special quadrature rules, but they are also sometimes blessings in disguise, since they can spoil a kernel's smoothing and make problems well-conditioned.

In §§18.4–18.7 we face up to the issues of inverse problems. §18.4 is an introduction to this large subject.

We should note here that wavelet transforms, already discussed in §13.10, are applicable not only to data compression and signal processing, but can also be used to transform some classes of integral equations into sparse linear problems that allow fast solution. You may wish to review §13.10 as part of reading this chapter.

Some subjects, such as *integro-differential equations*, we must simply declare to be beyond our scope. For a review of methods for integro-differential equations, see Brunner [4].

It should go without saying that this one short chapter can only barely touch on a few of the most basic methods involved in this complicated subject.

CITED REFERENCES AND FURTHER READING:

Delves, L.M., and Mohamed, J.L. 1985, *Computational Methods for Integral Equations* (Cambridge, U.K.: Cambridge University Press). [1]

Linz, P. 1985, *Analytical and Numerical Methods for Volterra Equations* (Philadelphia: S.I.A.M.). [2]

Atkinson, K.E. 1976, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind* (Philadelphia: S.I.A.M.). [3]

Brunner, H. 1988, in *Numerical Analysis 1987*, Pitman Research Notes in Mathematics vol. 170, D.F. Griffiths and G.A. Watson, eds. (Harlow, Essex, U.K.: Longman Scientific and Technical), pp. 18–38. [4]

Smithies, F. 1958, *Integral Equations* (Cambridge, U.K.: Cambridge University Press).

Kanwal, R.P. 1971, *Linear Integral Equations* (New York: Academic Press).

Green, C.D. 1969, *Integral Equation Methods* (New York: Barnes & Noble).

18.1 Fredholm Equations of the Second Kind

We desire a numerical solution for $f(t)$ in the equation

$$f(t) = \lambda \int_a^b K(t,s) f(s)\, ds + g(t) \tag{18.1.1}$$

The method we describe, a very basic one, is called the *Nystrom method*. It requires the choice of some approximate *quadrature rule*:

$$\int_a^b y(s)\, ds = \sum_{j=1}^{N} w_j y(s_j) \tag{18.1.2}$$

Here the set $\{w_j\}$ are the weights of the quadrature rule, while the N points $\{s_j\}$ are the abscissas.

What quadrature rule should we use? It is certainly possible to solve integral equations with low-order quadrature rules like the repeated trapezoidal or Simpson's rules. We will see, however, that the solution method involves $O(N^3)$ operations, and so the most efficient methods tend to use high-order quadrature rules to keep N as small as possible. For smooth, nonsingular problems, nothing beats Gaussian quadrature (e.g., Gauss-Legendre quadrature, §4.5). (For non-smooth or singular kernels, see §18.3.)

Delves and Mohamed [1] investigated methods more complicated than the Nystrom method. For straightforward Fredholm equations of the second kind, they concluded "... the clear winner of this contest has been the Nystrom routine ... with the N-point Gauss-Legendre rule. This routine is extremely simple.... Such results are enough to make a numerical analyst weep."

If we apply the quadrature rule (18.1.2) to equation (18.1.1), we get

$$f(t) = \lambda \sum_{j=1}^{N} w_j K(t, s_j) f(s_j) + g(t) \tag{18.1.3}$$

Evaluate equation (18.1.3) at the quadrature points:

$$f(t_i) = \lambda \sum_{j=1}^{N} w_j K(t_i, s_j) f(s_j) + g(t_i) \tag{18.1.4}$$

Let f_i be the vector $f(t_i)$, g_i the vector $g(t_i)$, K_{ij} the matrix $K(t_i, s_j)$, and define

$$\widetilde{K}_{ij} = K_{ij} w_j \tag{18.1.5}$$

Then in matrix notation equation (18.1.4) becomes

$$(\mathbf{1} - \lambda \widetilde{\mathbf{K}}) \cdot \mathbf{f} = \mathbf{g} \tag{18.1.6}$$

This is a set of N linear algebraic equations in N unknowns that can be solved by standard triangular decomposition techniques (§2.3) — that is where the $O(N^3)$ operations count comes in. The solution is usually well-conditioned, unless λ is very close to an eigenvalue.

Having obtained the solution at the quadrature points $\{t_i\}$, how do you get the solution at some other point t? You do *not* simply use polynomial interpolation. This destroys all the accuracy you have worked so hard to achieve. Nystrom's key observation was that you should use equation (18.1.3) as an interpolatory formula, maintaining the accuracy of the solution.

