17.4 A Worked Example: Spheroidal Harmonics

The best way to understand the algorithms of the previous sections is to see them employed to solve an actual problem. As a sample problem, we have selected the computation of spheroidal harmonics. (The more common name is spheroidal angle functions, but we prefer the explicit reminder of the kinship with spherical harmonics.) We will show how to find spheroidal harmonics, first by the method of relaxation (§17.3), and then by the methods of shooting (§17.1) and shooting to a fitting point (§17.2).

Spheroidal harmonics typically arise when certain partial differential equations are solved by separation of variables in spheroidal coordinates. They satisfy the following differential equation on the interval $-1 \le x \le 1$:

$$\frac{d}{dx}\left[(1-x^2)\frac{dS}{dx}\right] + \left(\lambda - c^2x^2 - \frac{m^2}{1-x^2}\right)S = 0 \tag{17.4.1}$$

Here m is an integer, c is the "oblateness parameter," and λ is the eigenvalue. Despite the notation, c^2 can be positive or negative. For $c^2 > 0$ the functions are called "prolate," while if $c^2 < 0$ they are called "oblate." The equation has singular points at $x = \pm 1$ and is to be solved subject to the boundary conditions that the solution be regular at $x = \pm 1$. Only for certain values of λ, the eigenvalues, will this be possible.

If we consider first the spherical case, where $c = 0$, we recognize the differential equation for Legendre functions $P_n^m(x)$. In this case the eigenvalues are $\lambda_{mn} = n(n+1)$, $n = m, m+1, \ldots$. The integer n labels successive eigenvalues for fixed m: When $n = m$ we have the lowest eigenvalue, and the corresponding eigenfunction has no nodes in the interval $-1 < x < 1$; when $n = m+1$ we have the next eigenvalue, and the eigenfunction has one node inside $(-1, 1)$; and so on.

A similar situation holds for the general case $c^2 \neq 0$. We write the eigenvalues of (17.4.1) as $\lambda_{mn}(c)$ and the eigenfunctions as $S_{mn}(x; c)$. For fixed m, $n = m, m+1, \ldots$ labels the successive eigenvalues.

The computation of $\lambda_{mn}(c)$ and $S_{mn}(x; c)$ traditionally has been quite difficult. Complicated recurrence relations, power series expansions, etc., can be found in references [1-3]. Cheap computing makes evaluation by direct solution of the differential equation quite feasible.

The first step is to investigate the behavior of the solution near the singular points $x = \pm 1$. Substituting a power series expansion of the form

$$S = (1 \pm x)^\alpha \sum_{k=0}^{\infty} a_k (1 \pm x)^k \tag{17.4.2}$$

in equation (17.4.1), we find that the regular solution has $\alpha = m/2$. (Without loss of generality we can take $m \ge 0$ since $m \to -m$ is a symmetry of the equation.) We get an equation that is numerically more tractable if we factor out this behavior. Accordingly we set

$$S = (1-x^2)^{m/2} y \tag{17.4.3}$$

We then find from (17.4.1) that y satisfies the equation

$$(1-x^2)\frac{d^2y}{dx^2} - 2(m+1)x\frac{dy}{dx} + (\mu - c^2x^2)y = 0 \tag{17.4.4}$$

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

where

$$\mu \equiv \lambda - m(m+1) \tag{17.4.5}$$

Both equations (17.4.1) and (17.4.4) are invariant under the replacement $x \to -x$. Thus the functions S and y must also be invariant, except possibly for an overall scale factor. (Since the equations are linear, a constant multiple of a solution is also a solution.) Because the solutions will be normalized, the scale factor can only be ± 1. If $n-m$ is odd, there are an odd number of zeros in the interval $(-1,1)$. Thus we must choose the antisymmetric solution $y(-x) = -y(x)$ which has a zero at $x = 0$. Conversely, if $n-m$ is even we must have the symmetric solution. Thus

$$y_{mn}(-x) = (-1)^{n-m} y_{mn}(x) \tag{17.4.6}$$

and similarly for S_{mn}.

