Numer. Math. 34, 125 – 141 (1980)

Numerische Mathematik
© by Springer-Verlag 1980

*Handbook Series Approximations**

Procedures for Kernel Approximation and Solution of Fredholm Integral Equations of the Second Kind

Günther Hämmerlin[1] and Larry L. Schumaker[2]

[1] Mathematisches Institut der Ludwig-Maximilians-Universität, Theresienstraße 39, 8000 München 2, Germany (Fed. Rep.)
[2] Department of Mathematics and Center for Numerical Analysis, The University of Texas at Austin, Austin, TX 78712, U.S.A.

Summary. The purpose of this paper is to present explicit ALGOL procedures for (1) the approximation of a kernel (surface) by tensor products of splines, and (2) the computation of approximate eigenvalues and eigenfunctions for Fredholm integral equations of the second kind.

Subject Classifications: AMS(MOS): 41A15, 65R05, 68A10; CR: 5.13, 5.18.

1. Introduction

In this paper we combine a method of approximating a kernel in two dimensions by tensor product splines with the application of computing approximate solutions for homogeneous Fredholm integral equations of the second kind. The procedure for approximating a kernel is based on a specific spline approximation operator whose precise definition is given in Sect. 2. The method is a particular example of a general class of such operators studied by Lyche and Schumaker [6], and has the advantage that it is extremely fast, is local, and satisfies optimal order error bounds. In addition to being a key ingredient in the package for solving Fredholm integral equations, this procedure can also be used for the general problem of fitting a surface to a given function on a rectangle. Thus, for the convenience of users who wish to fit surfaces, we include two associated procedures for the evaluation of the spline surface at a given point or at all the points of a given grid.

The procedure for solving Fredholm integral equations is based on the use of the kernel approximation as a degenerate kernel, thus reducing the problem to a matrix eigenvalue problem. This approach is reviewed in Sect. 3; see also Hämmerlin [3].

Formal parameter lists and descriptions of the procedures presented here can be found in Sect. 4. Section 5 contains their listings. Finally, in Sect. 6 we give some numerical examples.

* *Editor's Note.* In this fascile, prepublication of algorithms from the Approximations series of the Handbook for Automatic Computation is continued. Algorithms are published in ALGOL 60 reference language as approved by the IFIP. Contributions in this series should be styled after the most recently published ones

2. The Kernel Approximation Method

As a first step towards defining our kernel approximation method, we introduce the classical B-splines. Let $m \geqq 1$ and $k \geqq 0$ be integers, and set $h = 1/(k+1)$. Now let

$$y_i = (i-m)\,h, \qquad i = 1, 2, \ldots, 2m+k \tag{2.1}$$

and

$$N_i^m(x) = m(-1)^m h\,[y_i, \ldots, y_{i+m}]\,(y-x)_+^{m-1}, \qquad i = 1, 2, \ldots, m+k. \tag{2.2}$$

We have normalized the B-splines so that $\sum_{i=1}^{m+k} N_i^m(x) = 1$ for all $0 \leqq x \leqq 1$. Their values (as well as those of their derivatives) can be computed by convenient stable recursion relations (cf. deBoor [1]).

For each $i = 1, 2, \ldots, m+k$ let

$$t_{ip} = y_i + ph - h/2, \qquad p = 1, 2, \ldots, m \ldots. \tag{2.3}$$

These points lie in the interior of the support of N_i^m. Let $\alpha_1^{(1)} = 1$ and the $\{\alpha_p^{(m)}\}_{p=1}^m$ be defined as in Table 1 in [5] or in procedure *eker* in Sect. 5.1 of this article.

