PHY 387N: Relativity Theory II Spring 1998
Project 1 : Group V Scott Caveny, Will Chism, Scott Noble

1 Schwarzchild Geodesic Motion

1.1 Introduction

In this problem we integrate equations of motion for the timelike geodesics in the "equatorial plane" of the Schwarzschild metric:

$$ds^2 = -\left(1 - \frac{2M}{r}\right) dt^2 + \left(1 - \frac{2M}{r}\right)^{-1} dr^2 + r^2 d\Omega^2, \tag{1}$$

where $d\Omega^2$ is the metric on the unit 2-sphere. Due to the symmetry of the Schwarzchild metric under $\theta \to \pi - \theta$, a geodesic with initial position and tangent vector in the equatorial plane: $\theta = \pi/2$ lies entirely within the equatorial plane. A rotational isometry allows any initial geodesic be brought into this plane. Thus considering only equatorial geodesics does not restrict generality. More detailed discussion can be found in Wald(1984), pg. 138-9. The equatorial metric is $d\phi^2$. The equations of motion for timelike geodesics reduce to one second order differential equation in r:

$$\ddot{r} = -\frac{M}{r^2} + \frac{L^2}{r^3} - \frac{3ML^2}{r^4}, \tag{2}$$

where $L = r^2\dot{\phi}$ is a constant of the motion, and the dot denotes differentation w.r.t. *proper time*. In the following section we discuss the implementation of the our numerical analysis, followed by our results in the final section.

1.2 Integration

We integrate the equation of motion by writing it as two first order differential equations:

$$\begin{aligned} \dot{r} &= u & (3) \\ \dot{u} &= -\frac{M}{r^2} + \frac{L^2}{r^3} - \frac{3ML^2}{r^4}. & (4) \end{aligned}$$

Note that the r motion may be derived from an effective potential:

$$V_{eff} \equiv -\frac{M}{r} + \frac{L^2}{2r^2} - \frac{ML^2}{r^3}, \tag{5}$$

given by Wald (6.3.15). We also integrate equation $\dot{\phi} = L/r^2$, as we must output $x = rcos\phi$ and $y = rsin\phi$. The source code to integrate the equations of motion is in the file `orbit.f`, contained in the `problem1` subdirectory. As per the assignment instructions, the executable is in the file `orbit`, with usage: `orbit <L2> <R0> <Rdot0> <dtau> <tauf> [<tol>]`. We use the `lsoda` integrator, as suggested. The program is commented, with usage instructions if the input cannot be recognized. The equations to be integrated are in the file `fcn.f`, with common block in the file `fcn.inc`. The output is in the form required by the program `pp2d`. We use the z spatial coordinate, ignored by `pp2d`, to output the value of r and ϕ at each *proper time* step. This keeps motion in r and ϕ obvious. The source code is compiled with the makefile, also in the `problem1` directory. These are listed below.

```
c==============================================================
c     PHY387N: Project1, Problem1
c     Driver routine which integrates ODEs for massive
c     test-particle in the equatorial plane of the Schwarzschild
c     space-time.
c     Usage: orbit <L2> <R0> <Rdot0> <dtau> <tau_f> [<tau_f>].
c     Output in form of required input into the program pp2d.
c================================================================
      program orbit
      character*5 cdnm
      parameter(cdnm='orbit')
      integer iargc,i4arg,neq,istep,np
      real*8 r8arg,r8_never,M,K,R0,Rdot0,dtau,tau_f,L2
      parameter(neq=3,r8_never=-1.0d-60)
      logical input_ok
c--------------------------------------------------------------
c     lsoda variables
c--------------------------------------------------------------
      external fcn, jac
      real*8 y(neq),tbgn,tend,rtol,atol,tol,default_tol
```

```
      integer itol,itask,istate,iopt,jt,lrw,liw
      parameter(lrw=22+neq*16,liw=20+neq)
      real*8 rwork(lrw)
      integer iwork(liw)
      parameter(default_tol=1.0d-8)
      include 'fcn.inc'
c-----------------------------------------------------------
c     Parse command line arguments (initial values) ...
c-----------------------------------------------------------
      if(iargc() .lt. 5)then
            write(0,*) cdnm//': Insufficient initial values'//
     &        ' specified on command line.'
            write(0,*) 'Usage: '//cdnm//
     &     ' <L2> <R0> <Rdot0> <dtau> <tau_f> [<tol>].'
            go to 900
      end if
      input_ok = .true.
      L2=r8arg(1,r8_never)
      R0=r8arg(2,r8_never)
      Rdot0=r8arg(3,r8_never)
      dtau=r8arg(4,r8_never)
      tau_f=r8arg(5,r8_never)
c------------------------------------------------------------
c If values were inputted incorrectly, then inform the user
c  and then goto 900
c-------------------------------------------------------------
      if(L2 .eq. r8_never .or. R0 .eq. r8_never .or.
     &     Rdot0 .eq. r8_never .or. dtau .eq. r8_never .or.
     &     tau_f .eq. r8_never)then
         write(0,*) cdnm//': Initial values'//
     &        ' not recognizable as real numbers.'
         write(0,*) 'Usage: '//cdnm//
     &     ' <L2> <R0> <Rdot0> <dtau> <tau_f> [<tol>].'
         input_ok = .false.
      end if
      if( .not. input_ok)go to 900
c-----------------------------------------------------------
```

```
c     Get tolerance from command-line if specified,
c     otherwise use default ...
c------------------------------------------------------------
      tol = r8arg(6,default_tol)
c--------------------------------------------------------------------------
c     initial conditions
c--------------------------------------------------------------------------
      M=1.0
      K=1.0
      istep=int(tau_f/dtau)
      y(1)=R0
      y(2)=Rdot0
      y(3)=0.0
      np=2
c--------------------------------------------------------------------------
c     set lsoda parameters
c--------------------------------------------------------------------------
      itol=1
      rtol=tol
      atol=tol
      itask=1
      iopt=0
      jt=2
c--------------------------------------------------------------------------
c     integrate...
c--------------------------------------------------------------------------
      write(*,*) np
      write(*,*) 2.0d0
      write(*,*) (1.0d-1)
      do i=1,istep
         istate=1
         tbgn=real(i-1)*dtau
         tend=real(i)*dtau
         write(*,*) tend
      call lsoda(fcn,neq,y,tbgn,tend,itol,rtol,atol,itask,istate,iopt,
     &           rwork,lrw,iwork,liw,jac,jt)
      write(*,*) 0.0, 0.0, y(1)
```

```
      write(*,*) y(1)*cos(y(3)), y(1)*sin(y(3)), y(3)
      end do

 900  continue
      end

c=========================================================================
c     implements jacobian ('jt'=2 => dummy arg)
c=========================================================================
      subroutine jac
      implicit none
      include 'fcn.inc'
      return
      end
c=========================================================================
c     Driver routine for Schwarzchild spacetime
c=========================================================================
      SUBROUTINE fcn(neq,tau,y,yprime)
      include 'fcn.inc'
      INTEGER neq
      real*8 tau,y(neq),yprime(neq)
      yprime(1)=y(2)
      yprime(2)=-M*K/y(1)**2+L2/y(1)**3-3.0*M*L2/y(1)**4
      yprime(3)=sqrt(L2)/y(1)**2
      RETURN
      END
c--------------------------------------------------------------------
c     fcn.inc
c--------------------------------------------------------------------
      real*8 L2,M,K,R0,Rdot0,dtau
      common /  com_fcn /
     & L2,
     & M,
     & K,
     & R0,
     & Rdot0,
     & dtau
```

```
--------------------------------------------------------------------
.IGNORE:

F77_COMPILE = $(F77) $(F77FLAGS) $(F77CFLAGS)
F77_LOAD = $(F77) $(F77FLAGS) $(F77LFLAGS)

.f.o:
$(F77_COMPILE) $*.f

EXECUTABLES = orbit

all: $(EXECUTABLES)

ORBIT_LIBS = -lodepack -llinpack -lutilio
orbit: orbit.o fcn.o fcn.inc
$(F77_LOAD) orbit.o fcn.o $(ORBIT_LIBS) $(LIBBLAS) -o orbit

clean:
/bin/rm *.o
```

