

Chapter 17. Two Point Boundary Value Problems

17.0 Introduction

When ordinary differential equations are required to satisfy boundary conditions at more than one value of the independent variable, the resulting problem is called a *two point boundary value problem*. As the terminology indicates, the most common case by far is where boundary conditions are supposed to be satisfied at two points — usually the starting and ending values of the integration. However, the phrase “two point boundary value problem” is also used loosely to include more complicated cases, e.g., where some conditions are specified at endpoints, others at interior (usually singular) points.

The crucial distinction between initial value problems (Chapter 16) and two point boundary value problems (this chapter) is that in the former case we are able to start an acceptable solution at its beginning (initial values) and just march it along by numerical integration to its end (final values); while in the present case, the boundary conditions at the starting point do not determine a unique solution to start with — and a “random” choice among the solutions that satisfy these (incomplete) starting boundary conditions is almost certain *not* to satisfy the boundary conditions at the other specified point(s).

It should not surprise you that iteration is in general required to meld these spatially scattered boundary conditions into a single global solution of the differential equations. For this reason, two point boundary value problems require considerably more effort to solve than do initial value problems. You have to integrate your differential equations over the interval of interest, or perform an analogous “relaxation” procedure (see below), at least several, and sometimes very many, times. Only in the special case of linear differential equations can you say in advance just how many such iterations will be required.

The “standard” two point boundary value problem has the following form: We desire the solution to a set of N coupled first-order ordinary differential equations, satisfying n_1 boundary conditions at the starting point x_1 , and a remaining set of $n_2 = N - n_1$ boundary conditions at the final point x_2 . (Recall that all differential equations of order higher than first can be written as coupled sets of first-order equations, cf. §16.0.)

The differential equations are

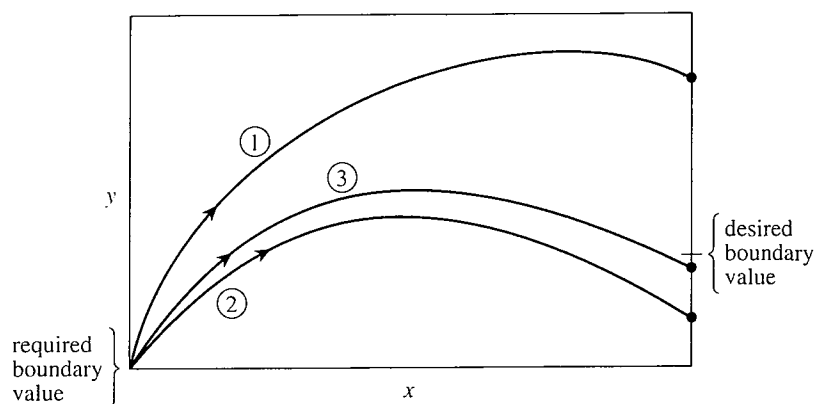


Figure 17.0.1. Shooting method (schematic). Trial integrations that satisfy the boundary condition at one endpoint are “launched.” The discrepancies from the desired boundary condition at the other endpoint are used to adjust the starting conditions, until boundary conditions at both endpoints are ultimately satisfied.

At x_1 , the solution is supposed to satisfy

$$B_{1j}(x_1, y_1, y_2, \dots, y_N) = 0 \quad j = 1, \dots, n_1 \quad (17.0.2)$$

while at x_2 , it is supposed to satisfy

$$B_{2k}(x_2, y_1, y_2, \dots, y_N) = 0 \quad k = 1, \dots, n_2 \quad (17.0.3)$$

There are two distinct classes of numerical methods for solving two point boundary value problems. In the *shooting method* (§17.1) we choose values for all of the dependent variables at one boundary. These values must be consistent with any boundary conditions for *that* boundary, but otherwise are arranged to depend on arbitrary free parameters whose values we initially “randomly” guess. We then integrate the ODEs by initial value methods, arriving at the other boundary (and/or any interior points with boundary conditions specified). In general, we find discrepancies from the desired boundary values there. Now we have a multidimensional root-finding problem, as was treated in §9.6 and §9.7: Find the adjustment of the free parameters at the starting point that zeros the discrepancies at the other boundary point(s). If we liken integrating the differential equations to following the trajectory of a shot from gun to target, then picking the initial conditions corresponds to aiming (see Figure 17.0.1). The shooting method provides a systematic approach to taking a set of “ranging” shots that allow us to improve our “aim” systematically.

As another variant of the shooting method (§17.2), we can guess unknown free parameters at both ends of the domain, integrate the equations to a common midpoint, and seek to adjust the guessed parameters so that the solution joins “smoothly” at the fitting point. In all shooting methods, trial solutions satisfy the differential equations “exactly” (or as exactly as we care to make our numerical integration), but the trial solutions come to satisfy the required boundary conditions only after the iterations are finished.

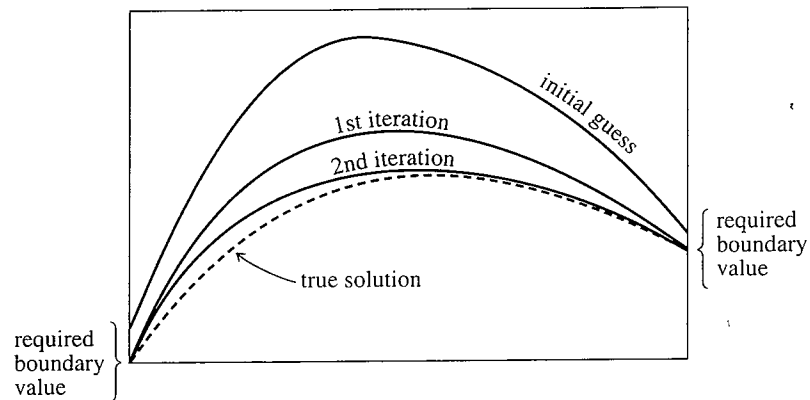


Figure 17.0.2. Relaxation method (schematic). An initial solution is guessed that approximately satisfies the differential equation and boundary conditions. An iterative process adjusts the function to bring it into close agreement with the true solution.

the integration. A trial solution consists of values for the dependent variables at each mesh point, *not* satisfying the desired finite-difference equations, nor necessarily even satisfying the required boundary conditions. The iteration, now called *relaxation*, consists of adjusting all the values on the mesh so as to bring them into successively closer agreement with the finite-difference equations and, simultaneously, with the boundary conditions (see Figure 17.0.2). For example, if the problem involves three coupled equations and a mesh of one hundred points, we must guess and improve three hundred variables representing the solution.

