TOV SOLVER.PY

USER GUIDE

PIERO TREVISAN

CONTENTS

1 STRUCTURE OF THE CODE

The code is divided in:

1.1 Packages

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
from astropy.constants import G, c, M_sun
from math import pi
from sys import argv

global K, Gamma, rho0_c                              #define globally EoS
     and central density
```

Imports the important packages for plotting, declaring numpy arrays, saving txt, $c, G, M_{\odot}, \pi$ values. Finally it declares ρ_c, polytropic costant and polytropic index as global variables.

1.2 Euler Method

```
def Eul(u , t, dt, rhs):                              #Euler Method
    n = len(u)
    k1 = np.zeros(n)
    up = np.zeros(n)
    k1 = dt*rhs(u,t)
    up = u + k1
    return up
```

The first function declares the Euler Method. It's written like a mathematical expression in the form $u_{n+1}(u,t) = u_n(t) + RHS_n(u,t)dt$. It takes as input the function u_n, t, the increment decided in 1.8 and the RHS decided in 1.5. It returns the function u_{n+1} incremented with the Euler method.

1.3 RK4 Method

```
def RK4(u, t, dt ,rhs):                           #Runge-Kutta 4 Method
    n = len(u)
    up = np.zeros(n)
    k1 = np.zeros(n)
    k2 = np.zeros(n)
    k3 = np.zeros(n)
    k4 = np.zeros(n)
    k1 = dt*rhs(u,t)
    k2 = dt*rhs(u + 0.5*k1, t + 0.5*dt)
    k3 = dt*rhs(u + 0.5*k2, t + 0.5*dt)
    k4 = dt*rhs(u + k3, t + dt)
    up = u + (k1 + 2*k2 + 2*k3 + k4) / 6.0
    return up
```

The second funcion declares the RK4 Method. The procedure is the same in 1.2 only this time the function returns u_{n+1} incremented with the RK4 method.

1.4 Negative pressure controller

```
def rho0fp(p):                                            #Advice for
    negative pressure
    if p <   0.0:
        print "Negative p found: %.2e" %(p)
        p = abs(p)
    return (p/K)**(1.0/Gamma) + p/(Gamma - 1.0)         #Energy density
```

This function simply prints a warning message when the solver finds negative pressure and returns the total mass-energy density.

1.5 TOV and continuity RHS

```
def TOVrhs(u,r):
    [m,p] = u
    rho = rho0fp(p)

    #Regularize system at the origin, Taylor series for m

    if r < 1e-3:

        m_rrr = 4.0/3.0*pi * (rho0_c**2/(Gamma - 1.0) + 4.0**(-1.0/Gamma
            )*(rho0_c)**(2.0/Gamma))
        m_rr = m_rrr * r
        m_r = m_rr * r
    else:
        m_r = m / r
        m_rr = m_r / r
        m_rrr = m_rr / r

    #The real TOV

    MRHS = 4.0*pi*r**2 * rho    #continuity
    PRHS = - m_rr * (rho + p)*(1.0 + 4.0*pi*p / m_rrr) * 1.0/(1.0 - 2.0*
        m_r)   #pressure equation


    return np.array([MRHS, PRHS])
```

This function takes as input the function u_n as a vector of $[Mass, Pressure]$. Then specifies the Mass continuity RHS and the TOV RHS. It returns an array where the values of $[MRHS, PRHS]_n$ are stored. The first part is the taylor expansion for m/r^3 used for $r < 10^{-3}$.

1.6 Differential equation solver

```
def tovint(rho0_c, dr):
    nsteps = 1000000
    sol = np.zeros((nsteps,2))      #store the solution
    n=0
    r=0.0
    p_c = K*rho0_c**Gamma           #Central Pressure
    u = np.array([0.0, p_c])        # [m, p](0,0)
    sol[0] = u

    #RK4 METHOD

    while  sol[n,1]> 1.0e-12 and  n < nsteps-1:                 #
        stopping condition 1.0e-12 = zeropressure
        u = RK4(u, r , dr, TOVrhs)
        n += 1
        sol[n] = u
        r+= dr
    '''#<---------------DECOMMENT HERE TO USE EULER METHOD
    #EULER METHOD:

    while sol[n,1]> 1.0e-12 and  n < nsteps-1:
        u = Eul(u, r , dr, TOVrhs)
        n += 1
        sol[n] = u
        r += dr
                  #DECOMMENT HERE TO USE EULER METHOD-------->'''

    return (r, sol[:n,0], sol[:n,1], rho0fp(p_c))              #(R, M,
        P, rho_c)
```

It takes as input the value of central density and the chosen stepsize. It computes the central pressure P_c for the given ρ_c and equation of state (simply K and γ in the polytropic case). It declares the initial radius $r = 0.0$, the maximum iterations ($N_{steps} = 100000$: can be reached if really small stepsize is chosen) and also u_0 as $[m, P]_0 = [0.0, P_c]$. Resolve the differential equation for the RHSs specified in 1.5 until pressure is below 10^{-12} for Euler Method or RK4 method (if you want to use one method please comment the indicated part). As output radius of the star, mass profile, pressure profile, density profiles is given.

1.7 Plotter

```
def tovplot(r, m, p):

    #generality of the plot

    rhost = str(rho0_c/1.6199e-18/1e14) #string for title

    ri = np.linspace(0.00001,r, len(m))
    fig = plt.figure(1)
    plt.xlim([0.00001,r])
    gs = gridspec.GridSpec(3,3)
    plt.rc('text', usetex= True)
    plt.rc('font', family = 'Iwona ')
    title = plt.suptitle(r'\textsc{Central density} = %s $\times$
        10$^{14}$ g cm$^{-3}$' %rhost, fontsize = 16)
    plt.subplots_adjust(wspace=0, hspace=0.38)