Once all bookeeping issues, such as input/output, visualization, etc..., were resolved, we set $M = 1$, and began testing.

1.3 Numerical Results

We first tested our program in the $L^2 \leq 12M^2$ regime, where there are no extrema of V_{eff}. For $L^2 = 6.0$ and $R_o = 5.0$, the value of $\dot{R}_o$ which separates bound orbits from trapped orbits is $\approx .5059644256$. Our runs show bound orbits for $\dot{R}_o = .50596444$, and unbound orbits for $\dot{R}_o = .50596445$. However our tolerance was set at 1.0×10^{-8}, so a precision of 1.0×10^{-7} is reasonable. The output of these runs are in the files `bound1` and `escape1`, contained in the `DATA` subdirectory (all files referenced are labeled and attached). We then decreased the tolerance to 1.0×10^{-12}, using the optional command line arguement. Trapped orbits exist for $\dot{R}_o = .50596442562$ and unbound orbits for $\dot{R}_o = .50596442563$, in agreement with the expected value. The output of these runs are in the files `bound2` and `escape2`, also contained in the `DATA` subdirectory. We performed similar tests in this regime with various values of L^2 and R_o. Our code behaves accurately to within the requested precision. From this point on, we use the default tolerance of 1.0×10^{-8}.

Next, we tested the regime $L^2 \geq 12M^2$, where the extrema of V_{eff} are given by $R_{\pm} = (L^2 \pm (L^4 - 12M^2)^{1/2})/2M$. Choosing $L^2 = 16.0$ yields $R_+ = 12.0$ and $R_- = 4.0$ We first checked the existence of these orbits for $\tau = 1.0 \times 10^{20}$. They remain circular. The output of these runs are contained in the files `cir.stb` and `cir.unstb`, respectively. We then perturb the unstable orbit, and find a small negative radial velocity (-1.0×10^{-16}) results in the capture of the particle by the black hole, while a small positive radial velocity (1.0×10^{-18}) leads to the escape of the particle. The output of these runs are contained in the files `circ.unstb-pert` and `circ.unstb+pert`. Finally, we perturb the stable orbit in order to see that the particle oscillates at the frequencies: $\omega_r^2 = M(R_+ - 6M)/R_+^3(R_+ - 3M)$ and $\omega_\phi^2 = M/R_+^2(R_+ - 3M)$, given by Wald (6.3.24-5). For $R_+ = 12.0$, $\omega_r^2 = 1/2592$ and $\omega_\phi^2 = 1/1292$. These values yield $\tau_r \approx 320$ and $\tau_\phi \approx 226$. By examination of the radial oscillations around $r = 12.0$ in the perturbed orbit file `circ.stbpert`, we see a radial cycle is completed in about $25\frac{1}{2}$ steps ($d\tau = .125 \times 10^2$), yielding a value $\tau_r \approx 319$. This is in good agreement with the expected value. By examination of the ϕ values in this file, we see an oscillation in ϕ is completed in about 18 steps, yielding a value $\tau_\phi \approx 225$, again, in nice agreement with the expected value. These values could be made more precise, but our code is does appear to be integrating the equations of motion quite accurately. Finally, we generate plots showing motion near a stable circular orbit and the inward spiral from a perturbed unstable circular orbit. These are contained in the files `stb.ps` and `unstb.ps`, respectively.

