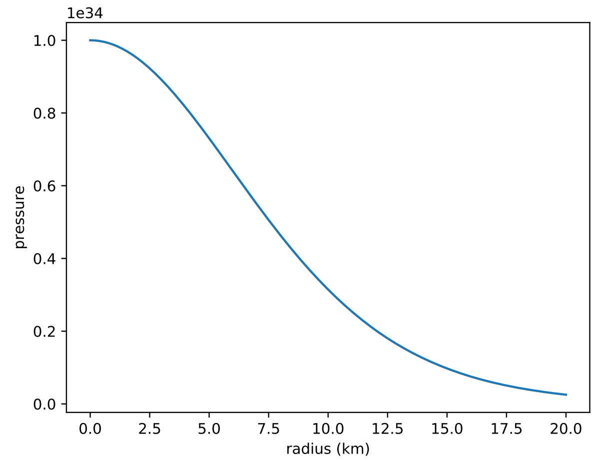
Now, I want to look at K. You can use the same equations to calculate it, so long as you have $\hbar, c, m_N$ in SI units. This yields a K of 1.23×10^{10}. Which is K_cgs $\times 10^5$. This is correct. You can check this because the units of K are m^3 kg$^{-1/3}$ s^{-2} which gives a factor of 10^5 when converted to cm^3 g$^{-1/3}$ s^{-2}. In order to do calculations later, we want to use geometric K. In order to go from SI or cgs to geometric, we need to use a set of transformations. For mass thats $M_g = M \times Gc^{-2}$ and for time its $t_g = t \times c$. This gives the a factor of $G^{-1/3}c^{-4/3}$. However, when you calculate the dimensionless K you get different solutions because of the factor of $c^{-4/3}$. You get (from SI) $K = 59.9$ and (from cgs) $K = 1291.2$. These numbers are significantly different... Maybe this will straighten itself out because the length units are different? I'll see

Again, we need to define the equation of state as well as the TOV equations.

```
def EOS(p):
   rho = (p/K)**(1./gamma)
   return rho

def TOV(r,y):
   M = y[0]
   p = y[1]

   rho = EOS(p)
   #print p
   dMdr = 4. * np.pi * rho * r**2.
   dpdr = - G * M * rho /r**2.
   #print dpdr

   return [dMdr,dpdr]
```

Again, I need to define the initial conditions. We have $M(\Delta r) = 0$ and $P(\Delta r) = 10^{33}$ Pa. (one bayre (cgs) = 0.1 Pa (SI)). So, I chose 10^{33} for an apples to apples comparison. I used solve_ivp to integrate the curve from 0 to 20km. Then I convert the output pressure back to bayres.





They agree! Furthermore all pressures give the same final mass ($M = 2.84 \times 10^{30}$ kg), which is expected in the n=3 case.

1.3 Geometric Units

For the Geometric unit code, I did not start from scratch. Since my SI and CGS codes are both working and are consistent with each other, I copied the SI unit code into a new file. I calculated the geometric K using the fact that the units of K are m^3 kg$^{-1/3}$ s^{-2}. Using the conversations from metric to geometric, this gives a factor of G$^{-1/3}c^{-4/3}$.

$$K[=]m^3 \times kg^{-1/3} \times (Gc^{-2})^{-1/3} s^{-2} \times c^{-2} = m^3 kg^{-1/3} s^{-2} \times (Gc^{-2})^{-1/3} c^{-2} = m^3 kg^{-1/3} s^{-2} \times G^{-1/3} c^{-4/3} \tag{1.8}$$

I define the pressure in SI, then multiply by Gc^{-4} to get the pressure in geometric units [L^{-2}]. The initial mass is zero, which is the same in both units. However, the converstion from mass in SI to mass in geometric units is G c^{-2}. After the integration, I convert the mass back to kg by dividing by G c^{-2}. The mass is consistent with previous results! ($M = 2.84 \times 10^{30}$ kg). The radii are also comparable. I summarize the results in the table below

Initial pressure (Pa)	Radius (m) SI run	Radius (m) Geometric run	Radius (m) dimensionless quantities
1.00e+30	$>$ 20000	$>$ 20000	$>$ 20000
1.29e+31	$>$ 20000	$>$ 20000	$>$ 20000
1.67e+32	$>$ 20000	$>$ 20000	$>$ 20000
2.15e+33	$>$ 20000	$>$ 20000	$>$ 20000
2.78e+34	$>$ 20000	$>$ 20000	$>$ 20000
3.59e+35	11366	11366	11365
4.64e+36	5996	5996	5995
5.99e+37	3162	3162	3162
7.74e+38	1668	1668	1668
1.00e+40	. 880	880	880

Since all of these are self consistent, I decided to check with values in the literature if possible. In [5] they have mass 1.243 $\mathrm{M}_{\odot}$. However, if you divide the mass obtained by my code ($M = 2.84 \times 10^{30}$ kg) by the mass of the sun you get 1.429 $\mathrm{M}_{\odot}$.

1.4 Geometric units with dimensionless pressure and density

Note This was successful, so I added it to the table above.

