TOV_solver-v4

April 1, 2025

[1]:
```
import time
from dataclasses import dataclass
from enum import Enum
from typing import Callable, Optional

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import interpolate
```

1 Mass of a Neutron Star

This program finds the maximum mass of a neutron star by solving the Tolman-Oppenheimer-Volkov (TOV) equations:

$$\frac{dm}{dr} = 4\,\pi\,r^2\,\epsilon,$$

$$\frac{dp}{dr} = -G\,(\epsilon + p)\,\frac{m + 4\,\pi\,r^3\,p}{r\,(r - 2\,G\,m)}.$$

The initial conditions are $m = 0$ and $p = p_c$ at $r = 0$ (the center of the star). We integrate the TOV equations until we reach $p = 0$.

However, dm/dr and dp/dr both equal zero at $r = 0$. To address this issue, we solve a different form of the TOV equations from $r = 0$ to $r = \Delta r$:

$$\frac{1}{r^2}\,\frac{dM}{dr} = 4\,\pi\,\epsilon,$$

$$\frac{1}{r}\,\frac{dp}{dr} = -G\,(\epsilon + p)\,\frac{M/r^3 + 4\,\pi\,p}{1 - 2\,G\,M/r},$$

so that

$$m\big|_{r=\Delta r} \approx \frac{4\,\pi}{3}\,\epsilon_c\,(\Delta r)^3,$$

$$p\big|_{r=\Delta r} \approx p_c - 2\,\pi\,G\,(\epsilon_c + p_c)\,(\epsilon_c/3 + p_c)\,(\Delta r)^2.$$

Then we integrate the usual form of the TOV equations starting with these values at $r = \Delta r$.

We solve the TOV equations starting with several values of p_c and plot the mass versus the radius. At a certain value of p_c, the mass is maximal.

The design of this program was informed by lecture notes by Aaron Smith (2012) and by Paul Romatschke (2021). We thank Prof. Romatschke, who kindly provided us with his Mathematica implementation.

1.1 Units

The TOV equations are written in units where $\hbar = 1$ and $c = 1$. Therefore, all variables have units that are powers of energy. As unit of energy we use GeV.

- M: GeV – divide M by M_sun_GeV to get M as a fraction of the mass of the sun
- r: 1/GeV – multiply r by hbar_c_GeV_km to get r in km
- p and ϵ: GeV^4 – divide p and ϵ by p_convert := hbar_c_GeV_fm**3 / 1e3 to get p and ϵ in MeV / fm^3
- G: $1/(\mathrm{GeV})^2$ – multiply G by hbar_SI c_SI c_sqr_GeV_per_kg**2 to get G in m $(\mathrm{m} / \mathrm{s})^2$ / kg

[2]:
```
c_SI = 299792458  # m / s
hbar_SI = 6.626093e-34 / (2 * np.pi)  # J s
J_per_GeV = 1e9 * 1.60217653e-19
c_sqr_GeV_per_kg = c_SI**2 / J_per_GeV  # GeV / kg
M_sun_SI = 1.989e30  # kg
M_sun_GeV = M_sun_SI * c_sqr_GeV_per_kg  # GeV
hbar_c_GeV_m = hbar_SI * c_SI / J_per_GeV  # GeV m
hbar_c_GeV_km = hbar_c_GeV_m / 1e3  # GeV km
hbar_c_GeV_fm = hbar_c_GeV_m / 1e-15  # GeV fm
m_neutron_SI = 1.674927471e-27 # kg
m_neutron_GeV = m_neutron_SI * c_sqr_GeV_per_kg # GeV
G_SI = 6.6743e-11  # J m / kg**2
G = G_SI / (hbar_SI * c_SI * c_sqr_GeV_per_kg**2)  # 1 / (GeV)**2
p_convert = hbar_c_GeV_fm**3 / 1e3
print(f"{hbar_c_GeV_km=}, {hbar_c_GeV_fm=}, {J_per_GeV=}, {c_sqr_GeV_per_kg=},␣
↪{M_sun_GeV=}, {m_neutron_GeV=}, {G=}, {1/hbar_c_GeV_km=}, {p_convert=}")
```

```
hbar_c_GeV_km=1.973276737501815e-19, hbar_c_GeV_fm=0.19732767375018148,
J_per_GeV=1.60217653e-10, c_sqr_GeV_per_kg=5.6095889679323764e+26,
M_sun_GeV=1.1157472457217496e+57, m_neutron_GeV=0.9395654663408475,
G=6.70880673995125e-39, 1/hbar_c_GeV_km=5.067712911195661e+18,
p_convert=7.683586562615894e-06
```

