Ay 190 Assignment 4

ODE Integration

January 22, 2013

1 The Code Skeleton

The core code that solves the stellar structure equations is implemented in `tov_integrate()`, which relies on a number of supporting routines to perform smaller segments of the work. This includes routines to grid the star, implement the stellar structure equations, and Runge-Kutta integrators of different orders. After specifying the conditions at the core of the star, the stellar structure equations are integrated at every grid point until the condition for the surface of the star (formulated in terms of pressure) is reached. In addition to the pressure and the interior mass, the specific internal energy and density are calculated at each grid point.

2 Implementation Notes

The white dwarf structure was computed by solving the simplified stellar structure equations

$$\frac{dP}{dr} = -\frac{GM(r)}{r^2}, \rho \qquad \frac{dM}{dr} = 4\pi\rho r^2$$

with a polytropic equation of state $P = K\rho^\Gamma$ for which values appropriate for a fully relativistic degenerate white dwarf, $K = 1.244 \times 10^1 5(0.5)^\Gamma$ dyn cm^{-2} (g^{-1} cm^3)$^\Gamma$ and $\Gamma = 4/3$, were used.

The 2nd, 3rd, and 4th order Runge-Kutta integration routines and the solution of the stellar structure equations were implemented in `tov_newt.py` reproduced in Appendix A.

3 Results

Integrating the simplified stellar structure equation using the fourth-order Runge-Kutta method over a grid with a resolution of $dr = 100$m (10^5 mesh points) produced a stellar mass of 1.47 $M_\odot$ and a radius of 1551.3 km.

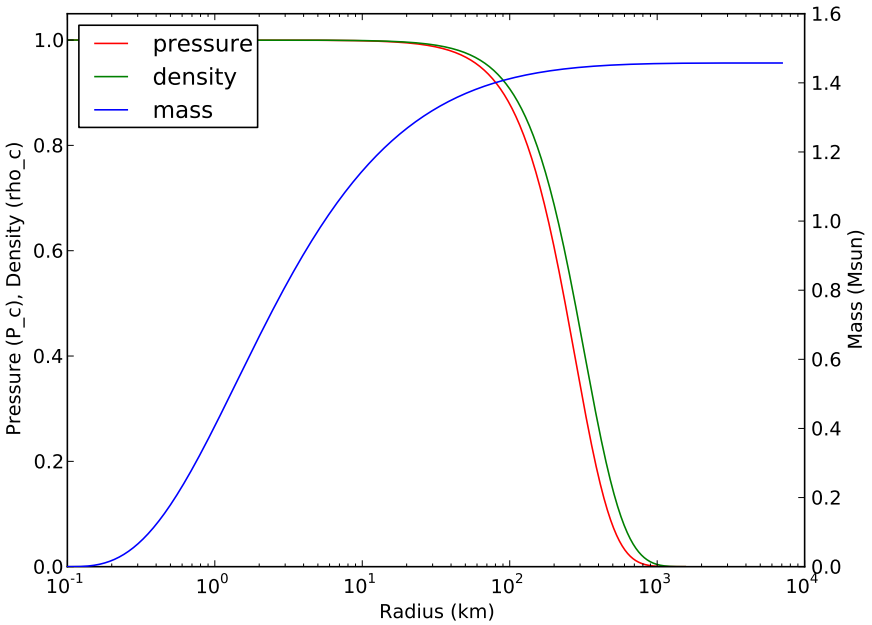

Figure 1: Stellar properties calculated by integrating the simplified stellar structure equations using a fourth-order Runge-Kutta method on a grid with 10^5 mesh points, for a resolution of $dr = 100$ m. The total stellar mass is 1.