We are now ready to define our approximation operator. Suppose that K is a function defined on the square $\tilde{U} = [-mh + 3h/2,\ 1 + mh - 3h/2] \times [-mh + 3h/2,\ 1 + mh - 3h/2]$. Then we set

$$QK(x,y) = \sum_{i=1}^{m+k} \sum_{j=1}^{m+k} (\lambda_{ij} K)\, N_i^m(x)\, N_j^m(y), \tag{2.4}$$

where

$$\lambda_{ij} K = \sum_{p=1}^{m} \sum_{q=1}^{m} \alpha_p^{(m)} \alpha_q^{(m)} K(t_{ip}, t_{jq}). \tag{2.5}$$

Q defines an operator mapping any function K defined on $\tilde{U}$ into a tensor product spline QK in $\mathcal{S} = \operatorname{span}\{N_i^m(x) N_j^m(y)\}_{i,j=1}^{m+k}$. In fact, QK can be defined as soon as we know all of the values $\{K(t_{ip}, t_{jq})\}$ appearing in (2.5). It is clear that QK can be computed directly without any matrix inversion. Since the B-splines have local support, it also follows that the value of $QK(x,y)$ at any given (x,y) depends only on the behavior of K in a relatively small neighborhood of (x,y).

Operators of the form (2.4) are studied in great detail in Lyche and Schumaker [6]. It is shown there that

$$Q\,p(x,y) = p(x,y), \qquad \text{all } (x,y) \in U \text{ and all } p \in \mathcal{P}_m \oplus \mathcal{P}_m, \tag{2.6}$$

where

$$U = [0,1] \times [0,1]$$

and

$$\mathcal{P}_m \oplus \mathcal{P}_m = \left\{ p(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} c_{ij}\, x^i y^j \right\}.$$

The paper of Lyche and Schumaker [6] contains an extensive collection of error bounds $\|K-QK\|_{L_q[U]}$, $1\leqq q\leqq\infty$, under various assumptions about the smoothness of K. Error bounds for the case where K is smooth except across the diagonal $x=y$ (as is often the case with the Green's kernels arising in integral equations) are given in Hämmerlin and Schumaker [5].

In closing this section we emphasize that the spline approximation operator Q provides approximations of the kernel K on the unit square U, but assumes that values of K can be computed at the points $\{t_{ip}, t_{jq}\}$ of the *slightly larger* square $\tilde{U}$. The design of Q in this way results in some substantial programming savings. There are related operators which require values of K only in U – see e.g. Lyche and Schumaker [6]. If it is desired to approximate a function K defined on a general rectangle $[a,b]\times[c,d]$, then by a simple change of variable the problem can be recast to one on the unit square.

3. The Integral Equation Method

Given a kernel in $L_2[U]$, we desire to find numbers κ_j and functions $\varphi_j\in L_2[0,1]$, $(j=1,2,\ldots)$ which solve the equation

$$\kappa\,\varphi(x)=\int_0^1 K(x,y)\,\varphi(y)\,dy. \tag{3.1}$$

Equation (3.1) is a Fredholm integral equation of the second kind, and κ_j and φ_j are eigenvalues and eigenfunctions, respectively. Generally, there will be an infinite sequence $\{\kappa_j\}$ and corresponding $\{\varphi_j\}$ solving (3.1).

Now, given an approximation

$$\tilde{K}(x,y)=\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\,B_i(x)\,B_j(y) \tag{3.2}$$

of K, where $\{B_i\}_1^n$ are any linearly independent functions on $[0,1]$, then the n eigenvalues $\{\tilde{\kappa}_j\}_1^n$ of the matrix

$$E=CP,\qquad C=(c_{ij})_{i,j=1}^n,\qquad P=\left(\int_0^1 B_i\,B_j\right)_{i,j=1}^n \tag{3.3}$$

may be regarded as approximations to n of the eigenvalues of (3.1) (cf. Hämmerlin [4]). Moreover, if $v_i=(v_{i1},\ldots,v_{in})^T$ is an eigenvector of E corresponding to the eigenvalue $\tilde{\kappa}_i$, then

$$\tilde{\varphi}_i(x)=\sum_{j=1}^{n} v_{ij}\,B_j(x) \tag{3.4}$$

will be an approximate eigenfunction of (3.1) corresponding to $\tilde{\kappa}_i$.

