TOV Solver problem

March 24, 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

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

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

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. 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 diff eq 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^{35}. 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 100 to get to cm. The code for this is:

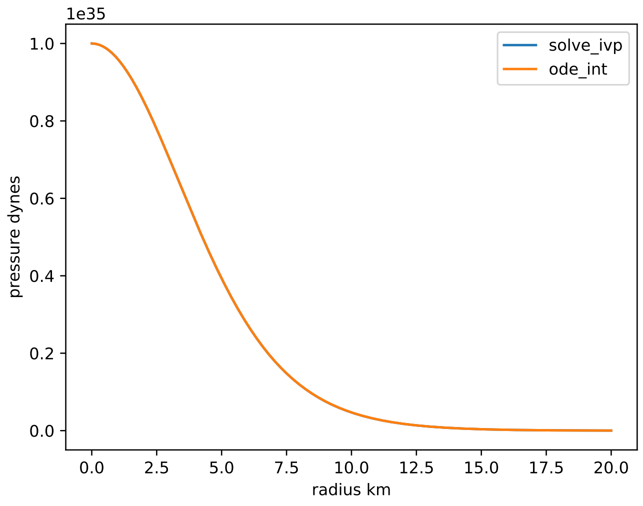
```
M0 = 0
p0 = 10.**34.
y0 = [M0,p0]

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
```

1.2 SI units

As another cross check, I tried to write the code from scratch in SI units

The constants defined in the Si system are:

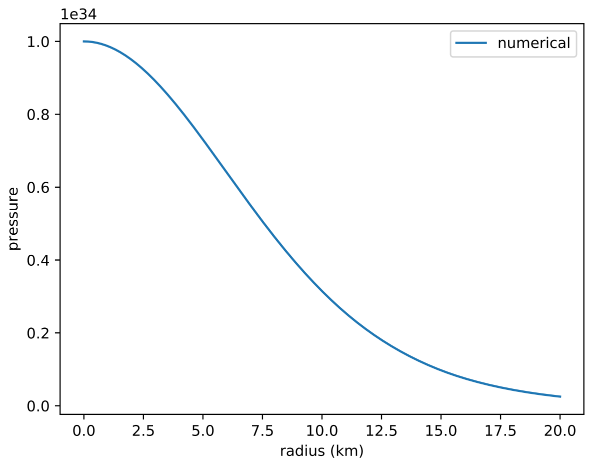
```
G = 6.67259 * 10.**(-11.) # m^3 kg^-1 s^-2
c = 3.00 * 10.**8. #m/s
hbar = 1.0546 * 10.**(-34.) # J * s
h = hbar*2.*np.pi
M_sun = 1.989 * 10.**(30.) # kg
n = 3./2.
gamma = (n+1.)/n

m_e = 9.11 * 10.**(-31.) # kg
m_h = 1.6749 * 10.**(-27.) # kg
```

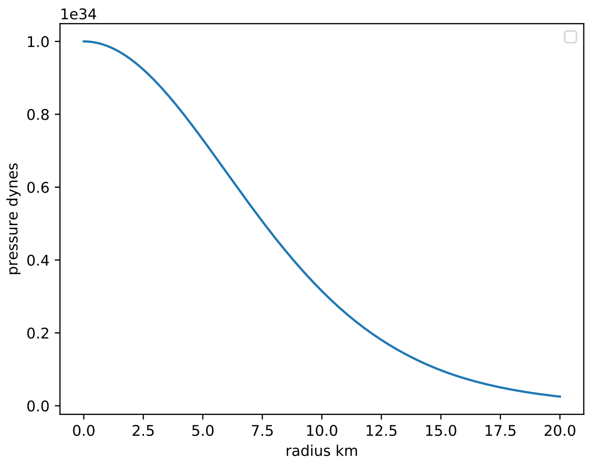
Now, I want to look at K. You can use the same equations to calculate it, so long as you have $\hbar, c, m_N$ in Si units. This yields a K of 1.23×10^{10}. Which is K_cgs $\times 10^5$. This is correct. You can check this because the units of K are m^3 kg$^{-1/3}$ s^{-2} which gives a factor of 10^5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





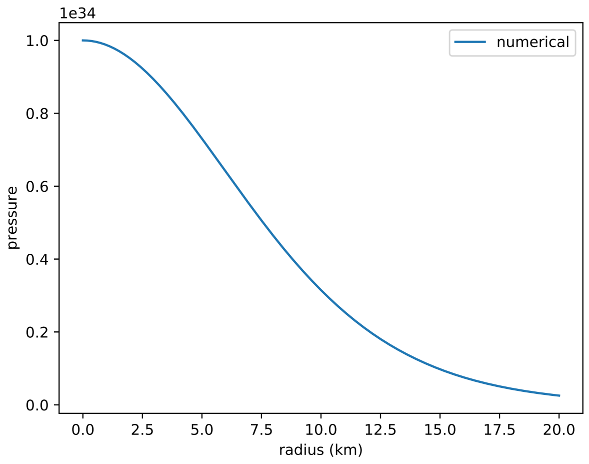
They agree! Furthermore all pressures give the same final mass ($M = 2.84 \times 10^{30}$ kg), which is expected in the n=3 case.

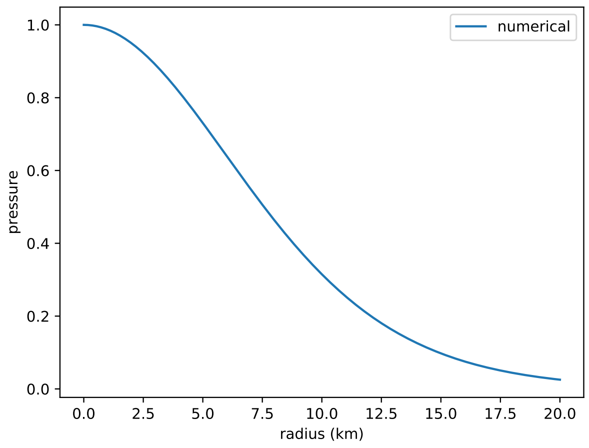
1.3 Geometric Units

I copied the SI unit code into a new file. I calculated the geometic 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$^{-1/3}c^{-4/3}$. I define the pressure in SI, then multiply by Gc^{-4} to get the geometric pressure. The initial mass is zero, which is the same in both units. After the integration, I convert the mass back to kg. The mass is consistent with previous results! ($M = 2.84 \times 10^{30}$ kg). The radii are also comparable

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

The next step is to use dimensionless density and pressure. This was successful, so I added it to the table above.





1.4 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.

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.