Mapana J Sci, 16, 1 (2017), 67-80
ISSN 0975-3303| https://doi:10.12723/mjs.40.5

Evaluation of Scipy.ode Integrators in Solving the Lane-Emden Equation for Polytropes as a Boundary Value Problem with a Fitting Method

M N Anandaram*

Abstract

The use of Scipy integrators like dopri5 and others in accurately solving the Lane-Emden equation of a polytrope as a two-point BVP with fitting is investigated by comparing the Emden radius with the extended precision reference value obtained by Boyd's Chebyshev spectral method. It is found that both dopri5 and dop853 integrators provide acceptable accuracy upto 14 decimal digits.

Keywords: Lane-Emden equation, two point BVP with fitting method, Scipy ode solvers

1. Introduction

The Lane-Emden equation is a well known non-linear second order differential equation which describes the structure of a polytrope. The polytrope of index **n** is a massive gas sphere in a state of hydrostatic equilibrium and is governed by a pressure-density relation of the form$P = K\rho^{1+1/n}$. The theory of the polytrope is described in [1]. In the notation of [1] the Lane-Emden equation with its two point central ($\xi = 0$) and surface ($\xi = \xi_1$) boundary conditions for the solution $\theta(\xi)$ and its slope $\theta'(\xi)$ reads

* Bangalore University, Bengaluru, India; mnanandaram@gmail.com

$$d^2\theta/d\xi^2 + (2/\xi)\, d\theta/d\xi + \theta^n = 0;\ \theta(0) = 1;\ \theta'(0) = 0;\ \theta(\xi_1) = 0 \quad (1.1)$$

where $\theta(\xi)$ is the solution (aka Lane-Emden function) and $0 \le n \le 5$ is the constant index for a given polytrope. The first zero of this solution denoted as $\xi = \xi_1$ yields the Emden radius of the polytrope. Now this is written as a system of two coupled first order ODEs using two new variables y and z defined by $y \equiv \theta$ and $z \equiv y' = d\theta/d\xi$ as

$$dy/d\xi = z\ ;\ dz/d\xi = -2\,(z/\xi) - y^n \quad (1.2)$$

with the two-point BCs now reading as $y(\xi = 0) = 1,\ z(\xi = 0) = 0\ ;\ y(\xi_1) = 0,\ z(\xi_1) < 0$. The right hand sides of (1.2) are used for integrating the Lane-Emden equation from either center or surface as the starting point. When the integration is started from the center ($\xi = 0$) a zero division singularity arises. This is avoided by fixing the value of the latter function in (1.2) from the slope of the power series expansion given by Equation (14) in [1] and repeated here:

$$z = d\theta/d\xi = -(1/3)\xi + (n/30)\xi^3 - [n(8n-5)/2520]\,\xi^5 + \ldots \quad (1.3)$$

It can now be seen using (1.3) that $2(z/\xi) = -2(1/3)$ as all other terms containing ξ vanish at the center and inserting this into the second expression in (1.2) yields $dz/d\xi = -(1/3)$ for all values of index **n** since $y \equiv \theta = 1$ there. This value can also be found by differentiating (1.3) again and evaluating it at the center. This is taken care of in the computer script by inserting an if-else statement (see line numbers 11 to 20 in the Appendix).

While there are many ways of solving this two-point BVP the bidirectional shooting method is applied here so that it is integrated simultaneously outward from the center and inward from the surface of the polytrope towards a selected fitting point in between. As the exact value of the scaled radius is unknown this inward integration is started from a reasonably guessed value. The two integrations do not meet at the fitting point at all as they are like a directed shooting method. Therefore the difference between them at the fitting point is used to adjust the value of the scaled radius,(ξ), in a proportionate way and the two-way integration is repeated. In this manner many attempts (~ 30) are made before the

two integrations converge at the fitting point to within a set difference tolerance limit (like ~1.0e-14). This fitting point is set at 90% of the radius so that the Lane-Emden function ($\theta(\xi)$) would have small values ($\theta \cong 0$) characteristic of the polytropic envelope. The algorithmic step sequence for carrying out this plan is taken from [2] and briefly described below.

