Tidal Deformabilities

August 31, 2020

1 Introduction

If we want to look at BNS in alternate theories of gravity using gravitational waves, we need to determine the tidal deformability. By definition the tidal deformability (sometimes called tidal polarizability) is the deformation a neutron star undergoes when exposed to tidal forces. Tidal forces induce quadrapole moments in the star. The deformation is described by the dimensionless Love number (k_2). k_2 depends on the neutron star structure and hence the mass and EOS.

The Love numbers were defined back in 1909 and 1912. They were initially defined to characterize the elastic response of the earth. There are three (h, k, l but only k is of interest here). k_2 is the cubical dilation (equal to the fractional change in volume). When a massive body deforms a new gravitational potential is induced as the radius changes from r to some $r \pm \delta r$. In this case the initial potential is($V(\theta, \phi)$) and the induced is ($k_2 V(\theta, \phi)$) [1].

2 Calculating GR Tidal Deformabilities

The tidal love number k_2 is a function of the compactness parameter ($\beta = GM/Rc^2$) and a quantity y_r [2–4].

$$
\begin{aligned}
k_2 = \frac{8\beta^5}{5}(1-2\beta)^2[2 - y_R + (y_R - 1)2\beta] \\
\times \Big[2\beta(6 - 3y_R + 3\beta(5y_R - 8)) \\
+ \ 4\beta^3\big(13 - 11y_R + \beta(3y_R - 2) + 2\beta^2(1 + y_R)\big) \\
+ \ 3(1-\beta)^2[2 - y_R + 2\beta(y_r - 1)]\ln(1-2\beta)\Big]^{-1}
\end{aligned}
$$

I cross checked this value in the three citations above. They use slightly different groupings of variables, but I confirmed that they are mathematically equivalent and also used both equaions in the code. I got the same value of k_2 for both.

```
k2 = ((8. * beta**5.)/5.) * (1. - 2. * beta)**2. * (2. - y_R + (y_R - 1.)
    * 2. * beta) \
       * ( 2. * beta * (6. - 3. * y_R + 3. * beta * (5. * y_R - 8.)) \
       + 4. * beta**3. * (13. - 11. * y_R + beta * (3.* y_R - 2.) + 2. *
          beta**2. * (1. + y_R)) \
       + 3. * (1. - 2. * beta)**2. * (2. - y_R + 2. * beta * (y_R - 1.)) *
          np.log(1. - 2. * beta)) **(-1.)


k2 = (8./5.) * (1.-2.* beta)**2. * beta**5. *( 2. * beta * (y_R - 1.)- y_R
    +2. ) \
       * (2 * beta * ( 4. * (y_R+1.)* beta**4. + (6. * y_R - 4.)* beta**3.
          + (26. - 22. * y_R)*beta**2. \
       + 3. * (5. * y_R-8.)* beta - 3.* y_R + 6.)+ 3*(1. - 2. * beta)**2. \
       *(2. * beta* (y_R - 1.) - y_R + 2.) * np.log(1. - 2. * beta))**(-1.)
```

The quantity y_r comes from a differential equation, in [4] . Where the functionals $F(r)$ and $Q(r)$ depend on energy density ($\mathcal{E}$), pressure (P), and mass (M).

$$r\frac{dy(r)}{dr} + y^2(r) + y(r)F(r) + r^2Q(r) = 0, \quad y(0) = 2 \quad y_R \equiv y(R) \tag{2.1}$$

$$F(r) = \left[1 - \frac{4\pi r^2 G}{c^4}(\mathcal{E}(r) - P(r))\right]\left(1 - \frac{2M(r)G}{rc^2}\right)^{-1} \tag{2.2}$$

$$\begin{aligned} r^2Q(r) = {} & \frac{4\pi r^2 G}{c^4}\left[5\mathcal{E}(r) + 9P(r) + \frac{\mathcal{E}(r) + P(r)}{dP(r)/d\mathcal{E}(r)}\right]\left(1 - \frac{2M(r)G}{rc^2}\right)^{-1} \\ & - \ 6\left(1 - \frac{2M(r)G}{rc^2}\right)^{-1}. \\ & - \ \frac{4M^2(r)G^2}{r^2c^4}\left(1 + \frac{4\pi r^3 P(r)}{M(r)c^2}\right)^2\left(1 - \frac{2M(r)G}{rc^2}\right)^{-2}. \end{aligned} \tag{2.3}$$