The boundary conditions on (17.4.4) require that y be regular at $x = \pm 1$. In other words, near the endpoints the solution takes the form

$$y = a_0 + a_1(1-x^2) + a_2(1-x^2)^2 + \ldots \tag{17.4.7}$$

Substituting this expansion in equation (17.4.4) and letting $x \to 1$, we find that

$$a_1 = -\frac{\mu - c^2}{4(m+1)} a_0 \tag{17.4.8}$$

Equivalently,

$$y'(1) = \frac{\mu - c^2}{2(m+1)} y(1) \tag{17.4.9}$$

A similar equation holds at $x = -1$ with a minus sign on the right-hand side. The irregular solution has a different relation between function and derivative at the endpoints.

Instead of integrating the equation from -1 to 1, we can exploit the symmetry (17.4.6) to integrate from 0 to 1. The boundary condition at $x = 0$ is

$$\begin{aligned} y(0) &= 0, \quad n-m \text{ odd} \\ y'(0) &= 0, \quad n-m \text{ even} \end{aligned} \tag{17.4.10}$$

A third boundary condition comes from the fact that any constant multiple of a solution y is a solution. We can thus *normalize* the solution. We adopt the normalization that the function S_{mn} has the same limiting behavior as P_n^m at $x = 1$:

$$\lim_{x \to 1} (1-x^2)^{-m/2} S_{mn}(x;c) = \lim_{x \to 1} (1-x^2)^{-m/2} P_n^m(x) \tag{17.4.11}$$

Various normalization conventions in the literature are tabulated by Flammer [1].

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

Imposing three boundary conditions for the second-order equation (17.4.4) turns it into an eigenvalue problem for λ or equivalently for μ. We write it in the standard form by setting

$$y_1 = y \tag{17.4.12}$$
$$y_2 = y' \tag{17.4.13}$$
$$y_3 = \mu \tag{17.4.14}$$

Then

$$y_1' = y_2 \tag{17.4.15}$$
$$y_2' = \frac{1}{1-x^2}\left[2x(m+1)y_2 - (y_3 - c^2x^2)y_1\right] \tag{17.4.16}$$
$$y_3' = 0 \tag{17.4.17}$$

The boundary condition at $x = 0$ in this notation is

$$\begin{aligned} y_1 &= 0, \quad n-m \text{ odd} \\ y_2 &= 0, \quad n-m \text{ even} \end{aligned} \tag{17.4.18}$$

At $x = 1$ we have two conditions:

$$y_2 = \frac{y_3 - c^2}{2(m+1)} y_1 \tag{17.4.19}$$
$$y_1 = \lim_{x\to 1}(1-x^2)^{-m/2}P_n^m(x) = \frac{(-1)^m(n+m)!}{2^m m!(n-m)!} \equiv \gamma \tag{17.4.20}$$

We are now ready to illustrate the use of the methods of previous sections on this problem.

Relaxation

If we just want a few isolated values of λ or S, shooting is probably the quickest method. However, if we want values for a large sequence of values of c, relaxation is better. Relaxation rewards a good initial guess with rapid convergence, and the previous solution should be a good initial guess if c is changed only slightly.

For simplicity, we choose a uniform grid on the interval $0 \le x \le 1$. For a total of M mesh points, we have

$$h = \frac{1}{M-1} \tag{17.4.21}$$
$$x_k = (k-1)h, \qquad k = 1, 2, \ldots, M \tag{17.4.22}$$

At interior points $k = 2, 3, \ldots, M$, equation (17.4.15) gives

$$E_{1,k} = y_{1,k} - y_{1,k-1} - \frac{h}{2}(y_{2,k} + y_{2,k-1}) \tag{17.4.23}$$

