Lecture 15

Two point boundary value problems

- Shooting method
- Relaxation methods.
- Adaptive meshes.
- Singularities in the domain.

The shooting method:

Let us assume that at point x_1 one has to specify N boundary values for the ODE, corresponding to the initial values of the y_i's. However, let us assume that these values are constrained by n_1 conditions. There are therefore, n_2=N-n_1 freely specifiable values. Let us think of these values as a vector V with n_2 components.

Choosing a V, we can now integrate the ODE with the routines we know. At x_2 we can evaluate a "discrepancy vector" F, of dimension n_2, which can be simply given by evaluating the n_2 boundary conditions at x_2, let us call them B_{2i}, $F_k = B_{2k}(x_2, \mathbf{y}) \qquad k = 1, \ldots, n_2$

We are now left with the task of finding a vector of values V that zeroes out the vector function F. We do this through multidimensional Newton-Raphson,

$$\mathbf{J} \cdot \delta\mathbf{V} = -\mathbf{F} \qquad \mathbf{V}^{\text{new}} = \mathbf{V}^{\text{old}} + \delta\mathbf{V} \qquad J_{ij} = \frac{\partial F_i}{\partial V_j}$$

And in general, one computes the Jacobian numerically,

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

Each shooting therefore requires n_2+1 operations, one to evaluate the discrepancy and n_2 to evaluate the derivatives.

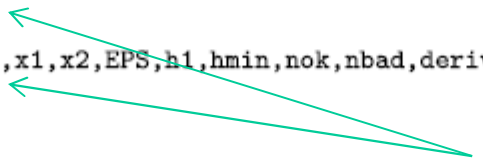
If the equations are linear, then one shooting suffices, since then one Newton-Raphson step produces the solution.

The ideas behind shooting to a midpoint, are essentially the same. One may prefer to do this in cases where the solution is very sensitive to V and produce wildly varying "shots". In such case the ODE routine may crash before reaching x_2.

```
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,xp,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,xp(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
```



User supplied.

Relaxation methods:

In relaxation methods we replace the ordinary differential equation by a finite difference equation (FDE) on a grid of points spanning the interval of interest, say from x_1 to x_2.

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

$$y_k - y_{k-1} - (x_k - x_{k-1})\, g\left[\tfrac{1}{2}(x_k + x_{k-1}), \tfrac{1}{2}(y_k + y_{k-1})\right] = 0$$

This particular discretization connects points at k, k-1, we could construct others that connect points further away from each other.

If we have N equations and M points in the grid, we wish a system of equations relating the NxM variables. We represent the N variables at a point x_k by a vector $\mathbf{y}_k$.

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

The $\mathbf{E}_k$ provide N equations for 2N variables, $\mathbf{y}_k$ and $\mathbf{y}_{k-1}$. Altogether we have MxN variables and (M-1)xN equations.

The remaining N equations come from the boundaries. We denote:

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

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

Where the vector B has only n_1 non-trivial components, whereas C has $N-n_1=n_2$ non trivial components. Arbitrarily, we take such components to be the first ones.

The solution to the problem is given by the N values of y at the M points. We will develop a method that starts with a guess and computes "corrections" to the values guessed Δy_k. The equations for these corrections are obtained expanding the equations in Taylor series,

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

For a solution we would like to demand that E(y+Δy) be zero.

The result is therefore a system of equations that can be cast in matrix form,

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

With $$S_{j,n} = \frac{\partial E_{j,k}}{\partial y_{n,k-1}}, \quad S_{j,n+N} = \frac{\partial E_{j,k}}{\partial y_{n,k}}, \quad n = 1, 2, \ldots, N$$

The matrix $S_{j,n}$ is an Nx2N matrix coupling the 2N variables y_k at points k and k-1.