The code for these three equations is as follows:

```
   temp1 = (2. * M * G)/ (r * c**2.)
   dpde = deriv_fromfile(p)


   F = (1. - ((4. * pi * G * r**2.)/(c**4.)) * (E - p) ) * (1. -
      temp1)**(-1.)


   r2Q = ((4. * pi * G * r**2.)/(c**4.)) * (5. * E + 9. * p + ((E +
      p)/(dpde))) * (1. - temp1)**(-1.) \
```

```
    - 6. * ( 1.- temp1)**(-1.) \
    - temp1**2. * (1. + ((4. * pi * p * r**3.)/(M * c**2.)))**2. * (1.
       - temp1)**(-2.)
#print "r2Q = ", r2Q


dydr = - (1./r) * (yval**2. + yval * F + r2Q )
```

But where do these equations come from?

The quantity here, y_R is a function of H which are the even-parity metric perturbation ($H = H_0 = H_2$). Thorne and Campolattaro have shown that odd parity perturbations are metric only fluctuations (they do not affect pressure and density of the object). These perturbations satisfy the second order differential equation.

$$H'' + C_1 H' + C_0 = 0 \tag{2.4}$$

$$C_1 = \frac{2}{r} + \frac{1}{2}(\nu' - \lambda') = \frac{2}{r} + e^{\lambda}\left[\frac{2m}{r^2} + 4\pi r(p - e)\right] \tag{2.5}$$

$$\begin{aligned} C_0 &= e^{\lambda}\left[- \frac{\ell(\ell+1)}{r^2} + 4\pi(e+p)\frac{de}{dp} + 4\pi(e+p)\right] + v'' + (v')^2 + \frac{1}{2r}(2 - r\nu')(3\nu' + \lambda') \\ &= e^{\lambda}\left[\frac{\ell(\ell+1)}{r^2} + 4\pi(e+p)\frac{de}{dp} + 4\pi(5e+9p)\right] - (\nu')^2 \end{aligned} \tag{2.6}$$

λ and ν come from the line elment $ds^2 = -e^{\nu(r)}dt^2 + e^{\lambda(r)}dr^2 + r^2(d\theta^2 + sin^2\theta d\phi^2)$

y_r is the logarithmic derivative of the metric function H.

$$y_r = \frac{rH'(r)}{H(r)} \tag{2.7}$$

I need two more (small) equations.

First, my code is written in terms of mass density and I need to multiply by c^2 to get the energy density which appears in the equation above.

$$\mathcal{E}(r) = \rho(r)c^2 \tag{2.8}$$

Secondly, all of the other derivatives are in terms of r and for the functional $Q(r)$ I need $dP(r)/d\mathcal{E}(r)$. This can be obtained from the equation of state which gives the pressure as a function density. (Because energy density is related to the mass density as described above, the derivatives of the two are also related also by a factor of c^2

In order to use Equation 2.1 in the code, it needs to be put in the following form:

$$\frac{dy(r)}{dr} = -\frac{1}{r}\left(y^2(r) + y(r)F(r) + r^2Q(r)\right) \tag{2.9}$$

The object of interest for gravitational waves is the dimensionless tidal deformability. This is related to k_2 by the following:

$$\Lambda = \frac{2}{3}k_2\beta^{-5} \tag{2.10}$$

Given that the equations are written in physical units, I will start with the SI code. In paper [3] they calculate the tidal deformabilitis of polytropes. This gives me two options: polytropic equation of state or eos from Ingo. On the surface, the polytropes are easier, but locating the polytropic index is tricky.

2.1 Polytropes

The paper [3], as mentioned, calculates the Love numbers of polytropes. The results are given in terms of compactness. Ideally, I would use the polytrope equation of state and compare to her results to ensure my code is correct. However, I'm not sure what K value she uses in the polytrope. I can't tell if the relation is independent of K value (but it seems to be). So I will look at the compactnesses generated using the K values I have already in the code. Where compactness is

$$C = \frac{GM}{c^2R} \tag{2.11}$$

The paper looks at values of n between .3 and 1.2 ($1.83 < \gamma < 4.33$). I start with $\gamma = 3$ ($n = .5$). I calculate the relativistic K value using the piecewise polytrope from this lecture.

