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.

This method is a special case of the *product Nystrom method* [3,4], where one factors out a singular term $p(t,s)$ depending on both t and s from K and constructs suitable weights for its Gaussian quadrature. The calculations in the general case are quite cumbersome, because the weights depend on the chosen $\{t_i\}$ as well as the form of $p(t,s)$.

We prefer to implement the product Nystrom method on a uniform grid, with a quadrature scheme that generalizes the extended Simpson's 3/8 rule (equation 4.1.5) to arbitrary weight functions. We discuss this in the subsections below.

3. Special quadrature formulas are also useful when the kernel is not strictly singular, but is "almost" so. One example is when the kernel is concentrated near $t = s$ on a scale much smaller than the scale on which the solution $f(t)$ varies. In that case, a quadrature formula can be based on locally approximating $f(s)$ by a polynomial or spline, while calculating the first few *moments* of the kernel $K(t,s)$ at the tabulation points t_i. In such a scheme the narrow width of the kernel becomes an asset, rather than a liability: The quadrature becomes exact as the width of the kernel goes to zero.

4. An infinite range of integration is also a form of singularity. Truncating the range at a large finite value should be used only as a last resort. If the kernel goes rapidly to zero, then

a Gauss-Laguerre [$w \sim \exp(-\alpha s)$] or Gauss-Hermite [$w \sim \exp(-s^2)$] quadrature should work well. Long-tailed functions often succumb to the transformation

$$s = \frac{2\alpha}{z+1} - \alpha \tag{18.3.1}$$

which maps $0 < s < \infty$ to $1 > z > -1$ so that Gauss-Legendre integration can be used. Here $\alpha > 0$ is a constant that you adjust to improve the convergence.

5. A common situation in practice is that $K(t, s)$ is singular along the diagonal line $t = s$. Here the Nystrom method fails completely because the kernel gets evaluated at (t_i, s_i). *Subtraction of the singularity* is one possible cure:

$$\begin{aligned}\int_a^b K(t,s)f(s)\,ds &= \int_a^b K(t,s)[f(s)-f(t)]\,ds + \int_a^b K(t,s)f(t)\,ds \\ &= \int_a^b K(t,s)[f(s)-f(t)]\,ds + r(t)f(t)\end{aligned} \tag{18.3.2}$$

where $r(t) = \int_a^b K(t,s)\,ds$ is computed analytically or numerically. If the first term on the right-hand side is now regular, we can use the Nystrom method. Instead of equation (18.1.4), we get

$$f_i = \lambda \sum_{\substack{j=1 \\ j\neq i}}^{N} w_j K_{ij}[f_j - f_i] + \lambda r_i f_i + g_i \tag{18.3.3}$$

Sometimes the subtraction process must be repeated before the kernel is completely regularized. See [3] for details. (And read on for a different, we think better, way to handle diagonal singularities.)

Quadrature on a Uniform Mesh with Arbitrary Weight

It is possible in general to find n-point linear quadrature rules that approximate the integral of a function $f(x)$, times an arbitrary weight function $w(x)$, over an arbitrary range of integration (a, b), as the sum of weights times n evenly spaced values of the function $f(x)$, say at $x = kh, (k+1)h, \ldots, (k+n-1)h$. The general scheme for deriving such quadrature rules is to write down the n linear equations that must be satisfied if the quadrature rule is to be exact for the n functions $f(x) = \text{const}, x, x^2, \ldots, x^{n-1}$, and then solve these for the coefficients. This can be done analytically, once and for all, if the moments of the weight function over the same range of integration,

$$W_n \equiv \frac{1}{h^n}\int_a^b x^n w(x)dx \tag{18.3.4}$$

are assumed to be known. Here the prefactor h^{-n} is chosen to make W_n scale as h if (as in the usual case) $b - a$ is proportional to h.

