

```

#generic import and constant definition list
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import h5py
import astropy.constants as cons
from matplotlib.colors import LogNorm
import astropy.units as u
import pandas as pd
import scipy.optimize as opt
#all of the important fundamental constants are put into cgs units
just for convenience
c=cons.c.cgs.value
G=cons.G.cgs.value
h=cons.h.cgs.value
hbar=cons.hbar.cgs.value
Msun=cons.M_sun.cgs.value
Rsun=cons.R_sun.cgs.value
Rearth=cons.R_earth.cgs.value
mp=cons.m_p.cgs.value
me=cons.m_e.cgs.value
mn=cons.m_n.cgs.value
kB=cons.k_B.cgs.value
mu_e=2 #mean mass per electron for He-core or C/O core composition
m_u = 1/cons.N_A.cgs.value #atomic mass unit in grams
from astropy.io import fits

```

part b.)

```

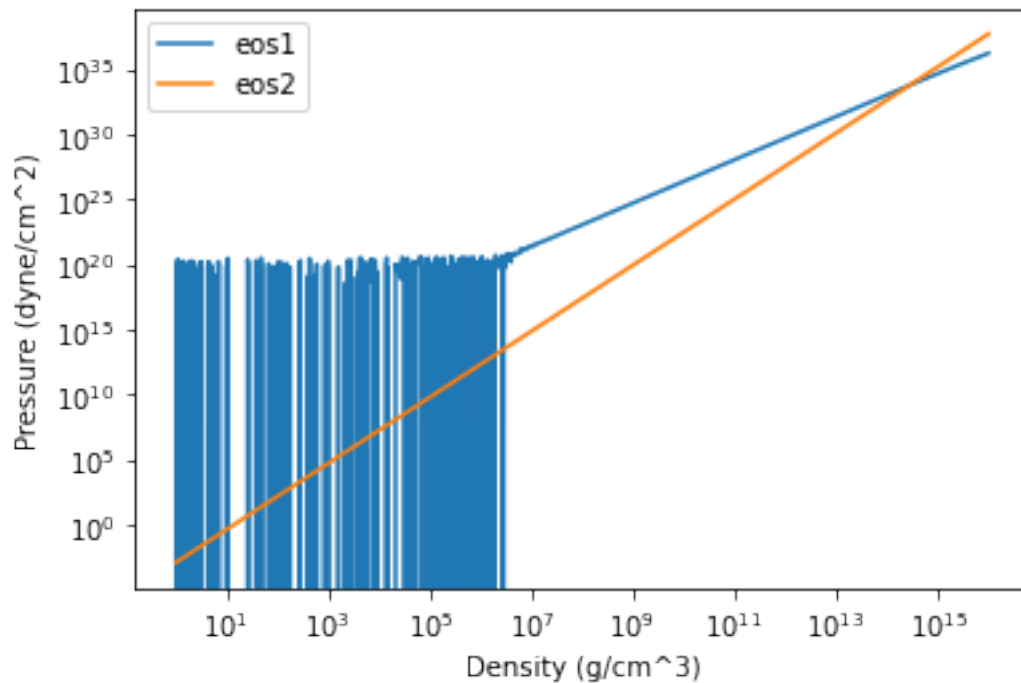
def eos1(rho):
    x = hbar*((3*np.pi**2*rho/mn)**(1/3))/(mn*c)
    phi = (1/(8*np.pi**2))*(x*np.sqrt(1+x**2)*(2*(x**2)/3 -
1)+np.log(x+np.sqrt(1+x**2)))
    P = (mn*(c**2)/(hbar/(mn*c))**3)*phi
    return P

def eos2(rho):
    #n is in cm^-3
    return 5.83e35*((rho*1e-39/mn)**(2.54))

rho = np.logspace(0,16,1000)
P1 = eos1(rho)
P2 = eos2(rho)
plt.plot(rho,P1,label='eos1')
plt.plot(rho,P2,label='eos2')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Density (g/cm^3)')
plt.ylabel('Pressure (dyne/cm^2)')

```

```
plt.legend()
plt.show()
```



I don't know why when I use log space I get those weird wiggles for low density. The only thing I could think of is that in that region it's no longer a valid equation of state.