This method can be realized with any choice $\{B_i\}_1^n$, and with any convenient approximation scheme to produce $\tilde{K}$. Piecewise constant and bilinear splines were used in Hämmerlin [3], and B-splines coupled with interpolation were suggested in Hämmerlin [4]. In view of the many desirable properties of the method suggested in Sect. 2 (in particular its considerable savings in computational effort over interpolation methods while still producing best order approximations), it would seem natural to use Q to produce $\tilde{K}$. Thus, we choose $n = m + k$ and $B_i = N_i^m$, $i = 1, 2, \ldots, m+k$. Since the N_i^m have support on $[y_i, y_{i+m}]$, it follows that P is a matrix with $2m-1$ bands. In fact, because we have used equally spaced knots, the matrix P will not only be symmetric but will have the following special form:

$$\left[\begin{array}{cccc|cccc}
P_{11} & P_{12} & \cdots & P_{1m} & 0 & \cdots & & \\
P_{12} & P_{22} & \cdots & P_{2m} & P_{1m} & \ddots & & \mathbf{0} \\
\cdots & & & & & & \ddots & \\
P_{1m} & P_{2m} & \cdots & P_{m,m} & P_{m-1,m} & \ddots & & \\
\hline
0 & P_{1m} & \ddots & & \ddots & & \ddots & \\
 & & \ddots & & P_{m,m} & P_{m-1,m} & \cdots & P_{1,m} \\
 & \mathbf{0} & & \ddots & P_{m-1,m} & & \cdots & P_{m-1,1} \\
 & & & & \cdots & & & \\
 & & & & P_{1,m} & P_{1,m-1} & \cdots & P_{1,1}
\end{array}\right]$$

where each element on the diagonals indicated by lines is constant along that diagonal. It follows that P will be completely determined by the uppertriangular part of the m-by-m (i.e. by the principal) matrix in the upper lefthand corner. Because of this special structure, the product CP can also be computed efficiently.

For $m=1$ the matrix P is the identity matrix. Tables 1–4 contain the values of the P_{ij} for $m = 2, 3, 4, 5$.

Table 1. Inner-products for $m=2$

$$\frac{h}{6}\begin{bmatrix} 2 & 1 \\ & 4 \end{bmatrix}$$

Table 2. Inner-products for $m=3$

$$\frac{h}{120}\begin{bmatrix} 6 & 13 & 1 \\ & 60 & 26 \\ & & 66 \end{bmatrix}$$

Table 3. Inner-products for $m=4$

$$\frac{h}{5{,}040}\begin{bmatrix} 20 & 129 & 60 & 1 \\ & 1{,}208 & 1{,}062 & 120 \\ & & 2{,}396 & 1{,}191 \\ & & & 2{,}416 \end{bmatrix}$$

Table 4. Inner-products for $m=5$

$$\frac{h}{362{,}880}\begin{bmatrix} 70 & 1{,}121 & 1{,}581 & 251 & 1 \\ & 22{,}880 & 44{,}117 & 13{,}027 & 502 \\ & & 133{,}310 & 87{,}113 & 14{,}608 \\ & & & 156{,}120 & 88{,}234 \\ & & & & 156{,}190 \end{bmatrix}$$

These inner-products of B-splines have been hand-computed. The values for higher m could also be done, but the values given here suffice in most cases. We also mention that these inner-products can be computed numerically by the programs in deBoor, Lyche, and Schumaker [2].

