TOV Solver problem

March 26, 2020

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

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

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

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 ($M(\Delta r) = 0$). 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:

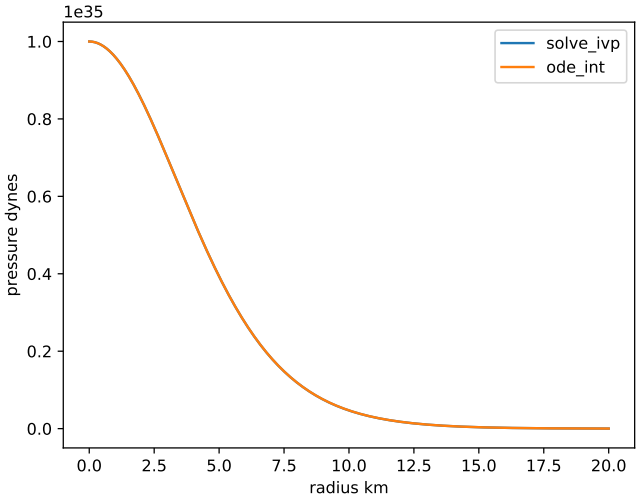
```
M0 = 0
p0 = 10.**34.
y0 = [M0,p0]

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
```

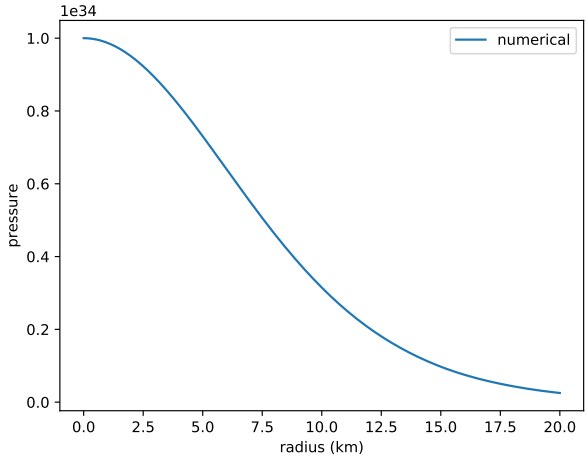
Now, I want to look at K. You can use the same equations to calculate it, so long as you have $\hbar, c, m_N$ in SI units. This yields a K of 1.23×10^{10}. Which is K_cgs $\times 10^5$. This is correct. You can check this because the units of K are $\mathrm{m}^3\ \mathrm{kg}^{-1/3}\ \mathrm{s}^{-2}$ which gives a factor of 10^5 when converted to $\mathrm{cm}^3\ \mathrm{g}^{-1/3}\ \mathrm{s}^{-2}$. 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

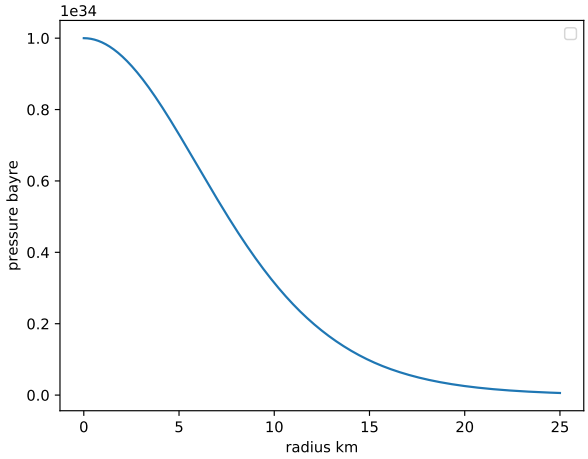
Again, we need to define the equation of state as well as the TOV equations.

```
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 $M_\odot$. 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 $M_\odot$.

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

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

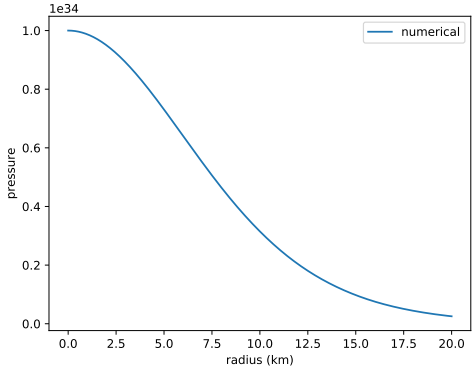
$$\frac{d(\bar{p}p_c)}{dr} = \frac{G(\bar{\rho}\rho_c)M}{r^2} \tag{1.11b}$$

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

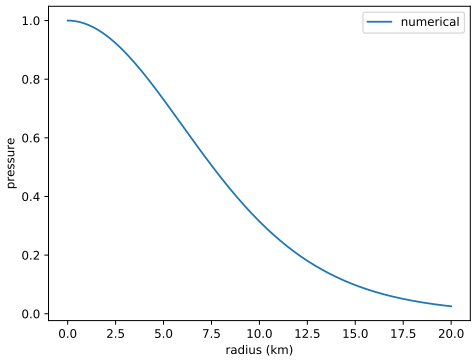
$$\frac{d\bar{p}}{dr} = \frac{G\bar{\rho}\rho_c M}{p_c r^2} \tag{1.11d}$$

The code for this is:

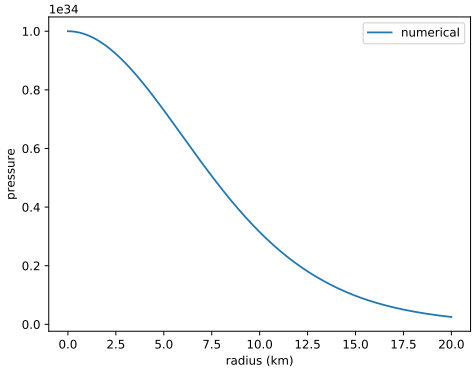
```
#equation of state
def EOS(p):

   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
```



(a) From Dimensionless Pressure Code



(b) From Dimensionless Pressure Code



(c) from SI Code

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.

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

1.5 EOS from a file

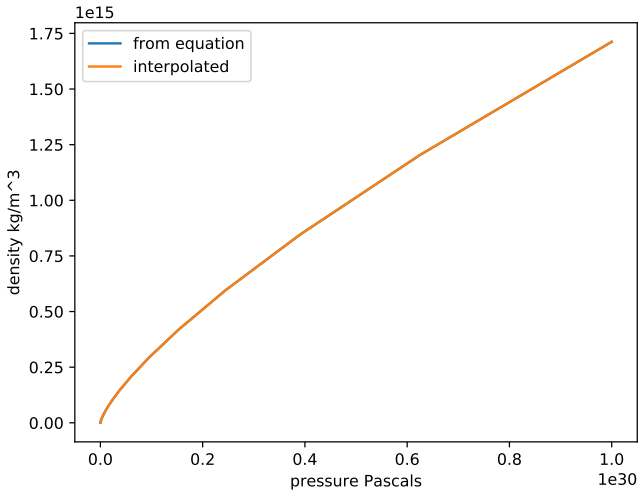
Since the EOS used for the real run will be from a table in a file, I want to set that up now. The first thing I did was generate a data file containing the tabulated equation of state. I generated the file in SI units, so that the code is ready to take a file in physical units. I'm not sure what units 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 $\mathrm{km}^{-2/3}$ 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.

2 General Relativistic

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.