PHYS 4347 Computational Project

Riley Smith, GTiD: 904084131

April 2025

1 Introduction and Background

Neutron stars (NS) are formed from the collapsed cores of massive stars that have exploded as supernovas. NS are incredibly dense objects, of the order of 10^{17} kg/m^3. Due to the density of NS, their internal structure and nature cannot be described accurately through the use of Newtonian physics. Instead, equations of state that describe the relationship between pressure and density are used.

These equations are developed through the use of theoretical modelling, astrophysical observations, and the consideration of other physics theories, such as quantum chromodynamics, and general relativity.

The purpose of this project is to numerically model different equations of state that describe the mass and radius of NS over a range of central densities. This is done through the integration of Tolman-Oppenheimer-Volkoff equations using a fourth-order Runge-Kutta method.

Three equations of state were modelled, originally developed by Wiringa et al. (1988), then parameterised for ease of integration by Kutschera & Kotlorz (1993). The equations of state are as follows:

AV14+UVII: $E(n) = 2.6511 + 76.744n - 183.611n^2 + 459.906n^3 - 122.832n^4$

UV14+UVII: $E(n) = 7.57891 - 1.23275n + 227.384n^2 - 146.596n^3 + 324.823n^4 - 120.355n^5$

UV14+TN1: $E(n) = 6.33041 - 28.1793n + 288.397n^2 - 65.2281n^3$

The results are shown in three plots: Mass v Density, Radius v Density, and Mass v Radius. These are compared to expected values from previous modelling of these equations of state. Figure 3a also includes empirical data from PSR J0030+0451 and PSR J0740+6620, this is in order to understand the validity of the equations of state. Figure 3a is also compared to Figure 4 from the paper by Lattimer & Prakash (2001).

2 Theoretical Framework

The equations that are used to solve the equations of state are as follows:
Structure equations:

$$\frac{du}{dR} = 4\pi\epsilon R^2$$

$$\frac{dt}{dR} = -4\pi R\left(P + \frac{u}{4\pi R^3}\right) \cdot \frac{P+\epsilon}{1 - \frac{2u}{R}}$$

Energy density and pressure from the equations of state:

$$\epsilon(n) = n\left[E(n) + m_n c^2\right]$$

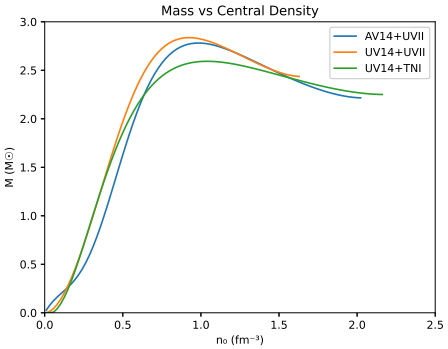
$$P(n) = n^2 \frac{dE}{dn}$$

$$u = \frac{M_r}{\tilde{M}}, \quad R = \frac{r}{\tilde{r}}$$

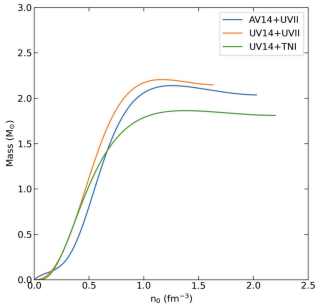
The initial conditions for the integration are as follows:

$$R = 0, \quad u = 0, \quad t = t_0$$

3 Results

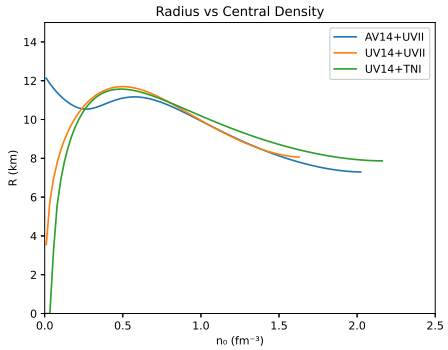


(a) Mass vs Density plotted from written code

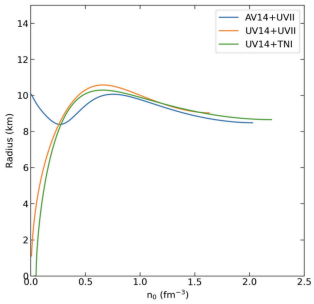


(b) Expected Plot of Mass vs Density

Figure 1: Plots of Mass vs Central Density

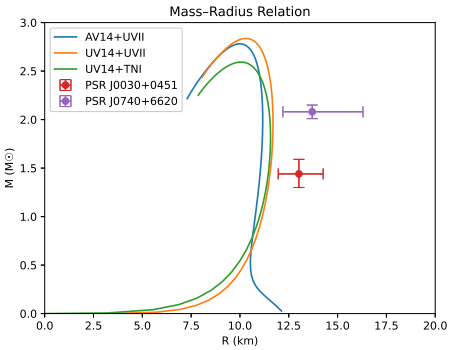


(a) Radius vs Density plotted from written code

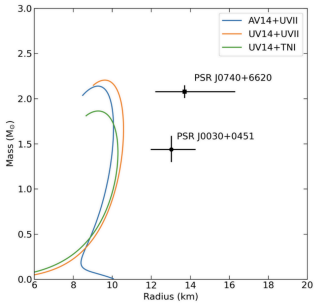


(b) Expected Plot of Radius vs Density

Figure 2: Plots of Radius vs Central Density



(a) Mass vs Radius plotted from written code



(b) Expected Plot of Mass vs Radius

Figure 3: Plots of Mass vs Radius

4 Discussion

The three equations of state that were plotted showed values with a $\sim 25\%$ inflation compared to that of the previously modelled equations of state, such as Figures 1b, 2b, and 3b. The issue that caused this was not found and thus should be investigated further in order to generate more accurate plots.

4.1 NS Mass v Density

As shown in Figure 1a, each equation of state follows the expected trend where mass increases with central density up to a maximum before sharply dropping off. The maximum mass is highest for the UV14+UVII model, indicating a stiffer equation of state that can support more mass against gravitational collapse. In contrast, the UV14+TNI model reaches its peak mass at a lower central density.

The UV14+UVII equation of state ends at the lowest central density, indicating that the NS becomes unstable first compared to other equations of state.

4.2 NS Radius v Density

In Figure 2a, all three models show that radius generally decreases with increasing central density. This is consistent with more compact, denser stars forming as pressure rises. The rate at which the radius decreases and the absolute values predicted differ between models. The UV14+TNI model predicts the smallest radii across most of the density range.

The UV14+TNI equation of state ends at the lowest central density, indicating that the NS becomes unstable first compared to other equations of state.

4.3 NS Mass v Radius

Figure 3a combines the trends of the other two plots and shows a clear peak in mass at a specific radius for each equation of state. Beyond this point, the mass drops off rapidly, where the star begins to be unstable. Observational data from PSR J0030+0451 and PSR J0740+6620 are also shown on the plot. These data points both UV14+UVII and UV14+TNI are most consistent with measured values, while neither passes through the recorded pulsar data, it could be reasoned that it is within an acceptable margin of error.

4.4 Comparison to Literature

Figure 4 from Lattimer & Prakash (2001) shows similar mass-radius trends. The shapes of our curves agree with theirs over the stable region. However, the models begin to diverge at high central densities.

5 Conclusion

This project explored the dependence of neutron star mass and radius on the equation of state used to describe dense nuclear matter. By numerically integrating the relativistic structure equations for a range of central densities, mass-radius relationships were generated for three parameterised equations of state: AV14+UVII, UV14+UVII, and UV14+TNI.