[3]:
```
class EOSType(Enum):
    FUNCTION = 0
    DATA = 1

@dataclass
class EOS:
    name: str
    eos_type: EOSType
```

```
    eps: Optional[Callable]
    e: Optional
    p: Optional
    rho: Optional
    eta: Optional
    p_scale: np.float64
    e_scale: np.float64
    r_scale: np.float64
    m_scale: np.float64
    p_min: np.float64
    p_max: np.float64
```

[4]:
```
debug = False
debug_plots = False
plots = False
timer = False
store_central_pressures = True
```

[5]:
```
def solve_TOV(eps, delta_r, m_0, p_0, p_min):
    """Solve the TOV equations with initial values m_0 and p_0 at r_0 =␣
 ↪delta_r"""
    # i is the iteration index
    i = 0
    # stored variables
    r = np.array([delta_r])
    m = np.array([m_0])
    p = np.array([p_0])

    while i < 10000:  # just in case
        # append the list of r with the current value of r
        r = np.append(r, r[i] + delta_r)

        eps_i = eps(p[i])

        # append the list of m
        dmdr = 4 * np.pi * r[i] ** 2 * eps_i
        # m[i+1] = m[i] + dr * dm/dr
        m = np.append(m, m[i] + delta_r * dmdr)

        # append the list of p
        dpdr = (
            -G
            * (eps_i + p[i])
            * (m[i] + 4 * np.pi * r[i] ** 3 * p[i])
            / (r[i] * (r[i] - 2 * G * m[i]))
        )
        # p[i+1] = p[i] + dr * dp/dr
```

```
        p = np.append(p, p[i] + delta_r * dpdr)

        if p[i + 1] <= p_min:
            # interpolate to find the radius more accurately
            alpha = (p_min - p[i]) / (p[i + 1] - p[i])
            r[i + 1] = (1 - alpha) * r[i] + alpha * r[i + 1]
            m[i + 1] = (1 - alpha) * m[i] + alpha * m[i + 1]
            p[i + 1] = p_min
            break

        i += 1
    #else:
        #print("ERROR: p did not decrease to 0")

    return r, m, p
```

We need to use a special form of the TOV equation near r = 0 because in the usual form of TOV, dm/dr and dp/dr both vanish at r = 0.

[6]:
```
def initial_m(eps, p_c, delta_r):
    m_0 = (4 * np.pi / 3) * eps(p_c) * delta_r**3
    return m_0
```

[7]:
```
def initial_p(eps, p_c, delta_r):
    eps_c = eps(p_c)
    p_0 = p_c - 2 * np.pi * G * (eps_c + p_c) * (eps_c / 3 + p_c) * delta_r**2
    return p_0
```

[8]:
```
def initial_values(eps, p_min, p_max, delta_r):
    """Calculate the initial values m_0 and p_0 at r_0 = delta_r"""
    n_p_core = 100
    delta = 1e-5
    p_core_min = (1 - delta) * p_min + delta * p_max
    p_core_max = 0.75 * p_min + 0.25 * p_max
    p_core = np.logspace(np.log10(p_core_min), np.log10(p_core_max), n_p_core)
    # delta = 1e-8
    # p_core_min = (1 - delta) * p_min + delta * p_max
    # p_core_max = 0.75 * p_min + 0.25 * p_max
    # p_core = np.linspace(p_core_min, p_core_max, n_p_core)

    m_0 = initial_m(eps, p_core, delta_r)
    p_0 = initial_p(eps, p_core, delta_r)

    if debug:
        print(f"{p_core=}")
        print(f"{m_0=}")
        print(f"{p_0=}")
```

```
    return m_0, p_0
```