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

Equation (17.4.16) gives

$$E_{2,k} = y_{2,k} - y_{2,k-1} - \beta_k \times \left[\frac{(x_k + x_{k-1})(m+1)(y_{2,k} + y_{2,k-1})}{2} - \alpha_k \frac{(y_{1,k} + y_{1,k-1})}{2} \right] \tag{17.4.24}$$

where

$$\alpha_k = \frac{y_{3,k} + y_{3,k-1}}{2} - \frac{c^2 (x_k + x_{k-1})^2}{4} \tag{17.4.25}$$

$$\beta_k = \frac{h}{1 - \frac{1}{4}(x_k + x_{k-1})^2} \tag{17.4.26}$$

Finally, equation (17.4.17) gives

$$E_{3,k} = y_{3,k} - y_{3,k-1} \tag{17.4.27}$$

Now recall that the matrix of partial derivatives $S_{i,j}$ of equation (17.3.8) is defined so that i labels the equation and j the variable. In our case, j runs from 1 to 3 for y_j at $k-1$ and from 4 to 6 for y_j at k. Thus equation (17.4.23) gives

$$\begin{aligned} S_{1,1} &= -1, & S_{1,2} &= -\frac{h}{2}, & S_{1,3} &= 0 \\ S_{1,4} &= 1, & S_{1,5} &= -\frac{h}{2}, & S_{1,6} &= 0 \end{aligned} \tag{17.4.28}$$

Similarly equation (17.4.24) yields

$$\begin{aligned} S_{2,1} &= \alpha_k \beta_k / 2, & S_{2,2} &= -1 - \beta_k (x_k + x_{k-1})(m+1)/2, \\ S_{2,3} &= \beta_k (y_{1,k} + y_{1,k-1})/4 & S_{2,4} &= S_{2,1}, \\ S_{2,5} &= 2 + S_{2,2}, & S_{2,6} &= S_{2,3} \end{aligned} \tag{17.4.29}$$

while from equation (17.4.27) we find

$$\begin{aligned} S_{3,1} &= 0, & S_{3,2} &= 0, & S_{3,3} &= -1 \\ S_{3,4} &= 0, & S_{3,5} &= 0, & S_{3,6} &= 1 \end{aligned} \tag{17.4.30}$$

At $x = 0$ we have the boundary condition

$$E_{3,1} = \begin{cases} y_{1,1}, & n - m \text{ odd} \\ y_{2,1}, & n - m \text{ even} \end{cases} \tag{17.4.31}$$

Recall the convention adopted in the `solvde` routine that for one boundary condition at $k = 1$ only $S_{3,j}$ can be nonzero. Also, j takes on the values 4 to 6 since the boundary condition involves only y_k, not y_{k-1}. Accordingly, the only nonzero values of $S_{3,j}$ at $x = 0$ are

$$\begin{aligned} S_{3,4} &= 1, & n - m \text{ odd} \\ S_{3,5} &= 1, & n - m \text{ even} \end{aligned} \tag{17.4.32}$$

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

At $x = 1$ we have

$$E_{1,M+1} = y_{2,M} - \frac{y_{3,M} - c^2}{2(m+1)} y_{1,M} \tag{17.4.33}$$

$$E_{2,M+1} = y_{1,M} - \gamma \tag{17.4.34}$$

Thus

$$S_{1,4} = -\frac{y_{3,M} - c^2}{2(m+1)}, \qquad S_{1,5} = 1, \qquad S_{1,6} = -\frac{y_{1,M}}{2(m+1)} \tag{17.4.35}$$

$$S_{2,4} = 1, \qquad S_{2,5} = 0, \qquad S_{2,6} = 0 \tag{17.4.36}$$

Here now is the sample program that implements the above algorithm. We need a main program, `sfroid`, that calls the routine `solvde`, and we must supply the subroutine `difeq` called by `solvde`. For simplicity we choose an equally spaced mesh of `m` = 41 points, that is, $h = .025$. As we shall see, this gives good accuracy for the eigenvalues up to moderate values of $n - m$.