Carrying out this prescription for the four-point case gives the result

$$\begin{aligned}&\int_a^b w(x)f(x)dx = \\ &\quad \frac{1}{6}f(kh)\Big[(k+1)(k+2)(k+3)W_0 - (3k^2+12k+11)W_1 + 3(k+2)W_2 - W_3\Big] \\ &\quad + \frac{1}{2}f([k+1]h)\Big[-k(k+2)(k+3)W_0 + (3k^2+10k+6)W_1 - (3k+5)W_2 + W_3\Big] \\ &\quad + \frac{1}{2}f([k+2]h)\Big[k(k+1)(k+3)W_0 - (3k^2+8k+3)W_1 + (3k+4)W_2 - W_3\Big] \\ &\quad + \frac{1}{6}f([k+3]h)\Big[-k(k+1)(k+2)W_0 + (3k^2+6k+2)W_1 - 3(k+1)W_2 + W_3\Big]\end{aligned} \tag{18.3.5}$$

While the terms in brackets superficially appear to scale as k^2, there is typically cancellation at both $O(k^2)$ and $O(k)$.

Equation (18.3.5) can be specialized to various choices of (a, b). The obvious choice is $a = kh$, $b = (k + 3)h$, in which case we get a four-point quadrature rule that generalizes Simpson's 3/8 rule (equation 4.1.5). In fact, we can recover this special case by setting $w(x) = 1$, in which case (18.3.4) becomes

$$W_n = \frac{h}{n+1}[(k+3)^{n+1} - k^{n+1}] \tag{18.3.6}$$

The four terms in square brackets equation (18.3.5) each become independent of k, and (18.3.5) in fact reduces to

$$\int_{kh}^{(k+3)h} f(x)dx = \frac{3h}{8}f(kh)+\frac{9h}{8}f([k+1]h)+\frac{9h}{8}f([k+2]h)+\frac{3h}{8}f([k+3]h) \tag{18.3.7}$$

Back to the case of general $w(x)$, some other choices for a and b are also useful. For example, we may want to choose (a, b) to be $([k + 1]h, [k + 3]h)$ or $([k + 2]h, [k + 3]h)$, allowing us to finish off an extended rule whose number of intervals is not a multiple of three, without loss of accuracy: The integral will be estimated using the four values $f(kh), \dots, f([k+3]h)$. Even more useful is to choose (a, b) to be $([k+1]h, [k+2]h)$, thus using four points to integrate a centered single interval. These weights, when sewed together into an extended formula, give quadrature schemes that have smooth coefficients, i.e., without the Simpson-like $2, 4, 2, 4, 2$ alternation. (In fact, this was the technique that we used to derive equation 4.1.14, which you may now wish to reexamine.)

All these rules are of the same order as the extended Simpson's rule, that is, exact for $f(x)$ a cubic polynomial. Rules of lower order, if desired, are similarly obtained. The three point formula is

$$\begin{aligned}\int_a^b w(x)f(x)dx = {} & \frac{1}{2}f(kh)\Big[(k+1)(k+2)W_0 - (2k+3)W_1 + W_2\Big] \\ & + f([k+1]h)\Big[-k(k+2)W_0 + 2(k+1)W_1 - W_2\Big] \\ & + \frac{1}{2}f([k+2]h)\Big[k(k+1)W_0 - (2k+1)W_1 + W_2\Big]\end{aligned} \tag{18.3.8}$$

Here the simple special case is to take, $w(x) = 1$, so that

$$W_n = \frac{h}{n+1}[(k+2)^{n+1} - k^{n+1}] \tag{18.3.9}$$

Then equation (18.3.8) becomes Simpson's rule,

$$\int_{kh}^{(k+2)h} f(x)dx = \frac{h}{3}f(kh) + \frac{4h}{3}f([k+1]h) + \frac{h}{3}f([k+2]h) \tag{18.3.10}$$

For nonconstant weight functions $w(x)$, however, equation (18.3.8) gives rules of one order less than Simpson, since they do not benefit from the extra symmetry of the constant case.