With all this adjustment, you may be surprised that relaxation is ever an efficient method, but (for the right problems) it really is! Relaxation works better than shooting when the boundary conditions are especially delicate or subtle, or where they involve complicated algebraic relations that cannot easily be solved in closed form. Relaxation works best when the solution is smooth and not highly oscillatory. Such oscillations would require many grid points for accurate representation. The number and position of required points may not be known *a priori*. Shooting methods are usually preferred in such cases, because their variable stepsize integrations adjust naturally to a solution's peculiarities.

Relaxation methods are often preferred when the ODEs have extraneous solutions which, while not appearing in the final solution satisfying all boundary conditions, may wreak havoc on the initial value integrations required by shooting. The typical case is that of trying to maintain a dying exponential in the presence of growing exponentials.

Good initial guesses are the secret of efficient relaxation methods. Often one has to solve a problem many times, each time with a slightly different value of some parameter. In that case, the previous solution is usually a good initial guess when the parameter is changed, and relaxation will work well.

Until you have enough experience to make your own judgment between the two methods, you might wish to follow the advice of your authors, who are notorious computer gunslingers: We always shoot first, and only then relax.

Problems Reducible to the Standard Boundary Problem

There are two important problems that can be reduced to the standard boundary value problem described by equations (17.0.1) – (17.0.3). The first is the *eigenvalue problem for differential equations*. Here the right-hand side of the system of differential equations depends on a parameter λ ,

$$\frac{dy_i(x)}{dx} = g_i(x, y_1, \dots, y_N, \lambda) \quad (17.0.4)$$

and one has to satisfy $N + 1$ boundary conditions instead of just N . The problem is overdetermined and in general there is no solution for arbitrary values of λ . For certain special values of λ , the eigenvalues, equation (17.0.4) does have a solution.

We reduce this problem to the standard case by introducing a new dependent variable

$$y_{N+1} \equiv \lambda \quad (17.0.5)$$

and another differential equation

$$\frac{dy_{N+1}}{dx} = 0 \quad (17.0.6)$$

An example of this trick is given in §17.4.

The other case that can be put in the standard form is a *free boundary problem*. Here only one boundary abscissa x_1 is specified, while the other boundary x_2 is to be determined so that the system (17.0.1) has a solution satisfying a total of $N + 1$ boundary conditions. Here we again add an extra constant dependent variable:

$$y_{N+1} \equiv x_2 - x_1 \quad (17.0.7)$$

$$\frac{dy_{N+1}}{dx} = 0 \quad (17.0.8)$$

We also define a new *independent* variable t by setting

$$x - x_1 \equiv t y_{N+1}, \quad 0 \leq t \leq 1 \quad (17.0.9)$$

The system of $N + 1$ differential equations for dy_i/dt is now in the standard form, with t varying between the known limits 0 and 1.

CITED REFERENCES AND FURTHER READING:

- Keller, H.B. 1968, *Numerical Methods for Two-Point Boundary-Value Problems* (Waltham, MA: Blaisdell).
- Kippenhan, R., Weigert, A., and Hofmeister, E. 1968, in *Methods in Computational Physics*, vol. 7 (New York: Academic Press), pp. 129ff.
- Eggleton, P.P. 1971, *Monthly Notices of the Royal Astronomical Society*, vol. 151, pp. 351–364.
- London, R.A., and Flannery, B.P. 1982, *Astrophysical Journal*, vol. 258, pp. 260–269.

17.1 The Shooting Method

In this section we discuss “pure” shooting, where the integration proceeds from x_1 to x_2 , and we try to match boundary conditions at the end of the integration. In the next section, we describe shooting to an intermediate fitting point, where the solution to the equations and boundary conditions is found by launching “shots” from both sides of the interval and trying to match continuity conditions at some intermediate point.

Our implementation of the shooting method exactly implements multidimensional, globally convergent Newton-Raphson (§9.7). It seeks to zero n_2 functions of n_2 variables. The functions are obtained by integrating N differential equations from x_1 to x_2 . Let us see how this works:

At the starting point x_1 there are N starting values y_i to be specified, but subject to n_1 conditions. Therefore there are $n_2 = N - n_1$ *freely specifiable* starting values. Let us imagine that these freely specifiable values are the components of a vector \mathbf{V} that lives in a vector space of dimension n_2 . Then you, the user, knowing the functional form of the boundary conditions (17.0.2), can write a subroutine that generates a complete set of N starting values \mathbf{y} , satisfying the boundary conditions at x_1 , from an arbitrary vector value of \mathbf{V} in which there are no restrictions on the n_2 component values. In other words, (17.0.2) converts to a prescription

$$y_i(x_1) = y_i(x_1; V_1, \dots, V_{n_2}) \quad i = 1, \dots, N \quad (17.1.1)$$

Below, the subroutine that implements (17.1.1) will be called `load`.

Notice that the components of \mathbf{V} might be exactly the values of certain “free” components of \mathbf{y} , with the other components of \mathbf{y} determined by the boundary conditions. Alternatively, the components of \mathbf{V} might parametrize the solutions that satisfy the starting boundary conditions in some other convenient way. Boundary conditions often impose algebraic relations among the y_i , rather than specific values for each of them. Using some auxiliary set of parameters often makes it easier to “solve” the boundary relations for a consistent set of y_i ’s. It makes no difference which way you go, as long as your vector space of \mathbf{V} ’s generates (through 17.1.1) all allowed starting vectors \mathbf{y} .

Given a particular \mathbf{V} , a particular $\mathbf{y}(x_1)$ is thus generated. It can then be turned into a $\mathbf{y}(x_2)$ by integrating the ODEs to x_2 as an initial value problem (e.g., using Chapter 16’s `odeint`). Now, at x_2 , let us define a *discrepancy vector* \mathbf{F} , also of dimension n_2 , whose components measure how far we are from satisfying the n_2 boundary conditions at x_2 (17.0.3). Simplest of all is just to use the right-hand sides of (17.0.3),

$$F_k = B_{2k}(x_2, \mathbf{y}) \quad k = 1, \dots, n_2 \quad (17.1.2)$$

As in the case of \mathbf{V} , however, you can use any other convenient parametrization, as long as your space of \mathbf{F} ’s spans the space of possible discrepancies from the desired boundary conditions, with all components of \mathbf{F} equal to zero if and only if the boundary conditions at x_2 are satisfied. Below, you will be asked to supply a