Since the boundary condition at $x = 0$ does not involve y_1 if $n - m$ is even, we have to use the `indexv` feature of `solvde`. Recall that the value of `indexv(j)` describes which column of `s(i,j)` the variable `y(j)` has been put in. If $n - m$ is even, we need to interchange the columns for y_1 and y_2 so that there is not a zero pivot element in `s(i,j)`.

The program prompts for values of m and n. It then computes an initial guess for y based on the Legendre function P_n^m. It next prompts for c^2, solves for y, prompts for c^2, solves for y using the previous values as an initial guess, and so on.

```
      PROGRAM sfroid
      INTEGER NE,M,NB,NCI,NCJ,NCK,NSI,NSJ,NYJ,NYK
      COMMON /sfrcom/ x,h,mm,n,c2,anorm                Communicates with difeq.
      PARAMETER (NE=3,M=41,NB=1,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     *     NSJ=2*NE+1,NYJ=NE,NYK=M)
C     USES plgndr,solvde
```

Sample program using `solvde`. Computes eigenvalues of spheroidal harmonics $S_{mn}(x;c)$ for $m \geq 0$ and $n \geq m$. In the program, m is `mm`, c^2 is `c2`, and γ of equation (17.4.20) is `anorm`.

```
      INTEGER i,itmax,k,mm,n,indexv(NE)
      REAL anorm,c2,conv,deriv,fac1,fac2,h,q1,slowc,
     *     c(NCI,NCJ,NCK),s(NSI,NSJ),scalv(NE),x(M),y(NE,M),plgndr
      itmax=100
      conv=5.e-6
      slowc=1.
      h=1./(M-1)
      c2=0.
      write(*,*)'ENTER M,N'
      read(*,*)mm,n
      if(mod(n+mm,2).eq.1)then                     No interchanges necessary.
        indexv(1)=1
        indexv(2)=2
        indexv(3)=3
      else                                         Interchange y1 and y2.
        indexv(1)=2
        indexv(2)=1
        indexv(3)=3
      endif
      anorm=1.                                     Compute γ.
```

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

```
      if(mm.NE.0)then
        q1=n
        do 11 i=1,mm
          anorm=-.5*anorm*(n+i)*(q1/i)
          q1=q1-1.
        enddo 11
      endif
      do 12 k=1,M-1                                   Initial guess.
        x(k)=(k-1)*h
        fac1=1.-x(k)**2
        fac2=fac1**(-mm/2.)
        y(1,k)=plgndr(n,mm,x(k))*fac2                 P_n^m from §6.8.
        deriv=-((n-mm+1)*plgndr(n+1,mm,x(k))-(n+1)*
     *       x(k)*plgndr(n,mm,x(k)))/fac1             Derivative of P_n^m from a recurrence re-
        y(2,k)=mm*x(k)*y(1,k)/fac1+deriv*fac2             lation.
        y(3,k)=n*(n+1)-mm*(mm+1)
      enddo 12
      x(M)=1.                                         Initial guess at x = 1 done separately.
      y(1,M)=anorm
      y(3,M)=n*(n+1)-mm*(mm+1)
      y(2,M)=(y(3,M)-c2)*y(1,M)/(2.*(mm+1.))
      scalv(1)=abs(anorm)
      scalv(2)=max(abs(anorm),y(2,M))
      scalv(3)=max(1.,y(3,M))
1     continue
      write (*,*) 'ENTER C**2 OR 999 TO END'
      read (*,*) c2
      if (c2.eq.999.) stop
      call solvde(itmax,conv,slowc,scalv,indexv,NE,NB,M,y,NYJ,NYK,
     *     c,NCI,NCJ,NCK,s,NSI,NSJ)
      write (*,*) ' M = ',mm,' N = ',n,
     *     ' C**2 = ',c2,' LAMBDA = ',y(3,1)+mm*(mm+1)
      goto 1                                          for another value of c^2.
      END



      SUBROUTINE difeq(k,k1,k2,jsf,is1,isf,indexv,ne,s,nsi,nsj,y,nyj,nyk)
      INTEGER is1,isf,jsf,k,k1,k2,ne,nsi,nsj,nyj,nyk,indexv(nyj),M
      REAL s(nsi,nsj),y(nyj,nyk)
      COMMON /sfrcom/ x,h,mm,n,c2,anorm
      PARAMETER (M=41)
        Returns matrix s(i,j) for solvde.
      INTEGER mm,n
      REAL anorm,c2,h,temp,temp2,x(M)
      if(k.eq.k1) then                    Boundary condition at first point.
        if(mod(n+mm,2).eq.1)then
          s(3,3+indexv(1))=1.             Equation (17.4.32).
          s(3,3+indexv(2))=0.
          s(3,3+indexv(3))=0.
          s(3,jsf)=y(1,1)                 Equation (17.4.31).
        else
          s(3,3+indexv(1))=0.             Equation (17.4.32).
          s(3,3+indexv(2))=1.
          s(3,3+indexv(3))=0.
          s(3,jsf)=y(2,1)                 Equation (17.4.31).
        endif
      else if(k.gt.k2) then               Boundary conditions at last point.
        s(1,3+indexv(1))=-(y(3,M)-c2)/(2.*(mm+1.))          Equation (17.4.35).
        s(1,3+indexv(2))=1.
        s(1,3+indexv(3))=-y(1,M)/(2.*(mm+1.))
        s(1,jsf)=y(2,M)-(y(3,M)-c2)*y(1,M)/(2.*(mm+1.))     Equation (17.4.33).
        s(2,3+indexv(1))=1.               Equation (17.4.36).
        s(2,3+indexv(2))=0.
```