The two point formula is simply

$$\int_{kh}^{(k+1)h} w(x)f(x)dx = f(kh)[(k+1)W_0 - W_1] + f([k+1]h)[-kW_0 + W_1] \tag{18.3.11}$$

Here is a routine `wwghts` that uses the above formulas to return an extended N-point quadrature rule for the interval $(a, b) = (0, [N - 1]h)$. Input to `wwghts` is a user-supplied routine, `kermom`, that is called to get the first four *indefinite-integral* moments of $w(x)$, namely

$$F_m(y) \equiv \int^y s^m w(s)ds \qquad m = 0, 1, 2, 3 \tag{18.3.12}$$

(The lower limit is arbitrary and can be chosen for convenience.) Cautionary note: When called with $N < 4$, `wwghts` returns a rule of lower order than Simpson; you should structure your problem to avoid this.

```
void wwghts(float wghts[], int n, float h,
    void (*kermom)(double [], double ,int))
```
Constructs in `wghts[1..n]` weights for the `n`-point equal-interval quadrature from 0 to $(\mathtt{n}-1)\mathtt{h}$ of a function $f(x)$ times an arbitrary (possibly singular) weight function $w(x)$ whose indefinite-integral moments $F_n(y)$ are provided by the user-supplied routine `kermom`.
```
{
    int j,k;
    double wold[5],wnew[5],w[5],hh,hi,c,fac,a,b;
```
Double precision on internal calculations even though the interface is in single precision.
```
    hh=h;
    hi=1.0/hh;
    for (j=1;j<=n;j++) wghts[j]=0.0;
```
Zero all the weights so we can sum into them.
```
    (*kermom)(wold,0.0,4);                  Evaluate indefinite integrals at lower end.
    if (n >= 4) {                           Use highest available order.
        b=0.0;                              For another problem, you might change
        for (j=1;j<=n-3;j++) {                  this lower limit.
            c=j-1;                          This is called k in equation (18.3.5).
            a=b;                            Set upper and lower limits for this step.
            b=a+hh;
            if (j == n-3) b=(n-1)*hh;       Last interval: go all the way to end.
            (*kermom)(wnew,b,4);
            for (fac=1.0,k=1;k<=4;k++,fac*=hi)      Equation (18.3.4).
                w[k]=(wnew[k]-wold[k])*fac;
            wghts[j] += (                           Equation (18.3.5).
                ((c+1.0)*(c+2.0)*(c+3.0)*w[1]
                -(11.0+c*(12.0+c*3.0))*w[2]
                +3.0*(c+2.0)*w[3]-w[4])/6.0);
            wghts[j+1] += (
                (-c*(c+2.0)*(c+3.0)*w[1]
                +(6.0+c*(10.0+c*3.0))*w[2]
                -(3.0*c+5.0)*w[3]+w[4])*0.5);
            wghts[j+2] += (
                (c*(c+1.0)*(c+3.0)*w[1]
                -(3.0+c*(8.0+c*3.0))*w[2]
                +(3.0*c+4.0)*w[3]-w[4])*0.5);
            wghts[j+3] += (
                (-c*(c+1.0)*(c+2.0)*w[1]
                +(2.0+c*(6.0+c*3.0))*w[2]
                -3.0*(c+1.0)*w[3]+w[4])/6.0);
            for (k=1;k<=4;k++) wold[k]=wnew[k];     Reset lower limits for moments.
        }
    } else if (n == 3) {                    Lower-order cases; not recommended.
        (*kermom)(wnew,hh+hh,3);
        w[1]=wnew[1]-wold[1];
        w[2]=hi*(wnew[2]-wold[2]);
        w[3]=hi*hi*(wnew[3]-wold[3]);
        wghts[1]=w[1]-1.5*w[2]+0.5*w[3];
        wghts[2]=2.0*w[2]-w[3];
        wghts[3]=0.5*(w[3]-w[2]);
    } else if (n == 2) {
        (*kermom)(wnew,hh,2);
        wghts[1]=wnew[1]-wold[1]-(wghts[2]=hi*(wnew[2]-wold[2]));
    }
}
```

We will now give an example of how to apply `wwghts` to a singular integral equation.