A Python Code

```
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy.constants import h as h_SI, c as c_SI, G as G_SI, physical_constants

# | convert constants to cgs |
h    = h_SI * 1e7               # Planck's constant [erg·s]
c    = c_SI * 1e2               # speed of light [cm/s]
G    = G_SI * 1e3               # gravitational constant [cm³/(g·s²)]
m_n  = physical_constants['neutron mass'][0] * 1e3   # neutron mass [g]

# | natural scales from O&V 1939 |
r_tilde   = (1/np.pi)*(h/(m_n*c))**1.5 * (c/np.sqrt(m_n*G))
M_tilde   = (1/np.pi)*(h/(m_n*c))**1.5 * (c**3/np.sqrt(m_n*G**3))
eps_scale = np.pi**2 * m_n**4 * c**5 / h**3
Msun    = 1.989e33    # g

# | EOS conversions |
MeV_to_erg = 1.602176634e-6
```

```
fm3_to_cm3 = 1e39
conv_P     = MeV_to_erg * fm3_to_cm3

# | polynomial fits for each EOS |
eos_coefs = {
    'AV14+UVII': [2.6511,  76.744,  -183.611, 459.906,  -122.832],
    'UV14+UVII': [7.57891, -1.23275, 227.384, -146.596, 324.823, -120.355],
    'UV14+TNI':  [6.33041, -28.1793, 288.397,  -65.2281]
}

def build_eos_table(coefs, n_max=2.5, N=1000):
    n     = np.linspace(0, n_max, N)
    P_t   = np.zeros(N)
    eps_t = np.zeros(N)
    dpdn  = np.zeros(N)

    for i, ni in enumerate(n):
        # polynomial E(n) and derivatives
        E_n     = sum(c*ni**j for j, c in enumerate(coefs))
        dE_dn   = sum(j*c*ni**(j-1) for j, c in enumerate(coefs) if j>0)
        d2E_dn2 = sum(j*(j-1)*c*ni**(j-2)
                      for j, c in enumerate(coefs) if j>1)

        # physical P, e, dP/dn
        P_phys   = conv_P * ni**2 * dE_dn
        eps_phys = (ni*fm3_to_cm3)*(E_n+939.0)*MeV_to_erg
        dpdn_phys= conv_P*(2*ni*dE_dn + ni**2*d2E_dn2)

        # convert to dimensionless
        P_t[i]   = P_phys   / eps_scale
        eps_t[i] = eps_phys / eps_scale
        dpdn[i]  = dpdn_phys/ eps_scale

    return n, P_t, eps_t, dpdn

def integrate_tov(nc, n_grid, P_t, eps_t, dpdn_t, h=1e-4):
    R   = 1e-6
    eps0= np.interp(nc, n_grid, eps_t)
    u   = (4/3)*np.pi*eps0*R**3
    t   = nc

    def derivs(R, u, t):
        P   = np.interp(t, n_grid, P_t)
        eps = np.interp(t, n_grid, eps_t)
        dp  = np.interp(t, n_grid, dpdn_t)
        if t<=0 or t>=n_grid[-1] or dp<=0:
```

```
            return None
        du   = 4*np.pi*R**2 * eps
        dPdr = - (eps+P)*(u+4*np.pi*R**3*P)/(R*(1-2*u/R))
        dt   = dPdr / dp
        return du, dt

    while True:
        if np.interp(t, n_grid, P_t) < 0:
            break
        k1 = derivs(R, u, t)
        k2 = derivs(R+0.5*h, u+0.5*h*k1[0], t+0.5*h*k1[1]) if k1 else None
        k3 = derivs(R+0.5*h, u+0.5*h*k2[0], t+0.5*h*k2[1]) if k2 else None
        k4 = derivs(R+  h,   u+  h*k3[0], t+  h*k3[1])   if k3 else None
        if None in (k1, k2, k3, k4):
            break
        u += (h/6)*(k1[0]+2*k2[0]+2*k3[0]+k4[0])
        t += (h/6)*(k1[1]+2*k2[1]+2*k3[1]+k4[1])
        R += h

    M    = u * M_tilde / Msun
    R_km = R * r_tilde / 1e5
    return M, R_km

# | compute results for each EOS |
nc_vals = np.linspace(0.01, 2.3, 100)
results = {}
for name, coefs in eos_coefs.items():
    n, P_t, eps_t, dpdn_t = build_eos_table(coefs)
    Ms, Rs = [], []
    for nc in nc_vals:
        M, R_km = integrate_tov(nc, n, P_t, eps_t, dpdn_t)
        Ms.append(M)
        Rs.append(R_km)
    results[name] = (np.array(Ms), np.array(Rs))

# | observational data |
obs = {
    'PSR J0030+0451': {'M':1.44, 'M_err':[0.14,0.15], 'R':13.02, 'R_err':[1.06,1.24]},
    'PSR J0740+6620': {'M':2.08, 'M_err':[0.07,0.07], 'R':13.7,  'R_err':[1.5, 2.6 ]}
}

# Save figures
subfolder = "Plots/"
os.makedirs(subfolder, exist_ok=True)

# 1) Mass vs. density w/ plunge cut
```

```
plt.figure()
for name, (Ms, _) in results.items():
    diffs = np.diff(Ms)               # step-by-step change
    idx   = np.where(diffs < -1.0)[0] # find plunge
    cut   = idx[0]+1 if len(idx) else len(Ms)
    plt.plot(nc_vals[:cut], Ms[:cut], label=name)
plt.xlabel('n_0 (fm^-3)')
plt.ylabel('M (M_o)')
plt.title('Mass vs Central Density')
plt.ylim(0,3); plt.xlim(0, 2.5); plt.legend()

filename = os.path.join(subfolder, "MassDensity.eps")
plt.gcf().savefig(filename, format="eps")

# 2) Radius vs. density w/ plunge cut
plt.figure()
for name, (_, Rs) in results.items():
    diffs = np.diff(Rs)
    idx   = np.where(diffs < -1.0)[0]
    cut   = idx[0]+1 if len(idx) else len(Rs)
    plt.plot(nc_vals[:cut], Rs[:cut], label=name)
plt.xlabel('n_0 (fm^-3)')
plt.ylabel('R (km)')
plt.title('Radius vs Central Density')
plt.ylim(0,15); plt.xlim(0, 2.5); plt.legend()

filename = os.path.join(subfolder, "RadiusDensity.eps")
plt.gcf().savefig(filename, format="eps")

# 3) Mass{Radius w/ plunge cut + error bars
plt.figure()
for name, (Ms, Rs) in results.items():
    diffs = np.diff(Rs)
    idx   = np.where(diffs < -1.0)[0]
    cut   = idx[0]+1 if len(idx) else len(Rs)
    plt.plot(Rs[:cut], Ms[:cut], label=name)
for key, d in obs.items():
    plt.errorbar([d['R']], [d['M']],
                 xerr=np.array(d['R_err']).reshape(2,1),
                 yerr=np.array(d['M_err']).reshape(2,1),
                 fmt='o', capsize=5, label=key)
plt.xlabel('R (km)')
plt.ylabel('M (M_o)')
plt.title('Mass{Radius Relation')
plt.ylim(0,3); plt.xlim(0,20); plt.legend(loc='upper left')
```

```
filename = os.path.join(subfolder, "MassRadius.eps")
plt.gcf().savefig(filename, format="eps")
plt.show()
```