Similarly, the boundary conditions can be written as,

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

At the first boundary, since E_1 only depends on y_1

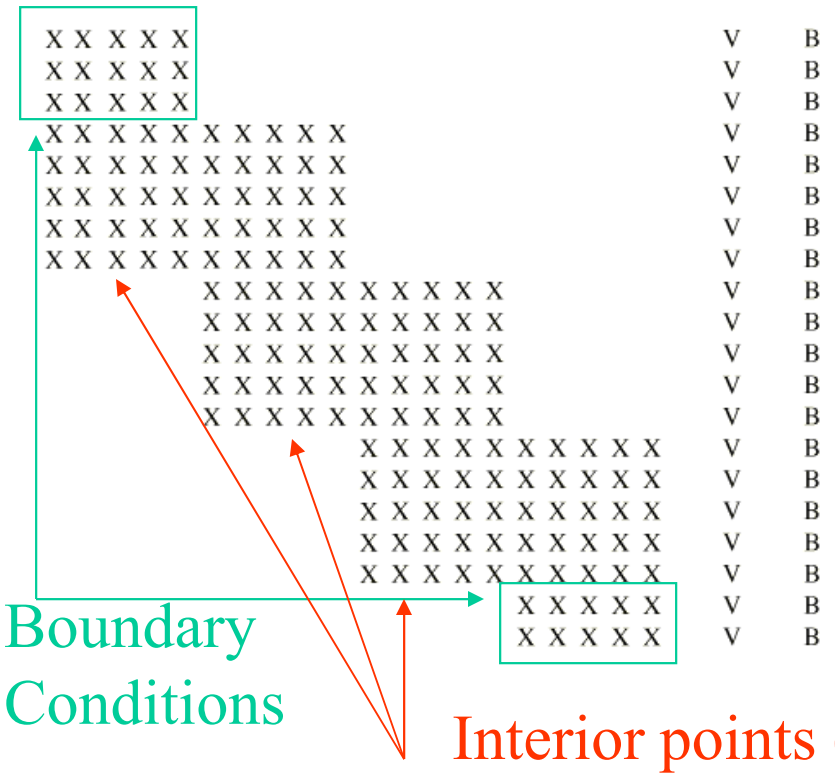
$$S_{j,n} = \frac{\partial E_{j,1}}{\partial y_{n,1}}, \quad n = 1, 2, \ldots, N$$

And at the second boundary,

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

We therefore have a system of linear equations that can determine all the Δy's. We can then proceed to iterate to improve the solution.

$$S_{j,n} = \frac{\partial E_{j,M+1}}{\partial y_{n,M}}, \quad n = 1, 2, \ldots, N$$



Example case of 5 variables and 4 mesh points. The vector V are the 20 unknowns. The matrix has a special form given the fact that S only connects neighboring points.

We could now solve, say, using Gaussian elimination. But perhaps we want to exploit the special structure of the matrix. Numerical recipes contains a detailed discussion of the method.

```
      SUBROUTINE solvde(itmax,conv,slowc,scalv,indexv,ne,nb,m,
     *     y,nyj,nyk,c,nci,ncj,nck,s,nsi,nsj)
      INTEGER itmax,m,nb,nci,ncj,nck,ne,nsi,nsj,
     *     nyj,nyk,indexv(nyj),NMAX
      REAL conv,slowc,c(nci,ncj,nck),s(nsi,nsj),
     *     scalv(nyj),y(nyj,nyk)
      PARAMETER (NMAX=10)          Largest expected value of ne.
C     USES bksub,difeq,pinvs,red
```

Driver routine for solution of two point boundary value problems by relaxation. `itmax` is the maximum number of iterations. `conv` is the convergence criterion (see text). `slowc` controls the fraction of corrections actually used after each iteration. `scalv(1:nyj)` contains typical sizes for each dependent variable, used to weight errors. `indexv(1:nyj)` lists the column ordering of variables used to construct the matrix `s` of derivatives. (The `nb` boundary conditions at the first mesh point must contain some dependence on the first `nb` variables listed in `indexv`.) The problem involves `ne` equations for `ne` adjustable dependent variables at each point. At the first mesh point there are `nb` boundary conditions. There are a total of `m` mesh points. `y(1:nyj,1:nyk)` is the two-dimensional array that contains the initial guess for all the dependent variables at each mesh point. On each iteration, it is updated by the calculated correction. The arrays `c(1:nci,1:ncj,1:nck)`, `s(1:nsi,1:nsj)` supply dummy storage used by the relaxation code; the minimum dimensions must satisfy: `nci=ne`, `ncj=ne-nb+1`, `nck=m+1`, `nsi=ne`, `nsj=2*ne+1`.

```
      INTEGER ic1,ic2,ic3,ic4,it,j,j1,j2,j3,j4,j5,j6,j7,j8,
     *     j9,jc1,jcf,jv,k,k1,k2,km,kp,nvars,kmax(NMAX)
      REAL err,errj,fac,vmax,vz,ermax(NMAX)
      k1=1                         Set up row and column markers.
      k2=m
      nvars=ne*m
      j1=1
      j2=nb
      j3=nb+1
      j4=ne
      j5=j4+j1
```

```
      j5=j4+j1
      j6=j4+j2
      j7=j4+j3
      j8=j4+j4
      j9=j8+j1
      ic1=1
      ic2=ne-nb
      ic3=ic2+1
      ic4=ne
      jc1=1
      jcf=ic3
      do 16 it=1,itmax                  Primary iteration loop.
        k=k1                            Boundary conditions at first point.
        call difeq(k,k1,k2,j9,ic3,ic4,indexv,ne,s,nsi,nsj,y,nyj,nyk)
        call pinvs(ic3,ic4,j5,j9,jc1,k1,c,nci,ncj,nck,s,nsi,nsj)
        do 11 k=k1+1,k2                 Finite difference equations at all point pairs.
          kp=k-1
          call difeq(k,k1,k2,j9,ic1,ic4,indexv,ne,s,nsi,nsj,y,nyj,nyk)
          call red(ic1,ic4,j1,j2,j3,j4,j9,ic3,jc1,jcf,kp,
     *         c,nci,ncj,nck,s,nsi,nsj)
          call pinvs(ic1,ic4,j3,j9,jc1,k,c,nci,ncj,nck,s,nsi,nsj)
        enddo 11
        k=k2+1                          Final boundary conditions.
        call difeq(k,k1,k2,j9,ic1,ic2,indexv,ne,s,nsi,nsj,y,nyj,nyk)
        call red(ic1,ic2,j5,j6,j7,j8,j9,ic3,jc1,jcf,k2,
     *       c,nci,ncj,nck,s,nsi,nsj)
        call pinvs(ic1,ic2,j7,j9,jcf,k2+1,c,nci,ncj,nck,s,nsi,nsj)
```