2 Hydrostatic stellar model

Here we numerically investigate a family of stellar models characterized by the *unphysical* equation of state $P = C\rho^\lambda$. Again, we use `Fortran 77` to implement the `ODEPACK` rountine `LSODA` when integrating the ODEs of this theory.

2.1 Introduction

Schwarzschild's solution for the interior of stars is

$$ds^2 = -\exp\phi\left(r\right)dt^2 + \left(1 - 2m\left(r\right)/r\right)^{-1}dr^2 + r^2 d\Omega^2. \tag{6}$$

For these solutions the equations of hydrostatic equilibrium can be written, in a nonstandard form, as

$$\frac{dm}{dr} = 4\pi r^2 \rho, \tag{7}$$

$$\frac{d\phi}{dr} = \frac{m + 4\pi r^3 P}{r\,(r - 2m)}, \tag{8}$$

and

$$\frac{dP}{dr} = -\,(P + \rho)\,\frac{d\phi}{dr}. \tag{9}$$

Equations 7 through 9 are a complete set of equations when augmented by an equation of state $P \equiv P\,(\rho)$. Note that these coupled ordinary differential equations are first order and independent of the value of $\phi\,(r)$. Also, by virtue of the order of these equations and the equation of state $P\,(\rho)$, the central density $\rho\,(0)$ and central mass $m\,(0) = 0$ are sufficient initial data for integration of our equations.