We here give two routines for use with linear Fredholm equations of the second kind. The routine `fred2` sets up equation (18.1.6) and then solves it by LU decomposition with calls to the routines `ludcmp` and `lubksb`. The Gauss-Legendre quadrature is implemented by first getting the weights and abscissas with a call to `gauleg`. Routine `fred2` requires that you provide an external function that returns $g(t)$ and another that returns λK_{ij}. It then returns the solution f at the quadrature points. It also returns the quadrature points and weights. These are used by the second routine `fredin` to carry out the Nystrom interpolation of equation (18.1.3) and return the value of f at any point in the interval $[a, b]$.

```
#include "nrutil.h"

void fred2(int n, float a, float b, float t[], float f[], float w[],
    float (*g)(float), float (*ak)(float, float))
```
Solves a linear Fredholm equation of the second kind. On input, a and b are the limits of integration, and n is the number of points to use in the Gaussian quadrature. g and ak are user-supplied external functions that respectively return $g(t)$ and $\lambda K(t, s)$. The routine returns arrays t[1..n] and f[1..n] containing the abscissas t_i of the Gaussian quadrature and the solution f at these abscissas. Also returned is the array w[1..n] of Gaussian weights for use with the Nystrom interpolation routine fredin.
```
{
    void gauleg(float x1, float x2, float x[], float w[], int n);
    void lubksb(float **a, int n, int *indx, float b[]);
    void ludcmp(float **a, int n, int *indx, float *d);
    int i,j,*indx;
    float d,**omk;

    indx=ivector(1,n);
    omk=matrix(1,n,1,n);
    gauleg(a,b,t,w,n);                Replace gauleg with another routine if not using
    for (i=1;i<=n;i++) {                  Gauss-Legendre quadrature.
        for (j=1;j<=n;j++)            Form 1 − λK̃.
            omk[i][j]=(float)(i == j)-(*ak)(t[i],t[j])*w[j];
        f[i]=(*g)(t[i]);
    }
    ludcmp(omk,n,indx,&d);            Solve linear equations.
    lubksb(omk,n,indx,f);
    free_matrix(omk,1,n,1,n);
    free_ivector(indx,1,n);
}
```

```
float fredin(float x, int n, float a, float b, float t[], float f[],
    float w[], float (*g)(float), float (*ak)(float, float))
```
Given arrays t[1..n] and w[1..n] containing the abscissas and weights of the Gaussian quadrature, and given the solution array f[1..n] from fred2, this function returns the value of f at x using the Nystrom interpolation formula. On input, a and b are the limits of integration, and n is the number of points used in the Gaussian quadrature. g and ak are user-supplied external functions that respectively return $g(t)$ and $\lambda K(t, s)$.
```
{
    int i;
    float sum=0.0;

    for (i=1;i<=n;i++) sum += (*ak)(x,t[i])*w[i]*f[i];
    return (*g)(x)+sum;
}
```

One disadvantage of a method based on Gaussian quadrature is that there is no simple way to obtain an estimate of the error in the result. The best practical method is to increase N by 50%, say, and treat the difference between the two estimates as a conservative estimate of the error in the result obtained with the larger value of N.

Turn now to solutions of the homogeneous equation. If we set $\lambda = 1/\sigma$ and $\mathbf{g} = 0$, then equation (18.1.6) becomes a standard eigenvalue equation

$$\widetilde{\mathbf{K}} \cdot \mathbf{f} = \sigma \mathbf{f} \tag{18.1.7}$$

which we can solve with any convenient matrix eigenvalue routine (see Chapter 11). Note that if our original problem had a symmetric kernel, then the matrix $\mathbf{K}$

is symmetric. However, since the weights w_j are not equal for most quadrature rules, the matrix $\widetilde{\mathbf{K}}$ (equation 18.1.5) is not symmetric. The matrix eigenvalue problem is much easier for symmetric matrices, and so we should restore the symmetry if possible. Provided the weights are positive (which they are for Gaussian quadrature), we can define the diagonal matrix $\mathbf{D} = \mathrm{diag}(w_j)$ and its square root, $\mathbf{D}^{1/2} = \mathrm{diag}(\sqrt{w_j})$. Then equation (18.1.7) becomes

$$\mathbf{K} \cdot \mathbf{D} \cdot \mathbf{f} = \sigma \mathbf{f}$$

Multiplying by $\mathbf{D}^{1/2}$, we get

$$\left(\mathbf{D}^{1/2} \cdot \mathbf{K} \cdot \mathbf{D}^{1/2}\right) \cdot \mathbf{h} = \sigma \mathbf{h} \tag{18.1.8}$$

where $\mathbf{h} = \mathbf{D}^{1/2} \cdot \mathbf{f}$. Equation (18.1.8) is now in the form of a symmetric eigenvalue problem.

Solution of equations (18.1.7) or (18.1.8) will in general give N eigenvalues, where N is the number of quadrature points used. For square-integrable kernels, these will provide good approximations to the lowest N eigenvalues of the integral equation. Kernels of *finite rank* (also called *degenerate* or *separable* kernels) have only a finite number of nonzero eigenvalues (possibly none). You can diagnose this situation by a cluster of eigenvalues σ that are zero to machine precision. The number of nonzero eigenvalues will stay constant as you increase N to improve their accuracy. Some care is required here: A nondegenerate kernel can have an infinite number of eigenvalues that have an accumulation point at $\sigma = 0$. You distinguish the two cases by the behavior of the solution as you increase N. If you suspect a degenerate kernel, you will usually be able to solve the problem by analytic techniques described in all the textbooks.

CITED REFERENCES AND FURTHER READING:

Delves, L.M., and Mohamed, J.L. 1985, *Computational Methods for Integral Equations* (Cambridge, U.K.: Cambridge University Press). [1]

Atkinson, K.E. 1976, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind* (Philadelphia: S.I.A.M.).

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