User supplied
Computes s

Diagonalizes
square portion
of matrix

Reduces
columns
of the matri

```
      call bksub(ne,nb,jcf,k1,k2,c,nci,ncj,nck)        Backsubstitution.
      err=0.
      do 13 j=1,ne                                     Convergence check, accumulate average error.
        jv=indexv(j)
        errj=0.
        km=0
        vmax=0.
        do 12 k=k1,k2                                  Find point with largest error, for each dependent variable.
          vz=abs(c(jv,1,k))
          if(vz.gt.vmax) then
            vmax=vz
            km=k
          endif
          errj=errj+vz
        enddo 12
        err=err+errj/scalv(j)                          Note weighting for each dependent variable.
        ermax(j)=c(jv,1,km)/scalv(j)
        kmax(j)=km
      enddo 13
      err=err/nvars
      fac=slowc/max(slowc,err)                         Reduce correction applied when error is large.
      do 15 j=1,ne                                     Apply corrections.
        jv=indexv(j)
        do 14 k=k1,k2
          y(j,k)=y(j,k)-fac*c(jv,1,k)
        enddo 14
      enddo 15
      write(*,100) it,err,fac                          Summary of corrections for this step. Point with largest
      if(err.lt.conv) return                             error for each variable can be monitored by writ-
    enddo 16                                             ing out kmax and ermax.
    pause 'itmax exceeded in solvde'                   Convergence failed.
100 format(1x,i4,2f12.6)
    return
    END
```

Automated allocation of grid points:

As usual, it is much better if the grid is not uniformly spaced, but that it captures where the "action" is in our solution. If we have an idea ahead of solving the problem, this can be achieved by a rescaling. We can even do better if we adapt the mesh finesse as we go along with the relaxations. To see how this works, let us consider a simple rescaling. Let us consider the variable q, which takes values 1 at k=1, 2 at k=2, etc and $\Delta q=1$ so $\frac{d\mathbf{y}}{dx} = \mathbf{g}$ becomes $\frac{d\mathbf{y}}{dq} = \mathbf{g}\frac{dx}{dq}$

The FDE can therefore be written as,

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

And we would like dq/dx to be large where the function changes rapidly.

The problem is that we do not know the exact proportionality factor. Without it, we cannot give a formula such that q will span from 1 to M when x spans the domain of integration.

The way to fix this is to add an equation for such relation Q(q) to the system of equations and solve the combined system. If we want the spacing to be proportional, we probably seek a linear relation,

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

Which we can write as,

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

To complete the prescription we introduce the function

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

Where $\phi(x)$ we choose.

In terms of this variable the equation reads,

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

And now we regards x as a function of q. The function $\phi(x)$ should be positive definite.

Singularities in the domain of integration:

It could happen that at some point in the domain of integration, the derivative is determined by a function,

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

Where $D(x_s,y)=0$. In most physical problems with finite answers this is compensated by the fact that the solution makes $N(x_s,y)=0$ as well and in such a way that the differential equation is well defined.

If one knows the boundary conditions at x_s, this is no problem, one simply starts the method from there and one can use a suitable analytic approximation for S at that boundary point. The method then proceeds without encountering any singularity.

If the singularity occurs in a point interior to the domain, then the situation is harder.

The way to handle singularities in the interior is akin to the free boundary problems we discussed before. Suppose as an example that we wish to integrate the equation,

$$\frac{dy}{dx} = \frac{N(x,y)}{D(x,y)}$$

Where both N and D are supposed to be zero at some point x_s that we do not know. We supplement the system with the equations,

$$z \equiv x_s - x_1 \qquad \frac{dz}{dx} = 0$$

And change the independent variable to t, defined by,

$$x - x_1 = tz, \qquad 0 \leq t \leq 1$$

The boundary conditions at t=1 therefore become,

$$N(x,y) = 0, \qquad D(x,y) = 0$$

Summary

- Shooting easily implemented.
- Relaxation much more complex. Use canned routines.
- Automated allocation of points can improve things quite a bit.
- Singularities are similar to free boundary problems.