2.2 Integration

As shown in the source code for `star.f` shown below – the program can be found in the `p387g5/problem2/` directory – the functions $P\,(r)$ and $m\,(r)$ are integrated for the usage $<$ `rho` $(0) >$, $<$ `C` $>$, and $< \lambda > <$ `dr` $>$. For a specified tolerance `tol` the integration proceeds until the pressure crosses an arbitrarty cutoff. To simplify our program we choose to use the tolerance of `LSODA` as that cutoff $P <$ `tol` ≈ 0, which physically corresponds to the surface of the star $P = 0$. The effect of choosing this arbitrary cut-off is discussed further below. In keeping with the assignment, the final output of `star.f` is of the form $<$ `R` $> <$ `rho` $> <$ `P` $> <$ `M` $>$. Here $<$ `M` $>$ denotes the star's mass and $<$ `R` $>$ its radius. The source code for `star.f` follows:

```
c-----------------------------------------------------------------
c
c "star.f" : Assignment #1, Problem #2
c PHY 387N: Relativity #2
c Group #5:  Caveny, Scott
c            Chism, Will
c            Noble, Scott
c------------------------------------------------------------------
```

```
      program        star

      implicit       none

c------------------------------------------------------------
c Setup command line variables and codename.
c------------------------------------------------------------
      character*4    cdnm
      parameter    ( cdnm = 'star' )

      integer        iargc,         indlnb,         i4arg
      real*8         r8arg

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )


c------------------------------------------------------------
c     neq = Order of system.
c------------------------------------------------------------
      integer        neq
      parameter    ( neq = 2 )


      integer        iR_out, idx

      integer        ieq

      logical        input_ok

      integer        default_check, icheck
      parameter    ( default_check = 0 )

      logical        check

      real*8         pi
```

```
      parameter    ( pi = 3.14159265358979323846d0 )

c-------------------------------------------------------------
c     Physics variables:
c
c     R=independent variable of integration
c     C = constant factor
c     lambda = constant exponetial
c     dR = interval of R to save or output values
c     press = pressure at R
c     Pressure() = function that calculates the pressure at R
c                    given rho, C, lambda
c     Density    = function that calculates density, rho, at R
c                    given P, C, lambda
c-------------------------------------------------------------

      real*8         C, lambda, dR, press, Pressure
      real*8         rho, rho_0, Density
      real*8         y0(neq)

c-----------------------------------------------------------
c Declare the variables for the finite differencing
c-------------------------------------------------------------

      integer        max_list
      parameter    ( max_list = 10000)
      real*8         M_list(max_list), P_list(max_list)
      real*8         R_list(max_list), Rho_list(max_list)
      real*8         del_M, del_P, del_R, dMdR, dPdR


c---------------------------------------------------------------------
c Define the unit numbers for the finite diff. data files
c------------------------------------------------------------------

      integer        File_dM, File_dP
      parameter    ( File_dM = 15 )
```

```
      parameter    ( File_dP = 16 )

c------------------------------------------------------------
c     LSODA Variables.
c
c     y(1) = "m" or the mass of star stuff inside shell at current radius
c     y(2) = "P" or the pressure at the radius R
c
c------------------------------------------------------------
      external       diffeqs,       jac

      real*8         y(neq)
      real*8         R_init,       R_end
      integer        itol
      real*8         rtol,       atol
      integer        itask,      istate,     iopt
      integer        lrw, i_irwork

      parameter    ( lrw = 22 + neq * 16 )
      real*8         rwork(lrw)

      integer        liw
      parameter    ( liw = 20 + neq )
      integer        iwork(liw)
      integer        jt

      real*8         tol
      real*8         default_tol
      parameter    ( default_tol = 1.0d-6 )

c---------------------------------------------------------------
c List common blocks
c---------------------------------------------------------------

      save /constants/
      common /constants/ C, lambda
```

```
c-----------------------------------------------------------
c     Parse command line arguments (initial values) ...
c-----------------------------------------------------------
      if( iargc() .lt. 4 ) then
            write(0,*) cdnm//': Insufficient initial values '//
     &                 ' specified on command line.'
            go to 900
      end if

      input_ok = .true.

c--------------------------------------------------------------
c Set initial m = 0
c---------------------------------------------------------------

      y0(1) = 0.0d0

c--------------------------------------------------------------
c Inputs:
c     #1 = rho_0 = Initial density
c     #2 = C = constant factor in P(rho)
c     #3 = lambda = constant exponent
c     #4 = dR = radius interval to save on
c     #5 = tol = tolerance of routine (OPTIONAL -- default=1.0d-6)
c-----------------------------------------------------------
c Get initial density, rho_0
c------------------------------------------------------------

      rho_0 = r8arg(1,r8_never)

c------------------------------------------------------------
c Get C, lambda, and dR
c------------------------------------------------------------

      C = r8arg(2,r8_never)
      lambda = r8arg(3,r8_never)
      dR = r8arg(4, r8_never)
```

```
      if(C .eq. 0.0d0) then
         write(0,*) cdnm//': C needs to be non-zero !!'
      endif

c--------------------------------------------------------------
c Set initial pressure
c------------------------------------------------------------

      y0(2) = Pressure(rho_0)

c-------------------------------------------------------------
c If values were inputted incorrectly, then inform the user
c  and then goto 900
c--------------------------------------------------------------

      if( y0(2) .eq. r8_never .or. C .eq. r8_never .or.
     &     lambda .eq. r8_never .or. dR .eq. r8_never) then
         write(0,*) cdnm//': Initial value Error!'
         input_ok = .false.
      end if


      if( .not. input_ok ) go to 900

c------------------------------------------------------------
c     Get tolerance from command-line if specified,
c     otherwise use default ...
c------------------------------------------------------------

      tol = r8arg(5,default_tol)

      icheck = i4arg(6, default_check)

      if( icheck .eq. 1) then
         check = .true.
```

```
c---------------------------------------------------------------
c Open the files which will contain data from the finite
c   differencing scheme (that is used to test the code).
c----------------------------------------------------------------