2. Bidirectional iterative integration and fitting method

The integration is carried out by using the python program ***le_fit.py*** [3] and modifying it where needed to achieve maximum accuracy. In order to start the inward integration of (1.2) from the surface let $3 \leq \xi_s \leq 10$ denote the guessed value of the scaled radius which will finally converge to the Emden radius ξ_1 and the slope at that point as $\alpha < 0$ (these are denoted as **xi_s, xi1** and **alpha** in the python script). The fitting point $\boldsymbol{\xi_{fit}}$ (**xi_fit**) is set as a fraction of $\boldsymbol{\xi_s}$ (say, **xi_fit = 0.9 xi_s**). The integrator is chosen in turns to be one of dopri5(), dop853(), vode() or lsoda() invoked from scipy.integrator.ode library. To maximize their accuracy the relative and absolute tolerance parameters were set to the minimum possible ($\sim 10^{-15}$). Now let **y_in($\boldsymbol{\xi_{fit}}$)** and **z_in($\boldsymbol{\xi_{fit}}$)** be the arrays so obtained as outputs of inward integration from the surface upto the fitting point. Similarly, in the case of outward integration from the center upto the same fitting point, let **y_out($\boldsymbol{\xi_{fit}}$)** and **z_out($\boldsymbol{\xi_{fit}}$)** be the arrays so obtained as outputs of the integrators. In order to match the respective inward and outward arrays at the fitting point the new functions which are required to be zeroed are defined as $Y(\alpha, \xi_s)$ and $Z(\alpha, \xi_s)$ and given by

$$Y(\alpha, \xi_s) \equiv y_{in(\xi_{fit})} - y_{out(\xi_{fit})} = 0 \tag{2.1a}$$

$$Z(\alpha, \xi_s) \equiv z_{in(\xi_{fit})} - z_{out(\xi_{fit})} = 0 \tag{2.1b}$$

The corrections needed are found from Taylor series expansion as

$$Y(\alpha + \Delta\alpha, \xi_s + \Delta\xi_s) = Y(\alpha, \xi_s) + \frac{\partial Y}{\partial \alpha}\Delta\alpha + \frac{\partial Y}{\partial \xi_s}\Delta\xi_s \sim 0 \tag{2.2}$$

$$Z(\alpha + \Delta\alpha, \xi_s + \Delta\xi_s) = Z(\alpha, \xi_s) + \frac{\partial Z}{\partial \alpha}\Delta\alpha + \frac{\partial Z}{\partial \xi_s}\Delta\xi_s \sim 0 \tag{2.3}$$

To get the partial derivatives in (2.2) and (2.3) integrations indicated in (2.1) are now repeated once to get $Y(\alpha + d\alpha, \xi_s)$ and $Z(\alpha + d\alpha, \xi_s)$ and again to get $Y(\alpha, \xi_s + d\xi_s)$ and $Z(\alpha, \xi_s + d\xi_s)$ so that they are computed as numerical differences given by

$$\partial Y/\partial\alpha = \big(Y(\alpha + d\alpha, \xi_s) - Y(\alpha, \xi_s)\big)/\Delta\alpha \qquad (2.4a)$$

$$\partial Y/\partial\xi_s = \big(Y(\alpha, \xi_s + d\xi_s) - Y(\alpha, \xi_s)\big)/\Delta\xi_s \qquad (2.4b)$$

$$\partial Z/\partial\alpha = \big(Z(\alpha + d\alpha, \xi_s) - Z(\alpha, \xi_s)\big)/\Delta\alpha \qquad (2.5a)$$

$$\partial Z/\partial\xi_s = \big(Z(\alpha, \xi_s + d\xi_s) - Z(\alpha, \xi_s)\big)/\Delta\xi_s \qquad (2.5b)$$