c.)

```
def eos1_zero(rho,Pin):
    x = hbar*(3*np.pi**2*rho/mn)**(1/3)/ (mn*c)
    n_n = (x**3)/(3*(np.pi**2)*(hbar/(mn*c))**3)
    phi = (1/(8*np.pi**2))*(x*np.sqrt(1+x**2)*(2*(x**2)/3 -
1)+np.log(x+np.sqrt(1+x**2)))
    P = (mn*(c**2)/(hbar/(mn*c))**3)*phi
    return P - Pin

def eos2_zero(rho,Pin):
    return 5.83e35*((rho*1e-39/mn)**(2.54)) - Pin

def get_rho_from_P_eos1(P):
    #guessRho = (P-1e10)/1e19
    guessRho = np.copy(P)
    guessRho_nonRel = (P/(5.3802e9))**(3/5)
    guessRho_rel = (P/(1.2293e15))**(3/4)

    guessRho[guessRho_rel<6e15] = guessRho_nonRel[guessRho_rel<6e15]
    guessRho[guessRho_rel>=6e15] = guessRho_rel[guessRho_rel>=6e15]
```

```

    rho1 = opt.fsolve(eos1_zero,guessRho,args= (P))
    return rho1

rho = np.logspace(0,16,100)
P1 = eos1(rho)
P2 = eos2(rho)
plt.plot(rho,P1,label='eos1')
plt.plot(rho,P2,label='eos2')

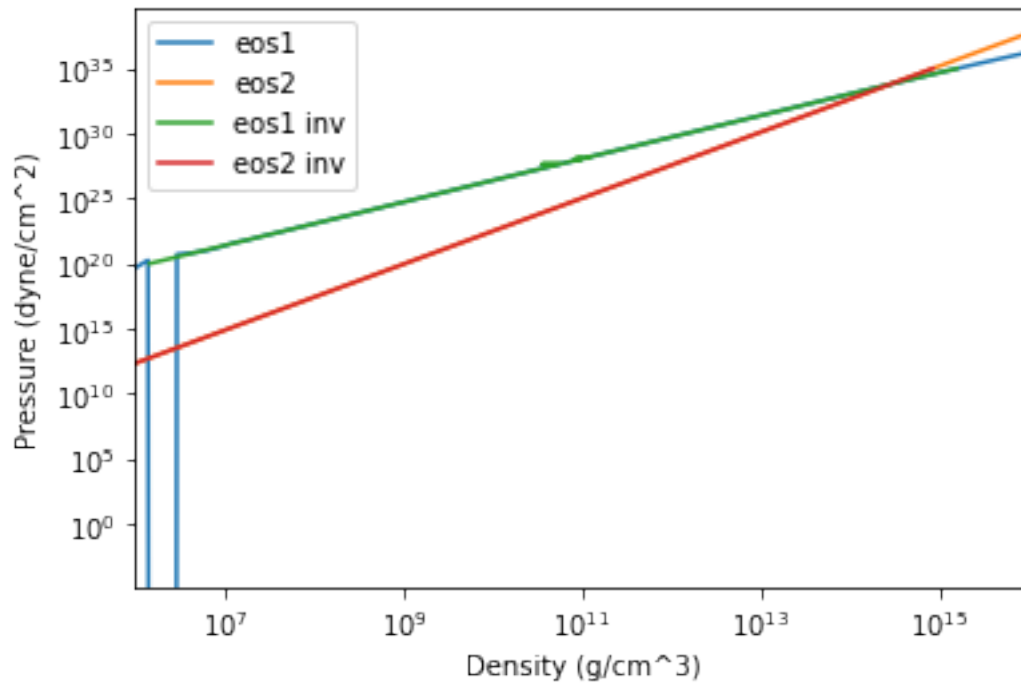
Pa = np.logspace(20,35,1000)
#I got this guess just by eyeballing the plot and then messing around
with it to get a linear approximation
rho1 = get_rho_from_P_eos1(Pa)
plt.plot(rho1,Pa,label='eos1 inv')

P = np.linspace(1e0,1e35,1000)
guessRho2 = ((P/5.83e35)**(1/2.54))*mn*1e39
plt.plot(guessRho2,P,label='eos2 inv')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Density (g/cm^3)')
plt.ylabel('Pressure (dyne/cm^2)')
plt.xlim(1e6,1e16)
plt.legend()
plt.show()

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/
ipykernel_launcher.py:2: RuntimeWarning: invalid value encountered in
power

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/scipy/
optimize/minpack.py:175: RuntimeWarning: The iteration is not making
good progress, as measured by the
improvement from the last ten iterations.
warnings.warn(msg, RuntimeWarning)