Now, as far as Newton-Raphson is concerned, we are nearly in business. We want to find a vector value of \mathbf{V} that zeros the vector value of \mathbf{F} . We do this by invoking the globally convergent Newton's method implemented in the routine `newt` of §9.7. Recall that the heart of Newton's method involves solving the set of n_2 linear equations

$$\mathbf{J} \cdot \delta\mathbf{V} = -\mathbf{F} \quad (17.1.3)$$

and then adding the correction back,

$$\mathbf{V}^{\text{new}} = \mathbf{V}^{\text{old}} + \delta\mathbf{V} \quad (17.1.4)$$

In (17.1.3), the Jacobian matrix \mathbf{J} has components given by

$$J_{ij} = \frac{\partial F_i}{\partial V_j} \quad (17.1.5)$$

It is not feasible to compute these partial derivatives analytically. Rather, each requires a *separate* integration of the N ODEs, followed by the evaluation of

$$\frac{\partial F_i}{\partial V_j} \approx \frac{F_i(V_1, \dots, V_j + \Delta V_j, \dots) - F_i(V_1, \dots, V_j, \dots)}{\Delta V_j} \quad (17.1.6)$$

This is done automatically for you in the routine `fdjac` that comes with `newt`. The only input to `newt` that you have to provide is the routine `funcv` that calculates \mathbf{F} by integrating the ODEs. Here is the appropriate routine:

```

C SUBROUTINE shoot(n2,v,f) is named "funcv" for use with "newt"
SUBROUTINE funcv(n2,v,f)
INTEGER n2,nvar,kmax,kount,KMAXX,NMAX
REAL f(n2),v(n2),x1,x2,dxsav,yp,eps
PARAMETER (NMAX=50,KMAXX=200,EPS=1.e-6) At most NMAX coupled ODEs.
COMMON /caller/ x1,x2,nvar
COMMON /path/ kmax,kount,dxsav,yp(KMAXX),yp(NMAX,KMAXX)
C USES derivs,load,odeint,rkqs,score
Routine for use with newt to solve a two point boundary value problem for nvar coupled
ODEs by shooting from x1 to x2. Initial values for the nvar ODEs at x1 are generated
from the n2 input coefficients v(1:n2), using the user-supplied routine load. The routine
integrates the ODEs to x2 using the Runge-Kutta method with tolerance EPS, initial stepsize
h1, and minimum stepsize hmin. At x2 it calls the user-supplied subroutine score to
evaluate the n2 functions f(1:n2) that ought to be zero to satisfy the boundary conditions
at x2. The functions f are returned on output. newt uses a globally convergent Newton's
method to adjust the values of v until the functions f are zero. The user-supplied subroutine
derivs(x,y,dydx) supplies derivative information to the ODE integrator (see Chapter
16). The common block caller receives its values from the main program so that funcv
can have the syntax required by newt. The common block path is included for compatibility
with odeint.
INTEGER nbad,nok
REAL h1,hmin,y(NMAX)
EXTERNAL derivs,rkqs
kmax=0
h1=(x2-x1)/100.
hmin=0.
call load(x1,v,y)
call odeint(y,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,rkqs)
call score(x2,y,f)
return
END

```

For some problems, initial conditions. It is s h1 as another returned a

A complete cycle of the N coupled ODEs: and n_2 for the partial d integrations. This illust boundary value problem

If the differential e since (17.1.3)–(17.1.4) useful, however, in mo

As given here, sho to integrate the ODEs, well be used.

You, the user, mus returns the n-vector y(1 given the freely specif subroutine `score(x2,y ending boundary condi starting vector v(1:n2 other obvious paramete`

In §17.4 we give a

CITED REFERENCES AN Acton, F.S. 1970, *Numeric ematical Association Keller, H.B. 1968, *Numeric Blaisdell).**

17.2 Shooting

The shooting meth be able to traverse the convergence to a corre wrong starting conditic encountering some inc of a square root might shooting would be sty

A different, but re of the set of ODEs. O the singular points, ana feasible to integrate in method to get through further numerical integ a singular point, if onl

For some problems the initial stepsize ΔV might depend sensitively upon the initial conditions. It is straightforward to alter `load` to include a suggested stepsize `h1` as another returned argument and feed it to `fdjac` via a common block.

A complete cycle of the shooting method thus requires $n_2 + 1$ integrations of the N coupled ODEs: one integration to evaluate the current degree of mismatch, and n_2 for the partial derivatives. Each new cycle requires a new round of $n_2 + 1$ integrations. This illustrates the enormous extra effort involved in solving two point boundary value problems compared with initial value problems.

If the differential equations are *linear*, then only one complete cycle is required, since (17.1.3)–(17.1.4) should take us right to the solution. A second round can be useful, however, in mopping up some (never all) of the roundoff error.

As given here, `shoot` uses the quality controlled Runge-Kutta method of §16.2 to integrate the ODEs, but any of the other methods of Chapter 16 could just as well be used.

You, the user, must supply `shoot` with: (i) a subroutine `load(x1, v, y)` which returns the n -vector $y(1:n)$ (satisfying the starting boundary conditions, of course), given the freely specifiable variables of $v(1:n2)$ at the initial point $x1$; (ii) a subroutine `score(x2, y, f)` which returns the discrepancy vector $f(1:n2)$ of the ending boundary conditions, given the vector $y(1:n)$ at the endpoint $x2$; (iii) a starting vector $v(1:n2)$; (iv) a subroutine `derivs` for the ODE integration; and other obvious parameters as described in the header comment above.

In §17.4 we give a sample program illustrating how to use `shoot`.

CITED REFERENCES AND FURTHER READING:

- Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America).
 Keller, H.B. 1968, *Numerical Methods for Two-Point Boundary-Value Problems* (Waltham, MA: Blaisdell).

17.2 Shooting to a Fitting Point

The shooting method described in §17.1 tacitly assumed that the “shots” would be able to traverse the entire domain of integration, even at the early stages of convergence to a correct solution. In some problems it can happen that, for very wrong starting conditions, an initial solution can’t even get from x_1 to x_2 without encountering some incalculable, or catastrophic, result. For example, the argument of a square root might go negative, causing the numerical code to crash. Simple shooting would be stymied.

A different, but related, case is where the endpoints are both singular points of the set of ODEs. One frequently needs to use special methods to integrate near the singular points, analytic asymptotic expansions, for example. In such cases it is feasible to integrate in the direction *away* from a singular point, using the special method to get through the first little bit and then reading off “initial” values for further numerical integration. However it is usually not feasible to integrate into

effort to obtain expansions of "wrong" solutions near the singular point (those not satisfying the desired boundary condition).

The solution to the above mentioned difficulties is *shooting to a fitting point*. Instead of integrating from x_1 to x_2 , we integrate first from x_1 to some point x_f that is between x_1 and x_2 ; and second from x_2 (in the opposite direction) to x_f .