The magnitudes of $\Delta\alpha$ and $\Delta\xi_s$ are adjusted to be as small as possible [2] by using a multiplying factor, **eps** typically set close to machine precision [3]. These steps are iterated for a number of times so that a good convergence is obtained in about 20 to 40 iterations. At the end of each run there would be two outputs xi_s and xi_in the average of which will be the required Emden radius xi1 and a third output for its slope at that point. There would also be a pair of two arrays comprising y_in (xi) and y_out (xi) and another pair comprising z_in (xi) and z_out (xi). The entire procedure was repeated with each of the aforementioned integrator backends and the corresponding value of the Emden radius is then compared with the highly precise reference value taken from [4] or computed from [5]. The results will be discussed in the next section. A sample plot of fitting at first and 28th iterations for **n = 3** is shown in Figure 1.

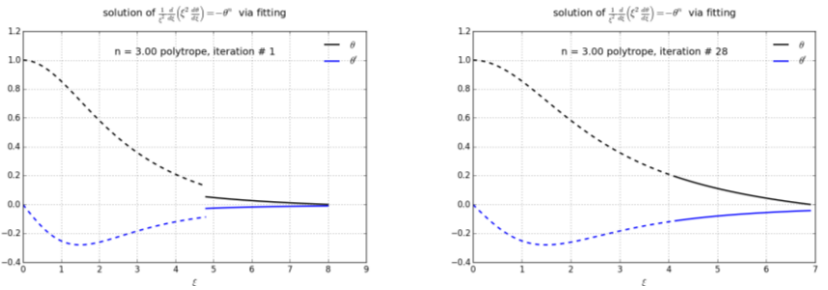

Figure 1 (left) First iteration and (right) 28th iteration of fitting y_out and z_out (dashed) with y_in and z_in (solid line) for a n=3 polytrope. Notice that xi_s is converging to its final value (~6.8968).

We can get the complete solution of the polytrope by correctly combining inward parts with outward parts got from the above fitting procedure. Additional tasks performed to do this are outlined here. Now $\theta = 0$ at the polytropic surface from where the inward solution y_in(xi) was found up to the fitting point whereas y_out(xi) starts from the center and ends at the same fitting point. Hence to get the complete solution $\theta(\xi)$ (or, y(xi)) running smoothly from center to the surface of the polytrope, the inward part y_in(xi) is now reversed and then merged with the outward part y_out(xi) after averaging out minute differences at the fitting point. Similarly the complete slope z(xi) is found by merging the reversed z_in(xi) array with z_out(xi) array. Now from [1] we note that as $T(\xi)/T_c \equiv \theta(\xi)$ this solution itself specifies the run of normalized temperature. Similarly we can find the normalized density $\rho(\xi)/\rho_c$ from θ^n and the normalized pressure $P(\xi)/P_c$ from θ^{n+1}. These parameters can be readily graphed in all-in-one plots as functions of normalized radius or normalized mass parameter and these show the structural properties of the given polytropic model. In order to do these computations a python function module Mer***ge2Get_LEPol(npol, xi_out, xi_in, y_out, y_in, z_out, z_in)*** was written and added to the earlier integration script [3]. The combined script is listed in the Appendix. All other polytropic model parameters can then be computed from expressions given in [1].

3. Discussion and Conclusion

While doing the bidirectional integration with each of the four integrators mentioned above in turn for each polytropic index the resulting value of the Emden radius was noted and then compared with the corresponding reference value obtained using extended precision Python script [5] based on Boyd's Chebyshev Spectral method [4]. The values produced by the two step size adapting integrators ***dopri5()*** and ***dop853()*** were closest to the reference value with the smallest difference and hence they are listed in Table 1. It may be noted that ***dopri5*** is based on a pair of embedded and optimized runge-kutta formulas of orders 5 and 4, found by Dormand and Prince, together with a dense output interpolation of order 4. Here the order 5 method is used as a

proxy for the exact value to estimate the error of the order 4 method which of course has a gross truncation error varying as the fourth power of the step size. If the error does not fall into a predetermined range relative to step size and problem scale then the step size is reduced or increased as needed so that the integration is steered to have a predetermined global error. Similarly ***dop853*** method is based on a pair of embedded formulas of orders 8 and 5 combined with a dense output interpolation of order 7.