It is better, computationally, to have values that have a similar order of magnitude and are as close to 1 as possible. If we use 10^{34} bayre, as I have above, that gives a geometric pressure of 10^{-15}. In order to bring that even closer to 0, we introduce a dimensionless scale factor p_c, which is equal to the central pressure. The scaling factors do not have to be the ones used here, these are simply convenient choices. Since the integration ends at $p = 0$, the integrator only deals with p values [0,1]. The density is also scaled. The scale factor ρ_c. This is calculated using equation (1.2) and K in geometric units: $\rho_c = (p_c/K)^{1/\gamma}$.

Explicitly $p = \bar{p}\, p_c$ and $\rho = \bar{\rho}\, \rho_c$. Where $\bar{p}$ and $\bar{\rho}$ are the dimensionless quantities.

Both the equation of state and the TOV equations need to be modified. This can be done by substituting these values into the original equations.

For the equation of state:

$$\rho = \left(\frac{p}{K}\right)^{1/\gamma} \tag{1.9a}$$

$$\bar{\rho}\, \rho_c = \left(\frac{\bar{p}\, p_c}{K}\right)^{1/\gamma} \tag{1.9b}$$

$$\bar{\rho} = \left(\frac{\bar{p}\, p_c}{K}\right)^{1/\gamma} \frac{1}{\rho_c} \tag{1.9c}$$

For the TOV equations

$$\frac{dM}{dr} = 4\pi\rho r^2 \tag{1.10a}$$

$$\frac{dM}{dr} = 4\pi\bar{\rho}\rho_c r^2 \tag{1.10b}$$

and

$$\frac{dp}{dr} = -\frac{G\rho M}{r^2} \tag{1.11a}$$

$$\frac{d(\bar{p}p_c)}{dr} = \frac{G(\bar{\rho}\rho_c)M}{r^2} \tag{1.11b}$$

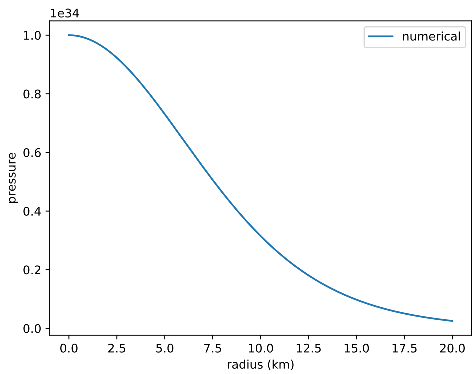
$$p_c\frac{d\bar{p}}{dr} = \frac{G\bar{\rho}\rho_c M}{r^2} \tag{1.11c}$$

$$\frac{d\bar{p}}{dr} = \frac{G\bar{\rho}\rho_c M}{p_c r^2} \tag{1.11d}$$

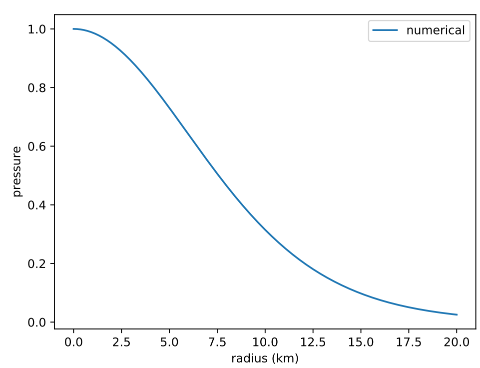
The code for this is:

```
#equation of state
def EOS(p):

   rho = (p_c * p/K_bar)**(1./gamma) / rho_c

   return rho

#TOV
def TOV(r,y):
   M = y[0]
   p = y[1]

   rho = EOS(p)
   #print p
   dMdr = 4. * np.pi * rho * rho_c * r**2.
   dpdr = - 1. * M * rho * rho_c/(r**2.* p_c)
   #print dpdr


   return [dMdr,dpdr]

#Define the scaling factors
for x in pressures:

   p_c = x
   rho_c = (p_c/K_bar)**(1./gamma)

   p_0 = x/p_c
```

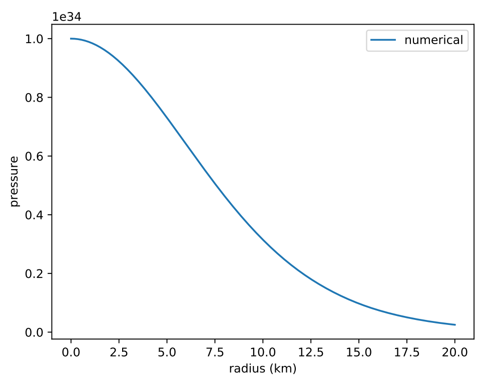
Once these changes are made, the code is run. I plotted the curve with the scaled p (i.e. p ranges between 0 and 1). I also converted the pressure back to CGS to make a comparison with the other two codes.



(a) From Dimensionless Pressure Code



(b) From Dimensionless Pressure Code



(c) from SI Code

As all of the results are consistent, I will move on to the next step.