We conclude this section with some remarks on how well the eigenvalues $\tilde{\kappa}_1, \ldots, \tilde{\kappa}_n$ of the matrix E approximate the true eigenvalues $\kappa_1, \ldots, \kappa_n, \ldots$ of the integral equation (3.1). Although our scheme can be used for any kernel, it is particularly well suited to symmetric kernels K. In this case, the eigenvalues of K are real, and the degenerate kernel $\tilde{K}$ is symmetric as well. The approximating integral equation with kernel $\tilde{K}$ is equivalent to the matrix equation with matrix E, and as a consequence the eigenvalues of E will be real. Thus, to compare the eigenvalues in this case, we suppose that they have been ordered according to their absolute values; i.e.,

$$|\kappa_1| \geqq |\kappa_2| \geqq \ldots \geqq 0 \quad \text{and} \quad |\tilde{\kappa}_1| \geqq |\tilde{\kappa}_2| \geqq \ldots \geqq |\tilde{\kappa}_n| \geqq 0,$$

where multiple eigenvalues are repeated as often as their multiplicity dictates. Then, as pointed out in [3, 4], a theorem of H. Weyl assures that

$$|\kappa_j - \tilde{\kappa}_j| \leqq \|K - \tilde{K}\|_{L_q[U]}, \qquad (j = 1, \ldots, n), \tag{3.5}$$

for any $1 \leqq q \leqq \infty$. (In general we get the best bounds by choosing $q = 1$.)

We can couple (3.5) with the error bounds for the spline approximation method $\tilde{K} = QK$ to predict the performance of the procedures presented in Sects. 4 and 5. The rate of convergence of the eigenvalues depends on the smoothness of K. Table 5 shows the expected rates of convergence using splines of order $m = 1, 2, \ldots, 5$ under the assumption that $K \in C^\sigma[\tilde{U}]$, $\sigma = 1, \ldots, 4$ (cf. Theorem 2.1 of [5]). Table 6 gives expected rates of convergence for the case where $K \in C^\sigma[\tilde{U}] \cap$

Table 5. Rate of convergence of eigenvalues for $K \in C^\sigma[\tilde{U}]$

$\sigma \backslash m$	1	2	3	4	5
1	h	h	h	h	h
2	h	h^2	h^2	h^2	h^2
3	h	h^2	h^3	h^3	h^3
4	h	h^2	h^3	h^4	h^4

Table 6. Rate of convergence of eigenvalues for Green's kernels in $C^\sigma[\tilde{U}] \cap C^{\sigma+2}(\tilde{T}_2)$

$\sigma \backslash m$	1	2	3	4	5
0	h	h^2	h^2	h^2	h^2
1	h	h^2	h^3	h^3	h^3
2	h	h^2	h^3	h^4	h^4
3	h	h^2	h^3	h^4	h^5
4	h	h^2	h^3	h^4	h^5

$C^{\sigma+2}[\tilde{T}_1] \cap C^{\sigma+2}[\tilde{T}_2]$, where $\tilde{T}_1 = \{(x, y) \in \tilde{U}:\ y \leqq x\}$ and $\tilde{T}_2 = \tilde{U} \setminus \tilde{T}_1$ (cf. Theorem 2.2 of [5]). This kind of smoothness assumption usually holds for the Green's kernels arising from ordinary differential equations.

4. Formal Parameter Lists and Description of the Program

4.1. General Organization of the Program. Procedures for Surface Approximation

The procedure *eker* constructs an approximation Q to a kernel *ker* defined on the square $\tilde{U}$. The user must supply a **real procedure** *ker*(x, y) with **real** parameters x, y which should be in the **value** list. If the user wants to evaluate Q only at one point (x, y) in the unit square, the procedure *kval* should be used. It is, however, not the most efficient way of producing all values of $Q(x, y)$ on a grid as is required in many applications. For such applications the procedure *grid* should be used. Both *grid* and *kval* call the procedure *ebspl* which computes the values of the m B-splines of order m with knot spacing $h = 1/(k+1)$ at a given point x in $[0, 1/(k+1)]$.