```



d.)

```
def dydr(m,P,phi,r,EOS):
    if EOS == 1:
        rho = get_rho_from_P_eos1(P)
    elif EOS == 2:
        rho = ((P/5.83e35)**(1/2.54))*mn*1e39
        #note that this is in the GR form from eqn 9.15-9.17 in the notes
    dmdr = 4*np.pi*(r**2)*rho
    dPdr = -G*(m+(4*np.pi*P*r**3)/c**2)*(rho+P/c**2)/((r**2)*(1-
2*G*m/(r*c**2)))
    #dPdr = -G*m*rho/(r**2)
    dphidr = (G*m/((c**2)*r**2))*(1+(4*np.pi*P*r**3)/(m*c**2))/(1-
2*G*m/(r*c**2))
    dphidr = G*m/(r**2)
    #print(dphidr)

    return np.array([dmdr,dPdr,dphidr])
```

e.)

```
def rk4_step(r, y, dr=0.5, EOS=1):
    #performs a single RK4 step
    #taken from my homework 1 and just slightly modified
    m = y[0]
    P = y[1]
    phi = y[2]
```

```

    k1 = dydr(m, P, phi, r, EOS=EOS)
    k2 = dydr(m + dr/2*k1[0], P + dr/2*k1[1], phi + dr/2*k1[2], r +
dr/2, EOS=EOS)
    k3 = dydr(m + dr/2*k2[0], P + dr/2*k2[1], phi + dr/2*k2[2], r +
dr/2, EOS=EOS)
    k4 = dydr(m + dr*k3[0], P + dr*k3[1], phi + dr*k3[2], r + dr,
EOS=EOS)

    dm = (k1[0] + 2*k2[0] + 2*k3[0] + k4[0]) * dr / 6
    dP = (k1[1] + 2*k2[1] + 2*k3[1] + k4[1]) * dr / 6
    dphi = (k1[2] + 2*k2[2] + 2*k3[2] + k4[2]) * dr / 6
    m_new = m + dm
    P_new = P + dP
    phi_new = phi + dphi

    return m_new, P_new, phi_new

```

f.)

```

def integrate(r0,y0, dr=500, EOS=1):
    #taken from my homework 1 and modified slightly
    m0 = y0[0]
    P0 = y0[1]
    phi0 = y0[2]
    maxIterations=100000
    i=0
    r = np.array([r0])
    m = np.array([m0])
    P = np.array([P0])
    phi = np.array([phi0])
    P_central = P0
    while i < maxIterations:
        # Perform a single RK4 step
        m_new, P_new, phi_new = rk4_step(r[-1], np.array([m[-1], P[-
1], phi[-1]]), dr=dr, EOS=EOS)
        # Update the values for the next iteration
        r = np.append(r, r[-1] + dr)
        m = np.append(m, m_new)
        P = np.append(P, P_new)
        phi = np.append(phi, phi_new)
        dmdr, dPdr, dphidr = dydr(m_new, P_new, phi_new, r[-1], EOS)
        dr = 0.1*min(np.abs(m_new/dmdr), np.abs(P_new/dPdr))
        # Check for stopping condition (the pressure is 10^-10 of the
central pressure)
        if P[-1]/P_central < 1e-10:
            return r, m, P, phi
        #if P becomes negative, also stop integration
        if P[-1] < 0:
            return r, m, P, phi

```

```

        i += 1
    print("Maximum iterations reached in integrate_star. Returning
last values. P_C = ", P[0])
    return r, m, P, phi

```

g.)

test case:

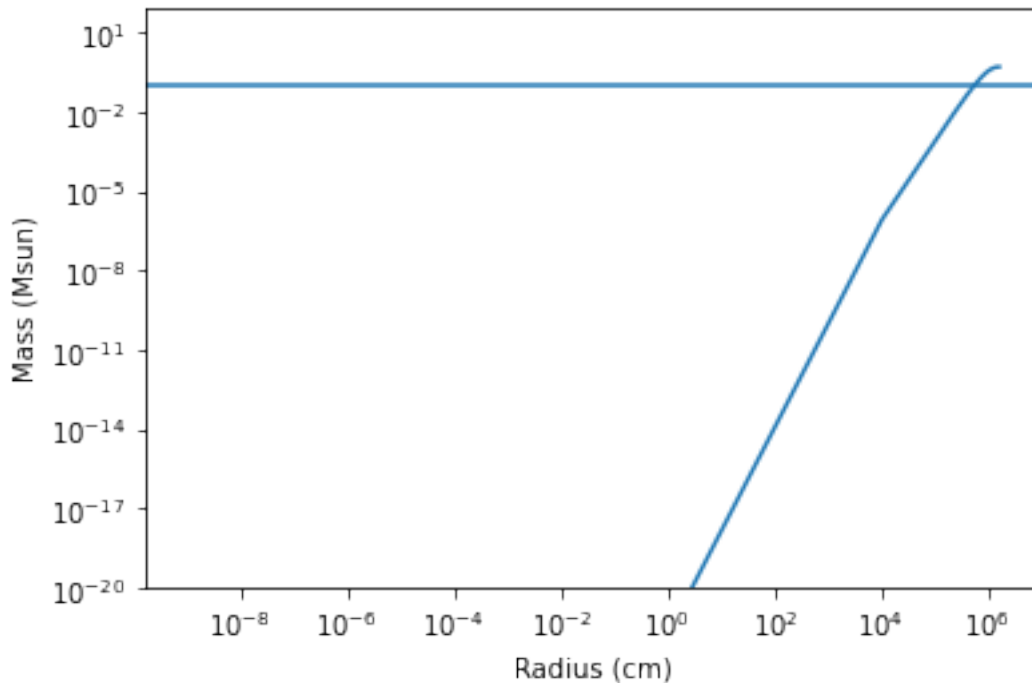
```

P0 = 1e34
dr = 1e4
#I'm starting with random initial central pressure just to see if the
code runs and an initial mass of electron mass so its tiny but not 0
(so I won't get nans)
r, m, P, phi = integrate(1e-9, np.array([mn, P0, 0]), dr=dr, EOS=1)
plt.plot(r, m/Msun)
plt.xscale('log')
plt.yscale('log')
plt.ylim(1e-20, 1e2)
plt.axhline(y=0.103)
plt.xlabel('Radius (cm)')
plt.ylabel('Mass (Msun)')
plt.show()

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/
ipykernel_launcher.py:14: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is
deprecated. If you meant to do this, you must specify 'dtype=object'
when creating the ndarray

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/scipy/
optimize/minpack.py:175: RuntimeWarning: The iteration is not making
good progress, as measured by the
improvement from the last ten iterations.
warnings.warn(msg, RuntimeWarning)

```



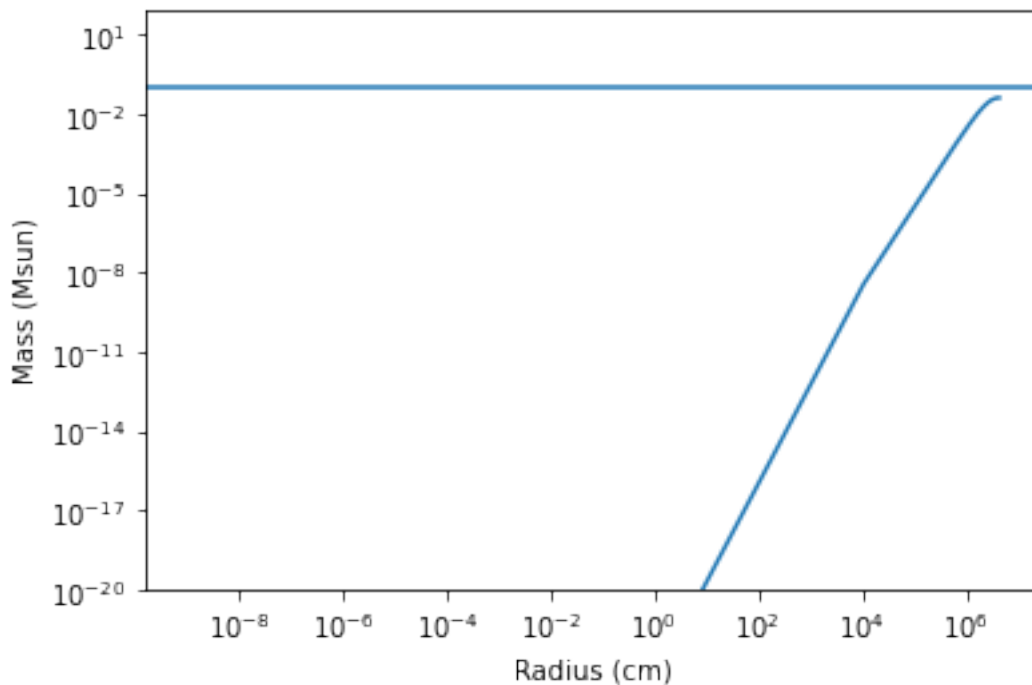
```
P0 = 1e30
dr = 1e4
#I'm starting with random initial central pressure just to see if the
code runs and an initial mass of electron mass so its tiny but not 0
(so I won't get nans)
r, m, P, phi = integrate(1e-9, np.array([mn, P0, 0]), dr=dr, EOS=1)
plt.plot(r, m/Msun)
plt.xscale('log')
plt.yscale('log')
plt.ylim(1e-20, 1e2)
plt.axhline(y=0.103)
plt.xlabel('Radius (cm)')
plt.ylabel('Mass (Msun)')
plt.show()
```