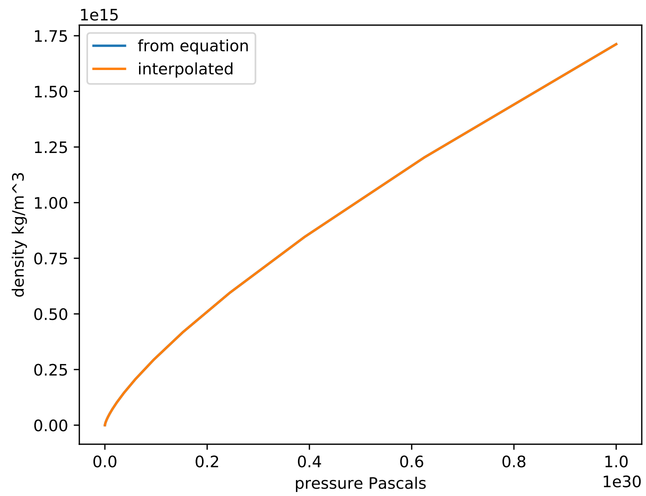
1.5 EOS from a file

Since the EOS used for the real run will be from a table in a file, I want to set that up now. The first thing I did was generate a data file containing the tabulated equation of state. I generated the file in SI units, so that the code is ready to take a file in physical units. I'm not sure what units Ingo uses for his EOS. However, the infrastructure is in place to use whatever physical units.

I use np.loadtxt to read in the data. Then I can use the pressures as x values and the denstiies are y values for the np.interp. I also tested scipy.interp1d and it yielded the same results as np.interp. The code for reading in the file and the new EOS function is this:

```
#Read in the eos file and save the data

data = np.loadtxt("relativistic_polytrope.dat",skiprows=1,delimiter='\t')
data = data.transpose()

eos_ps = data[0]
eos_pg = eos_ps * G * c**(-4.)

eos_rhos = data[1]
eos_rhog = eos_rhos * G * c**(-2.)


def EOS_fromfile(p):
   ps = eos_ps/p_c
   rhos = eos_rhos/rho_c

   rho = np.interp(p,ps,rhos)
   return rho
```

To check the quality of the interpolation and to ensure that the conversion to geometric units and dimensionless parameters was correct, I compared the interpolated results to the exact equation (with the code for that taken from the dimensionless quantities code)



You can see that both versions of the equation of state are the same. I also calculated the percent error and it is 10^{-10}.

All of the work above is done with $n = 3$ or $\gamma = 4/3$ which is the ultra relativistic fermi gas. Since there was a small difference between my results and the results in [5], I decided to run the code for the nonrelativistic case ($n = 3/2$ and $\gamma = 5/3$).

1.6 Non Relativistic Case

For this, I copied the SI unit code. I chose the SI case because that minimizes the number of unit conversions that I need to make to compare answers with [5]. The fewer transformations I have to make, the less likely I am to have a mistake there. I only use the SI code because all of the codes are consistent with each other.

As before, the first need to calculate the value of K. It is **different** than the value in the relativistic case. Again [5] and [2] have different equations. I calculate both and compare the numerical values as a cross check against print errors.

$$K_{nr} = \frac{\hbar^2}{15\pi^2 m_e}\left(\frac{3\pi^2}{\mu m_H}\right)^{5/3} \tag{1.12}$$

$$K_{nr} = \frac{1}{20}\left(\frac{3}{\pi}\right)^{2/3}\frac{h^2}{m_e(m_H\mu)^{5/3}} \tag{1.13}$$

According to [2] this has a numerical value of $0.991 \times 10^{13}\mu^{-5/3}$ in cgs units. I use $\mu = 0.5$ just as I did before. The code returns $K_{nr} = 9.91 \times 10^6$. To check that this is consistent with the CGS value in the paper, one needs to know the units of K (again these are different than before). The units are km$^{-2/3}$ m^4 s^{-2}. This was calculated using the equations above and confirmed by making sure that the units of $K\rho^{5/3}$ returns units of pressure. The conversion from SI to CGS is 10^6, meaning that the value of $K_{nr} = 9.91 \times 10^6$ is consistent with the published value.

The equation of state is defined in the same way, but with a different value of γ.

```
n = 3./2.
gamma = (n+1.)/n

def EOS(p):

    rho = (p/K)**(1./gamma)

    return rho
```

In [5], they use $p = \epsilon_0 \bar{p}$. For the nonrelativstic case they have $\epsilon_0 = 2.488 \times 10^{37}$ ergs / cm^3. They staste that the nonrelativstic polytrope is only valid for pressures $\bar{p}(0) < 4 \times 10^{-15}$. This means that $p_0 < 10^{23}$ bayre or $p < 10^{22}$ Pa. They give the following results

$\bar{p}$	p (Pa)	R (km)	M ($M_\odot$)
10^{-15}	10^{22}	10,620	0.3941
10^{-16}	10^{21}	13,360	0.1974

However, these are not the results that I get. I get

p (Pa)	R (km)	M ($M_\odot$)
10^{22}	39,676	11.04
10^{21}	49,934	5.53

The order of magnitude of the radius is correct, but the numbers are off and the masses are off. The masses are both off by a factor of 3.7 and the masses are both off by a factor of 28.