If (as before) the number of boundary conditions imposed at x_1 is n_1 , and the number imposed at x_2 is n_2 , then there are n_2 freely specifiable starting values at x_1 and n_1 freely specifiable starting values at x_2 . (If you are confused by this, go back to §17.1.) We can therefore define an n_2 -vector $V_{(1)}$ of starting parameters at x_1 , and a prescription $\text{load1}(x_1, v_1, y)$ for mapping $V_{(1)}$ into a y that satisfies the boundary conditions at x_1 ,

$$y_i(x_1) = y_i(x_1; V_{(1)1}, \dots, V_{(1)n_2}) \quad i = 1, \dots, N \quad (17.2.1)$$

Likewise we can define an n_1 -vector $V_{(2)}$ of starting parameters at x_2 , and a prescription $\text{load2}(x_2, v_2, y)$ for mapping $V_{(2)}$ into a y that satisfies the boundary conditions at x_2 ,

$$y_i(x_2) = y_i(x_2; V_{(2)1}, \dots, V_{(2)n_1}) \quad i = 1, \dots, N \quad (17.2.2)$$

We thus have a total of N freely adjustable parameters in the combination of $V_{(1)}$ and $V_{(2)}$. The N conditions that must be satisfied are that there be agreement in N components of y at x_f between the values obtained integrating from one side and from the other,

$$y_i(x_f; V_{(1)}) = y_i(x_f; V_{(2)}) \quad i = 1, \dots, N \quad (17.2.3)$$

In some problems, the N matching conditions can be better described (physically, mathematically, or numerically) by using N different functions F_i , $i = 1 \dots N$, each possibly depending on the N components y_i . In those cases, (17.2.3) is replaced by

$$F_i[y(x_f; V_{(1)})] = F_i[y(x_f; V_{(2)})] \quad i = 1, \dots, N \quad (17.2.4)$$

In the program below, the user-supplied subroutine $\text{score}(x_f, y, f)$ is supposed to map an input N -vector y into an output N -vector F . In most cases, you can dummy this subroutine as the identity mapping.

Shooting to a fitting point uses globally convergent Newton-Raphson exactly as in §17.1. Comparing closely with the routine shoot of the previous section, you should have no difficulty in understanding the following routine shootf . The main differences in use are that you have to supply both load1 and load2 . Also, in the calling program you must supply initial guesses for $v_1(1:n_2)$ and $v_2(1:n_1)$. Once again a sample program illustrating shooting to a fitting point is given in §17.4.

C SUBROUTINE $\text{shootf}(n, v, f)$ is named "funcv" for use with "newt"

```

SUBROUTINE funcv(n,v,f)
  INTEGER n,nvar,nn2,kmax,kount,KMAXX,NMAX
  REAL f(n),v(n),x1,x2,xf,dxsav,yp,EPS
  PARAMETER (NMAX=50,KMAXX=200,EPS=1.e-6) At most NMAX equations.
  COMMON /caller/ x1,x2,xf,nvar,nn2
  COMMON /path/ kmax,kount,dxsav,yp(KMAXX),yp(NMAX,KMAXX)

```

C USES derivs , load1 , load2 , odeint , rkqs , score

Routine for use with newt to solve a two point boundary value problem for $nvar$ coupled ODEs by shooting from x_1 and x_2 to a fitting point x_f . Initial values for the $nvar$

ODEs at x_1 (x_2) are supplied routine load1 (array $v(1:n_1+n_2)$ ($v_1(1), v_2(1)$), $v_2(1)$ fine integrates the ODE stepsize h_1 , and minit to evaluate the $nvar$ returned on output. Uses of v until the function supplies derivative info caller receives its required by newt . See compatibility with od

```

INTEGER i,nbad,nok
REAL h1,hmin,f1(NMAX)
EXTERNAL derivs,rkqs
kmax=0
h1=(x2-x1)/100.
hmin=0.
call load1(x1,v,y)
call odeint(y,nvar,x1)
call score(xf,y,f1)
call load2(x2,v(nn2+1))
call odeint(y,nvar,x2)
call score(xf,y,f2)
do i=1,n
  f(i)=f1(i)-f2(i)
enddo
return
END

```

There are boundary conditions — the integration into solution being matched

CITED REFERENCES
 Acton, F.S. 1970, *Numerical Solution of Ordinary Differential Equations*, Wiley-Interscience.
 Keller, H.B. 1968, *Numerical Solution of Two Point Boundary Value Problems*, Prentice-Hall.
 Stoer, J., and Bulirsch, R. §§7.3.5–7.3.6. [1]

17.3 Relaxation

In relaxation methods (FDEs) on a grid or mesh we could replace a general

with an algebraic equation

$$y_k - y_{k-1} = \Delta x$$

ODEs at x_1 (x_2) are generated from the n_2 (n_1) coefficients v_1 (v_2), using the user-supplied routine `load1` (`load2`). The coefficients v_1 and v_2 should be stored in a single array $v(1:n_1+n_2)$ in the main program by an EQUIVALENCE statement of the form $(v_1(1), v(1)), (v_2(1), v(n_2+1))$. The input parameter $n = n_1+n_2 = nvar$. The routine integrates the ODEs to xf using the Runge-Kutta method with tolerance EPS , initial stepsize h_1 , and minimum stepsize $hmin$. At xf it calls the user-supplied subroutine `score` to evaluate the $nvar$ functions f_1 and f_2 that ought to match at xf . The differences f are returned on output. `newt` uses a globally convergent Newton's method to adjust the values of v until the functions f are zero. The user-supplied subroutine `derivs(x,y,dydx)` supplies derivative information to the ODE integrator (see Chapter 16). The common block caller receives its values from the main program so that `funcv` can have the syntax required by `newt`. Set $nn_2 = n_2$ in the main program. The common block path is for compatibility with `odeint`.

```

INTEGER i,nbad,nok
REAL h1,hmin,f1(NMAX),f2(NMAX),y(NMAX)
EXTERNAL derivs,rkqs
kmax=0
h1=(x2-x1)/100.
hmin=0.
call load1(x1,v,y)           Path from x1 to xf with best trial values v1.
call odeint(y,nvar,x1,xf,EPS,h1,hmin,nok,nbad,derivs,rkqs)
call score(xf,y,f1)
call load2(x2,v(nn2+1),y)    Path from x2 to xf with best trial values v2.
call odeint(y,nvar,x2,xf,EPS,h1,hmin,nok,nbad,derivs,rkqs)
call score(xf,y,f2)
do 11 i=1,n
    f(i)=f1(i)-f2(i)
enddo 11
return
END

```

There are boundary value problems where even shooting to a fitting point fails — the integration interval has to be partitioned by several fitting points with the solution being matched at each such point. For more details see [1].