[9]:
```
def radius_mass_pressure_relation(eos):
    """Calculate the radius-mass-pressure relation for the specified EOS
    by solving the TOV equations for the specified EOS for several choices
    for the pressure at the core of the neutron star."""

    # integration step size
    n_steps = 500
    delta_r = eos.r_scale / n_steps

    # initial values at r = delta_r
    m_0, p_0 = initial_values(eos.eps, eos.p_min, eos.p_max, delta_r)

    R = np.zeros(len(p_0), dtype=np.float64)
    M = np.zeros(len(p_0), dtype=np.float64)
    P = np.zeros(len(p_0), dtype=np.float64)

    # n_plots = 5
    # plot_freq = len(p_0) / n_plots
    plot_times = [len(p_0)//2, len(p_0) - 1]

    for i in range(len(p_0)):
        r, m, p = solve_TOV(eos.eps, delta_r, m_0[i], p_0[i], eos.p_min)

        # if debug_plots and i % plot_freq == 0:
        if debug_plots and i in plot_times:
            r_values = r * hbar_c_GeV_km
            m_values = m / M_sun_GeV
            p_values = p / p_convert
            rlim = eos.r_scale * hbar_c_GeV_km
            mlim = eos.m_scale / M_sun_GeV
            plim = eos.p_scale / p_convert
            fig, ax = plt.subplots(1, 2, figsize=[2 * 6.4, 4.8])
            ax[0].set(xlabel=r"radius [$\text{km}$]", ylabel=r"mass␣
↪[$\text{M}_\text{Sun}$]")
            ax[0].set(xlim=[0.0, rlim], ylim=[0.0, mlim])
            ax[0].plot(r_values, m_values, ".")
            ax[1].set(xlabel=r"radius [$\text{km}$]", ylabel=r"pressure␣
↪[$\text{MeV}\cdot\text{fm}^{-3}$]")
            ax[1].set(xlim=[0.0, rlim], ylim=[0.0, plim])
            ax[1].plot(r_values, p_values, ".")
            plt.show()

        R[i] = r[-1] * hbar_c_GeV_km
        M[i] = m[-1] / M_sun_GeV
        P[i] = p[0] / p_convert
```

```
    if debug:
        print(R)
        print(M)
        print(P)

    return np.array([R, M, P])
```

[10]:
```
def analyze_model(eos):
    if timer:
        start = time.time()
    R_M_P = radius_mass_pressure_relation(eos)
    if timer:
        end = time.time()
        print((end - start) / 60, "m")

    if debug:
        print(f"{R_M_P[0]=}")
        print(f"{R_M_P[1]=}")

    title = eos.name
    fig, ax = plt.subplots(1, 2, figsize=[2 * 8.4, 4.8])
    ax[0].set(xlabel=r"radius [$\text{km}$]", ylabel=r"mass␣
 ↪[$\text{M}_\text{Sun}$]")
    ax[0].set(title=title)
    rlim = eos.r_scale * hbar_c_GeV_km
    mlim = eos.m_scale / M_sun_GeV
    plim = eos.p_scale / p_convert
    ax[0].set(xlim=[0.0, rlim], ylim=[0, mlim])
    ax[0].plot(R_M_P[0], R_M_P[1], ".")
    ax[1].set(xlabel=r"radius [$\text{km}$]", ylabel=r"core pressure␣
 ↪[$\text{MeV}\cdot\text{fm}^{-3}$]")
    ax[1].set(title=title)
    ax[1].set(xlim=[0.0, rlim], ylim=[0.0, plim])
    ax[1].plot(R_M_P[0], R_M_P[2], ".")
    plt.show()
```