         open(File_dM, file='dm_dr.dat', status='NEW',
     &        access='SEQUENTIAL')

         open(File_dP, file='dP_dr.dat', status='NEW',
     &        access='SEQUENTIAL')

      else
         check = .false.
      endif


c     write(*,*) 'rho_0 = ', rho_0
c     write(*,*) 'dR = ', dR
c     write(*,*) 'C = ', C
c     write(*,*) 'lambda = ', lambda

c------------------------------------------------------------
c     Reset the LSODA optional input array space
c------------------------------------------------------------

      do i_irwork = 5, 10
         iwork(i_irwork) = 0
         rwork(i_irwork) = 0.0
      end do

c-------------------------------------------------------------
c Set Maxsteps = 1000
c    where Maxsteps = Maximum # of steps LSODA does per call
c-------------------------------------------------------------

      iwork(6) = 10000
```

```
c------------------------------------------------------------
c Set the LSODA inputs.
c   NOTE: iopt must be 1 if we want to use the optional inputs
c--------------------------------------------------------------

      itol   = 1
      rtol   = tol
      atol   = tol
      itask  = 1
      iopt   = 1
      jt     = 2

c-----------------------------------------------------------
c     Initialize the solution ...
c-----------------------------------------------------------
      do ieq = 1 , neq
         y(ieq) = y0(ieq)
      end do

      rho = rho_0

c---------------------------------------------------------
c Find initial pressure.
c
c-----------------------------------------------------------

      R_init = 0.0d0
      R_end = 0.0d0
      iR_out = 0

c-----------------------------------------------------------
c     Do the integration ...
c-----------------------------------------------------------

      do 600 while( y(2) .gt. tol )
         istate = 1
         iR_out = iR_out + 1
```

```
c-------------------------------------------------------------
c Find density
c----------------------------------------------------------

        rho = Density(y(2))

c--------------------------------------------------------------
c Output stuff
c----------------------------------------------------------

        write(*,*) R_end, rho, y(2), y(1)

        if(check .and. ( iR_out .lt. 10000 )) then
           M_list(iR_out) = y(1)
           P_list(iR_out) = y(2)
           R_list(iR_out) = R_end
           Rho_list(iR_out) = rho
        endif

c----------------------------------------------------------------
c Set endpoint of integration
c----------------------------------------------------------------

        R_end = R_init + dR

        call lsoda(diffeqs,neq,y,R_init,R_end,
    &        itol,rtol,atol,itask,
    &        istate,iopt,rwork,lrw,iwork,liw,jac,jt)

        if( istate .lt. 0 ) then
           write(0,*) 'Lsoda Error #',istate,'!! at loop #', iR_out
           go to 500
        end if


 600  continue
```

```
c-----------------------------------------------------------------------
c Now do the finite differencing if we won't to "check" out the
c the integration procedure.
c-----------------------------------------------------------------------

      if(check) then
         do idx = 2, (iR_out - 1)
            del_M = M_list(idx+1) - M_list(idx-1)
            del_P = P_list(idx+1) - P_list(idx-1)
            del_R = R_list(idx+1) - R_list(idx-1)

            dMdR = 4.0d0 * pi * Rho_list(idx) * R_list(idx)**2

            if( R_list(idx) .eq. 0.0d0 ) then
               dPdR = 0.0d0
            else
               dPdR = -( P_list(idx) + Rho_list(idx) ) *
     &              (M_list(idx) + 4.0d0*pi*P_list(idx)*R_list(idx)**3)
     &              / (R_list(idx)*(R_list(idx) - 2.0d0*M_list(idx) ))
            endif

            write(File_dM, *) R_list(idx), dMdR, (del_M/del_R)
            write(File_dP, *) R_list(idx), dPdR, (del_P/del_R)

         enddo

c-----------------------------------------------------------------
c Close the finite diff. data files.
c------------------------------------------------------------------

         close(File_dM)
         close(File_dP)

      endif

c----------------------------------------------------------------------
```

```
c This is where a successful run ends.
c-----------------------------------------------------------------

 500  continue

      stop


 900  continue

      write(0,*) 'usage: '//cdnm//
     &     ' <rho_0> <C> <lambda> <dR> [<tol> <icheck>]'
      write(0,*) ' '
      stop

      end

c---------------------------------------------------------------------
c     Implements differential equations.
c     Called by ODEPACK routine LSODA.
c---------------------------------------------------------------------
      subroutine diffeqs(neq,r,y,ydot)
         implicit    none

         save /constants/
         common /constants/ C, lambda

         real*8      C, lambda, pi
         parameter  ( pi = 3.14159265358979323846d0 )

         integer     neq
         real*8      r,     y(neq),    ydot(neq), rho_temp
         real*8      Density


         rho_temp = Density( y(2) )
```

```
        ydot(1) =  4.0d0 * pi * rho_temp * r**2

        if( r .eq. (2.0d0*y(1)) .and. r .ne. 0.0d0) then
           write(0,*)'r = 2m !!!!  Big Problem!!!!'
        endif

c-----------------------------------------------------------------
c Check to see if r=0.
c   If so, then in order to prevent hitting a pole, we set ydot(2)=0
c          when r=0
c----------------------------------------------------------------------

        if( r .eq. 0.0d0 ) then
           ydot(2) = 0.0d0

        else
           ydot(2) = -(y(2) + rho_temp) * (y(1) + 4.0d0*pi*y(2)*r**3)
     &          / (r*(r - 2.0d0*y(1)))

        endif

        return
      end

c-----------------------------------------------------------------------
c     Implements Jacobian  (optional) ...
c-----------------------------------------------------------------------
      subroutine jac
        implicit   none


        return
      end

c----------------------------------------------------------------------------
c   *Pressure*  -- definition
c---------------------------------------------------------------------------
```

```
      real*8 function Pressure(rho_tmp)

         save /constants/
         common /constants/ C, lambda

         real*8 rho_tmp
         real*8 C, lambda, pi
         parameter    ( pi = 3.14159265358979323846d0 )

         Pressure = C * (rho_tmp ** lambda)

         return

      end

c------------------------------------------------------------------------
c *Density* -- definition
c--------------------------------------------------------------------

      real*8 function Density(press_tmp1)

         save /constants/
         common /constants/ C, lambda

         real*8 press_tmp1
         real*8 C, lambda, pi
         parameter    ( pi = 3.14159265358979323846d0 )


         if(lambda .eq. 0.0d0) then
            Density = 1.0d0
         else
            Density = (press_tmp1/C)**(1.0d0/lambda)
         endif

c         write(*,*) 'LAMBDA = ', lambda
```

```
c          write(*,*) 'C !!! = ', C

         return

      end
```