Worked Example: A Diagonally Singular Kernel

As a particular example, consider the integral equation

$$f(x) + \int_0^\pi K(x,y)f(y)dy = \sin x \tag{18.3.13}$$

with the (arbitrarily chosen) nasty kernel

$$K(x,y) = \cos x \cos y \times \begin{cases} \ln(x-y) & y < x \\ \sqrt{y-x} & y \ge x \end{cases} \tag{18.3.14}$$

which has a logarithmic singularity on the left of the diagonal, combined with a square-root discontinuity on the right.

The first step is to do (analytically, in this case) the required moment integrals over the singular part of the kernel, equation (18.3.12). Since these integrals are done at a fixed value of x, we can use x as the lower limit. For any specified value of y, the required indefinite integral is then either

$$F_m(y;x) = \int_x^y s^m(s-x)^{1/2}ds = \int_0^{y-x}(x+t)^m t^{1/2}dt \qquad \text{if } y > x \tag{18.3.15}$$

or

$$F_m(y;x) = \int_x^y s^m \ln(x-s)ds = \int_0^{x-y}(x-t)^m \ln t\, dt \qquad \text{if } y < x \tag{18.3.16}$$

(where a change of variable has been made in the second equality in each case). Doing these integrals analytically (actually, we used a symbolic integration package!), we package the resulting formulas in the following routine. Note that w$(j+1)$ returns $F_j(y;x)$.

```
#include <math.h>

extern double x;                    Defined in quadmx.

void kermom(double w[], double y, int m)
Returns in w[1..m] the first m indefinite-integral moments of one row of the singular part of
the kernel. (For this example, m is hard-wired to be 4.) The input variable y labels the column,
while the global variable x is the row. We can take x as the lower limit of integration. Thus,
we return the moment integrals either purely to the left or purely to the right of the diagonal.
{
    double d,df,clog,x2,x3,x4,y2;

    if (y >= x) {
        d=y-x;
        df=2.0*sqrt(d)*d;
        w[1]=df/3.0;
        w[2]=df*(x/3.0+d/5.0);
        w[3]=df*((x/3.0 + 0.4*d)*x + d*d/7.0);
        w[4]=df*(((x/3.0 + 0.6*d)*x + 3.0*d*d/7.0)*x+d*d*d/9.0);
    } else {
        x3=(x2=x*x)*x;
        x4=x2*x2;
        y2=y*y;
        d=x-y;
        w[1]=d*((clog=log(d))-1.0);
        w[2] = -0.25*(3.0*x+y-2.0*clog*(x+y))*d;
        w[3]=(-11.0*x3+y*(6.0*x2+y*(3.0*x+2.0*y))
            +6.0*clog*(x3-y*y2))/18.0;
        w[4]=(-25.0*x4+y*(12.0*x3+y*(6.0*x2+y*
            (4.0*x+3.0*y)))+12.0*clog*(x4-(y2*y2)))/48.0;
    }
}
```

Next, we write a routine that constructs the quadrature matrix.

```
#include <math.h>
#include "nrutil.h"
#define PI 3.14159265

double x;                           Communicates with kermom.

void quadmx(float **a, int n)
```
Constructs in a[1..n][1..n] the quadrature matrix for an example Fredholm equation of the second kind. The nonsingular part of the kernel is computed within this routine, while the quadrature weights which integrate the singular part of the kernel are obtained via calls to wwghts. An external routine kermom, which supplies indefinite-integral moments of the singular part of the kernel, is passed to wwghts.
```
{
    void kermom(double w[], double y, int m);
    void wwghts(float wghts[], int n, float h,
        void (*kermom)(double [], double ,int));
    int j,k;
    float h,*wt,xx,cx;

    wt=vector(1,n);
    h=PI/(n-1);
    for (j=1;j<=n;j++) {
        x=xx=(j-1)*h;               Put x in global variable for use by kermom.
        wwghts(wt,n,h,kermom);
        cx=cos(xx);                 Part of nonsingular kernel.
        for (k=1;k<=n;k++) a[j][k]=wt[k]*cx*cos((k-1)*h);
        Put together all the pieces of the kernel.
        ++a[j][j];                  Since equation of the second kind, there is diagonal piece
    }                                   independent of h.
    free_vector(wt,1,n);
}
```