B Lattimer & Prakash (2001) Figure 2

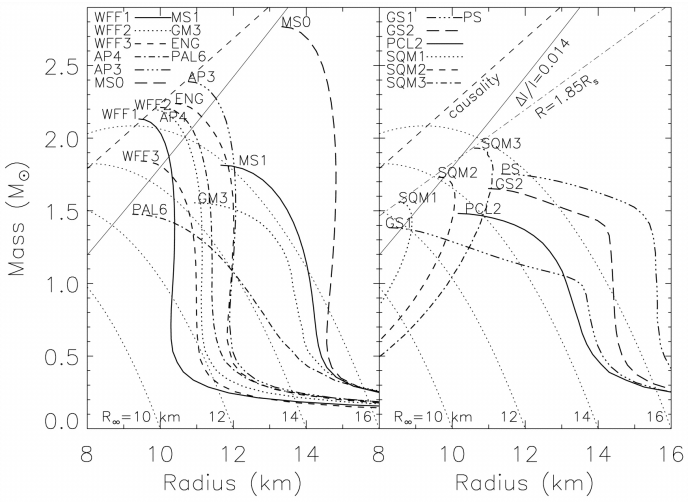


Figure 4: Figure 2 from J. M. Lattimer and M. Prakash showing NS Mass vs Radius