In the case of n = 3 polytrope the dopri5 result differs from the reference value by 2.3×10^{-15} whereas the dop853 result differs by 3.05×10^{-14}. In other cases the dop853 results differs less or even same as the result from dopri5. So the conclusion is that both these are suitable for use as integrators in this problem. If lower accuracy is acceptable the other two integrators may also be used. The preference for dopri5 is dependent on setting the fitting point (**xi_fit**) close to the surface at **90%** of the Emden radius (**xi_s**) of the polytrope. This has the advantage that the fitting iterations converge quickly. In addition the inward and outward parts of the solution (**y** and **z**) at the fit point are almost equal and so the merger of the corresponding arrays has negligible error (a sample print out is given in the Appendix). Further the numerical difference factor ***eps*** should be set to 10^{-15} or so. This is fixed by trial and error so that the result is closest to the reference value shown in the second column of Table 1 at least upto first 14 digits.

Table 1 Comparing Emden Radii from dopri5() and dop853() with the Reference value from Boyd's Chebyshev Spectral Method

n	Reference value [4]	dopri5() (this work)	dop853() (this work)
0.5	2.7526980540652	2.75269805406500634	2.75269805406500900
1.0	3.1415926535897932	3.14159265358981177	3.14159265358980777
1.5	3.6537537362191223	3.65375373621913013	3.65375373621912969
2.0	4.35287459594612468	4.35287459594613413	4.35287459594613679
2.5	5.35527545901077946	5.35527545901080337	5.35527545901080426
3.0	6.89684861937696037	6.89684861937695803	6.89684861937699090
3.25	8.01893752727151142	8.01893752727152176	8.01893752727152176
3.5	9.53580534424485044	9.53580534424484583	9.53580534424487070
4.0	14.97154634883809510	14.97154634883809621	14.97154634883809976
4.5	31.83646324469428526	31.83646324469370725	31.83646324469442135

Acknowledgement

I wish to thank Mike Zingale for information used from [2] and [3] and Nikola Merkov for assistance in developing and hosting the Python version of Boyd's script in [5].

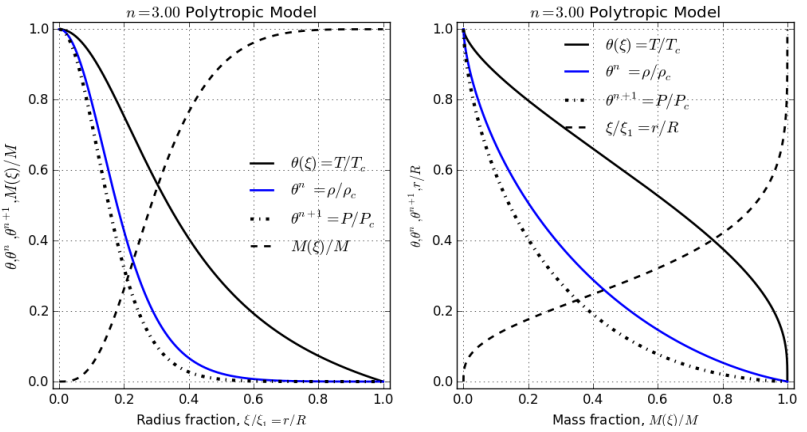

Figure 2 The n=3 polytrope structure as drawn against radius fraction (left) and mass fraction (right).

References

[1] M.N. Anandaram, "***On Emden's Polytropes: Gas Globes in Hydrostatic Equilibrium***", Mapana J Sci, Vol.12, No. 1, pp. 85-114, 2014.

[2] http://bender.astro.sunysb.edu/classes/stars/notes/models.pdf

[3] http://bender.astro.sunysb.edu/classes/stars/notes/le-fit.py

[4] J.P. Boyd, "***Chebyshev Spectral Methods and the Lane-Emden Problem***", Numer. Math. Theor. Meth. Appl., Vol. 4, No. 2, pp. 142-157, 2011.