1.7 Exact Polynomial Solutions

Polytropes have exact solutions for n=0,1,5.

Equations 1.6 and 1.7 can be combined into one equation called the Poisson equation:

$$\frac{1}{r^2}\frac{d}{dr}\frac{r^2}{\rho(r)}\frac{dP}{dr} = -4\pi G\rho(r) \tag{1.14}$$

In order to solve this, a fimensionless function $\theta(r)$ is defined such that:

$$\rho = \rho_c\,\theta^n \tag{1.15a}$$

$$P = P_c\,\theta^{n+1} \tag{1.15b}$$

In this case rho_c is the central pressure and P_c is the central pressure and they are related through the polytrope equation. Using the theta substitution you get the following differential equation.

$$\frac{(n+1)}{4\pi G}\frac{P_c}{\rho_c^2}\frac{1}{r^2}\frac{d}{dr}r^2\frac{d\theta}{dr} = -\theta^n \tag{1.16}$$

Next a dimensionless radius (ξ) is defined via $r = r_n\xi$. Here r_n is the Emden length and is defined as follows:

$$r_n^2 = \frac{(n+1)K}{4\pi G}\rho_c^{\frac{1}{n}-1} \tag{1.17}$$

This transforms the diff eq into the Lane-Emden equation

$$\frac{1}{\xi^2}\frac{d}{d\xi}\xi^2\frac{d\theta}{d\xi} = -\theta^n \tag{1.18}$$

1.7.1 n=1

For the $n = 0$ case you have a homogenous sphere.

$$\theta_0(\xi) = 1 - \frac{1}{6}\xi^2 \tag{1.19}$$

In this case $xi_1 = \sqrt{6}$

However, I'm not sure how to program this is (n+1)/n diverges.

1.7.2 n=1

For the n=1 case the radius of the star is the same for all initial pressures.

In this case $\theta(\xi)$ can be written as follows

$$\theta_1(\xi) = \frac{\sin\xi}{\xi} \tag{1.20}$$

The pressure drops to zero at a point ξ_1 and for $n = 1$, $\xi_1 = \pi$

To compare the results from the code to the Lane-Emden solution. I first calculate the radius of the star using the following.

$$R_{star} = r_n\xi_1 = \left[\frac{(n+1)P_c}{4\pi G\rho_c^2}\right]^{1/2}\pi \tag{1.21}$$

I then compare this to the radius calculated by the numerical integrator and calculate the percent error. The code is as follows:

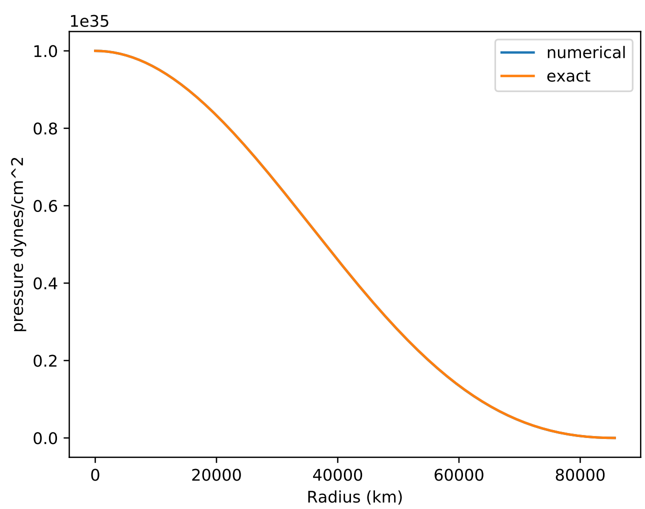
```
R = (((1.+1.)*p0 )/(4*np.pi * g_cgs * rho0**2))**0.5 * np.pi
print "Exact Radius = ", R
#Exact solution of the pressure
print "R_calc - R_exact = ", r[-1] - R
error = abs((r[-1] - R)/R)
print "percent error is ", error
```

I find that $R_{star} = 85,721$ km in both cases. The percent error is 0.0004.

To get the pressure curve from the Lane-Emden solution, I first convert the radius to xi, then calculate $theta$ by equation 1.19, and finally use 1.15b to get the pressure. Then I plot the two to compare.

```
R_s = (((1.+1.)*p0 )/(4*np.pi * g_cgs * rho0**2))**0.5
xi = r / R_s
print max(xi)
theta = np.sin(xi) / xi
P = p0 * theta**(n+1)
```

The two plots are consistent



1.8 Numerical Convergence

For cases where the exact solution is unknown, it is important to test the convergence of the integrator on a solution. This can be done by using smaller and smaller integrator steps. The answer should converge.

If not, it may be possible to extrapolate from the curve of step size vs answer the answer at infinitely small step.

Using my geometric code, I defined a series of step sizes using the np.logspace to range from 1m to 10^{-4}m. I then used this as max step size and calculated the mass/radius for $p_0 10^{40}$ Pa. The masses all agree up to 1.4331011306 $M_\odot$, which is plenty precision for this work. There is more fluctution in the radius, but they all agree to 879.79.