This gives me a range of compactnesses from 5×10^{-4} to 0.38.

I need to have the derivative of pressure with respect to energy density and since I am using the polytrope, I have an exact equation for the derivative. (This only works for polytropes, but it will allow me to check that my equations are working and then I can turn to how to properly define the derivative.). The values of k_2 seem reasonable. I have
$(C = 0.002, k_2 = 0.449)$ vs $(C = 10^{-5}, k_2 = 0.449)$
$(C = 0.14, k_2 = 0.169)$ vs $(C = 0.15, k_2 = 0.173)$
$(C = 0.20, k_2 = 0.096)$ vs $(C = 0.20, k_2 = 0.095)$
$(C = 0.26, k_2 = 0.047)$ vs $(C = 0.15, k_2 = 0.057)$

The next step then

2.2 Chiral EFT

The actual equations of state that I use are the Chiral- EFT equations provided by Ingo. These are in tables of pressure and energy density. I already set up the code to read in an equation of state from a file and use it in my TOV notes. Essentially its broken down into

1. read in file
2. Convert values from mev to SI units
3. define eos function by interpolating

The Code:

```
#-------------------------------------------------
#  EOS FROM FILE
#-------------------------------------------------

from_mev = 1.602176565 * 10.**(32) #This is to SI, to cgs its 10^33

# READ IN THE EOS

data = np.loadtxt("EOS_files/EOS_nsat_ind1.dat",skiprows=0,delimiter='\t')
data = data.transpose()
print np.shape(data)

eos_ps = data[1]
eos_ps = eos_ps * from_mev  #pressure (SI)
eos_Es = data[2]
eos_Es = eos_Es * from_mev   #Energy density (SI)
eos_rhos = eos_Es / c**2.   #Mass Density (SI)


def EOS_fromfile(p):
   rho = np.interp(p,eos_ps,eos_rhos)
   return rho
```

Looking at the equation for the functional $Q(r)$ you can see that there is a $dP(r)/d\mathcal{E}(r)$ term. Since the equation i'm using does not have a functional form that I can use to calculate the derivative (Like I did with the polynomials), I need another way to do it.

Recall the definition of the derivative:

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.12}$$

In this case we have $f(x) = P(\mathcal{E})$. It's not practical to find the limit of h, instead I choose a sufficiently small h.

I want to define a function that takes P as an input. In this case, I am started with $(f(x+h)-f(x)$ instead of x or h. Since I want the derivative at P, instead of taking P as one of the end points (either $f(x+h)$ or $f(x)$) I choose a small value δ and set $f(x+h) = P+\delta$ and $f(x) = P-\delta$. The next step is to calculate h. I do this using the EOS function to get rho at $P+\delta$ and $P-\delta$. Subtracting these two values gives h. This gives:

$$\frac{dP}{d\mathcal{E}} = \frac{(P+\delta)-(P-\delta)}{\mathcal{E}(P+\delta)-\mathcal{E}(P-\delta)} = \frac{2\delta}{\mathcal{E}(P+\delta)-\mathcal{E}(P-\delta)} \tag{2.13}$$

Choosing the value of δ is important because if it is too large then there will be errors introduced to the calculation. I fixed this value by trial and error. Since P (SI) is n the order of 10^{30}, I started with$\delta = 1$ and then rapidly increased because the difference was too small for the equation to work. I ended up with $\delta = 10^{20}$. This seems very large, but it is 10^{-10} times smaller than P. I tested this on the polynomial equation of state and compared it to the explicit derivation. With this value of $delta$, I got a percent error of 7×10^{-8}. Since I want the code to work for various values of P and to continue to work in geometric units (where $P \approx 10^{-10}$), I define δ relative to P instead of as an absolute value. Thus, $\delta = 10^{-10} \times P$.

The code for this is:

```
def deriv_fromfile(p):
    p2 = p + p*10**-10.
    p1 = p - p*10**-10.
    E2 = EOS_fromfile(p2) * c**2.
    E1 = EOS_fromfile(p1) * c**2.
    h = E2-E1

    deriv = 2.*p*10**(-10.)/h
    if p < 0:
        deriv = 0

    return deriv
```

Note: the integrator drops into negative values of pressure when it's in the last few steps. Since the EOS is only defined for $P > 0$, it doesnt make sense t find the derivative of the EOS when $P < 0$. So, I set the derivative = 0. (This value shouldnt matter because the integrator outputs only results for $P > 0$.