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/ipykernel\_launcher.py:14: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/scipy/optimize/minpack.py:175: RuntimeWarning: The iteration is not making good progress, as measured by the improvement from the last ten iterations.  
 warnings.warn(msg, RuntimeWarning)

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/ipyker

```
nel_launcher.py:2: RuntimeWarning: invalid value encountered in power
```



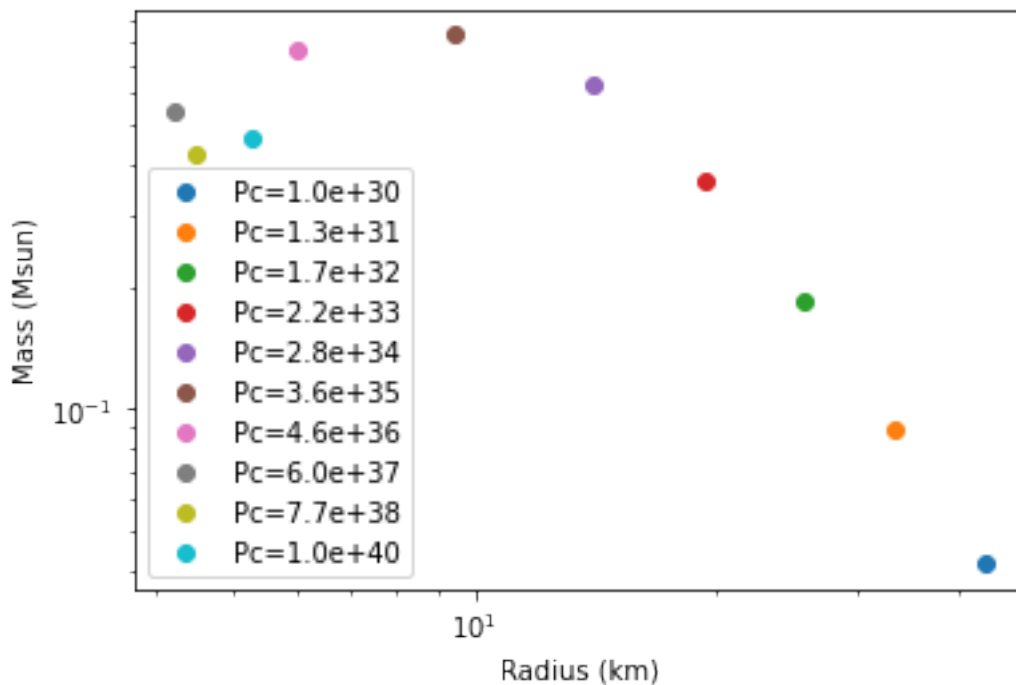
Real Values

```
P0 = 1e30
dr = 1e4
eos1 = []
#I'm starting with random initial central pressure just to see if the
code runs and an initial mass of electron mass so its tiny but not 0
(so I won't get nans)
for Pc in np.logspace(30,40,10):
    r, m, P, phi = integrate(1e-9, np.array([mn, Pc, 0]), dr=dr,
EOS=1)
    plt.scatter(r[-1]/1e5, m[-1]/Msun, label=f'Pc={Pc:.1e}')
    eos1.append((r[-1], m[-1], Pc))
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Radius (km)')
plt.ylabel('Mass (Msun)')
plt.legend()
plt.show()

/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/
ipykernel_launcher.py:14: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is
deprecated. If you meant to do this, you must specify 'dtype=object'
```