Finally, we solve the linear system for any particular right-hand side, here $\sin x$.

```
#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#define PI 3.14159265
#define N 40                        Here the size of the grid is specified.

int main(void) /* Program fredex */
```
This sample program shows how to solve a Fredholm equation of the second kind using the product Nystrom method and a quadrature rule especially constructed for a particular, singular, kernel.
```
{
    void lubksb(float **a, int n, int *indx, float b[]);
    void ludcmp(float **a, int n, int *indx, float *d);
    void quadmx(float **a, int n);
    float **a,d,*g,x;
    int *indx,j;

    indx=ivector(1,N);
    a=matrix(1,N,1,N);
    g=vector(1,N);
    quadmx(a,N);                    Make the quadrature matrix; all the action is here.
    ludcmp(a,N,indx,&d);            Decompose the matrix.
    for (j=1;j<=N;j++) g[j]=sin((j-1)*PI/(N-1));
    Construct the right hand side, here sin x.
    lubksb(a,N,indx,g);             Backsubstitute.
    for (j=1;j<=N;j++) {            Write out the solution.
        x=(j-1)*PI/(N-1);
```

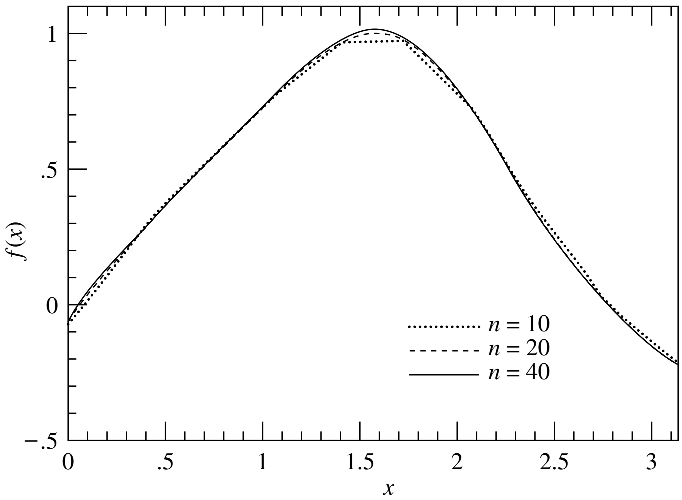

Figure 18.3.1. Solution of the example integral equation (18.3.14) with grid sizes $N = 10, 20$, and 40. The tabulated solution values have been connected by straight lines; in practice one would interpolate a small N solution more smoothly.

```
        printf("%6.2d %12.6f %12.6f\n",j,x,g[j]);
    }
    free_vector(g,1,N);
    free_matrix(a,1,N,1,N);
    free_ivector(indx,1,N);
    return 0;
}
```

With $N = 40$, this program gives accuracy at about the 10^{-5} level. The accuracy increases as N^4 (as it should for our Simpson-order quadrature scheme) *despite* the highly singular kernel. Figure 18.3.1 shows the solution obtained, also plotting the solution for smaller values of N, which are themselves seen to be remarkably faithful. Notice that the solution is smooth, even though the kernel is singular, a common occurrence.

CITED REFERENCES AND FURTHER READING:

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York). [1]

Stroud, A.H., and Secrest, D. 1966, *Gaussian Quadrature Formulas* (Englewood Cliffs, NJ: Prentice-Hall). [2]

Delves, L.M., and Mohamed, J.L. 1985, *Computational Methods for Integral Equations* (Cambridge, U.K.: Cambridge University Press). [3]

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.). [4]

18.4 Inverse Problems and the Use of A Priori Information

Later discussion will be facilitated by some preliminary mention of a couple of mathematical points. Suppose that **u** is an "unknown" vector that we plan to determine by some minimization principle. Let $\mathcal{A}[\mathbf{u}] > 0$ and $\mathcal{B}[\mathbf{u}] > 0$ be two positive functionals of **u**, so that we can try to determine **u** by either

$$\text{minimize:} \quad \mathcal{A}[\mathbf{u}] \qquad \text{or} \qquad \text{minimize:} \quad \mathcal{B}[\mathbf{u}] \tag{18.4.1}$$