[5] https://github.com/nikola-m/another-chebpy/blob/master/boyd_polytropes.py

Appendix

The source code listing of the complete Python script with line numbers is given below:

Python Script: LaneEmdenSol_Fit.py

```
1   # -*- coding: utf-8 -*-
    from __future__ import division #, printfunction
    import scipy  # scipy includes all of math and numpy!
```

```
from scipy.integrate import ode
import matplotlib.pyplot as plt

#Original Python Script by Mike Zingale has been taken from
 #http://bender.astro.sunysb.edu/classes/stars/notes/le-fit.py
( line Nos. 11 to 148)
#Additional Script added by M.N. Anandaram to output complete
model.

def rhs(xi, H, n):
    """ input: [y, z];  output/return: [dy = z, d2y = dz] """
    y = H[0]; z = H[1]
    dy = z
    if (xi == 0.0):
        d2y = -1.0/3.0     # <==> dz = (2.0/3.0) - y**n = 2/3 - 1 = -1/3
    else:
        d2y = -2.0 * z/xi - y**n    # dz

    return scipy.array([dy,d2y])

def le_integrate(xi_start, xi_end, H0, n):
    # use the explicit (adams) integrator from the VODE package
    #r = ode(rhs).set_integrator("vode", method="adams",   #"bdf"
    #                  atol=2.e-15, rtol=3.e-14, nsteps=15000, order=12)
    r = ode(rhs).set_integrator("dopri5",   #"dop853",  #"lsoda",
                atol=2.e-15, rtol=3.e-14) #, nsteps=15000)
    r.set_initial_value(H0, xi_start)

    # pass n into the rhs() routine
    r.set_f_params(n)
    xi_out = [xi_start] # store starting values
    y_out = [H0[0]]
    z_out = [H0[1]]

    # we want to know what the solution looks like on some regular
grid
    xi = scipy.linspace(xi_start, xi_end, 800)
    iend = 1
    if (xi_end > xi_start):
        while r.successful() and r.t < xi_end:
            r.integrate(xi[iend])
            xi_out.append(r.t)
            y_out.append(r.y[0])
```

```
            z_out.append(r.y[1])
            iend += 1
        elif (xi_end < xi_start):
          while r.successful() and r.t > xi_end:
            r.integrate(xi[iend])
            xi_out.append(r.t)
            y_out.append(r.y[0])
            z_out.append(r.y[1])
            iend += 1

        return scipy.array(xi_out), scipy.array(y_out), scipy.array(z_out)

    # initial guesses for the unknowns -- if we aren't careful with the
    # guess at the outer boundary, we can get 2 roots.  Here we know
    that
    # n = 1 has xi_s = pi
    n = 3.0    # <--- Here choose any value of n like 0.5,1.,1.5,2.,3.,4.,4.5 or
    4.9
    if (n > 2.0):    xi_s = 10.0
    else:    xi_s = 10.0
    alpha = -0.15     # guesstimated slope dy/dxi at xi_s    (known that
    alpha < 0)
    # set numerical differentiation factor (this multiplies alpha, xi_s)
    eps = 5.0e-15 #eps = 1.0e-8

    # main iteration loop
    converged = 0
    iterno = 1
    while not converged:
      # fitting point set here at 90%
      xi_fit = xi_s * 0.9
      # baseline integration
      # outward from the center
      H0 = scipy.array([1.0,0.0]) # y[0]= 1; y'[0] = 0