4 Convergence

The masses were recomputed over grids of three different resolutions, $dr_1 = 10^3$ (100 mesh points), $dr_2 = 10^4$, and $dr_3 = 10^5$. The self-convergence factor

$$Q_s = \frac{|M_*(dr_3) - M_*(dr_2)|}{|M_*(dr_2) - M_*(dr_1)|}$$

where $dr_1 > dr_2 > dr_3$ and which, for a convergence order n should equal

$$Q_s = \frac{dr_3^n - dr_2^n}{dr_2^n - dr_1^n}$$

was calculated. The self-convergence factor agreed well for all orders of Runge-Kutta (see Table 1).

Table 1: Convergence of stellar mass

Order	$M_*(dr_1)$	$M_*(dr_2)$	$M_*(dr_3)$	Q_s	expected Q_s
2	1.4591519945	1.45744000872	1.45742358791	0.00959	0.01
3	1.4574053995	1.45742340502	1.45742342266	0.000980	0.001
4	1.45742468681	1.4574234228	1.45742342268	0.0000949	0.0001

Appendices

The Python modules and scripts used in this assignment include

A The Stellar Structure Solver: tov_newt.py

```
#!/opt/local/bin/python

import sys
from scipy import *
from pylab import *
from time import time
import matplotlib.pyplot as plt
from matplotlib.axes import *


# global constants (cgs)
ggrav = 6.67e-8
clite = 3.0e10
msun = 1.99e33

# EOS for neutron stars:
# polyG = 2.0
# polyK = 100.0 * 5.55e38/6.1755e17**polyG

# EOS for white dwarfs:
polyG = 4.0/3.0
polyK = 1.244e15*0.5**polyG

# central values
rhoc = 1.0e10
pressc = polyK * rhoc**polyG

# minimum pressure
rhomin = 1.0e-10*rhoc
min_press = polyK * rhomin**polyG

# grid
rmax = 1.0e9


# SUPPORTING ROUTINES -------------------------------------------------------

def set_grid(rmax,nzones):
    """Sets up the grid and return the radius array and dr."""

    dr=float(rmax)/nzones
    rad=arange(0,rmax,dr)

    return (rad,dr)
```

```
def tov_RHS(r,data):
    """Implements simplified stellar structure equations."""
    # data = [pressure, m_baryons]

    rhs = zeros(2)

    mass = data[1]
    press = max(data[0],min_press)
    rho = (press/polyK) ** (1.0/polyG)

    rhs[0] = -ggrav*mass*rho/(r*r) if r != 0.0 else 0
    rhs[1] = 4*pi*rho*r*r
    return rhs


def tov_RK2(old_data,r,dr):
    """Integrates the stellar structure equations over [r,r+dr] via RK2."""

    k1=dr*tov_RHS(r,old_data)
    k2=dr*tov_RHS(r+dr/2,old_data+k1/2)

    return old_data + k2


def tov_RK3(old_data,r,dr):
    """Integrates the stellar structure equations over [r,r+dr] via RK3."""

    k1=dr*tov_RHS(r,old_data)
    k2=dr*tov_RHS(r+dr/2,old_data+k1/2)
    k3=dr*tov_RHS(r+dr,old_data-k1+2*k2)

    return old_data + (k1 + 4*k2 + k3)/6

first=1
def tov_RK4(old_data,r,dr):
    """Integrates the stellar structure equations over [r,r+dr] via RK4."""
    global first

    k1=dr*tov_RHS(r,old_data)
    k2=dr*tov_RHS(r+dr/2,old_data+k1/2)
    k3=dr*tov_RHS(r+dr/2,old_data+k2/2)
    k4=dr*tov_RHS(r+dr,old_data+k3)

    return old_data + (k1 + 2*k2 + 2*k3 + k4)/6


mul=10000
def tov_integrate(rmax,nzones):
    """Integrates the stellar structure equations over the entire star."""

    # set up grid
    (rad,dr) = set_grid(rmax,nzones)
```

```
    # initialize some variables