2 General Relativistic

2.1 Polytrope

When moving to relativistic physics, there are three equations. The additonal equation is for the relativistic analog of the gravitional potential (denote ϕ)

In physical units:

$$\frac{dM}{dr} = 4\pi\rho r^2 \tag{2.1}$$

$$\frac{dp}{dr} = -G(\rho + P/c^2)\left(\frac{m + 4\pi r^3 P/c^2}{r(r - 2Gm/c^2)}\right) \tag{2.2}$$

$$\frac{d\phi}{dr} = \frac{m + 4\pi r^3 p}{r(r - 2GM/c^2)} \tag{2.3}$$

To use this in the code, I need to set G=c=1 and substitute in the dimensionless variables.

$$\frac{dM}{dr} = 4\pi\bar{\rho}\rho_c r^2 \tag{2.4}$$

$$\frac{dp}{dr} = -(\bar{\rho}\rho_c + \bar{p}p_c)\left(\frac{m + 4\pi r^3 \bar{p}p_c}{r(r - 2m)}\right)\frac{1}{p_c} \tag{2.5}$$

$$\frac{d\phi}{dr} = \frac{m + 4\pi r^3 \bar{p}p_c}{r(r - 2M)} \tag{2.6}$$

This is implemented as follows:

```
def TOV(r,y):
    M = y[0]
    p = y[1]
    phi = y[2]

    rho = EOS(p)

    temp1 = r * (r - 2. * M )
```

```
    temp2 = M + 4 * np.pi * r**3. * p * p_c

    dMdr = 4. * np.pi * rho * rho_c * r**2.
    dpdr = - 1. * (p * p_c + rho * rho_c) * ( temp2 / temp1 ) * (1./p_c)
    dphidr = temp2 / temp1

    return [dMdr,dpdr,dphidr]
```

Of course, in order to solve this system of equations, you need an equation of state. I found a Reference on GR TOV. They use a compound polytrope. Piecewise polytropes are commonly used equations of state. In this case, they have only two equations of state one for low density regiomns (non relativistic) and one for high density regions (relativistic equations needed). The transition denisty they use is $\rho_t = 5 \times 10^{17}$ kg m^{-3}.

For the densities $\rho < \rho_t$ they have $\gamma = 5/3$. The constant used is defined as

$$K_0 = \frac{(3\pi^2)^{2/3}}{5} \frac{\hbar^2}{m_h^{8/3}} \tag{2.7}$$

For densities $\rho > \rho_t$ they use $\gamma = 3$. They then use continuity to calculate the new value of K.

$$P_{rel}(\rho_t) = P_{nr}(\rho_t) \tag{2.8}$$

$$K_{nr}\rho_t^{5/3} = K_r\rho_t^3 \tag{2.9}$$

$$K_r = K_{nr}\rho_t^{5/3-3} \tag{2.10}$$

In order to use this equation of state in my code with geometric units, I need to convert the K values to geometric units. The first thing I do is to convert K_{nr}. The first step to this is to determine the units of K_{nr}. The only two quantities with units in the definition of K are $\hbar$ and m_h.

$$[K] = (Js)^2 kg^{-8/3} = \left(\frac{kgm^2}{s}\right)^2 kg^{-8/3} = kg^{-2/3} m^4 s^{-2} \tag{2.11}$$

You can then check this by making sure that

$$[p] = [K][\rho^{5/3}] \tag{2.12}$$

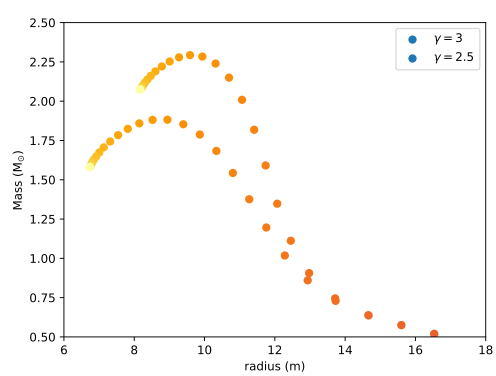
After performing this check, I need to convert to geometric units. Using conversions, there's a factor of Gc^{-2} for kilograms and c for seconds. Substituting this into th units of K gives a multiplicative factor of $G^{-2/3}c^{-2/3}$.

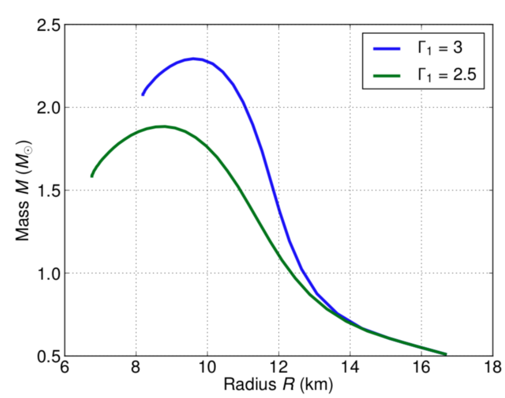
In order to use Equation 2.10 to calculate K_{rel}, I need ρ_t in geometric units. This is done simply by multiplying Gc^{-2}. After that, I can just calculate K_{rel} using 2.10. The paper also looks at $\gamma = 2.5$ for the relativistic case, so I looked at this as well.