      xi_out, y_out, z_out = le_integrate(0.0, xi_fit, H0, n)
      # inward from xi_s
      H0 = scipy.array([0.0,alpha])  # y[xi_s] = 0; y'[xi_s] = alpha
      xi_in, y_in, z_in = le_integrate(xi_s, xi_fit, H0, n)
      # the two functions we want to zero
      nin = len(y_in)
      nout = len(y_out)
      Ybase = y_in[nin-1] - y_out[nout-1]
      Zbase = z_in[nin-1] - z_out[nout-1]
```

```
        # now do alpha + eps*alpha, xi_s
        # inward from xi_s
90      H0  =  scipy.array([0.0,alpha*(1.0+eps)])
        xi_in, y_in, z_in  =   le_integrate(xi_s, xi_fit, H0, n)

        Ya = y_in[nin-1] - y_out[nout-1]
        Za = z_in[nin-1] - z_out[nout-1]
        # our derivatives
        dYdalpha = (Ya-Ybase)/(alpha*eps)
        dZdalpha = (Za-Zbase)/(alpha*eps)

        # now do alpha, xi_s + eps*xi_s inward from xi_s
100     H0 = scipy.array([0.0,alpha])
        xi_in, y_in, z_in = le_integrate(xi_s*(1.0+eps), xi_fit, H0, n)
        Yxi = y_in[nin-1] - y_out[nout-1]
        Zxi = z_in[nin-1] - z_out[nout-1]
        # our derivatives
        dYdxi_s = (Yxi-Ybase)/(xi_s*eps)
106     dZdxi_s = (Zxi-Zbase)/(xi_s*eps)
        # compute the correction for our two parameters
        if (dZdxi_s - dZdalpha*dYdxi_s/dYdalpha == 0.0):
            dxi_s = 2.0*dxi_s
110     else:
            dxi_s = - (Zbase - dZdalpha*Ybase/dYdalpha)/  (dZdxi_s -
    dZdalpha*dYdxi_s/dYdalpha)
        dalpha = -(Ybase + dYdxi_s*dxi_s)/dYdalpha
        # limit the changes per iteration
        if (abs(dalpha) > 0.1*abs(alpha)):
            dalpha = 0.1*abs(alpha)*scipy.copysign(1.0,dalpha)
        if (abs(dxi_s) > 0.1*abs(xi_s)):
            dxi_s = 0.1*abs(xi_s)*scipy.copysign(1.0,dxi_s)
        #print "corrections: %3.10e, %3.10e, %3.16f " %(dalpha, dxi_s, xi_s)
        alpha += dalpha
120     xi_s += dxi_s
        #print ("corrections: %3.10e, %3.10e, %3.17f " %(dalpha, dxi_s,
    xi_s))
        iterno += 1
        print ("corrections: %3.10e, %3.10e, %3.17f " %(dalpha, dxi_s,
    xi_s))

        if (abs(dalpha) < eps*abs(alpha) and abs(dxi_s) < eps*abs(xi_s)):
            converged = 1
            print("\nLEEq solutions converge at xi_fit after %3d
```

```
      iterations"%iterno)
        #plt.figure()
        plt.clf()
130     plt.plot(xi_in, y_in, color="k",lw=2, label=r"$\theta$")
        plt.plot(xi_out, y_out, color="k", ls="--",lw=2)
        plt.plot(xi_in, z_in, color="b",lw=2, label=r"$\theta'$")
        plt.plot(xi_out, z_out, color="b",lw=2, ls="--")
        plt.grid()
        plt.xlabel(r"$\xi$", fontsize=14)
        plt.ylabel(r"$\theta,\, \theta'$", fontsize=14)
        #plt.ylim(-0.02,1.02)
        t = plt.title(r"solution of $\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2
      \frac{d\theta}{d\xi}\ \right) = -\theta^n$ via fitting")
        t.set_y(1.05)
140     ax = plt.gca()
        plt.text(0.5, 0.90, "n = %3.2f polytrope, iteration # %d" % (n,
      iterno),
              transform=ax.transAxes,                    fontsize=11,