In order to test this, Ingo provided me with tables and plots from when he first developed his own code. It shows $F(r)$, r^2Q, and y_r for the SLy equation of state. It even breaks r^2Q down into the three terms.

1.
$$\frac{4\pi r^2 G}{c^4}\left[5\mathcal{E}(r)+9P(r)+\frac{\mathcal{E}(r)+P(r)}{dP(r)/d\mathcal{E}(r)}\right]\left(1-\frac{2M(r)G}{rc^2}\right)^{-1} \tag{2.14}$$

2.
$$-6\left(1-\frac{2M(r)G}{rc^2}\right)^{-1} \tag{2.15}$$

3.

$$-\frac{4M^2(r)G^2}{r^2c^4}\left(1+\frac{4\pi r^3P(r)}{M(r)c^2}\right)^2\left(1-\frac{2M(r)G}{rc^2}\right)^{-2} \tag{2.16}$$

Note I will attach the page of plots at the end for reference.

To do a 1:1 comparison. I need to use the FPS or SLy equation of state. I found the tables for these online. The tables are quite sparse, but the website also provides a code to calculate the eos. I downloaded the .c file. The current code is designed to be interactive. I'm going to change it so that it prints the data to a file instead of a screen. This is done by changing the main() function from the code. This portion f the code DOES NOT do any calculations, so I am not in danger of messing up the EOS. I generated a list of pressures similar to those in my chiral-eft files using a python script to make logarithmically spaced points then put those points through the code.

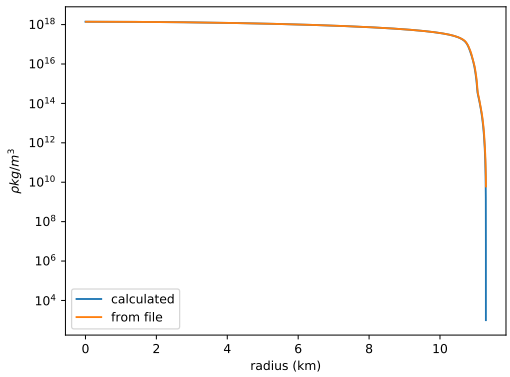
Also these tables generate pressure and mass density in CGS units, so the code to read in files will need to be adjusted slightly. (Recall that 1 Bayre (cgs) = 0.1 PA (SI) and 1 g/cm^3 = 10^3 kg/m^3.

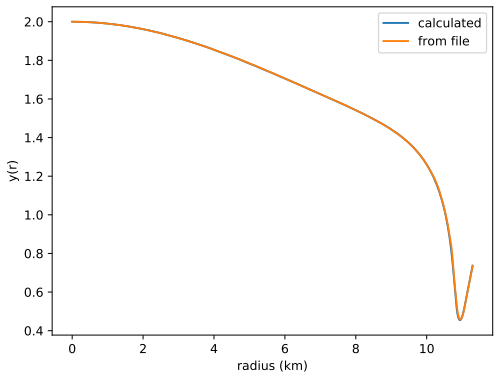
In order to compare the functionals, I had to reproduce them after the integration (It doesn't make sense to collect them in the integrator because it will loop back over time steps multiple times. I used the integrators output radius, pressure, and mass (using solve_ivp's dense_output='true').