The code for this is as follows:

```
K_nr = (3.0*pi**2)**(2.0/3.0) * hbar**2. / (5.0*m_h**(8.0/3.0))
    #nonrelativistic K

K_nr_bar = K_nr * G**(-2./3.) * c**(-2./3.)
gamma_1 = 5./3. #non-relativistic degenercy at low densities
gamma_2 = 3.  #relativistic degenercy at high densities

rho_t = 5. * 10.**17. #Transition Density
rho_t_bar = rho_t * G * c**(-2.) #to geometric units

p_t = K_nr * rho_t ** gamma_1 #Transition pressure

p_t_bar = K_nr_bar * rho_t_bar **gamma_1 #Transition pressure G=c=1

K_r_bar = K_nr_bar * rho_t_bar ** (gamma_1 - gamma_2) #Relativistic K,
    calculated using continuity
```

Comparing my plot to the plot from the lecture notes:





2.2 Chiral EFT

As the goal of this project is to eventually use Chiral EFT in scalar tensor theories and gravitational wave forms, I first want to check the GR version of the code. The EOS are provided by Ingo Tews. They consist of three columns: baryon density (fm^{-3}), pressure (MeV fm^{-3}), energy density (MeV fm^{-3}). To use these, I will read in the three columns, toss out the first one, convert to SI units. I have been using mass density (but using E=mc^2, I know that energy density is mass density x c^2). In the case where c=1, mass density and energy density are the same.

I have already developed the code for reading in files above. I added this to the gr_tov code and then tested it by generating a file with the piecewise polytrope above and comparing the results.

The code that I have converts SI units to geometric units. So, I will first need to change from the nuclear units used by Ingo to SI. According to his email multiplying the columns by $1.602176565 \times 10^{33}$ will give erg/cm^3 or to dyne/cm^2. To go from dyne/cm^2 (bayres) to pascals you can use the fact that 1 bayre = 0.1 pascal. For energy density 1 erg/cm^2 is 10^{-1} J/m^3 (10^2 cm per m cubed and then. multiplied by 10^7 ergs per Joule.

In order to generate the mass-radius curve, I have to explore a range of pressures. Ingo suggested exploring from 1 MeV fm^{-3} and up. So I start with $p = 10^{32}$ Pa. I go up to the maximum pressure given in the EOS file.

I started with the "EOS_nsat_ind1" file.

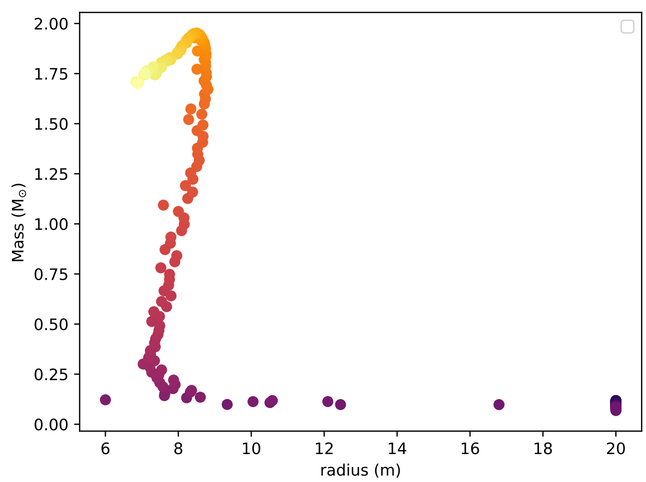


Figure 2: This is the mass radius curve from the first result. You can see that there are a few outliers on the curve. I suspect these are from numerical errors.

Since I believe the scatter in the above curve comes from numerical errors, I will do a few things to fix it.

1. Start with a smaller initial radius
2. Reduce the error tolerance on the integrator
3. reduce max step size of the integrator

By reducing the max stepsize to 1m I get the following

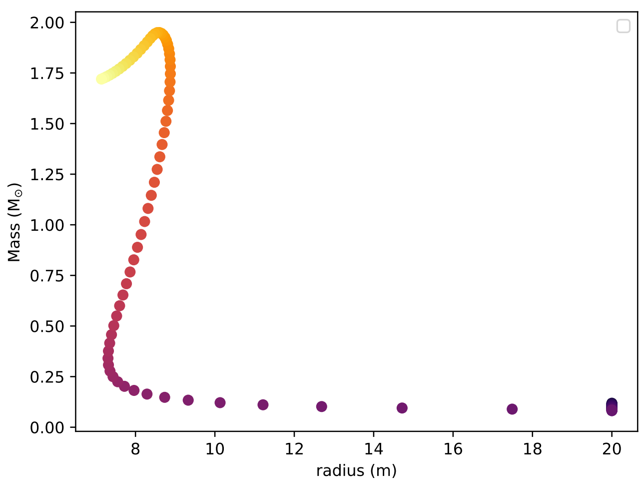


Figure 3: Max Step Size = 1.