      horizontalalignment="center")
        plt.text(0.5, 0.85, "Emden radius, xi_s: %3.17f" %(xi_s),
             transform=ax.transAxes,                     fontsize=11,
      horizontalalignment="center")
        plt.text(0.5, 0.80, "fitting point: xi_fit / xi_s = %.3f"%(xi_fit/xi_s),
             transform=ax.transAxes,                     fontsize=11,
      horizontalalignment="center")
        plt.legend(loc="best", frameon=False)
148     #plt.show()

150   # The following was added by M.N. Anandaram to print all results
      at the
      #  fitting point,to compute all the model solutions and show them as
      graphs
      print "\nExamine both _in and _out solutions to merge them at the
      fitting point:"
      print "xi_s  = %3.17f" %(xi_s)
      print "xi_in = %3.17f" %xi_in[0]
      print "xi1 = (xi_s+xi_in[0])/2 = %3.17f" %((xi_s+xi_in[0])/2);
156   print "fitting point at xi = %3.17f"% xi_fit
157   print "both in, out fitpoints same?: %3.17f ;  %3.17f"%(xi_out[-
      1],xi_in[-1])
      print "theta at fitpoint: %3.17f;  %3.17f"%(y_out[-1],y_in[-1])
      print "theta' at fitpoint: %3.17f;  %3.17f"%(z_out[-1],z_in[-1])
      print "theta' = dtheta/dxi at xi1: %3.17f "%(z_in[0])
```

```
print "[xi**2 * theta'] AT xi1 : %3.17f " %(xi_s**2*z_in[0])

def Merge2Get_LEPol(npol,xi_out,xi_in,y_out,y_in,z_out,z_in):
    # merge xi_in, y_in, z_in vectors carefully with xi_out, y_out, z_out
    xi_in = xi_in[:-1] # duplicate xi_fit point deleted
    xi_in = xi_in[::-1] # reversed for merging with xi_out
    xi = scipy.hstack((xi_out,xi_in)) # merged radius vector
    # adjust smooth continuity of _in vectors at fitting point and then merge
    adj_yout_yin_fit = y_out[-1] / y_in[-1] # ratio y_out/y_in at fit point
    print"adj_yout_yin_fit = %.17f"%adj_yout_yin_fit  #should be ~= 1
    #y_out[-1] = (y_out[-1] + y_in[-1])/2.0 # averaged at xi_fit point
    y_in = adj_yout_yin_fit * y_in #adjust fit-point transition of y_in
    y_in = y_in[:-1] # duplicate y_in[-1] point deleted
    yxi = scipy.hstack((y_out, y_in[::-1])) # reversed y_in merged with y_out
    adj_zout_zin_fit = z_out[-1] / z_in[-1] # ratio z_out/z_in at fit point
    print"adj_zout_zin_fit = %.17f"%adj_zout_zin_fit #should be ~= 1
    #z_out[-1] = (z_out[-1] + z_in[-1])/2.0 # averaged at xi_fit point
    z_in = adj_zout_zin_fit * z_in #adjust fit-point transition of z_in
    z_in = z_in[:-1] # duplicate z_in[-1] point deleted
    zxi = scipy.hstack((z_out, z_in[::-1])) # reversed z_in merged with z_out
    # compute mass fraction from xi,zi; density and pressure fractions from yi
    mxi = xi*xi*zxi # mxi = xi^2.dtheta = M(xi) / [4*pi* (r_n)^3 * rho_c]
    rhoxi = yxi**npol  # density fraction, theta**n = rho(xi) / rho_c
    pgasxi = yxi**(npol+1.0) # pressure fraction, theta**(n+1) = P(xi) / P_c
    return ( scipy.array(xi), scipy.array(mxi), scipy.array(yxi),
             scipy.array(rhoxi),
scipy.array(pgasxi),scipy.array(zxi) )
#Now get polytrope model data: (Rxi and Mxi not normalized here)