    #presure subplot

    ax1 = plt.subplot(gs[0,:])
    ax1.set_xscale("log")
    line1, = ax1.plot(ri, p, 'r--')
    ax1.set_ylabel(r'P(r) [dyn cm$^2$]', fontsize = 14)
    lb1, ub1 = ax1.get_ylim()
    ax1.set_xlim(0.00001,r+5)
    ax1.set_ylim(1, ub1 + 1e34)
    ax1.set_title(r'\textit{Pressure profile}', fontsize = 14)

    #mass subplot

    ax2 = plt.subplot(gs[1,:])
    ax2.set_xscale("log")
    line2, = ax2.plot(ri, m, 'b-')
    lb2, ub2 = ax2.get_ylim()
    ax2.set_yticks( np.linspace(lb2, ub2, 5))
    plt.ylabel(r'm(r) [M$_{\odot}$]', fontsize = 14)
    ax2.set_ylim(lb2-0.2, ub2+0.2)
    ax2.set_xlim(0.00001,r+5)
    ax2.set_title(r'\textit{Mass profile}', fontsize = 14)

    #density subplot

    ax3 = plt.subplot(gs[2,:])
    ax3.set_xscale("log")
    line3, = ax3.plot(ri,rho2, 'g-')

    lb3, ub3 = ax3.get_ylim()
    ax3.set_ylim(lb3, ub3 + 1e14)
    ax3.set_ylabel(r'$\rho (r)$ [g cm$^{-3}$]', fontsize = 14)
    ax3.set_xlabel(r'r [km]',fontsize = 14)
    ax3.set_xlim(0.00001,r+5)
    ax3.set_title(r'\textit{Density profile}', fontsize = 14)


    plt.savefig('plot %s_%s_%s.png' % (rhost,str(m[-1]),str(r)))     #
        save figure giving central density, mass, radius


    plt.show()
```

It simply plots the pressure, mass, density proflies for the values given in the output of 1.6 in a value - log(r) diagram.

1.8 Code runner

```
if __name__ == "__main__":

    # MODEL: EOS

    script, rho0_c0, K, Gamma = argv                #CHOOSE CENTRAL
        DENSITY IN CGS, K IN G=C=Msun=1 units, GAMMA when you run the
        program
    K = float(K)
    Gamma = float(Gamma)
    rho0_c0 = float(rho0_c0)
```

```
                                                    #K = 30000 in  G
                                                        =C=Msun=1
                                                        units
                                                        corresponds
                                                        K = 1.98183e
                                                        -6 in cgs
#TYPICAL MODEL

#rho0_c0 = 2.2   e14
#K = 30000
#Gamma = 2.75


rho0_c = rho0_c0*1.6199e-18             #convert in G=C=Msun=1

M_sun = M_sun.value
c = c.value
G = G.value
M_solar_G_over_c_sq = M_sun * G/ c**2   #Msun=G=C=1 lenght convert


dr = 1e-4                               #Stepsize

(r, m , p ,rho_c) = tovint(rho0_c, dr)      #RESOLVE TOV

global rho1, rho2

rho1 = (p / K) ** (1. / Gamma) + p/(Gamma - 1.0)
rho2 = rho1/(1.6199e-18)
Press =  (r, m , p ,rho_c)[2]/1.8063e-39
Prhom = (Press, rho2, m)
rhoMR = ([(rho0_c0, m[-1], r*M_solar_G_over_c_sq/1000.0)])

print 'stopped at n = %d, r = %.4f km, m = %.4e M_solar and p = %.4e
    ' %(len(m), r*M_solar_G_over_c_sq / 1000.0,m[-1], p[-1])
print 'Stellar mass: M = %.11e' %m[-1]
print 'Stellar radius. R = %.11e [km]' %(r*M_solar_G_over_c_sq
    /1000.0)

print 'schwartzRadius = %.4e [km]' %(2.**m[-1]*M_solar_G_over_c_sq
    /1000.0)

tovplot(r*M_solar_G_over_c_sq / 1000.0, m, p/1.8063e-39)
    #PLOT PRESSURE, MASS, DENSITY PROFILE



np.savetxt('prhomRK4%s.txt' %str(rho0_c0), np.transpose(Prhom), fmt=
     '%.8e', header = 'PRESSURE          DENSITY       MASS
    MODEL=%s' %str(rho0_c0)) #save pressure and density and mass
    profile
    #SAVE DENSITY, FINAL MASS, FINAL RADIUS

    datafile_path = "PATH\massradiusgraf.txt"
    datafile_id = open(datafile_path, 'a+')
    np.savetxt(datafile_id, rhoMR , fmt='%.11f')
    datafile_id.close()
```

It declares ρ_c, K, $Gamma$ for the user input when the code is run by the user. Proper conversion for the various units chosen are specified (like from cgs to $c = G = M_\odot = 1$). The stepsize is specified as well. Furthermore the 1.6 solver is run for the ρ_c and stepsize chosen. Finally the plotter starts for the output value of 1.6 converted in cgs units for the pressure and density, solar mass for the mass, km for the radius. In the end pressure, cumulative mass, density for each step (given a central density) are stored in a "prhom(# densityvalue).txt" file.

If you are planning to run the code multiple times for different central densities, the code creates a massradiusgraf.txt file where ρ_c, Final Mass, Final Radius are printed. Then it stores there the values for every run.