3 Scalar Tensor Theory

Scalar tensor theory has an additional dynamical scalar field (ϕ). This means that there is an additional equation in the system of equation. The equations for M, P and ϕ are also modified. (The details of the derivation of the equations is in a seperate set of notes) There are two ways to express the TOV equations in scalar tensor theory: the Einstein frame and the Jordan frame. The Jordan frame is the metric frame. The Einstein frame reformulates the Einstein equation so that it looks more like the GR version. According to [**?**] the Einstein frame is better for numerical analysis, so I will use this. I will also start with the simplest Scalar Tensor Theory: Fierz, Jordan, Brans and Dike (FJBD) Theory ($\omega' = \lambda = 0$). ω is the one arbitrary parameter and scalar tensor theory goes to general relativity as $\omega \to \infty$. Since [**?**] uses a polytropic equation of state, I will also use a polytrope for the initial tests.

$$\frac{dM}{dr} = \frac{\kappa^2}{2} r^2 A^4(\varphi)\rho + \frac{1}{2} r(r-2M)\psi^2 \tag{3.1}$$

$$\frac{d\nu}{dr} = \kappa^2 \frac{r^2 A^4(\varphi) p}{r-2M} + r\psi^2 + \frac{2M}{r(r-2M)} \tag{3.2}$$

$$\frac{d\varphi}{dr} = \psi \tag{3.3}$$

$$\frac{d\psi}{dr} = \frac{\kappa^2}{2} \frac{rA^4(\varphi)}{r-2M} \left[\alpha(\varphi)(\rho-3p) + r\psi(\rho-p)\right] - \frac{2(r-M)}{r(r-2M)}\psi \tag{3.4}$$

$$\frac{dp}{dr} = -(\rho+p)\left[\frac{\kappa^2}{2} \frac{r^2 A^4(\varphi)p}{r-2M} + \frac{1}{2} r\psi^2 + \frac{M}{r(r-2M)} + \alpha(\varphi)\psi\right] \tag{3.5}$$

Here they have defined $\nu(r) = 2\Phi(r)$ We need explicit defintions for $A(\varphi)$ and $\alpha(\varphi)$. Since we are starting with the FJBD theory

$$A(\varphi) = e^{\tilde{\alpha}\varphi} = \Omega^{-1} \tag{3.6}$$

and $\tilde{\alpha} = \tilde{\alpha}_0$ because we are starting with FJBD theory and $\omega(\varphi) = const$. They explore a range of the ccouipling constant ($\tilde{\alpha}$) such that $10^{-4} \leq \tilde{\alpha} \leq 10^{-2}$. The results discussed are for $\tilde{\alpha} = 10^{-3}$ so I will also start with this value for comparison reasons.
The value of Ω comes from the transformation from the Jordan frame to the Einstein frame.

$$\Omega^2(\phi) = \phi \tag{3.7}$$

Damour and Esposito-Farese define $\tilde{\alpha}$ as follows:

$$\tilde{\alpha}(\varphi) \equiv \frac{d\ln A(\varphi)}{d\varphi} = \frac{1}{\sqrt{3 + 2\omega(\phi)}} \tag{3.8}$$

Since we have a explicit function for A in the FJBD theory, I will start with the explicit definition of alpha. For more complex iterations later, I can implement a derivative calculator. The baryonic mass is different from the mass above and is defined as:

$$\frac{d\bar{M}}{dr} = 4\pi m_b n A^3(\varphi)\frac{r^2}{\sqrt{1-2M/r}} \tag{3.9}$$

In order to solve these equations, you need initial conditions. These are all defined at some small Δr because the computer cannot handle $1/r$ terms at $r = 0$. The scalar field φ is actually defined at $r = \infty$, but for the computational integration we need to start at $r = 0$. Therefore, the central scalar field is initially a guess.

$$M(\Delta r) = 0 \tag{3.10}$$

$$\nu(\Delta r) = 0 \tag{3.11}$$

$$\varphi(\Delta r) = \varphi_c \tag{3.12}$$

$$p(\Delta r) = p_c \tag{3.13}$$

$$\psi(\Delta r) = \frac{4\pi}{3}\Delta r A^4(\varphi_c)\alpha(\varphi_c)(\rho_c - 3p_c) \tag{3.14}$$

$$\bar{M}(\Delta r) = 0 \tag{3.15}$$

When the integrator reaches the boundary of the star ($p = 0$), I will have to compare the value of the scalar field to the value at infinity, The initial guess for the central scalar field is updated and the integration is done again. The value of the scalar field at infinity (φ_0) is related to the value of the scalar field at the surface of the star (φ_s) by the following

$$\varphi_0 = \varphi_s - \frac{1}{2}\alpha_A\hat{\nu}_s \tag{3.16}$$

This is a function of α_A and $\hat{\nu}_s$ which are defined at the surface of the star based on the following:

$$R \equiv r_s \tag{3.17}$$

$$\alpha_A \equiv \frac{2\psi_s}{\nu_s'} \tag{3.18}$$

$$Q_1 \equiv (1+\alpha_A^2)^{1/2} \tag{3.19}$$

$$Q_2 \equiv \left(1 - \frac{2M_s}{R}\right)^{1/2} \tag{3.20}$$

$$\hat{\nu}_s = -\frac{2}{Q_1}\tanh^{-1}\left(\frac{Q_1}{1+2(R\nu_S')^{-1}}\right) \tag{3.21}$$