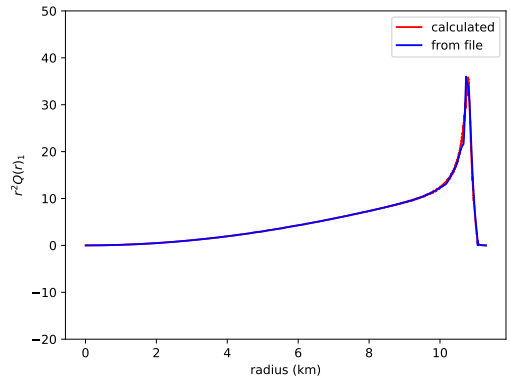
```
rho_plot = []
E_plot = []
F_plot = []
r2Q_1 = []
r2Q_2 = []
r2Q_3 = []


for x in range(0,len(p)):
   temp = EOS_fromfile(p[x])
   rho_plot.append(temp)
   E_plot.append(temp*c**2.)
   temp1 = (2. * m[x] * G)/ (r[x] * c**2.)

   dpde = deriv_fromfile(p[x])

   F_plot.append((1. - ((4. * pi * G * r[x]**2.)/(c**4.)) * (E_plot[x]
      - p[x]) ) * (1. - temp1)**(-1.))
   r2Q_1.append(((4. * pi * r[x]**2. * G )/ c**4.) * ( 5. * E_plot[x] +
      9. * p[x] + (E_plot[x]+p[x])/(dpde))*(1.-temp1)**(-1.))
   r2Q_2.append(- 6. * (1. - temp1)**(-1.))
   r2Q_3.append(- temp1**2. * (1. + ((4. * pi * r[x]**3. * p[x])/(m[x]
      * c**2.)))**2. * (1. - temp1)**(-2.))
```

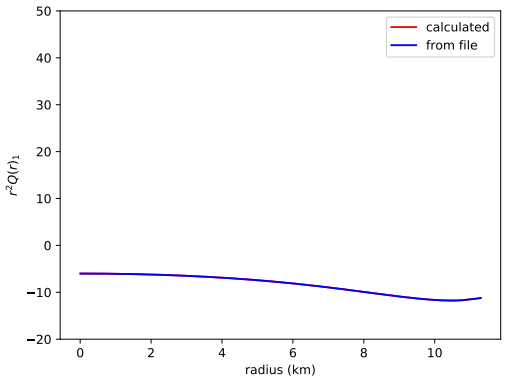
Then, I plotted the values calculated here against the ones provide by Ingo.

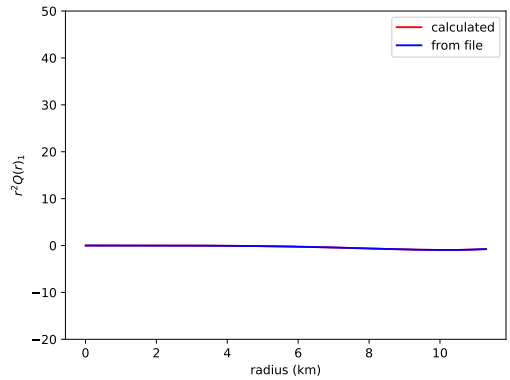
Using this, I ran the code with Chiral-EFT EOS ($index = 1$). The mass, radius, tidal deformability tables are available in the git repo for our paper, so I plotted the tidal deformability as a function of neutron star

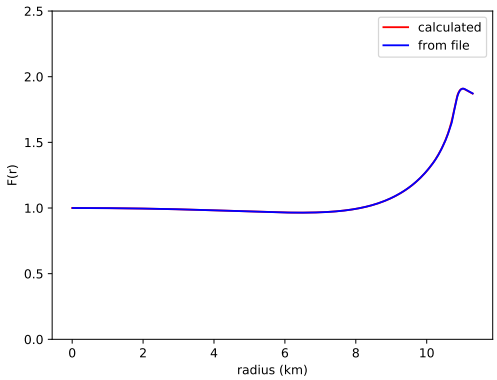

$\rho kg/m^3$
10^{18}
10^{16}
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
calculated
from file
0
2
4
6
8
10
radius (km)

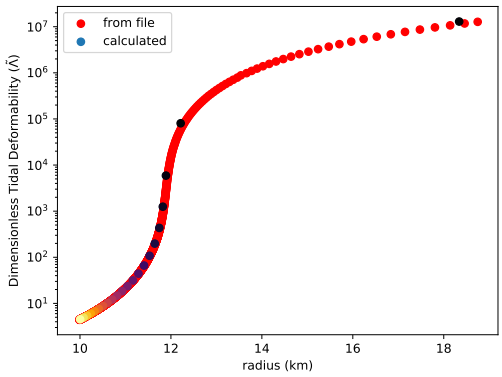

y(r)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
calculated
from file
0
2
4
6
8
10
radius (km)