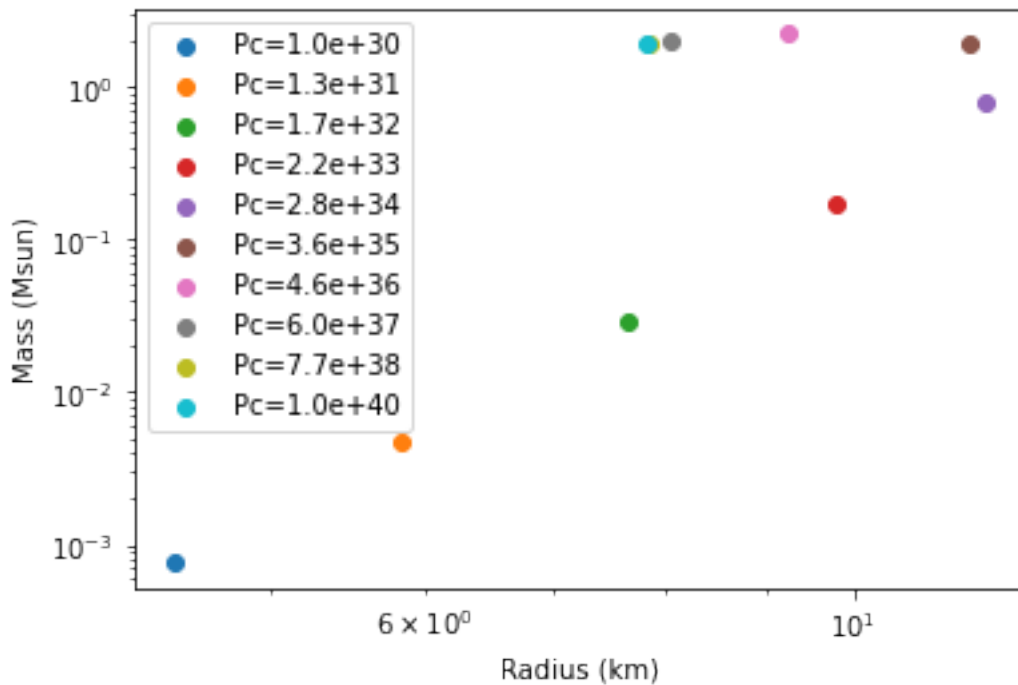
when creating the ndarray

```
/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/scipy/
optimize/minpack.py:175: RuntimeWarning: The iteration is not making
good progress, as measured by the
improvement from the last ten iterations.
warnings.warn(msg, RuntimeWarning)
/home/tegan/anaconda3/envs/research/lib/python3.6/site-packages/ipyker
nel_launcher.py:2: RuntimeWarning: invalid value encountered in power
```

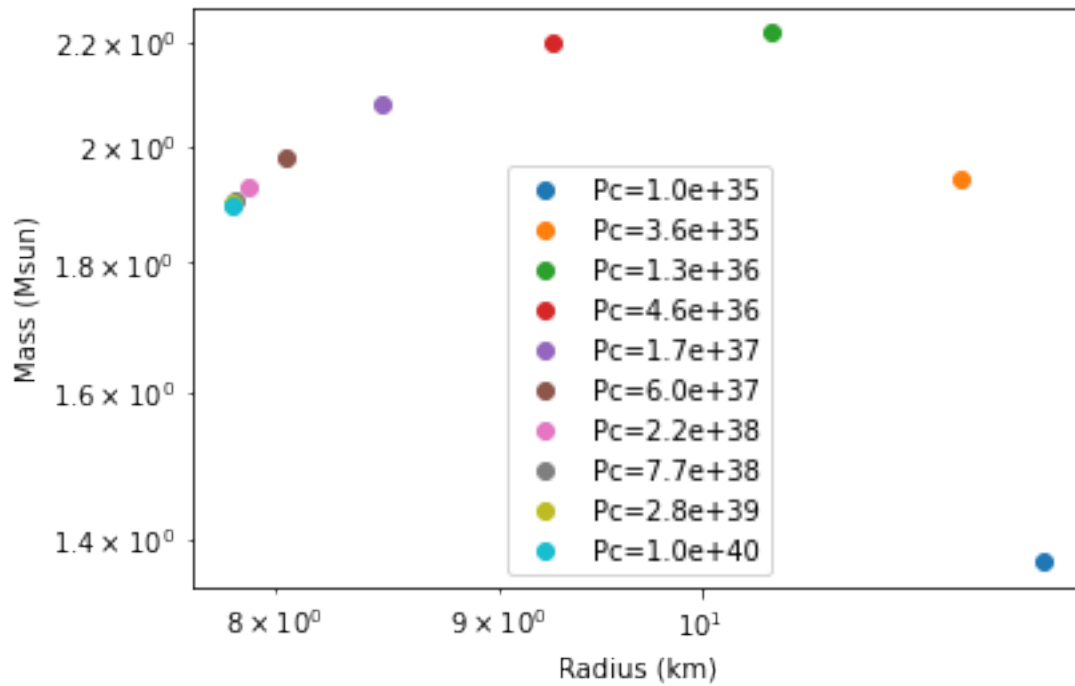


```
P0 = 1e30
dr = 1e4
eos2 = []
#I'm starting with random initial central pressure just to see if the
code runs and an initial mass of electron mass so its tiny but not 0
(so I won't get nans)
for Pc in np.logspace(30,40,10):
    r, m, P, phi = integrate(1e-9, np.array([mn, Pc, 0]), dr=dr,
EOS=2)
    plt.scatter(r[-1]/1e5, m[-1]/Msun, label=f'Pc={Pc:.1e}')
    eos2.append((r[-1], m[-1], Pc))
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Radius (km)')
plt.ylabel('Mass (Msun)')
```

```
plt.legend()
plt.show()
```



