18.2 Volterra Equations

Let us now turn to Volterra equations, of which our prototype is the Volterra equation of the second kind,

$$f(t) = \int_a^t K(t,s)f(s)\,ds + g(t) \tag{18.2.1}$$

Most algorithms for Volterra equations march out from $t = a$, building up the solution as they go. In this sense they resemble not only forward substitution (as discussed in §18.0), but also initial-value problems for ordinary differential equations. In fact, many algorithms for ODEs have counterparts for Volterra equations.

The simplest way to proceed is to solve the equation on a mesh with uniform spacing:

$$t_i = a + ih, \quad i = 0, 1, \ldots, N, \qquad h \equiv \frac{b-a}{N} \tag{18.2.2}$$

To do so, we must choose a quadrature rule. For a uniform mesh, the simplest scheme is the trapezoidal rule, equation (4.1.11):

$$\int_a^{t_i} K(t_i,s)f(s)\,ds = h\left(\tfrac{1}{2}K_{i0}f_0 + \sum_{j=1}^{i-1} K_{ij}f_j + \tfrac{1}{2}K_{ii}f_i\right) \tag{18.2.3}$$

Thus the trapezoidal method for equation (18.2.1) is:

$$\begin{aligned} f_0 &= g_0 \\ (1 - \tfrac{1}{2}hK_{ii})f_i &= h\left(\tfrac{1}{2}K_{i0}f_0 + \sum_{j=1}^{i-1} K_{ij}f_j\right) + g_i, \qquad i = 1, \ldots, N \end{aligned} \tag{18.2.4}$$

(For a Volterra equation of the first kind, the leading 1 on the left would be absent, and g would have opposite sign, with corresponding straightforward changes in the rest of the discussion.)

Equation (18.2.4) is an explicit prescription that gives the solution in $O(N^2)$ operations. Unlike Fredholm equations, it is not necessary to solve a system of linear equations. Volterra equations thus usually involve less work than the corresponding Fredholm equations which, as we have seen, do involve the inversion of, sometimes large, linear systems.

The efficiency of solving Volterra equations is somewhat counterbalanced by the fact that *systems* of these equations occur more frequently in practice. If we interpret equation (18.2.1) as a *vector* equation for the vector of m functions $f(t)$, then the kernel $K(t,s)$ is an $m \times m$ matrix. Equation (18.2.4) must now also be understood as a vector equation. For each i, we have to solve the $m \times m$ set of linear algebraic equations by Gaussian elimination.

The routine `voltra` below implements this algorithm. You must supply an external function that returns the kth function of the vector $g(t)$ at the point t, and another that returns the (k,l) element of the matrix $K(t,s)$ at (t,s). The routine `voltra` then returns the vector $f(t)$ at the regularly spaced points t_i.

```
      SUBROUTINE voltra(n,m,t0,h,t,f,g,ak)
      INTEGER m,n,MMAX
      REAL h,t0,f(m,n),t(n),g,ak
      EXTERNAL ak,g
      PARAMETER (MMAX=5)
C     USES ak,g,lubksb,ludcmp
        Solves a set of m linear Volterra equations of the second kind using the extended trapezoidal
        rule. On input, t0 is the starting point of the integration and n-1 is the number of steps
        of size h to be taken. g(k,t) is a user-supplied external function that returns g_k(t), while
        ak(k,l,t,s) is another user-supplied external function that returns the (k,l) element
        of the matrix K(t,s). The solution is returned in f(1:m,1:n), with the corresponding
        abscissas in t(1:n).
      INTEGER i,j,k,l,indx(MMAX)
      REAL d,sum,a(MMAX,MMAX),b(MMAX)
      t(1)=t0
      do 11 k=1,m                                 Initialize.
        f(k,1)=g(k,t(1))
      enddo 11
      do 16 i=2,n                                 Take a step h.
        t(i)=t(i-1)+h
        do 14 k=1,m
          sum=g(k,t(i))                           Accumulate right-hand side of linear equations in
          do 13 l=1,m                                 sum.
            sum=sum+0.5*h*ak(k,l,t(i),t(1))*f(l,1)
            do 12 j=2,i-1
              sum=sum+h*ak(k,l,t(i),t(j))*f(l,j)
            enddo 12
            if(k.eq.l)then                        Left-hand side goes in matrix a.
              a(k,l)=1.
            else
              a(k,l)=0.
            endif
            a(k,l)=a(k,l)-0.5*h*ak(k,l,t(i),t(i))
          enddo 13
          b(k)=sum
        enddo 14
        call ludcmp(a,m,MMAX,indx,d)              Solve linear equations.
        call lubksb(a,m,MMAX,indx,b)
        do 15 k=1,m
          f(k,i)=b(k)
        enddo 15
      enddo 16
      return
      END
```