$$\nu_s' = R\psi_s + \frac{2M_s}{R(R-2M_s)} \tag{3.22}$$

For the actual code, numerical scale factors are introduced to reduce numerical errors. Specifically: $r = r_0\hat{r}$, $M = r_0 M$, $\rho = \rho_0\hat{\rho}$, and $p = \rho_0\hat{\rho}$. Same as before, I choose $\rho_0 = \rho_c$. The choice of using r_0 for both mass and radius and ρ_0 for both density and pressure means that when substituted into the TOV equations (3.1-3.5) means that the equations remain largely the same except for a factor of $\rho_0 r_0^2$. However, r_0 can be chosen such that $\rho_0 r_0^2 = 1$ and the equations remain unchanged. (The EOS does need to be scaled).

According to [**?**] initial tests show that the value of the scalar field does not change significantly, si they choose the initial scalar field to be he same order of magnitude as the value at infnity. They restrain $\varphi_0 < 4.3 \times 10^{-3}$ for he 'spontaneous scalarization' case based on Demour and Esposito-Farese.

3.1 Equation of State

The thesis starts by using a polytropic equation of state and so I will do that as well. However, it's a different form than the ones used in previous sections. It's defined as two separate equations for pressure and density. **I will not be using this EOS in my write up!**

$$\rho = m_b n + \frac{K m_b n_0}{\Gamma - 1}\left(\frac{n}{n_0}\right)^\Gamma \tag{3.23}$$

$$p = K m_b n_0 \left(\frac{n}{n_0}\right)^\Gamma \tag{3.24}$$

There are several constants that need to be defined. Two are $n_0 = 0.1\text{fm}^{-3}$ and $m_b = 1.66 \times 10^{-27}\text{kg}$ The other two (K, γ) can vary. They define three seperate EOS as follows:

	Γ	K
EOS I	2.00	0.1
EOS II	2.34	0.0195
EOS III	2.46	0.00936

The paper has defined the constant K in such a way that you need to use the values given above for m_b and n_0 even though they are in different units. This was determined by comparing the equation given to other equations of state. Furthermore, the code returns the pressure and density in geometric units. To be used in SI, you need to transform with the appropriate factors of G and c.

It's obvious that this equation of state differs quite a bit from the others. In fact, it doesnt give reasonable mass radius curves when solved with GR. The way its formulated in the thesis is unclear. It gives a second formulation in the section about scaling for the numerical solver. I will attempt to implement this. They give the equation for $\hat{\rho}$ in terms of $\hat{p}$

$$\hat{\rho} = \frac{\hat{p}}{\Gamma - 1} + A\hat{p}^{1/\Gamma} \tag{3.25}$$

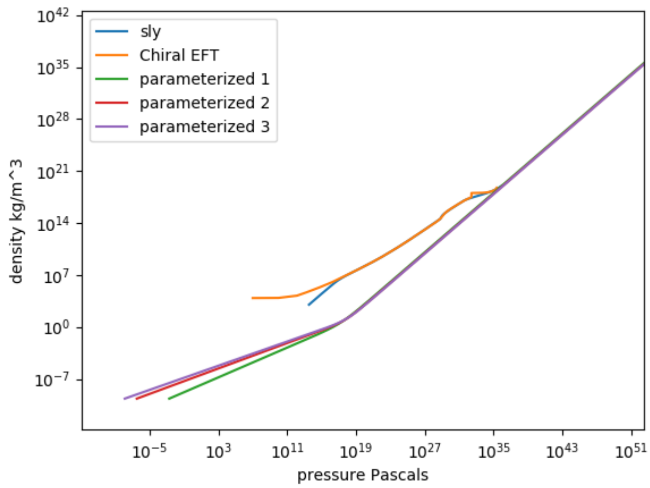


Figure 4: comparison of different EOS

Where the constant A is defined thus:

$$A = \frac{m_b n_0}{(K m_b n_0)^{1/\Gamma} \rho_c^{1-1/\Gamma}} \tag{3.26}$$

They then find the central pressure by solving

$$\frac{\hat{p_c}}{\Gamma - 1} + A\hat{p}^{1/\Gamma} = 1 \tag{3.27}$$

The explored a range of central densities between $(10^{15}, 10^{21})$ kg/m^3.

References

[1] Shmuel Balberg and Stuart L. Shapiro. The properties of matter in white dwarfs and neutron stars. 2000.

[2] Jill Knapp. Equation of state, 2011.

[3] Ann-Sofie Nielson. Lecture 5-2 polytropes, 2019.

[4] A. M. Oliveira, H. E. S. Velten, J. C. Fabris, and I. G . Salako. Newtonian view of general relativistic stars. *Eur. Phys. J.*

[5] Richard Silber and Sanjay Reddy. Neutron stars for undergraduates. *Am.J.Phys*, 72:892–905, 2004.