CITED REFERENCES AND FURTHER READING:

- Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America).
- Keller, H.B. 1968, *Numerical Methods for Two-Point Boundary-Value Problems* (Waltham, MA: Blaisdell).
- Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §§7.3.5–7.3.6. [1]

17.3 Relaxation Methods

In *relaxation methods* we replace ODEs by approximate *finite-difference equations* (FDEs) on a grid or mesh of points that spans the domain of interest. As a typical example, we could replace a general first-order differential equation

$$\frac{dy}{dx} = g(x, y) \quad (17.3.1)$$

with an algebraic equation relating function values at two points k $k - 1$.

The form of the FDE in (17.3.2) illustrates the idea, but not uniquely: There are many ways to turn the ODE into an FDE. When the problem involves N coupled first-order ODEs represented by FDEs on a mesh of M points, a solution consists of values for N dependent functions given at each of the M mesh points, or $N \times M$ variables in all. The relaxation method determines the solution by starting with a guess and improving it, iteratively. As the iterations improve the solution, the result is said to *relax* to the true solution.

While several iteration schemes are possible, for most problems our old standby, multi-dimensional Newton's method, works well. The method produces a matrix equation that must be solved, but the matrix takes a special, "block diagonal" form, that allows it to be inverted far more economically both in time and storage than would be possible for a general matrix of size $(MN) \times (MN)$. Since MN can easily be several thousand, this is crucial for the feasibility of the method.

Our implementation couples at most pairs of points, as in equation (17.3.2). More points can be coupled, but then the method becomes more complex. We will provide enough background so that you can write a more general scheme if you have the patience to do so.

Let us develop a general set of algebraic equations that represent the ODEs by FDEs. The ODE problem is exactly identical to that expressed in equations (17.0.1)–(17.0.3) where we had N coupled first-order equations that satisfy n_1 boundary conditions at x_1 and $n_2 = N - n_1$ boundary conditions at x_2 . We first define a mesh or grid by a set of $k = 1, 2, \dots, M$ points at which we supply values for the independent variable x_k . In particular, x_1 is the initial boundary, and x_M is the final boundary. We use the notation \mathbf{y}_k to refer to the entire set of dependent variables y_1, y_2, \dots, y_N at point x_k . At an arbitrary point k in the middle of the mesh, we approximate the set of N first-order ODEs by algebraic relations of the form

$$0 = \mathbf{E}_k \equiv \mathbf{y}_k - \mathbf{y}_{k-1} - (x_k - x_{k-1}) \mathbf{g}_k(x_k, x_{k-1}, \mathbf{y}_k, \mathbf{y}_{k-1}), \quad k = 2, 3, \dots, M \quad (17.3.3)$$

The notation signifies that \mathbf{g}_k can be evaluated using information from both points $k, k - 1$. The FDEs labeled by \mathbf{E}_k provide N equations coupling $2N$ variables at points $k, k - 1$. There are $M - 1$ points, $k = 2, 3, \dots, M$, at which difference equations of the form (17.3.3) apply. Thus the FDEs provide a total of $(M - 1)N$ equations for the MN unknowns. The remaining N equations come from the boundary conditions.

At the first boundary we have

$$0 = \mathbf{E}_1 \equiv \mathbf{B}(x_1, \mathbf{y}_1) \quad (17.3.4)$$

while at the second boundary

$$0 = \mathbf{E}_{M+1} \equiv \mathbf{C}(x_M, \mathbf{y}_M) \quad (17.3.5)$$

The vectors \mathbf{E}_1 and \mathbf{B} have only n_1 nonzero components, corresponding to the n_1 boundary conditions at x_1 . It will turn out to be useful to take these nonzero components to be the *last* n_1 components. In other words, $E_{j,1} \neq 0$ only for $j = n_2 + 1, n_2 + 2, \dots, N$. At the other boundary, only the first n_2 components of \mathbf{E}_{M+1} and \mathbf{C} are nonzero: $E_{j,M+1} \neq 0$ only for $j = 1, 2, \dots, n_2$.

The "solution" of the FDE problem in (17.3.3)–(17.3.5) consists of a set of variables $y_{j,k}$, the values of the N variables y_j at the M points x_k . The algorithm we describe below requires an initial guess for the $y_{j,k}$. We then determine increments $\Delta y_{j,k}$ such that $y_{j,k} + \Delta y_{j,k}$ is an improved approximation to the solution.

Equations for the increments are developed by expanding the FDEs in first-order Taylor series with respect to small changes $\Delta \mathbf{y}_k$. At an interior point, $k = 2, 3, \dots, M$ this gives:

$$\begin{aligned} \mathbf{E}_k(\mathbf{y}_k + \Delta \mathbf{y}_k, \mathbf{y}_{k-1} + \Delta \mathbf{y}_{k-1}) &\approx \mathbf{E}_k(\mathbf{y}_k, \mathbf{y}_{k-1}) \\ &+ \sum_{n=1}^N \frac{\partial \mathbf{E}_k}{\partial y_{n,k-1}} \Delta y_{n,k-1} + \sum_{n=1}^N \frac{\partial \mathbf{E}_k}{\partial y_{n,k}} \Delta y_{n,k} \end{aligned} \quad (17.3.6)$$

For a solution we want the updated value $\mathbf{E}(\mathbf{y} + \Delta \mathbf{y})$ to be zero, so the general set of equations at an interior point can be written in matrix form as

$$\sum_{n=1}^N S_{j,n} \Delta y_{n,k-1} + \sum_{n=N+1}^{2N} S_{j,n} \Delta y_{n-N,k} = -E_{j,k}, \quad j = 1, 2, \dots, N \quad (17.3.7)$$

where

$$S_{j,n} =$$

The quantity $S_{j,n}$ is a block of N equations c

Similarly, the algebraic Taylor series for increments find at the first bound

$$\sum_{n=1}^N$$

where

At the second bound

$$\sum_{n=1}^N$$

where

We thus have in the corrections $\Delta \mathbf{y}$, it is a special structure, because of the typical structure of points, with 3 boundary blocks of nonzero entries. The condition $S_{j,n}$ at points, coupling variables corresponding to the s

We can solve each Gaussian elimination through a number of operations, elements in a special block §2.2, if you are unfamiliar with Gaussian elimination, as dividing rows of coefficients and adding appropriate multiples. We take advantage of the form that we wish to solve a small subset of the reduced system. Once the solution progresses. Once the solution is quickly by a backsubstitution.