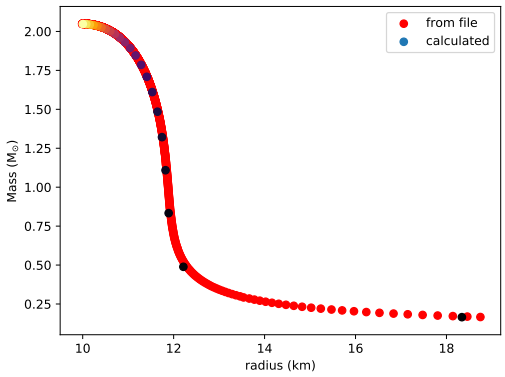

$r^2Q(r)_1$
50
40
30
20
10
0
−10
−20
calculated
from file
0
2
4
6
8
10
radius (km)

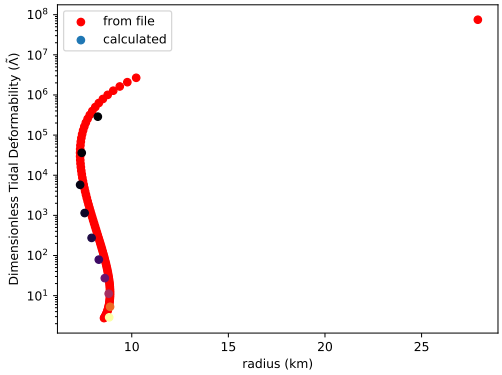

$r^2Q(r)_1$
50
40
30
20
10
0
−10
−20
calculated
from file
0
2
4
6
8
10
radius (km)


$r^2Q(r)_1$
50
40
30
20
10
0
−10
−20
calculated
from file
0
2
4
6
8
10
radius (km)


F(r)
2.5
2.0
1.5
1.0
0.5
0.0
calculated
from file
0
2
4
6
8
10
radius (km)







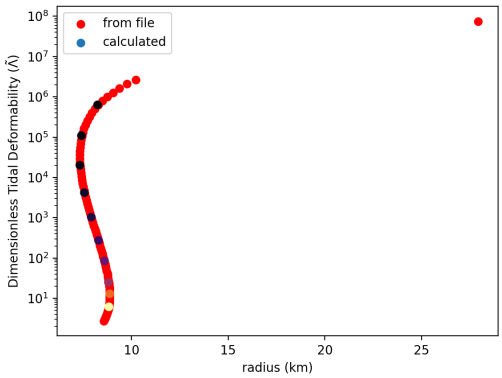
radius.

It sees that the answer is roughly correct, but there are large errors. This can be caused by using too large a step size in the integrators. I can test this by doing a cnvergence test with increasingly smaller step. To do this, I tested a range of step sizes for the same central pressure and output the tidal deformability. This showed a wide range of tidal deformabilities, which tells me that I need to use a smaller step size than the one needed for just the mass radius calculation (step_size = 1.).

It's obvious that the integrator is not converging with the step size initially used. I will need to do a run to see at what step size it converges and to what value. These runs take a long time, however. First, I will do a run at various pressures with a small step size to compare this to the actual values and see if this resolves the discrepancy between my results and Ingo's.

Table 1: $P_c = 10^{34}$

Step Size	Tidal deformability
1.0×10^{1}	1133
3.6×10^{0}	1139
1.3×10^{0}	1141
4.6×10^{-1}	1145
1.7×10^{-1}	1146
6.0×10^{-2}	4160
2.2×10^{-2}	3981
7.7×10^{-3}	4307
2.8×10^{-3}	4265
1.0×10^{-3}	4093



Since the values agree better, I will run a convergence test on atlas.

Running the code with an appropriate step size $\approx 10^{-3}$ is slow. I want to see if I can reduce the error in another way. One way is to try and make the derivative function more accurate. I tried to make the δP value smaller, but when I dropped it an order of magnitude (from 10^{-10} to 10^{-11}) the tidal deformability value was way off. The answer is also off if I go to a larger step size of 10^{-9}. I made a run to calculate the percent error from a range of different values. After I did a broad range show in the table below, I narrowed in on the segment where the error is the lowest (6.81×10^{-11} , 1.14×10^{-10}) to get a