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

```
        s(2,3+indexv(3))=0.
        s(2,jsf)=y(1,M)-anorm              Equation (17.4.34).
    else                                   Interior point.
        s(1,indexv(1))=-1.                 Equation (17.4.28).
        s(1,indexv(2))=-.5*h
        s(1,indexv(3))=0.
        s(1,3+indexv(1))=1.
        s(1,3+indexv(2))=-.5*h
        s(1,3+indexv(3))=0.
        temp=h/(1.-(x(k)+x(k-1))**2*.25)
        temp2=.5*(y(3,k)+y(3,k-1))-c2*.25*(x(k)+x(k-1))**2
        s(2,indexv(1))=temp*temp2*.5                              Equation (17.4.29).
        s(2,indexv(2))=-1.-.5*temp*(mm+1.)*(x(k)+x(k-1))
        s(2,indexv(3))=.25*temp*(y(1,k)+y(1,k-1))
        s(2,3+indexv(1))=s(2,indexv(1))
        s(2,3+indexv(2))=2.+s(2,indexv(2))
        s(2,3+indexv(3))=s(2,indexv(3))
        s(3,indexv(1))=0.                                         Equation (17.4.30).
        s(3,indexv(2))=0.
        s(3,indexv(3))=-1.
        s(3,3+indexv(1))=0.
        s(3,3+indexv(2))=0.
        s(3,3+indexv(3))=1.
        s(1,jsf)=y(1,k)-y(1,k-1)-.5*h*(y(2,k)+y(2,k-1))           Equation (17.4.23).
        s(2,jsf)=y(2,k)-y(2,k-1)-temp*((x(k)+x(k-1))              Equation (17.4.24).
*            *.5*(mm+1.)*(y(2,k)+y(2,k-1))-temp2*
*            .5*(y(1,k)+y(1,k-1)))
        s(3,jsf)=y(3,k)-y(3,k-1)                                  Equation (17.4.27).
    endif
    return
    END
```

You can run the program and check it against values of $\lambda_{mn}(c)$ given in the tables at the back of Flammer's book [1] or in Table 21.1 of Abramowitz and Stegun [2]. Typically it converges in about 3 iterations. The table below gives a few comparisons.