Furthermore, the one block of the matrix is reduced to zero (2) elimination of the section contains unity; remaining nonzero coefficients are the steps schematically. Consider the block conditions. We have n

where

$$S_{j,n} = \frac{\partial E_{j,k}}{\partial y_{n,k-1}}, \quad S_{j,n+N} = \frac{\partial E_{j,k}}{\partial y_{n,k}}, \quad n = 1, 2, \dots, N \quad (17.3.8)$$

The quantity $S_{j,n}$ is an $N \times 2N$ matrix at each point k . Each interior point thus supplies a block of N equations coupling $2N$ corrections to the solution variables at the points $k, k-1$.

Similarly, the algebraic relations at the boundaries can be expanded in a first-order Taylor series for increments that improve the solution. Since \mathbf{E}_1 depends only on \mathbf{y}_1 , we find at the first boundary:

$$\sum_{n=1}^N S_{j,n} \Delta y_{n,1} = -E_{j,1}, \quad j = n_2 + 1, n_2 + 2, \dots, N \quad (17.3.9)$$

where

$$S_{j,n} = \frac{\partial E_{j,1}}{\partial y_{n,1}}, \quad n = 1, 2, \dots, N \quad (17.3.10)$$

At the second boundary,

$$\sum_{n=1}^N S_{j,n} \Delta y_{n,M} = -E_{j,M+1}, \quad j = 1, 2, \dots, n_2 \quad (17.3.11)$$

where

$$S_{j,n} = \frac{\partial E_{j,M+1}}{\partial y_{n,M}}, \quad n = 1, 2, \dots, N \quad (17.3.12)$$

We thus have in equations (17.3.7)–(17.3.12) a set of linear equations to be solved for the corrections $\Delta \mathbf{y}$, iterating until the corrections are sufficiently small. The equations have a special structure, because each $S_{j,n}$ couples only points $k, k-1$. Figure 17.3.1 illustrates the typical structure of the complete matrix equation for the case of 5 variables and 4 mesh points, with 3 boundary conditions at the first boundary and 2 at the second. The 3×5 block of nonzero entries in the top left-hand corner of the matrix comes from the boundary condition $S_{j,n}$ at point $k=1$. The next three 5×10 blocks are the $S_{j,n}$ at the interior points, coupling variables at mesh points (2,1), (3,2), and (4,3). Finally we have the block corresponding to the second boundary condition.

We can solve equations (17.3.7)–(17.3.12) for the increments $\Delta \mathbf{y}$ using a form of Gaussian elimination that exploits the special structure of the matrix to minimize the total number of operations, and that minimizes storage of matrix coefficients by packing the elements in a special blocked structure. (You might wish to review Chapter 2, especially §2.2, if you are unfamiliar with the steps involved in Gaussian elimination.) Recall that Gaussian elimination consists of manipulating the equations by elementary operations such as dividing rows of coefficients by a common factor to produce unity in diagonal elements, and adding appropriate multiples of other rows to produce zeros below the diagonal. Here we take advantage of the block structure by performing a bit more reduction than in pure Gaussian elimination, so that the storage of coefficients is minimized. Figure 17.3.2 shows the form that we wish to achieve by elimination, just prior to the backsubstitution step. Only a small subset of the reduced $MN \times MN$ matrix elements needs to be stored as the elimination progresses. Once the matrix elements reach the stage in Figure 17.3.2, the solution follows quickly by a backsubstitution procedure.

Furthermore, the entire procedure, except the backsubstitution step, operates only on one block of the matrix at a time. The procedure contains four types of operations: (1) partial reduction to zero of certain elements of a block using results from a previous step, (2) elimination of the square structure of the remaining block elements such that the square section contains unity along the diagonal, and zero in off-diagonal elements, (3) storage of the remaining nonzero coefficients for use in later steps, and (4) backsubstitution. We illustrate the steps schematically by figures.

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 $-1 \leq x \leq 1$:

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

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 $\lambda_{mn} = n(n+1)$, $n = m, m+1, \dots$. 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 , $n = m, m+1, \dots$ 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 $m \geq 0$ since $m \rightarrow -m$ 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

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

where

Both equations $x \rightarrow -x$. Thus the overall scale factor is also a solution.) only be ± 1 . If $n - m$ Thus we must choose at $x = 0$. Conversely

and similarly for S .

The boundary conditions other words, near the

Substituting this expansion

Equivalently,

A similar equation The irregular solution the endpoints.

Instead of integrating (17.4.6) to integrate

A third boundary condition of a solution y is a normalization that that

$$\lim_{x \rightarrow 1} (1-x^2)$$

Various normalizations

where

$$\mu \equiv \lambda - m(m+1) \quad (17.4.5)$$

Both equations (17.4.1) and (17.4.4) are invariant under the replacement $x \rightarrow -x$. 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

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

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

$$y = a_0 + a_1(1-x^2) + a_2(1-x^2)^2 + \dots \quad (17.4.7)$$

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

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

Equivalently,

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

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

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

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 $x = 1$:

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

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

$$y_1 = y \quad (17.4.12)$$

$$y_2 = y' \quad (17.4.13)$$

$$y_3 = \mu \quad (17.4.14)$$

Then

$$y_1' = y_2 \quad (17.4.15)$$

$$y_2' = \frac{1}{1-x^2} [2x(m+1)y_2 - (y_3 - c^2x^2)y_1] \quad (17.4.16)$$

$$y_3' = 0 \quad (17.4.17)$$

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

$$y_1 = 0, \quad n - m \text{ odd} \quad (17.4.18)$$

$$y_2 = 0, \quad n - m \text{ even}$$

At $x = 1$ we have two conditions:

$$y_2 = \frac{y_3 - c^2}{2(m+1)} y_1 \quad (17.4.19)$$

$$y_1 = \lim_{x \rightarrow 1} (1-x^2)^{-m/2} P_n^m(x) = \frac{(-1)^m (n+m)!}{2^m m! (n-m)!} \equiv \gamma \quad (17.4.20)$$

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 $0 \leq x \leq 1$. For a total of M mesh points, we have

$$h = \frac{1}{M-1} \quad (17.4.21)$$

$$x_k = (k-1)h, \quad k = 1, 2, \dots, M \quad (17.4.22)$$

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

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

Equation (17.4.16) gives

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

where

Finally, equation (17.