1.2 Equations of state

[11]:
```
def plot_function_model(ax, eos):
    # ax.set(xlim=[0, eos.p_scale], ylim=[0, eos.e_scale])
    ax.set(xlim=[0, eos.p_scale / p_convert], ylim=[0.0, eos.e_scale /␣
 ↪p_convert])
    p_arr = np.linspace(eos.p_min, eos.p_max, 100)
    ax.plot(p_arr / p_convert, eos.eps(p_arr) / p_convert, label=eos.name)
    ax.legend()
```

```
def plot_data_model(ax, eos):
    ax[0].set(xlabel=r"pressure [$\text{MeV}\cdot\text{fm}^{-3}$]",␣
 ↪ylabel=r"internal energy density [$\text{MeV}\cdot\text{fm}^{-3}$]")
    ax[0].set(xlim=[0.0, eos.p_scale / p_convert], ylim=[0.0, eos.e_scale /␣
 ↪p_convert])
    ax[0].plot(eos.p / p_convert, eos.e / p_convert, label=eos.name)
    ax[1].set(xlabel=r"pressure [$\text{MeV}\cdot\text{fm}^{-3}$]",␣
 ↪ylabel=r"total energy density [$\text{MeV}\cdot\text{fm}^{-3}$]")
    ax[1].set(xlim=[0.0, eos.p_scale / p_convert], ylim=[0.0, eos.e_scale /␣
 ↪p_convert])
    ax[1].plot(eos.p / p_convert, (eos.e + eos.rho) / p_convert, label=eos.name)
    ax[1].plot(eos.p / p_convert, eos.eps(eos.p) / p_convert, "--", label=eos.
 ↪name + " interpolated")
    # ax[2].set(xlabel=r"pressure [$\text{MeV}\cdot\text{fm}^{-3}$]",␣
 ↪ylabel="enthalpy-1")
    # ax[2].plot(eos.p, eos.eta, label=eos.name)
    for axis in ax:
        axis.legend(fontsize=12)


def plot_model(ax, eos):
    if eos.eos_type == EOSType.FUNCTION:
        plot_function_model(ax, eos)
    else:
        plot_data_model(ax, eos)
```

1.2.1 Toy model: MIT bag model

The equation of state for the MIT bag model is:

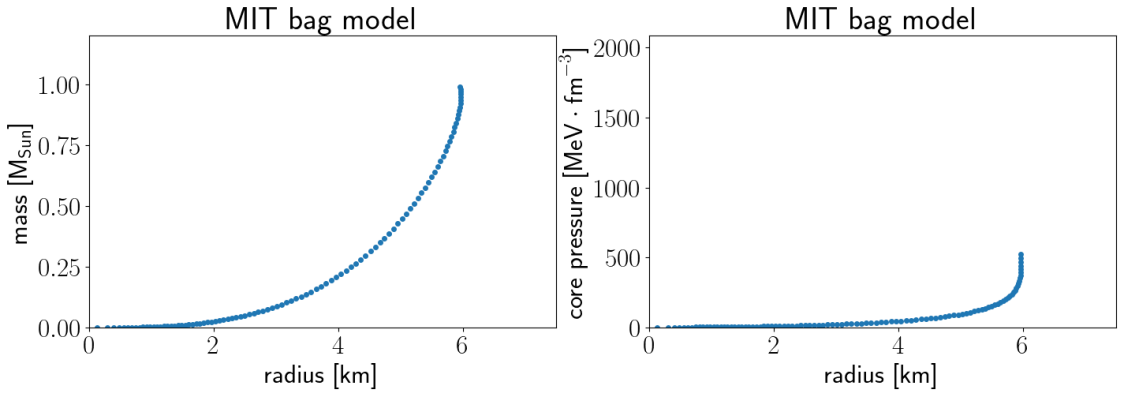
$$\epsilon(p) = 3\,p + 4\,B^4.$$

Here B is the bag constant, with value 0.2 GeV.

[12]:
```
# MIT bag model EOS

# bag constant B:
B = 0.2  # GeV


def eps_bag(p):
    return 3 * p + 4 * B**4


def bag_model():
    # p_scale: typical pressure at core of a neutron star
    p_scale = 10 * B**4
```

```
    eos = EOS(
        name="MIT bag model",
        eos_type=EOSType.FUNCTION,
        eps=eps_bag,
        e=None,
        p=None,
        rho=None,
        eta=None,
        p_scale=p_scale,
        p_min=1e-3 * p_scale,
        p_max=p_scale,
        e_scale=eps_bag(p_scale),
        r_scale=7.5 / hbar_c_GeV_km,
        m_scale=1.2 * M_sun_GeV,
    )
    return eos


if plots:
    fig, ax = plt.subplots(1, 1, figsize=[8.4, 4.8])
    ax.set(xlabel=r"pressure [$\text{MeV}\cdot\text{fm}^{-3}$]",␣
 ↪ylabel=r"energy density [$\text{MeV}\cdot\text{fm}^{-3}$]")
    eos = bag_model()
    plot_model(ax, eos)
    plt.show()
```