Selected Output of `sfroid`

m	n	c^2	λ_{exact}	$\lambda_{\texttt{sfroid}}$
2	2	0.1	6.01427	6.01427
		1.0	6.14095	6.14095
		4.0	6.54250	6.54253
2	5	1.0	30.4361	30.4372
		16.0	36.9963	37.0135
4	11	-1.0	131.560	131.554

Shooting

To solve the same problem via shooting (§17.1), we supply a subroutine `derivs` that implements equations (17.4.15)–(17.4.17). We will integrate the equations over the range $-1 \le x \le 0$. We provide the subroutine `load` which sets the eigenvalue y_3 to its current best estimate, `v(1)`. It also sets the boundary values of y_1 and

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

y_2 using equations (17.4.20) and (17.4.19) (with a minus sign corresponding to $x = -1$). Note that the boundary condition is actually applied a distance dx from the boundary to avoid having to evaluate y_2' right on the boundary. The subroutine score follows from equation (17.4.18).

```
      PROGRAM sphoot
        Sample program using shoot. Computes eigenvalues of spheroidal harmonics S_mn(x;c) for
        m ≥ 0 and n ≥ m. Be sure that routine funcv for newt is provided by shoot (§17.1).
      INTEGER i,m,n,nvar,N2
      PARAMETER (N2=1)
      REAL c2,dx,gamma,q1,x1,x2,v(N2)
      LOGICAL check
      COMMON /sphcom/ c2,gamma,dx,m,n          Communicates with load, score, and derivs.
      COMMON /caller/ x1,x2,nvar               Communicates with shoot.
C     USES newt
      dx=1.e-4                                 Avoid evaluating derivatives exactly at x = −1.
      nvar=3                                   Number of equations.
1     write(*,*) 'input m,n,c-squared (999 to end)'
      read(*,*) m,n,c2
      if (c2.eq.999.) stop
      if ((n.lt.m).or.(m.lt.0)) goto 1
      gamma=1.0                                Compute γ of equation (17.4.20).
      q1=n
      do 11 i=1,m
        gamma=-0.5*gamma*(n+i)*(q1/i)
        q1=q1-1.0
      enddo 11
      v(1)=n*(n+1)-m*(m+1)+c2/2.0              Initial guess for eigenvalue.
      x1=-1.0+dx                               Set range of integration.
      x2=0.0
      call newt(v,N2,check)                    Find v that zeros function f in score.
      if(check)then
        write(*,*)'shoot failed; bad initial guess'
      else
        write(*,'(1x,t6,a)') 'mu(m,n)'
        write(*,'(1x,f12.6)') v(1)
        goto 1
      endif
      END

      SUBROUTINE load(x1,v,y)
      INTEGER m,n
      REAL c2,dx,gamma,x1,y1,v(1),y(3)
      COMMON /sphcom/ c2,gamma,dx,m,n
        Supplies starting values for integration at x = −1 + dx.
      y(3)=v(1)
      if(mod(n-m,2).eq.0)then
        y1=gamma
      else
        y1=-gamma
      endif
      y(2)=-(y(3)-c2)*y1/(2*(m+1))
      y(1)=y1+y(2)*dx
      return
      END

      SUBROUTINE score(x2,y,f)
      INTEGER m,n
      REAL c2,dx,gamma,x2,f(1),y(3)
      COMMON /sphcom/ c2,gamma,dx,m,n
        Tests whether boundary condition at x = 0 is satisfied.
      if (mod(n-m,2).eq.0) then
        f(1)=y(2)
```

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

```
      else
        f(1)=y(1)
      endif
      return
      END

      SUBROUTINE derivs(x,y,dydx)
      INTEGER m,n
      REAL c2,dx,gamma,x,dydx(3),y(3)
      COMMON /sphcom/ c2,gamma,dx,m,n
        Evaluates derivatives for odeint.
      dydx(1)=y(2)
      dydx(2)=(2.0*x*(m+1.0)*y(2)-(y(3)-c2*x*x)*y(1))/(1.0-x*x)
      dydx(3)=0.0
      return
      END
```