Now recall that i labels defined so that i labels 3 for y_j at $k-1$ and:

Similarly equation (1

$$S_{2,1} = \alpha_k \beta_k / 2,$$

$$S_{2,3} = \beta_k (y_{1,k} -$$

$$S_{2,5} = 2 + S_{2,2}:$$

while from equation

At $x = 0$ we h

Recall the convention at $k = 1$ only $S_{3,j}$ boundary condition values of $S_{3,j}$ at x

Equation (17.4.16) gives

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

where

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

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

Finally, equation (17.4.17) gives

$$E_{3,k} = y_{3,k} - y_{3,k-1} \quad (17.4.27)$$

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

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

Similarly equation (17.4.24) yields

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

while from equation (17.4.27) we find

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

At $x = 0$ we have the boundary condition

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

Recall the convention adopted in the `solvrde` 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

$$S_{3,4} = 1, \quad n - m \text{ odd}$$

At $x = 1$ we have

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

$$E_{2,M+1} = y_{1,M} - \gamma \quad (17.4.34)$$

Thus

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

$$S_{2,4} = 1, \quad S_{2,5} = 0, \quad S_{2,6} = 0 \quad (17.4.36)$$

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  $\gamma$  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  $y_1$  and  $y_2$ .
indexv(1)=2
indexv(2)=1
indexv(3)=3
endif
anorm=1. Compute  $\gamma$ .

```

```

if(mm.NE.0)then
q1=n
do 11 i=1,mm
anorm=-.5*ar
q1=q1-1.
enddo 11
endif
do 12 k=1,M-1
x(k)=(k-1)*h
fac1=1.-x(k)**2
fac2=fac1**(-mm/
y(1,k)=plgndr(n,
deriv=-((n-mm+1)
* x(k)*plgndr
y(2,k)=mm*x(k)*y
y(3,k)=n*(n+1)-n
enddo 12
x(M)=1.
y(1,M)=anorm
y(3,M)=n*(n+1)-mm*(n
y(2,M)=(y(3,M)-c2)*j
scalv(1)=abs(anorm)
scalv(2)=max(abs(anc
scalv(3)=max(1.,y(3,
1 continue
write(*,*)'ENTER (
read(*,*)c2
if(c2.eq.999.)stop
call solvde(itmax,cc
* c,NCI,NCJ,NCK,s
write(*,*)'M=',
* 'C**2=',c2,'
goto 1
END

```

```

SUBROUTINE difeq(k,b
INTEGER is1,isf,jsf,
REAL s(nsi,nsj),y(ny
COMMON /sfrcom/ x,h,
PARAMETER (M=41)
Returns matrix s(i
INTEGER mm,n
REAL anorm,c2,h,theta
if(k.eq.k1) then
if(mod(n+mm,2).e
s(3,3+indexv
s(3,3+indexv
s(3,3+indexv
s(3,jsf)=y(1
else
s(3,3+indexv
s(3,3+indexv
s(3,3+indexv
s(3,jsf)=y(2
endif
else if(k.gt.k2) the
s(1,3+indexv(1)):
s(1,3+indexv(2)):
s(1,3+indexv(3)):
s(1,jsf)=y(2,M)-
s(2,3+indexv(1)):
s(2,3+indexv(2)):

```

```

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
  x(k)=(k-1)*h
  fac1=1.-x(k)**2
  fac2=fac1**(-mm/2.)
  y(1,k)=plgndr(n,mm,x(k))*fac2
  deriv=-((n-mm+1)*plgndr(n+1,mm,x(k))-(n+1)*
*      x(k)*plgndr(n,mm,x(k)))/fac1
  y(2,k)=mm*x(k)*y(1,k)/fac1+deriv*fac2
  y(3,k)=n*(n+1)-mm*(mm+1)
enddo 12
x(M)=1.
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
END

```

Initial guess.

P_n^m from §6.8.

Derivative of P_n^m from a recurrence relation.

Initial guess at $x = 1$ done separately.

for another value of c^2 .