Xi,Mxi,Txi,Dxi,Pxi,Zxi = Merge2Get_LEPol(n,xi_out,xi_in,y_out,y_in,z_out,z_in)
G = 6.67259e-8;  Msun = 1.989e33;  Rsun = 6.9599e10;  Lsun = 3.826e33;
```

```
190  mass = 1.0;  rad = 1.0;  Xi1 = Xi[-1];  Zxi1 = Zxi[-1];  Mxi1 = Mxi[-1]
     Pc = 9.048e+14 * mass * mass / rad**4 /(n + 1.0) / Zxi1**2
     #D_mean = Msun/(4*pi/3)/Rsun**3 #
     D_mean = 1.408436186 * mass / rad ** 3
     Dc = D_mean * (- Xi1 /Zxi1 / 3.0)
     alpha_n = rad / Xi1  # Radius scaling Factor
     GBE = -3.80e+48 * 3.0/(5.0-n) * mass * mass / rad
     print "---------------------------------------------------------------"
     print " The basic properties of the Lane-Emden Polytrope are : "
     print "---------------------------------------------------------------"
200  print " Polytropic index selected      : ", n
     print " Radius parameter(Xi1)          : %.17f" %(Xi1)
     print " Radius scaling Factor, alpha_n  : %.14f " %(alpha_n)
     print " Slope, [dtheta/dxi] AT Xi1      : %.17f" %(Zxi1)
     print " Mass parameter, [Xi1**2*Zxi1]   : %.17f" %(Mxi1)
     print " Central Pressure (Pc)          : ", Pc
     print " Central density (Dc)           : ", Dc
207  print " EOS Constant, K=Pc/Dc**(1+1/n)      : %.14e "
     %(Pc/Dc**(1.0+1.0/n))
208  print " Mean density (D_mean)          : ", D_mean
     print " Central Condensation, Dc/D_mean : ", Dc/D_mean
210  print " Binding Energy (GBE)           : ", GBE
     print "---------------------------------------------------------------"
     # Rx and Mx are now normalized here
     Rxf = Xi/Xi1 # max(Rx) # get radius fraction (normalized radius)
     here
     Mxf = Mxi/Mxi1 # mass fraction (normalized mass) M(xi) / M
     where  M == M(xi1)
     # and show them as all-in-one plots vs Radius fraction
     plt.figure(figsize=(12,6)) #(7,10))
     plt.subplot(121) #(211)
     plt.plot(Rxf,Txi,"k",lw=2,label=r"$\theta(\xi)=T/T_c$");
     plt.plot(Rxf,Dxi,"b",lw=2,label=r"$\theta^n=\rho/\rho_c$")
220  plt.plot(Rxf,Pxi,"k-.",lw=3,label=r"$\theta^{n+1}=P/P_c$")
     plt.plot(Rxf,Mxf,"k--",lw=2,label=r"$M(\xi)/M$")
     plt.xlim(-0.02,1.02); plt.ylim(-0.02,1.02)
     plt.title(r"$n = %.2f$ Polytropic Model"%n)
     plt.xlabel(r"Radius fraction, $\xi/\xi_1=r/R$")
     plt.ylabel(r"$\theta,\theta^n,\theta^{n+1},M(\xi)/M$",
     fontsize=14)
     plt.grid();plt.legend(loc="best", frameon=False)
     # also show them as all-in-one plots vs Mass fraction
     plt.subplot(122)  #(212)
```

```
    plt.plot(Mxf,Txi,"k",lw=2,label=r"$\theta(\xi)=T/T_c$")
230 plt.plot(Mxf,Dxi,"b",lw=2,label=r"$\theta^n=\rho/\rho_c$")
    plt.plot(Mxf,Pxi,"k-.",lw=3,label=r"$\theta^{n+1}=P/P_c$")
    plt.plot(Mxf,Rxf,"k--",lw=2,label=r"$\xi/\xi_1=r/R$")
    plt.xlim(-0.02,1.02); plt.ylim(-0.02,1.02);
    plt.grid();plt.legend(loc="best", frameon=False)
    plt.title(r"$n = %.2f$ Polytropic Model"%n)
    plt.xlabel(r"Mass fraction, $M(\xi)/M$")
    plt.ylabel(r"$\theta,\theta^n,\theta^{n+1},r/R$")
238 plt.show()
```