(Of course these will generally give different answers for **u**.) As another possibility, now suppose that we want to minimize $\mathcal{A}[\mathbf{u}]$ subject to the *constraint* that $\mathcal{B}[\mathbf{u}]$ have some particular value, say b. The method of Lagrange multipliers gives the variation

$$\frac{\delta}{\delta\mathbf{u}}\left\{\mathcal{A}[\mathbf{u}] + \lambda_1(\mathcal{B}[\mathbf{u}] - b)\right\} = \frac{\delta}{\delta\mathbf{u}}\left(\mathcal{A}[\mathbf{u}] + \lambda_1\mathcal{B}[\mathbf{u}]\right) = 0 \tag{18.4.2}$$

where λ_1 is a Lagrange multiplier. Notice that b is absent in the second equality, since it doesn't depend on **u**.

Next, suppose that we change our minds and decide to minimize $\mathcal{B}[\mathbf{u}]$ subject to the constraint that $\mathcal{A}[\mathbf{u}]$ have a particular value, a. Instead of equation (18.4.2) we have

$$\frac{\delta}{\delta\mathbf{u}}\left\{\mathcal{B}[\mathbf{u}] + \lambda_2(\mathcal{A}[\mathbf{u}] - a)\right\} = \frac{\delta}{\delta\mathbf{u}}\left(\mathcal{B}[\mathbf{u}] + \lambda_2\mathcal{A}[\mathbf{u}]\right) = 0 \tag{18.4.3}$$

with, this time, λ_2 the Lagrange multiplier. Multiplying equation (18.4.3) by the constant $1/\lambda_2$, and identifying $1/\lambda_2$ with λ_1, we see that the actual variations are exactly the same in the two cases. Both cases will yield the same one-parameter family of solutions, say, $\mathbf{u}(\lambda_1)$. As λ_1 varies from 0 to ∞, the solution $\mathbf{u}(\lambda_1)$ varies along a so-called *trade-off curve* between the problem of minimizing $\mathcal{A}$ and the problem of minimizing $\mathcal{B}$. Any solution along this curve can equally well be thought of as either (i) a minimization of $\mathcal{A}$ for some constrained value of $\mathcal{B}$, or (ii) a minimization of $\mathcal{B}$ for some constrained value of $\mathcal{A}$, or (iii) a weighted minimization of the sum $\mathcal{A} + \lambda_1\mathcal{B}$.

The second preliminary point has to do with *degenerate* minimization principles. In the example above, now suppose that $\mathcal{A}[\mathbf{u}]$ has the particular form

$$\mathcal{A}[\mathbf{u}] = |\mathbf{A}\cdot\mathbf{u} - \mathbf{c}|^2 \tag{18.4.4}$$

for some matrix **A** and vector **c**. If **A** has fewer rows than columns, or if **A** is square but degenerate (has a nontrivial nullspace, see §2.6, especially Figure 2.6.1), then minimizing $\mathcal{A}[\mathbf{u}]$ will *not* give a unique solution for **u**. (To see why, review §15.4, and note that for a "design matrix" **A** with fewer rows than columns, the matrix $\mathbf{A}^T\cdot\mathbf{A}$ in the normal equations 15.4.10 is degenerate.) *However*, if we add any multiple λ times a nondegenerate quadratic form $\mathcal{B}[\mathbf{u}]$, for example $\mathbf{u}\cdot\mathbf{H}\cdot\mathbf{u}$ with **H** a positive definite matrix, then minimization of $\mathcal{A}[\mathbf{u}] + \lambda\mathcal{B}[\mathbf{u}]$ *will* lead to a unique solution for **u**. (The sum of two quadratic forms is itself a quadratic form, with the second piece guaranteeing nondegeneracy.)