```

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
  if(mod(n+mm,2).eq.1)then
    s(3,3+indexv(1))=1.
    s(3,3+indexv(2))=0.
    s(3,3+indexv(3))=0.
    s(3,jsf)=y(1,1)
  else
    s(3,3+indexv(1))=0.
    s(3,3+indexv(2))=1.
    s(3,3+indexv(3))=0.
    s(3,jsf)=y(2,1)
  endif
else if(k.gt.k2) then
  s(1,3+indexv(1))=-(y(3,M)-c2)/(2.*(mm+1.))
  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.))

```

Boundary condition at first point.

Equation (17.4.32).

Equation (17.4.31).

Equation (17.4.32).

Equation (17.4.31).

Boundary conditions at last point.

Equation (17.4.35).

Equation (17.4.33).

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 $k = 1$, $q = 2$ at $k = 2$, 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 ,

$$\frac{dy}{dx} = g \quad (17.5.1)$$

becomes

$$\frac{dy}{dq} = g \frac{dx}{dq} \quad (17.5.2)$$

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

$$y_k - y_{k-1} - \frac{1}{2} \left[\left(g \frac{dx}{dq} \right)_k + \left(g \frac{dx}{dq} \right)_{k-1} \right] = 0 \quad (17.5.3)$$

or some related version. Note that dx/dq should accompany 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 $y(x)$, 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 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

$$\frac{d^2 Q}{dq^2} = 0 \quad (17.5.4)$$

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

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

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

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

where $\phi(x)$ is chosen by

Notice that $\phi(x)$ should be everywhere positive. Or

To use automated reparametrization of a set of equations, i.e., to also become new dependent variables, it will be composed from one to investigate quickly for mesh spacing. (A special case is handled analytically, i.e., dQ/dx those in 17.5.5, and two

As an example of with one dependent var

or

where Δ and δ are constants in x if it alone were present, changing rapidly. The condition is about as "attractive" to the solution according to taste. Other terms in the first term will

CITED REFERENCES
Eggleton, P. P. 1971, *Mathematics of the Earth and Planetary Interiors*, Kippenhan, R., Weigert, M., eds. vol. 7 (New York: Wiley)

17.6 Handling or Sing

Singularities can occur where there is a point x_s at which

where the denominator of the equations usually come from the boundary conditions, such as to make N a finite value. This constraint that $D(x_s, y)$ satisfy a condition, an algebraic condition, an algebraic

We discussed a "method" to handle the singularities. In those problems you

where $\phi(x)$ is chosen by us. Written in terms of the mesh variable q , this equation is

$$\frac{dx}{dq} = \frac{\psi}{\phi(x)} \quad (17.5.7)$$

Notice that $\phi(x)$ should be chosen to be positive definite, so that the density of mesh points is everywhere positive. Otherwise (17.5.7) can have a zero in its denominator.

To use automated mesh spacing, you add the three ODEs (17.5.5) and (17.5.7) to your set of equations, i.e., to the array $y(j,k)$. Now x becomes a dependent variable! Q and ψ also become new dependent variables. Normally, evaluating ϕ requires little extra work since it will be composed from pieces of the g 's that exist anyway. The automated procedure allows one to investigate quickly how the numerical results might be affected by various strategies for mesh spacing. (A special case occurs if the desired mesh spacing function Q can be found analytically, i.e., dQ/dx is directly integrable. Then, you need to add only two equations, those in 17.5.5, and two new variables x, ψ .)

As an example of a typical strategy for implementing this scheme, consider a system with one dependent variable $y(x)$. We could set

$$dQ = \frac{dx}{\Delta} + \frac{|d \ln y|}{\delta} \quad (17.5.8)$$

or

$$\phi(x) = \frac{dQ}{dx} = \frac{1}{\Delta} + \left| \frac{dy/dx}{y\delta} \right| \quad (17.5.9)$$

where Δ and δ are constants that we choose. The first term would give a uniform spacing in x if it alone were present. The second term forces more grid points to be used where y is changing rapidly. The constants act to make every logarithmic change in y of an amount δ about as "attractive" to a grid point as a change in x of amount Δ . You adjust the constants according to taste. Other strategies are possible, such as a logarithmic spacing in x , replacing dx in the first term with $d \ln x$.

CITED REFERENCES AND FURTHER READING:

Eggleton, P. P. 1971, *Monthly Notices of the Royal Astronomical Society*, vol. 151, pp. 351–364.
 Kippenhan, R., Weigert, A., and Hofmeister, E. 1968, in *Methods in Computational Physics*, vol. 7 (New York: Academic Press), pp. 129ff.

17.6 Handling Internal Boundary Conditions or Singular Points

Singularities can occur in the interiors of two point boundary value problems. Typically, there is a point x_s at which a derivative must be evaluated by an expression of the form

$$S(x_s) = \frac{N(x_s, y)}{D(x_s, y)} \quad (17.6.1)$$

where the denominator $D(x_s, y) = 0$. In physical problems with finite answers, singular points usually come with their own cure: Where $D \rightarrow 0$, there the physical solution y must be such as to make $N \rightarrow 0$ simultaneously, in such a way that the ratio takes on a meaningful value. This constraint on the solution y is often called a *regularity condition*. The condition that $D(x_s, y)$ satisfy some special constraint at x_s is entirely analogous to an extra boundary condition, an algebraic relation among the dependent variables that must hold at a point.

We discussed a related situation earlier, in §17.2, when we described the "fitting point

where x_s is the unknown location of the singularity, and change the independent variable to t by setting

$$x - x_1 = tz, \quad 0 \leq t \leq 1 \quad (17.6.4)$$

The boundary conditions at $t = 1$ become

$$N(x, y) = 0, \quad D(x, y) = 0 \quad (17.6.5)$$

Use of an adaptive mesh as discussed in the previous section is another way to overcome the difficulties of an internal singularity. For the problem (17.6.2), we add the mesh spacing equations

$$\frac{dQ}{dq} = \psi \quad (17.6.6)$$

$$\frac{d\psi}{dq} = 0 \quad (17.6.7)$$

with a simple mesh spacing function that maps x uniformly into q , where q runs from 1 to M , the number of mesh points:

$$Q(x) = x - x_1, \quad \frac{dQ}{dx} = 1 \quad (17.6.8)$$

Having added three first-order differential equations, we must also add their corresponding boundary conditions. If there were no singularity, these could simply be

$$\text{at } q = 1 : \quad x = x_1, \quad Q = 0 \quad (17.6.9)$$

$$\text{at } q = M : \quad x = x_2 \quad (17.6.10)$$

and a total of N values y_i specified at $q = 1$. In this case the problem is essentially an initial value problem with all boundary conditions specified at x_1 and the mesh spacing function is superfluous.

However, in the actual case at hand we impose the conditions

$$\text{at } q = 1 : \quad x = x_1, \quad Q = 0 \quad (17.6.11)$$

$$\text{at } q = M : \quad N(x, y) = 0, \quad D(x, y) = 0 \quad (17.6.12)$$

and $N - 1$ values y_i at $q = 1$. The "missing" y_i is to be adjusted, in other words, so as to make the solution go through the singular point in a regular (zero-over-zero) rather than irregular (finite-over-zero) manner. Notice also that these boundary conditions do not directly impose a value for x_2 , which becomes an adjustable parameter that the code varies in an attempt to match the regularity condition.

In this example the singularity occurred at a boundary, and the complication arose because the location of the boundary was unknown. In other problems we might wish to continue the integration beyond the internal singularity. For the example given above, we could simply integrate the ODEs to the singular point, then as a separate problem recommence the integration from the singular point on as far we care to go. However, in other cases the singularity occurs internally, but does not completely determine the problem: There are still some more boundary conditions to be satisfied further along in the mesh. Such cases present no difficulty in principle, but do require some adaptation of the relaxation code given in §17.3. In effect all you need to do is to add a "special" block of equations at the mesh point where the internal boundary conditions occur, and do the proper bookkeeping.

Figure 17.6.1 illustrates a concrete example where the overall problem contains 5 equations with 2 boundary conditions at the first point, one "internal" boundary condition, and two final boundary conditions. The figure shows the structure of the overall matrix equations along the diagonal in the vicinity of the special block. In the middle of the domain, blocks typically involve 5 equations (rows) in 10 unknowns (columns). For each block prior to the special block, the initial boundary conditions provided enough information to zero the first two columns of the blocks. The five EDEs eliminate five more columns. At the

equation: the internal boundary condition. This effectively reduces the required storage of unreduced coefficients by one column for the rest of the grid, and allows us to reduce to zero the first three columns of subsequent blocks. The subroutines `red`, `pinvs`, `bksub` can readily handle these cases with minor recoding, but each problem makes for a special case, and you will have to make the modifications as required.

CITED REFERENCES AND FURTHER READING:

London, R.A., and Flannery, B.P. 1982, *Astrophysical Journal*, vol. 258, pp. 260–269.

Chapter 1

18.0 Introduction

Many people, on the solution of integral equations, it was almost never large and growing literature. Monographs have been published, but one activity is that there are many possible pitfalls; often a single case.

There is a close relationship between linear, integral relations and plain old linear algebra in a finite-dimensional space. In most computational work, integral equations are treated as Fredholm equations.

An inhomogeneous equation.

Here $f(t)$ is the unknown function on the right side." (In integral equations, $K(t, s)$ is conventionally written as the kernel. Equation (18.0.1) is

whose solution is $f(t)$. Equation (18.0.2), equation (18.0.3) is the homogeneous case, and as we shall see, this is the same as the inhomogeneous case.

The analog of (18.0.1) is