[13]:
```
eos = bag_model()
analyze_model(eos)
```



1.2.2 Models from I. Tews

[14]:
```
# opens .dat files to be converted into CSVs
# change pathprefix on different computers
EOS_files = "/home/tplohr/proj/SF 24-25/data/Peps_024_new/"
mass_radius_files = "/home/tplohr/proj/SF 24-25/diettim NMMA master␣
 ↪EOS-chiralEFT_MTOV/"


def load_EOS_file(number):
    file = open(EOS_files + str(number) + ".dat", "r")
    return file


def load_mass_radius_file(number):
    file = open(mass_radius_files + str(number) + ".dat", "r")
    return file
```

[15]:
```
def plot_mass_radius(eos_number):
    f = load_mass_radius_file(eos_number)
    df = pd.read_csv(f, delimiter="\t", names=["r", "m", "deform"])
    R_M = np.array([df["r"], df["m"]])
    # ax.set(xlim=[0, eos.p_scale], ylim=[0, eos.e_scale])
    title = f"{eos_number}.dat"
    fig, ax = plt.subplots(1, 2, figsize=[2 * 8.4, 4.8])
    ax[0].set(xlabel=r"radius [$\text{km}$]", ylabel=r"mass␣
 ↪[$\text{M}_\text{Sun}$]")
    ax[0].set(title=title + " from Tews")
    rlim = eos.r_scale * hbar_c_GeV_km
    mlim = eos.m_scale / M_sun_GeV
    ax[0].set(xlim=[0.0, rlim], ylim=[0.0, mlim])
    ax[0].plot(R_M[0], R_M[1], ".")
    plt.show()
```

[16]:
```
def tews_model(eos_number):
    f = load_EOS_file(eos_number)
    # n = number density of neutrons, p = pressure, e = total energy density
    df = pd.read_csv(f, delimiter="\t", names=["n", "p", "eps"])
    # n data is in units of fm^-3
    # p and eps data is in units of MeV / fm^3
    p_EOS = np.array(df["p"]) * p_convert
    eps_EOS = np.array(df["eps"]) * p_convert
    rho_EOS = np.array(df["n"]) * m_neutron_GeV * hbar_c_GeV_fm**3
    e_EOS = eps_EOS - rho_EOS
    eta_EOS = (e_EOS + p_EOS) / rho_EOS
    eps = interpolate.interp1d(p_EOS, eps_EOS)
    p_scale = np.max(p_EOS)
```