Shooting to a Fitting Point

For variety we illustrate `shootf` from §17.2 by integrating over the whole range $-1+dx \le x \le 1-dx$, with the fitting point chosen to be at $x=0$. The routine `derivs` is identical to the one for `shoot`. Now, however, there are two load routines. The routine `load1` for $x=-1$ is essentially identical to `load` above. At $x=1$, `load2` sets the function value y_1 and the eigenvalue y_3 to their best current estimates, `v2(1)` and `v2(2)`, respectively. If you quite sensibly make your initial guess of the eigenvalue the same in the two intervals, then `v1(1)` will stay equal to `v2(2)` during the iteration. The subroutine `score` simply checks whether all three function values match at the fitting point.

```
      PROGRAM sphfpt
        Sample program using shootf. Computes eigenvalues of spheroidal harmonics S_mn(x;c)
        for m ≥ 0 and n ≥ m. Be sure that routine funcv for newt is provided by shootf (§17.2).
        The routine derivs is the same as for sphoot.
      INTEGER i,m,n,nvar,nn2,N1,N2,NTOT
      REAL DXX
      PARAMETER (N1=2,N2=1,NTOT=N1+N2,DXX=1.e-4)
      REAL c2,dx,gamma,q1,x1,x2,xf,v1(N2),v2(N1),v(NTOT)
      LOGICAL check
      COMMON /sphcom/ c2,gamma,dx,m,n
        Communicates with load1, load2, score, and derivs.
      COMMON /caller/ x1,x2,xf,nvar,nn2      Communicates with shootf.
      EQUIVALENCE (v1(1),v(1)),(v2(1),v(N2+1))
C     USES newt
      nvar=NTOT                              Number of equations.
      nn2=N2
      dx=DXX                                 Avoid evaluating derivatives exactly at x = ±1.
1     write(*,*) 'input m,n,c-squared (999 to end)'
      read(*,*) m,n,c2
      if (c2.eq.999.) stop
      if ((n.lt.m).or.(m.lt.0)) goto 1
      gamma=1.0                              Compute γ of equation (17.4.20).
      q1=n
      do 11 i=1,m
        gamma=-0.5*gamma*(n+i)*(q1/i)
        q1=q1-1.0
      enddo 11
      v1(1)=n*(n+1)-m*(m+1)+c2/2.0           Initial guess for eigenvalue and function value.
      v2(2)=v1(1)
```

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

```
v2(1)=gamma*(1.-(v2(2)-c2)*dx/(2*(m+1)))
x1=-1.0+dx                              Set range of integration.
x2=1.0-dx
xf=0.                                   Fitting point.
call newt(v,NTOT,check)                 Find v that zeros function f in score.
if(check)then
    write(*,*)'shootf failed; bad initial guess'
else
    write(*,'(1x,t6,a)') 'mu(m,n)'
    write(*,'(1x,f12.6)') v1(1)
    goto 1
endif
END

SUBROUTINE load1(x1,v1,y)
INTEGER m,n
REAL c2,dx,gamma,x1,y1,v1(1),y(3)
COMMON /sphcom/ c2,gamma,dx,m,n
    Supplies starting values for integration at x = -1 + dx.
y(3)=v1(1)
if(mod(n-m,2).eq.0)then
    y1=gamma
else
    y1=-gamma
endif
y(2)=-(y(3)-c2)*y1/(2*(m+1))
y(1)=y1+y(2)*dx
return
END

SUBROUTINE load2(x2,v2,y)
INTEGER m,n
REAL c2,dx,gamma,x2,v2(2),y(3)
COMMON /sphcom/ c2,gamma,dx,m,n
    Supplies starting values for integration at x = 1 - dx.
y(3)=v2(2)
y(1)=v2(1)
y(2)=(y(3)-c2)*y(1)/(2*(m+1))
return
END

SUBROUTINE score(xf,y,f)
INTEGER i,m,n
REAL c2,gamma,dx,xf,f(3),y(3)
COMMON /sphcom/ c2,gamma,dx,m,n
    Tests whether solutions match at fitting point x = 0.
do 12 i=1,3
    f(i)=y(i)
enddo 12
return
END
```

CITED REFERENCES AND FURTHER READING:

Flammer, C. 1957, *Spheroidal Wave Functions* (Stanford, CA: Stanford University Press). [1]

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §21. [2]

Morse, P.M., and Feshbach, H. 1953, *Methods of Theoretical Physics*, Part II (New York: McGraw-Hill), pp. 1502ff. [3]

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).