We performed three tests on our program. First, using values of $P, r, \rho, andm$ of each integration loop we calculated the values $\Delta P/\Delta r$ and $\Delta m/\Delta r$ and compared the result to the actual values of dP/dr and dm/dr at that instant of the integration (here, $\Delta P/\Delta r$ and $\Delta m/\Delta r$ are symmetric approximations of the above mentioned derivatives:

$$\Delta P/\Delta r = \frac{(P(r+h) - P(r-h))}{2h}$$

$$\Delta m/\Delta r = \frac{(m(r+h) - m(r-h))}{2h}$$

where $\Delta P/\Delta r \to dP/dr$ and $\Delta m/\Delta r \to dm/dr$ as $h \to 0$. The results of this finite differencing are plotted versus r in figures 2.1 and 2.2. In those figures good agreement between $\Delta P/\Delta r$ and dP/dr are shown. Similar results hold for the comparison of $\Delta m/\Delta r$ and dm/dr. Finally, also we wrote an independent program that integrates over $\rho(r)$ in place of $P(r)$. Comparing the results of the two programs for a set of values of $\rho(0)$, λ, and C we found good numerical agreement in general; in some instances agreement was to well over five decimal places.

2.3 Discussion

To investigate this family of stellar models it it useful to first recover physical units (in the cgs system) from the geometric units. Using $P_{geo} = c^4/GP_{cgs}$ and $\rho_{geo} = (c^2/G)\,\rho_{cgs}$ the value of C in our equation of state satisfies $C_{geo} = \left(c^{2\lambda-4}/G^{\lambda-1}\right) C_{cgs}$, where the speed of light c is $3.00 \times 10^{10} cms^{-1}$ and Newton's constant G is $6.67 \times 10^{-8} cm^3 g^{-1} s^{-2}$. Searches over λ with C fixed in geometric units then correspond to searches over λ and C in physical units.

Embedding `star.f` into a program `p_space.f` we performed parameter space surveys over the region $1 \leq \rho() \leq 10^{17}$, $-2 \leq \lambda \leq 2$, and $10^{-3} \leq C \leq 10^3$. The program `p`$_s$`pace.f` surveys regions of parameter space according to parameters defined within the program and writes output to specified files. The source code of `p`$_s$`pace.f` is included below with appropriate commentary included.

Our motivation for choosing to study this region is threefold. Firstly, the region includes the Newtonian sector $P \ll \rho$. Secondly, the region includes the fully relativistic regime $P \approx \rho$ and also exotic models obeying $P > \rho$. Finally, the region contains the physically relavant stellar densities[1] $10^5 gc^{-3} \leq \rho(0) \leq 10^{17} gc^{-3}$ where planets satisfy $\rho(0) \ll 10^5 gc^{-3}$, and white dwarfs and neutron stars are respectively $\rho_{WD}(0) \leq 10^{8.38} gc^{-3}$ and $10^{13.43} gc^{-3} \rho_N(0) \ll 10^{15.78} gc^{-3}$. The region can then be compared to the usual physical results of relativistic hydrostatic equilibrium.

```
c---------------------------------------------------------------
c
c "p_space.f" : Assignment #1, Problem #2
c This program surveys the parameter space of star.f
c----------------------------------------------------------------

      program        p_space

      implicit       none


c-----------------------------------------------------------
c     neq = Order of system.
c-----------------------------------------------------------
      integer        neq
      parameter    ( neq = 2 )


      integer        iR_out, idx

      integer        ieq

      logical        input_ok
```