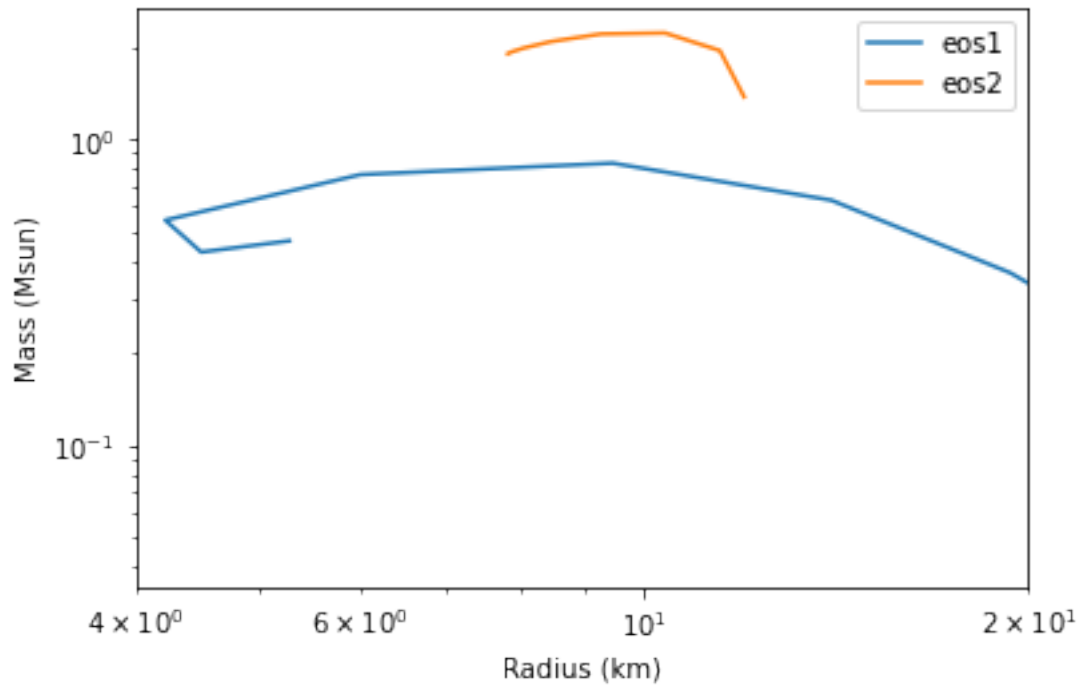
```
P0 = 1e30
dr = 1e4
eos2 = []
#I'm starting with random initial central pressure just to see if the
#code runs and an initial mass of electron mass so its tiny but not 0
#(so I won't get nans)
#If I go higher than 10^40 for the Pc then the code takes forever
#which I feel shouldn't be happening
#but if I go lower than 10^30 than the values turn back in a non-
#physical way
for Pc in np.logspace(35,40,10):
    r, m, P, phi = integrate(1e-9, np.array([mn, Pc, 0]), dr=dr,
EOS=2)
    plt.scatter(r[-1]/1e5, m[-1]/Msun, label=f'Pc={Pc:.1e}')
    eos2.append((r[-1], m[-1], Pc))
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Radius (km)')
plt.ylabel('Mass (Msun)')
plt.legend()
plt.show()
```



```

eos1 = np.array(eos1)
eos2 = np.array(eos2)
plt.plot(eos1[:,0]/1e5, eos1[:,1]/Msun, label='eos1')
plt.plot(eos2[:,0]/1e5, eos2[:,1]/Msun, label='eos2')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Radius (km)')
plt.ylabel('Mass (Msun)')
plt.xlim(4,20)
plt.legend()
plt.show()

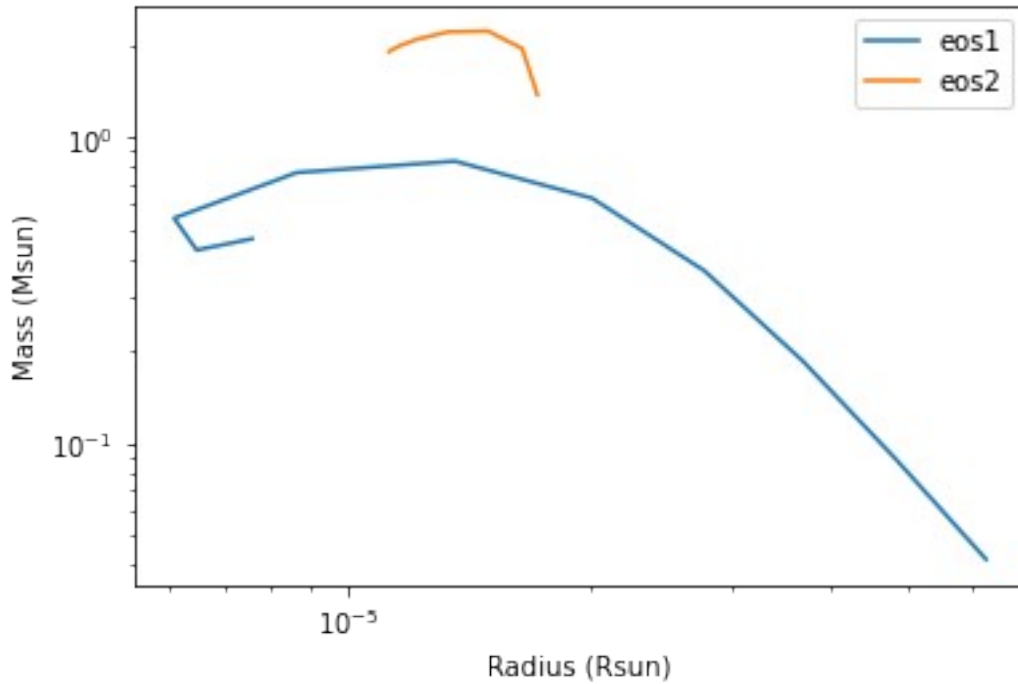
```



```

eos1 = np.array(eos1)
eos2 = np.array(eos2)
plt.plot(eos1[:,0]/Rsun, eos1[:,1]/Msun, label='eos1')
plt.plot(eos2[:,0]/Rsun, eos2[:,1]/Msun, label='eos2')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Radius (Rsun)')
plt.ylabel('Mass (Msun)')
plt.legend()
plt.show()