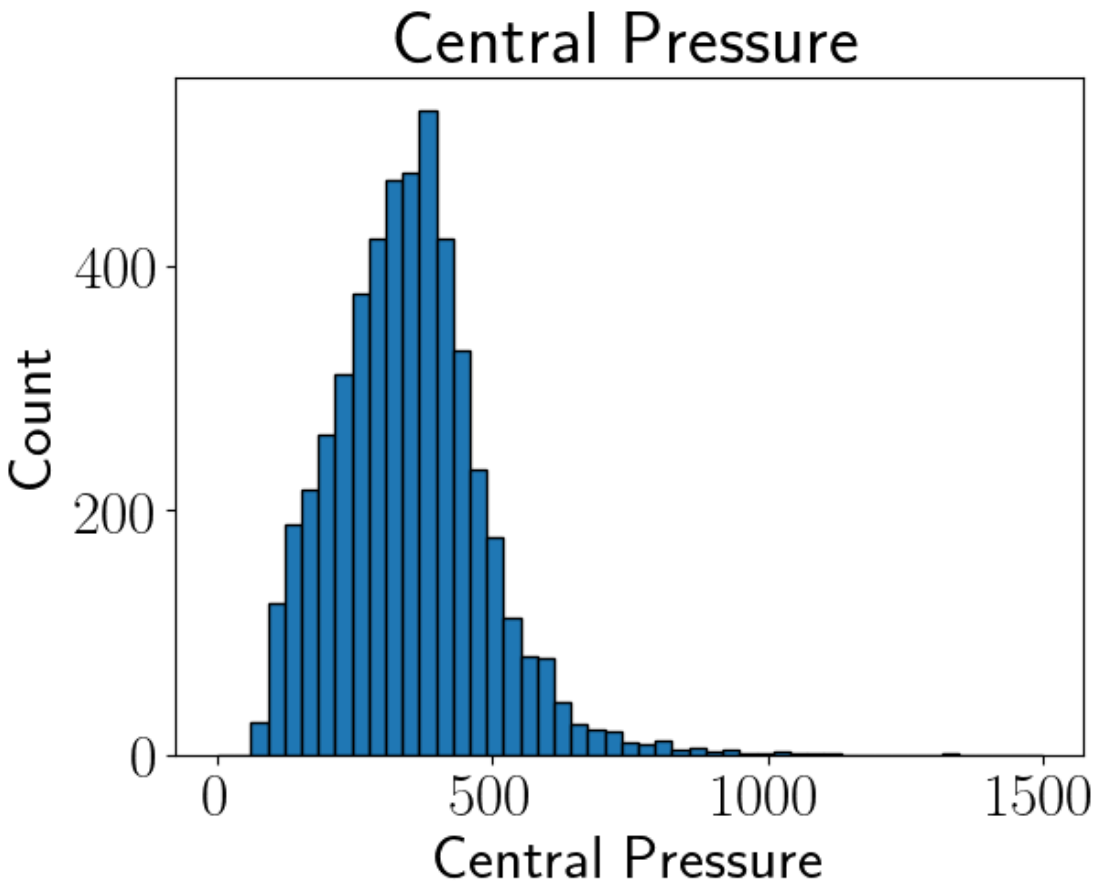
```
    delta = 1e-10
    eos = EOS(
        name=f"{eos_number}.dat",
        eos_type=EOSType.DATA,
        eps=eps,
        p=p_EOS,
        e=e_EOS,
        rho=rho_EOS,
        eta=eta_EOS,
        p_scale=p_scale,
        p_min=(1 - delta) * np.min(p_EOS) + delta * p_scale,
        p_max=p_scale,
        e_scale=np.max(eps_EOS),
        r_scale=20.0 / hbar_c_GeV_km,
        m_scale=3.5 * M_sun_GeV,
    )
    return eos


n_eos = 5000
freq = 100
eos_numbers = [(j + 1) * freq for j in range(n_eos // freq)]
# eos_numbers = [1023, 2101, 3500, 4900]

if plots:
    fig, ax = plt.subplots(1, 2, figsize=[2 * 8.4, 4.8])
    for eos_number in eos_numbers:
        eos = tews_model(eos_number)
        plot_model(ax, eos)
    plt.show()
```

[ ]:
```
if store_central_pressures:
    central_pressures = np.zeros(5000)
    for i in range(5000):
        eos = tews_model(i+1)
        R_M_P = radius_mass_pressure_relation(eos)
        central_pressures[i] = R_M_P[2][-1]
        print(i, "/5000")
```

[ ]:
```
import os
current_time = time.localtime()
filename = time.strftime('%m-%d_%H', current_time)  # Month-Day_Hour (e.g.␣
 ↪01-20_15)
file_path = os.path.join("/home/tplohr/proj/SF 24-25/saves/TOV/", f"{filename}.
 ↪txt")

# Save the probabilities to the text file
```

```
np.savetxt(file_path, central_pressures)
```

[21]:
```
bins1 = np.linspace(0, 1500, 50)
plt.xlabel("Central Pressure")
plt.ylabel("Count")
plt.title("Central Pressure")
plt.hist(central_pressures, bins=bins1, edgecolor = "black")
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/histograms/
 ↪hist_eps_start_posterior")
plt.show()
```



[ ]:
```

```