[1]See for example Figure 24.2 [?].

```
      integer        default_check, icheck
      parameter    ( default_check = 0 )

      logical        check

      real*8         pi
      parameter    ( pi = 3.14159265358979323846d0 )

c------------------------------------------------------------
c     Physics variables:
c
c     R =independent variable of integration
c     C = constant factor
c     lambda = constant exponetial
c     dR = interval of R to save or output values
c     press = pressure at R
c     Pressure() = function that calculates the pressure at R
c                    given rho, C, lambda
c     Density    = function that calculates density, rho, at R
c                    given P, C, lambda
c------------------------------------------------------------

      real*8         C, lambda, dR, press, Pressure
      real*8         rho, rho_0, Density
      real*8         y0(neq)
      integer        ilam, ilam_beg, ilam_end
      real*8         lam_beg, delta_lam
      integer        irho_exp, irho_exp_beg, irho_exp_end
      real*8         rho_factor, rho_beg, delta_rho
      integer        iC_exp, iC_exp_beg, iC_exp_end
      real*8          C_factor, delta_C, C_beg


      parameter    ( ilam_beg = 1, ilam_end = 1)
      parameter    ( lam_beg = 0.10d0 )
      parameter    ( delta_lam = 0.0d0)
```

```
      parameter    ( rho_factor = 1.0d0 )
      parameter    ( rho_beg = 10.1d0 )
      parameter    ( irho_exp_beg = 1, irho_exp_end =100 )
      parameter    ( delta_rho = 1.0d0)
      parameter    ( C_factor = 1.0d0, C_beg =0.0d0 )
      parameter    ( iC_exp_beg = 1, iC_exp_end = 1)
      parameter    ( delta_C =0.0d0)

      parameter    ( dR = 1.0d0 )

      real*8         tol
      parameter    ( tol = 1.0d-8 )


      integer        file_m, file_rho, file_r, file_lam, file_c
      parameter    ( file_m = 17, file_rho = 18 )
      parameter    ( file_r = 19, file_lam = 20 )
      parameter    ( file_c = 21 )

c-----------------------------------------------------------
c     LSODA Variables.
c
c     y(1) = "m" or the mass of star stuff inside shell at current radius
c     y(2) = "P" or the pressure at the radius R
c
c-----------------------------------------------------------
      external       diffeqs,        jac

      real*8         y(neq)
      real*8         R_init,       R_end
      integer        itol
      real*8         rtol,       atol
      integer        itask,      istate,     iopt
      integer        lrw, i_irwork

      parameter    ( lrw = 22 + neq * 16 )
      real*8         rwork(lrw)
```

```
      integer        liw
      parameter    ( liw = 20 + neq )
      integer        iwork(liw)
      integer        jt

      real*8         default_tol
      parameter    ( default_tol = 1.0d-6 )

c----------------------------------------------------------------
c List common blocks
c----------------------------------------------------------------

      save /constants/
      common /constants/ C, lambda

c-------------------------------------------------------------
c     Reset the LSODA optional input array space
c-------------------------------------------------------------

      do i_irwork = 5, 10
         iwork(i_irwork) = 0
         rwork(i_irwork) = 0.0
      end do

c--------------------------------------------------------------
c Set Maxsteps = 1000
c    where Maxsteps = Maximum # of steps LSODA does per call
c--------------------------------------------------------------

      iwork(6) = 10000

c--------------------------------------------------------------
c Set the LSODA inputs.
c   NOTE: iopt must be 1 if we want to use the optional inputs
c----------------------------------------------------------------
```

```
      itol   = 1
      rtol   = tol
      atol   = tol
      itask  = 1
      iopt   = 1
      jt     = 2

c-------------------------------------------------------------
c open files
c--------------------------------------------------------------

c      open(file_m, file='mass.dat', status='NEW',
c     &     access='SEQUENTIAL')

      open(file_rho, file='2lam01_rho_C_1.dat', status='NEW',
     &     access='SEQUENTIAL')

c      open(file_lam, file='lambda.dat', status='NEW',
c     &     access='SEQUENTIAL')

c      open(file_c, file='C.dat', status='NEW',
c     &     access='SEQUENTIAL')

c      open(file_r, file='radius.dat', status='NEW',
c     &     access='SEQUENTIAL')




c-------------------------------------------------------------
c Start the looping
c------------------------------------------------------------

      do iC_exp = iC_exp_beg, iC_exp_end
         do irho_exp = irho_exp_beg, irho_exp_end
            do ilam =ilam_beg, ilam_end
```

```
c-----------------------------------------------------------
c     Initialize the solution ...
c-----------------------------------------------------------

               lambda = lam_beg + delta_lam*dble(ilam)
               rho_0 =  rho_beg
&                                + delta_rho*dble(irho_exp)
               C = C_factor**(C_beg
&                                + delta_C*dble(iC_exp))

               y(1) = 0.0d0
               y(2) = Pressure(rho_0)

               R_init = 0.0d0
               R_end = 0.0d0
               iR_out = 0

c-----------------------------------------------------------
c     Do the integration ...
c-----------------------------------------------------------

               do 747 while( y(2) .gt. tol )
                  istate = 1
                  iR_out = iR_out + 1

c-------------------------------------------------------------
c     Find density
c----------------------------------------------------------

                  rho = Density(y(2))

c----------------------------------------------------------------
c     Set endpoint of integration
c----------------------------------------------------------------

                  R_end = R_init + dR
```

```
              call lsoda(diffeqs,neq,y,R_init,R_end,
     &                     itol,rtol,atol,itask,
     &                     istate,iopt,rwork,lrw,iwork,liw,jac,jt)


c-----------------------------------------------------------------
c     Check for LSODA error.  If there is one, don't quit the program,
c     just go onto to the next set of parameters that we're looping over
c---------------------------------------------------------------------

              if( istate .lt. 0 ) then
                 write(0,*) 'Lsoda Error #',istate,
     &                     '!! at loop #', iR_out
                 write(0,*) 'At: iC_exp = ',iC_exp,
     &                     ' irho_exp = ',irho_exp,
     &                     ' ilam = ', ilam

                 y(2) = 1.0d-5 * tol

              end if

 747          continue

c-------------------------------------------------------------------
c End of LSODA loop.  Output the results now.
c--------------------------------------------------------------------

            if(istate .ge. 0) then
              write(*,825) rho_0, lambda, y(1), R_end, C
              write(file_rho,*) rho_0,  y(1), R_end
c               write(file_m,*) y(1)
c               write(file_r,*) R_end
c               write(file_lam,*) lambda
c               write(file_c,*) C

            endif
```

```
 825          format(5E25.16)


           enddo
         enddo
      enddo


c-----------------------------------------------------------------
c End of do loops.
c--------------------------------------------------------------------

 500  continue

      stop


 900  continue

      stop

      end

c------------------------------------------------------------------------
c     Implements differential equations.
c     Called by ODEPACK routine LSODA.
c------------------------------------------------------------------------
      subroutine diffeqs(neq,r,y,ydot)
         implicit    none

         save /constants/
         common /constants/ C, lambda

         real*8      C, lambda, pi
         parameter  ( pi = 3.14159265358979323846d0 )

         integer     neq
```

```
        real*8      r,     y(neq),    ydot(neq), rho_temp
        real*8      Density


        rho_temp = Density( y(2) )

        ydot(1) =  4.0d0 * pi * rho_temp * r**2

        if( r .eq. (2.0d0*y(1)) .and. r .ne. 0.0d0) then
           write(0,*)'r = 2m !!!!  Big Problem!!!!'
        endif

c-----------------------------------------------------------------
c Check to see if r=0.
c   If so, then in order to prevent hitting a pole, we set ydot(2)=0
c           when r=0
c----------------------------------------------------------------------

        if( r .eq. 0.0d0 ) then
           ydot(2) = 0.0d0

        else
           ydot(2) = -(y(2) + rho_temp) * (y(1) + 4.0d0*pi*y(2)*r**3)
     &           / (r*(r - 2.0d0*y(1)))

        endif



        return
      end

c-----------------------------------------------------------------------
c     Implements Jacobian  (optional) ...
c-----------------------------------------------------------------------
      subroutine jac
        implicit    none
```

```
         return
      end

c---------------------------------------------------------------------------
c   *Pressure*  -- definition
c--------------------------------------------------------------------------

      real*8 function Pressure(rho_tmp)

         save /constants/
         common /constants/ C, lambda

         real*8 rho_tmp
         real*8 C, lambda, pi
         parameter    ( pi = 3.14159265358979323846d0 )

         Pressure = C * (rho_tmp ** lambda)

         return

      end

c------------------------------------------------------------------------
c *Density* -- definition
c---------------------------------------------------------------------

      real*8 function Density(press_tmp1)

         save /constants/
         common /constants/ C, lambda

         real*8 press_tmp1
         real*8 C, lambda, pi
         parameter    ( pi = 3.14159265358979323846d0 )
```

```
    if(lambda .eq. 0.0d0) then
       Density = 1.0d0
    else
       Density = (press_tmp1/C)**(1.0d0/lambda)
    endif

    return

end
```