```



I vaguely recover the plot, but not very well and only over a limited region.

```
print("EoS 1 (R (cm), M (g), Pc (dyne cm^-2)):")
print(eos1)
print("EoS 2 (R (cm), M (g), Pc (dyne cm^-2)):")
print(eos2)
```

```
EoS 1 (R (cm), M (g), Pc (dyne cm^-2)):
[[4.33932917e+06 8.28620121e+31 1.00000000e+30]
 [3.34739705e+06 1.76400713e+32 1.29154967e+31]
 [2.56569983e+06 3.67786684e+32 1.66810054e+32]
 [1.93380545e+06 7.25731981e+32 2.15443469e+33]
 [1.40097450e+06 1.25272506e+33 2.78255940e+34]
 [9.44803570e+05 1.65533260e+33 3.59381366e+35]
 [5.99672257e+05 1.51849596e+33 4.64158883e+36]
 [4.21988034e+05 1.07819870e+33 5.99484250e+37]
 [4.50189125e+05 8.48148005e+32 7.74263683e+38]
 [5.27797529e+05 9.22956138e+32 1.00000000e+40]]

EoS 2 (R (cm), M (g), Pc (dyne cm^-2)):
[[1.19714353e+06 2.72685192e+33 1.00000000e+35]
 [1.14615357e+06 3.85963870e+33 3.59381366e+35]
 [1.03815219e+06 4.40749012e+33 1.29154967e+36]
 [9.25320306e+05 4.37384529e+33 4.64158883e+36]
 [8.45619369e+05 4.13191036e+33 1.66810054e+37]
 [8.04568387e+05 3.93295528e+33 5.99484250e+37]
 [7.88652916e+05 3.82907848e+33 2.15443469e+38]
 [7.83752489e+05 3.78746007e+33 7.74263683e+38]]
```

```
[7.82447672e+05 3.77327292e+33 2.78255940e+39]
[7.82082327e+05 3.76891338e+33 1.00000000e+40]]

print("For EOS 1 the maximum mass is", max(eos1[:,1]/Msun))
print("For EOS 2 the maximum mass is", max(eos2[:,1]/Msun))

For EOS 1 the maximum mass is 0.8324906388524562
For EOS 2 the maximum mass is 2.2165903450852333
```

The BJ (aka EOS 2) maximum mass is a bit high compared to the table, but not by much. The ideal EOS (EOS 1) is also a tiny bit high compared to the expected 0.7 solar masses, but it's not too bad.