Procedures for Fredholm Integral Equations. The procedure *fred* computes approximate eigenvalues of the Fredholm integral equation (3.1) using a tensor product degenerate kernel. It calls the procedure *eker* which computes the B-spline coefficients of the spline approximation. At the end of *fred* the user must insert a call to a procedure which computes eigenvalues of an $(n \times n)$-matrix. It should output the **complex** vector *eigval* or the two vectors *reigval* and *ieigval* consisting of the real and imaginary parts, respectively.

4.2. Formal Parameter List and Description of the Procedure eker

procedure *eker*(m, k, ker, c);
value m, k; **integer** m, k;
real procedure *ker*;
array c;

Parameter list:

m	**integer** input parameter with $m \geqq 1$ defining the order of the spline.
k	**integer** input parameter with $k > 0$ giving the number of equispaced knots in $(0, 1)$.
ker	**real procedure** parameter defining a function *ker*(x, y) on $-mh \leqq x, y \leqq 1 + mh$, where $h = 1/(k+1)$. The two parameters of *ker* must be **real** and should be in the **value** list of *ker*.
c	**array** $[1{:}m+k,\ 1{:}m+k]$ output parameter. After the call of *eker*, c contains the evaluated spline coefficients.

4.3. *Formal Parameter List and Description of the Procedure ebspl*

procedure *ebspl*(m, x, k, bx);
value m, x, k; **integer** m, k;
real x;
array bx;

Parameter list:

$\left.\begin{matrix} m \\ k \end{matrix}\right\}$ as in *eker.*

x **real** input parameter, $0 \leqq x \leqq 1/k+1$.

bx **array** $[1:m+1]$, output parameter. The components 1 to m contain the values of the B-splines at x. The component $m+1$ is for workspace.

4.4. *Formal Parameter List and Description of the Procedure kval*

real procedure *kval*(m, i, j, k, x, y, c);
value m, i, k, x, y;
integer m, i, k, j;
real x, y;
array c;

Parameter list:

$\left.\begin{matrix} m \\ k \end{matrix}\right\}$ as in *eker*

i, j **integer** input. See x, y.

x, y **real** input parameters giving the coordinates of the point where the spline is to be evaluated. It is assumed that
i) $0 \leqq x, y \leqq 1$
ii) if $x<1$ then $ih \leqq x < (i+1)h$
if $x=1$ then $i=k$
iii) if $y<1$ then $jh \leqq y < (j+1)h$
if $y=1$ then $j=k$.

c **array** $[1:m+k, 1:m+k]$ input parameters giving the coefficient of the spline. Normally c is output of the procedure *eker.*

kval **real** output giving the value of the spline at the point (x, y).

4.5. *Formal Parameter List and Description of the Procedure grid*

procedure *grid*(m, n, k, c, g);
value m, n, k;
integer m, n, k;
array c, g;

Parameter list:

$\left.\begin{array}{l} m \\ k \end{array}\right\}$ as in *eker*.

n	**integer** input parameter describing how fine the grid should be.
c	**array** $[1:m+k,\ 1:m+k]$ input parameter giving the coefficients of the spline. Normally *c* is output of the procedure *eker*.
g	**array** $[0:n(k+1),\ 0:n(k+1)]$ output parameter giving the values of the spline at the grid points.

4.6. Formal Parameter List and Description of the Procedure fred

procedure *fred*(*m, k, ker, eigval*);
    or
procedure *fred*(*m, k, ker, reigval, ieigval*);
value *m, k*;
integer *m, k*;
real procedure *ker*;
complex array *eigval*;
    or
array *reigval, ieigval*; (if **complex arrays** are not supported.)

Parameter list:

$\left.\begin{array}{l} m \\ k \\ ker \end{array}\right\}$ as in *eker*.

eigval	**complex array** $[1:m+k]$, output array. It contains $m+k$ approximations to the eigenvalues of the integral equation respectively.
reigval	**real array** $[1:m+k]$ containing $m+k$ approximations to the real part of the eigenvalues of the integral equation.
ieigval	**real array** $[1:m+k]$ containing $m+k$ approximations to the imaginary part of the eigenvalues of the integral equation.