    tovdata = zeros((nzones,2)) # [pressure, m_baryons]
    tovout  = zeros((nzones,4)) # [rho, pressure, eps, mass]

    # central values
    tovdata[0,0] = polyK * rhoc**polyG
    tovdata[0,1] = 0.0
    tovout[0,0] = rhoc
    tovout[0,1] = pressc
    print 'r={0:5}, {1:>17} {2:>17}'.format(0.,tovdata[0,0],tovdata[0,1])

    isurf = 0  # to track the surface (where press <= press_min)
    for i in range(nzones-1):

        # integrate one step
        p = 1 if i<2 or i%mul==0 else 0
        tovdata[i+1,:] = tov_RK4(tovdata[i,:],rad[i],dr)
        #if i%mul == 0: print 'r={0:5}, {1:>17} {2:>17} msun'.format(rad[i]/1.0e5,tovdata[i+1,0],tovdata

        # check if press below 0
        if(tovdata[i+1,0] <= min_press):
            isurf = i
            break

        # compute stellar properties
        tovout[i+1,0] = (tovdata[i+1,0]/polyK)**(1.0/polyG)      # density
        tovout[i+1,1] = tovdata[i+1,0]                           # pressure
        tovout[i+1,2] = tovdata[i+1,0]/((polyG-1)*tovout[i+1,0]) # eps
        if (i+1 > isurf and isurf > 0): tovout[i+1,3] = tovdata[isurf,1]
        else: tovout[i+1,3] = tovdata[i+1,1]                     # mass


    return (tovout,isurf,dr)


# SOLVE STELLAR STRUCTURE EQNS AND CHECK CONVERGENCE ----------------------------

# for convergence:
# number of points
na=array([1,10,100],dtype='int')*1000
# to store
masses = zeros(len(na))
drs = zeros(len(na))
isurfs = zeros(len(na))
tovs = []

for i in range(len(na)):
    t=time()
    (tov_star,isurf,dr) = tov_integrate(rmax,na[i])
    print 'took',time()-t,'sec'
    print '  istar =',isurf,'dr =',dr/1.0e5,'km'
    print '  mass =',tov_star[isurf,3]/msun,'msun'
    print '  radius =',dr*isurf/1.0e5,'km'
```

```
    print '  density =',tov_star[isurf,0]
    print '  pressure =',tov_star[isurf,1]
    print '  eps =',tov_star[isurf,2]

    masses[i] = tov_star[isurf,3]/msun
    drs[i] = dr
    isurfs[i] = isurf
    tovs.append(tov_star)

# convergence: self-convergence factor
order=4
Qs=abs((masses[2]-masses[1])/(masses[1]-masses[0]))
Qs_target=abs((drs[2]**order-drs[1]**order)/(drs[1]**order-drs[0]**order))
print 'Qs =',Qs,'Qs target =',Qs_target


# PLOT RESULTS ---------------------------------------------------------------

ls=['-.','--','-'] #line styles

#plot masses, pressures, density for best resolution run
r=arange(0.,drs[-1]*isurf/1e5,drs[-1]/1e5) # in km
rect=[0.1,0.1,0.8,0.8]

print r[-1], tovs[-1][isurf,3]/msun

a1=axes(rect)
a1.yaxis.tick_left()
plot(r,tovs[-1][:isurf,1]/pressc,'r',  # plot pressure
     r,tovs[-1][:isurf,0]/rhoc,'g')    #   and density
ylim([0,1.05])
xscale('log')
ylabel('Pressure (P_c), Density (rho_c)')
xlabel('Radius (km)')

a2=axes(rect,frameon=False)
a2.yaxis.tick_right()
plot([0],[0],'r',[0],[0],'g',r,tovs[-1][:isurf,3]/msun,'b') # plot mass
a2.yaxis.set_label_position('right')
ylabel('Mass (Msun)')
a2.set_xticks([])

legend(['pressure','density','mass'],loc='best')
savefig('star_properties.pdf')
show()
```