For nonlinear Volterra equations, equation (18.2.4) holds with the product $K_{ii}f_i$ replaced by $K_{ii}(f_i)$, and similarly for the other two products of K's and f's. Thus for each i we solve a nonlinear equation for f_i with a known right-hand side. Newton's method (§9.4 or §9.6) with an initial guess of f_{i-1} usually works very well provided the stepsize is not too big.

Higher-order methods for solving Volterra equations are, in our opinion, not as important as for Fredholm equations, since Volterra equations are relatively easy to solve. However, there is an extensive literature on the subject. Several difficulties arise. First, any method that achieves higher order by operating on several quadrature points simultaneously will need a special method to get started, when values at the first few points are not yet known.

Second, stable quadrature rules can give rise to unexpected instabilities in integral equations. For example, suppose we try to replace the trapezoidal rule in

the algorithm above with Simpson's rule. Simpson's rule naturally integrates over an interval $2h$, so we easily get the function values at the even mesh points. For the odd mesh points, we could try appending one panel of trapezoidal rule. But to which end of the integration should we append it? We could do one step of trapezoidal rule followed by all Simpson's rule, or Simpson's rule with one step of trapezoidal rule at the end. Surprisingly, the former scheme is unstable, while the latter is fine!

A simple approach that can be used with the trapezoidal method given above is Richardson extrapolation: Compute the solution with stepsize h and $h/2$. Then, assuming the error scales with h^2, compute

$$f_{\rm E} = \frac{4f(h/2) - f(h)}{3} \tag{18.2.5}$$

This procedure can be repeated as with Romberg integration.

The general consensus is that the best of the higher order methods is the *block-by-block method* (see [1]). Another important topic is the use of variable stepsize methods, which are much more efficient if there are sharp features in K or f. Variable stepsize methods are quite a bit more complicated than their counterparts for differential equations; we refer you to the literature [1,2] for a discussion.

You should also be on the lookout for singularities in the integrand. If you find them, then look to §18.3 for additional ideas.

CITED REFERENCES AND FURTHER READING:

Linz, P. 1985, *Analytical and Numerical Methods for Volterra Equations* (Philadelphia: S.I.A.M.). [1]

Delves, L.M., and Mohamed, J.L. 1985, *Computational Methods for Integral Equations* (Cambridge, U.K.: Cambridge University Press). [2]

18.3 Integral Equations with Singular Kernels

Many integral equations have singularities in either the kernel or the solution or both. A simple quadrature method will show poor convergence with N if such singularities are ignored. There is sometimes art in how singularities are best handled.

We start with a few straightforward suggestions:

1. Integrable singularities can often be removed by a change of variable. For example, the singular behavior $K(t,s) \sim s^{1/2}$ or $s^{-1/2}$ near $s = 0$ can be removed by the transformation $z = s^{1/2}$. Note that we are assuming that the singular behavior is confined to K, whereas the quadrature actually involves the product $K(t,s)f(s)$, and it is this product that must be "fixed." Ideally, you must deduce the singular nature of the product before you try a numerical solution, and take the appropriate action. Commonly, however, a singular kernel does *not* produce a singular solution $f(t)$. (The highly singular kernel $K(t,s) = \delta(t-s)$ is simply the identity operator, for example.)

2. If $K(t,s)$ can be factored as $w(s)\overline{K}(t,s)$, where $w(s)$ is singular and $\overline{K}(t,s)$ is smooth, then a Gaussian quadrature based on $w(s)$ as a weight function will work well. Even if the factorization is only approximate, the convergence is often improved dramatically. All you have to do is replace `gauleg` in the routine `fred2` by another quadrature routine. Section 4.5 explained how to construct such quadratures; or you can find tabulated abscissas and weights in the standard references [1,2]. You must of course supply $\overline{K}$ instead of K.