5. ALGOL Programs

5.1. The ALGOL Procedures for Kernel Approximation

procedure *eker*(*m, k, ker, c*);
value *m, k*; **integer** *m, k*;
real procedure *ker*; **array** *c*;
begin
comment purpose: The procedure constructs an approximation $Q(x,y)=\sum\sum c_{ij}\,N_i^m(x)\,N_j^m(y)$ to a kernel *ker*. The coefficients are computed as $c_{ij}=\lambda_{ij}\,ker$, where λ_{ij} are defined in (2.5);

```
comment input:
    m, an integer with m ≧ 1 defining the order of the spline,
    k, an integer with k > 0 giving the number of equispaced knots in (0, 1),
    ker, a real procedure defining a function ker(x, y) on −mh ≦ x, y ≦ 1 + mh,
        where h = 1/(k + 1);
comment output:
    c, the array of spline coefficients c_ij, i, j = 1, 2, ..., m + k;
integer i, j, m1, km1, km, p, q, ip, jq;
real sum, h, h2, v, w;
m1 := m − 1; km1 := k + m1; h := 1/(k + 1); h2 := h/2; km := k + m;
begin
    array kv[−m1:km1, −m1:km1], a[1:m];
    switch s := l1, l2, l3, l4, l5;
    go to s[m];
    l2: a[1] := a[2] :=  .5; go to next;
    l3: a[1] := a[3] := − .125;
        a[2] := 1.25; go to next;
    l4: a[1] := a[4] := − .14583333;
        a[2] := a[3] :=  .64583333; go to next;
    l5: a[1] := a[5] :=  .04079861;
        a[2] := a[4] := − .3715277; a[3] := 1.6614583;
    next:
    l1: v := h2 − h * m1;
comment: We now sample the kernel;
for i := −m1 step 1 until km1 do
begin
    w := h2 − h * m1;
    for j := −m1 step 1 until km1 do
        begin
        kv[i, j] := ker(v, w); w := w + h;
        end j;
    v := v + h;
end i;
if m > 1 then go to l6;
comment: If m = 1 we compute the B-spline coefficients;
for i := 1 step 1 until km do
for j := 1 step 1 until km do
    c[i, j] := kv[i − 1, j − 1];
go to fin;
l6:; comment: If m > 1, we compute the B-spline coefficients here;
    for i := 1 step 1 until km do
    begin
        ip := i − m − 1;
        for j := 1 step 1 until km do
        begin
            sum := 0; jq := j − m − 1;
            for p := 1 step 1 until m do
            for $q:=1$ step 1 until $m$ do
                $sum:=sum+a[p]*a[q]*kv[ip+p, jq+q]$;
            $c[i,j]:=sum$;
        end $j$;
    end $i$;
fin: end;
end eker;
```

Procedure *eker* can be used in the degenerate kernel approach to numerical solution of integral equations as discussed in the following section. For applications of *eker* to surface fitting, it is desirable to have evaluation procedures to produce the value of the spline.