17.5 Automated Allocation of Mesh Points

In relaxation problems, you have to choose values for the independent variable at the mesh points. This is called *allocating* the grid or mesh. The usual procedure is to pick a plausible set of values and, if it works, to be content. If it doesn't work, increasing the number of points usually cures the problem.

If we know ahead of time where our solutions will be rapidly varying, we can put more grid points there and less elsewhere. Alternatively, we can solve the problem first on a uniform mesh and then examine the solution to see where we should add more points. We then repeat the solution with the improved grid. The object of the exercise is to allocate points in such a way as to represent the solution accurately.

It is also possible to automate the allocation of mesh points, so that it is done "dynamically" during the relaxation process. This powerful technique not only improves the accuracy of the relaxation method, but also (as we will see in the next section) allows internal singularities to be handled in quite a neat way. Here we learn how to accomplish the automatic allocation.

We want to focus attention on the independent variable x, and consider two alternative reparametrizations of it. The first, we term q; this is just the coordinate corresponding to the mesh points themselves, so that $q = 1$ at $\mathtt{k} = 1$, $q = 2$ at $\mathtt{k} = 2$, and so on. Between any two mesh points we have $\Delta q = 1$. In the change of independent variable in the ODEs from x to q,

$$\frac{d\mathbf{y}}{dx} = \mathbf{g} \tag{17.5.1}$$

becomes

$$\frac{d\mathbf{y}}{dq} = \mathbf{g}\frac{dx}{dq} \tag{17.5.2}$$

In terms of q, equation (17.5.2) as an FDE might be written

$$\mathbf{y}_k - \mathbf{y}_{k-1} - \tfrac{1}{2}\left[\left(\mathbf{g}\frac{dx}{dq}\right)_k + \left(\mathbf{g}\frac{dx}{dq}\right)_{k-1}\right] = 0 \tag{17.5.3}$$

or some related version. Note that dx/dq should accompany $\mathbf{g}$. The transformation between x and q depends only on the *Jacobian* dx/dq. Its reciprocal dq/dx is proportional to the density of mesh points.

Now, given the function $\mathbf{y}(x)$, or its approximation at the current stage of relaxation, we are supposed to have some idea of how we want to specify the density of mesh points. For example, we might want dq/dx to be larger where $\mathbf{y}$ is changing rapidly, or near to the boundaries, or both. In fact, we can probably make up a formula for what we would like dq/dx to be proportional to. The problem is that we do not know the proportionality constant. That is, the formula that we might invent would not have the correct integral over the whole range of x so as to make q vary from 1 to M, according to its definition. To solve this problem we introduce a second reparametrization $Q(q)$, where Q is a new independent variable. The relation between Q and q is taken to be *linear*, so that a mesh spacing formula for dQ/dx differs only in its unknown proportionality constant. A linear relation implies

$$\frac{d^2Q}{dq^2} = 0 \tag{17.5.4}$$

or, expressed in the usual manner as coupled first-order equations,

$$\frac{dQ(x)}{dq} = \psi \qquad \frac{d\psi}{dq} = 0 \tag{17.5.5}$$

where ψ is a new intermediate variable. We add these two equations to the set of ODEs being solved.

Completing the prescription, we add a third ODE that is just our desired mesh-density function, namely

$$\phi(x) = \frac{dQ}{dx} = \frac{dQ}{dq}\frac{dq}{dx} \tag{17.5.6}$$

Sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books or CDROMs, visit website
http://www.nr.com or call 1-800-872-7423 (North America only), or send email to directcustserv@cambridge.org (outside North America).