In Progress

Table 2: EOS = 1nsat, SI unit code, step size = 4×10^{-3}

δP	Avg % Error (run 1)	Avg % Error (run 2)
3.16×10^{-11}	72%	72%
4.08×10^{-11}	66%	66%
5.27×10^{-11}	57%	57%
6.81×10^{-11}	47%	46%
8.80×10^{-11}	18%	19%
1.14×10^{-10}	24%	24%
1.47×10^{-10}	116%	116%
1.90×10^{-10}	1628%	1629 %
2.44×10^{-10}	1089%	1090%
3.16×10^{-10}	615%	616 %

Table 3: EOS = 1nsat, SI unit code, step size = 5×10^{-3}

δP	Avg % Error (run 1)	Avg % Error (run 2)
7.00×10^{-11}	45%	45
7.56×10^{-11}	40%	40
8.11×10^{-11}	32%	32
8.67×10^{-11}	21%	21
9.22×10^{-11}	15%	15
9.78×10^{-11}	5%	5
1.03×10^{-10}	7%	7
1.09×10^{-10}	16%	16
1.14×10^{-10}	25%	25
1.20×10^{-10}	36%	36

Table 4: EOS = 1nsat, SI unit code, $\delta P = 10^{-10}$

Step Size (m)	Avg % Error (run 1)	Avg % Error (run 2)
$1.00 \times 10^{+1}$	65%	65%
5.62×10^{-1}	65%	65%
3.16×10^{-2}	8.8%	8.8%
1.78×10^{-3}	3.8%	3.8%
1.00×10^{-4}	1.9%	1.9%

Table 5: EOS = SLY, SI unit code, $\delta P = 10^{-10}$

Step Size (m)	Avg % Error (run 1)	Avg % Error (run 2)
$1.00 \times 10^{+1}$	-%	%
5.62×10^{-1}	-%	%
3.16×10^{-2}	-%	%
1.78×10^{-3}	-%	%
1.00×10^{-4}	-%	%

2.3 Geometric Units

To reduce numerical errors, the code should be done in geometric units with dimensionless variables. For this to work, I need to change the diff eq for y_R. This is done by adjusting the equations for the functionals $F(r)$ and $Q(r)$, since there are no factors of G or c elsewhere in the equation. Luckily, $0 \leq y(r) \leq 2.5$ so this does not need to be scaled. I also want to introduce the dimensionless variables for P and ρ. The are defined as $P = \bar{P}P_c$ and $\mathcal{E} = \rho = \bar{\rho}\rho_c = \bar{\mathcal{E}}\mathcal{E}_c$

$$F(r) = \left[1 - 4\pi r^2(\mathcal{E}_c\bar{\mathcal{E}}(r) - P_c\bar{P}(r))\right]\left(1 - \frac{2M(r)}{r}\right)^{-1} \tag{2.17}$$

$$\begin{aligned} r^2Q(r) = 4\pi r^2 &\left[5\mathcal{E}_c\bar{\mathcal{E}}(r) + 9P_c\bar{P}(r) + \frac{\mathcal{E}_c\bar{\mathcal{E}}(r) + P_c\bar{P}(r)}{dP(r)/d\mathcal{E}(r)}\right]\left(1 - \frac{2M(r)}{r}\right)^{-1} \\ - \; & 6\left(1 - \frac{2M(r)}{r}\right)^{-1} \\ - \; & \frac{4M^2(r)}{r^2}\left(1 + \frac{4\pi r^3 P_c\bar{P}(r)}{M(r)}\right)^2\left(1 - \frac{2M(r)}{r}\right)^{-2} \end{aligned} \tag{2.18}$$

The issue of the derivative function is complex. I believe that I need the derivative of P in geometric units with respect to $\mathcal{E}/\rho$. The P in the TOV function is scaled. The main difference is the need to change from dimensionless pressure to pressure in geometric units. But the process goes as follows:

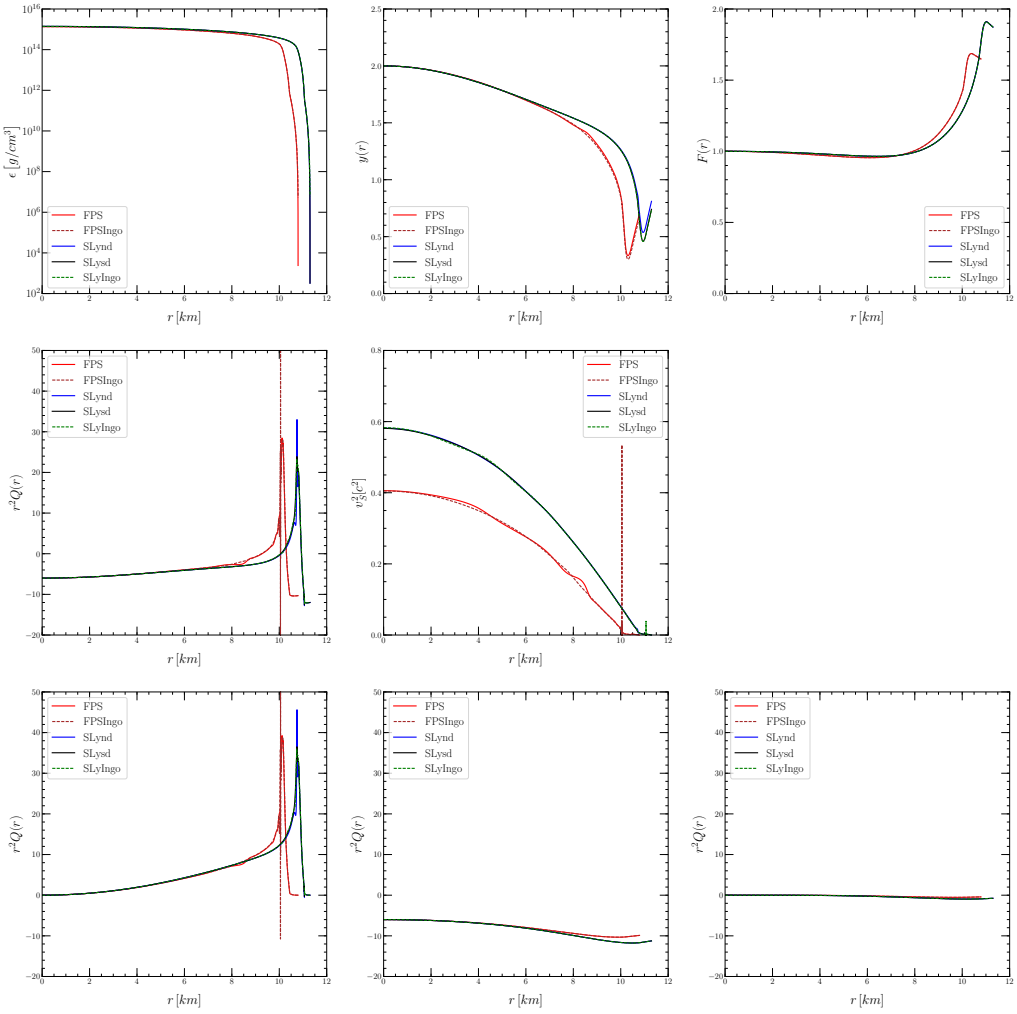
1. Take dimensionless pressure as a variable (same)
2. multiply by scale factor to get geometric pressure
3. use EOS_geometric_from file instead of EOS_scaled_fromfile to get geometric density ($\mathcal{E} = \rho$)
4. use the same algorithm as before to calculate the derivative

3 Scalar Tensor Theory

Tidal deformability is a function of the metric. The metric is the same in STT and GR. This might mean that the tidal deformability equations are THE SAME! I need to look at the details of the derivation though.

4 Tidal Deformabilities and Gravitational Waves

5 SLY Plots



References

[1] D. C. Agnew. *Earth Tides. Treatise on Geophysics and Geodesy.* New York. Elsevier, 2007.

[2] Cecilia Chirenti, Camilo Posada, and Victor Guedes. Where is love? tidal deformability in the black hole compactness limit. 2020.

[3] Tanja Hinderer. Tidal love numbers of neutron stars. *The Astrophysical Journal*, 677(2):1216–1220, Apr 2008.

[4] Ch. C. Moustakidis, T. Gaitanos, Ch. Margaritis, and G. A. Lalazissis. Bounds on the speed of sound in dense matter, and neutron star structure. *Physical Review C*, 95(4), Apr 2017.

[5] Junjie Zhao, Lijing Shao, Zhoujian Cao, and Bo-Qiang Ma. Reduced-order surrogate models for scalar-tensor gravity in the strong field regime and applications to binary pulsars and gw170817. *Physical Review D*, 100(6), Sep 2019.