2.4 Numerical Results

This section consists of four parts. Firstly, for fixed C and λ the influence of the central $\rho(0)$ on the total mass M and radius R is discussed. Secondly, we discuss the behavior of M and R with ρ and C fixed and λ varying. Thirdly we discuss the role of the cutoff and finally, we discuss the role of the step size dR.

Figures E, F and G depict how stellar mass varies with central density, ρ_o, with $\lambda = 0.1$, 0.5, and 1 respectively (all with $C = 1$). From each of these plots, one can see that stellar mass, M, initially increases rapidly in a short interval of ρ_o. For $\lambda = 0.1, 0.5$, we see that M finally reaches a flat direction. These two cases are nearly Newtonian cases, since, initially $P < \rho$. However, for $\lambda = 1$, this initial rise in M is followed by an equally rapid drop-off for increasing λ. In the case where $\lambda = 0.5$, we see that there is a noticeable dip in M after its initial rise, and that this dip becomes a drop to nearly zero when $\lambda = 1$. This suggests that as λ increases towards 1 from $\lambda = 0.5$.

In every encountered case, the stellar radius, R, does not vary significantly (i.e. more than our resolution dR) with ρ_o. This seen in Figures H and I for $C = 1$ and $\lambda = 0.1, 1$ respectively. For most of our runs, we used a $dR = 1$. The parameter dR defines the resolution will we have in finding R values (that is, the difference between any two R values calculated will be an integer multiple of dR). We did run our model for smaller dR values; it seemed as though the size of dR was strongly inversely proportional to the program's run-time and volatility. Hence, we decided to sacrifice maximal precision for efficiency and stability.

In Figures J and K, we still have $\lambda = 1$ but now have $C = 10^5$ and $C = 2x10^5$, respectively. By comparison with Figure G, we see that the increase in C has decreased M dramatically. Also, we see M values of Figure J are approximately twice that of Figure K. So, it appears as though M is inversely proportional to C.

Figures L and M show how $R vs \rho$ is not qualitatively altered by the changing C by a factor of 2; remaining very slowly varying over ρ. For larger values of C, up to $C = 10^5$ our results show this structure persists; e.g, figure M.

As seen in Figure B, for fixed $C = 1$ and $\rho = 1$ and using the cutoff of $P < 10^{-8}$, the total mass M versus λ shows a resonance of height $M_{max} \approx 470$ located at $\lambda \approx 0.92$ and having a FWHM (Full Width at Half-Maximum) of 0.6. It is evident in the plot that the resonance peak migrates as the result of the variation in the cut-off. Using a cut-off of 10^{-12} the resonance in M is centered at $\lambda \approx 1.5$ and reaches a maximum of approximately the same value as that of the lower cut-off. The strengthening of peaks with lower cut-offs indicates the presence of a long tail in the function $P \equiv P(r)$. Cutting off at smaller values of P then naturally produces stars of greater mass. Finally, we note that for these values $C \approx 1$ and $\lambda \approx 1$ the star is in the fully relativistic regime with $P \approx \rho_o$.

We also studied the change in the total radius with the change in both the cut-off and λ. As shown in Figure C, for $C \approx 1$ and $\rho \approx 1$, the total radius of the stars are not sensitive to the cut-off for $\lambda < 0.6$: both the 10^{-12} and 10^{-8} cut-offs for small λ tend towards a radius of 4×10^3 .

Figure D demonstrates how stellar masses and radii vary parametrically together with respect to λ (with $C = \rho_o = 1$); distributions are shown for both cut-off values. These parametric plots of mass versus radius exhibit both physical and unphysical characteristics. First, the radii of the more massive stars are approximately equal to half of the maximum stellar radii allowed. Again, the value of the cut-off plays a noticeable role; the decrease of the cut-off seems to produce more massive stars at small radii. But there is a bold caveat, for the cutoff of 10^{-8} the ratio M/R satisfies the bound $M/R < 4/9$ of General Relativity, which is known to be independent of any given equation of state. However, for the cutoff 10^{-12} the M/R ratio does *not* satisfy the bound and therefore cannot be trusted. This parameteric plot also shows that arbitrarily small mass stars of arbitrarily small radii are seen for small λ when using the cutoff of 10^-8, but are not seen at the lower, relativity violating cutoff 10^{-12}. Although the two curves disagree with respect to the bound $M/R < 4/9$ and the presence of stars of arbitrarily small mass, they do agree on the presence and value of a maximum mass when M is a regarded as a function of R.

References

[1] C. J. Misner, K. Thorne, J. A. Wheeler, *Gravitation* (W.H. Freeman and Co., San Francisco, 1970).

[2] R. M. Wald, *GeneralRelativity*, (University of Chicago Press, Chicago, 1984).