Suppose we wish to evaluate the spline

$$s(x,y)=\sum_{i=1}^{m+k}\sum_{j=1}^{m+k} c_{ij}B_i(x)B_j(y). \tag{4.1}$$

If we know that the point (x,y) lies in the rectangle $U_{ij}=[ih,(i+1)h)\times[jh,(j+1)h)$, then by the support properties of the B-splines it follows that at most $\{B_v(x)\}_{v=i-m+1}^{i}$ and $\{B_\mu(y)\}_{\mu=j+1-m}^{j}$ can have non-zero values. The values of the B-splines can be computed by the well-known recursions (cf. [1]). Since we are using equally spaced points, the standard algorithm can be made somewhat more efficient.

```
procedure ebspl(m, x, k, bx);
value m, x, k; integer m, k; real x; array bx;
begin
comment purpose: To compute the values of the $m$ $B$-splines of order $m$ with
    knot spacing $h=1/(k+1)$ which have value at a given point;
comment input:
    $m$, an integer with $m\geq 1$ defining the order of the spline,
    $k$, an integer with $k\geq 0$ giving the number of equispaced knots in (0, 1),
    $x$, a real number satisfying $0\leq x\leq 1/(k+1)$;
comment output:
    bx, an array of $m$ real numbers containing the values of the $B$-splines at $x$.
        The array bx must be dimensioned as $bx[1:m+1]$ in the calling sequence.
        The extra component is for workspace;
integer i, j, j1, mp1;
real t, t1, prod;
$t1:=x*(k+1)$; $prod:=1$; $mp1:=m+1$;
for $i:=1$ step 1 until mp1 do
    $bx[i]:=0$;
$bx[m]:=1$;
for $j:=2$ step 1 until $m$ do
begin
    $j1:=j-1$; $t:=t1+j1$;
    for $i:=m-j1$ step 1 until $m$ do
    begin
        $bx[i]:=bx[i]*t+bx[i+1]*(j-t)$;
        $t:=t-1$;
    end i;
    prod:=prod/j1;
end j;
for i:=1 step 1 until m do
    bx[i]:=bx[i]*prod;
end ebspl;

real procedure kval(m, i, j, k, x, y, c);
value m, i, j, k, x, y; integer m, i, j, k; real x, y; array c;
begin
comment purpose: To compute the value of a spline s as in (4.1);
comment input:
    m, an integer with m≧1, defining the order of the spline,
    i, j, integers with 0≦i, j≦k,
    x, y, real numbers. It is assumed that if x<1 and y<1, then ih≦x≦(i+1)h
        and jh≦y<(j+1)h, where h=1/(k+1). If x=1, then i should be k and
        if y=1, then j should have value k.
    c, an array dimensioned as c[1:m+k, 1:m+k] giving the coefficients of the
        spline;
comment output:
    kval, a real number giving the value of the spline at the point (x, y);
integer nu, mu; real sum, temp, h;
array bx, by[1:m+1];
h:=1/(k+1);
ebspl(m, x-i*h, k, bx); ebspl(m, y-j*h, k, by); sum:=0;
for mu:=1 step 1 until m do
begin
    temp:=0;
    for nu:=1 step 1 until m do
        temp:=c[i+nu, j+mu]*bx[nu]+temp;
    sum:=sum+temp*by[mu];
end mu;
kval:=sum;
end kval;
```

Procedure *kval* can be used to evaluate the spline $s(x, y)$ of (4.1) at any (x, y) in the unit square. It is, however, not the most efficient way of producing all of the values of $s(x, y)$ on a grid, as is required in many applications. The following more efficient procedure is designed for such applications.

```
procedure grid(m, n, k, c, g);
value m, n, k; integer m, n, k; array c, g;
begin
comment purpose: To produce the values of a spline s defined as in (4.1) by a
    coefficient array c at all points of the form (i/nk, j/nk), i, j=0, 1, ..., nk, where
    nk=n(k+1);
```

```
comment input:
    m, an integer with $m \geqq 1$ defining the order of the spline,
    k, an integer with $k \geqq 0$ giving the number of knots in (0, 1),
    n, an integer with $n \geqq 1$ describing how fine the grid should be,
    c, an array dimensioned as $c[1:m+k, 1:m+k]$ giving the coefficients of s;
comment output:
    g, an array giving the values of the spline on the grid points. This array
        should be dimensioned as $g[0:n(k+1), 0:n(k+1)]$ in the call;
integer i, i1, in1, j, j1, jn1, mu, nu, n1, km, kp1, nkp1;
real sum, h;
array b[0:n, 1:m+1], isum[1:k+m], bx, by[1:m+1];
n1:=n-1; km:=k+m; kp1:=k+1; nkp1:=n*kp1; h:=1/nkp1;
for i:=0 step 1 until n do
begin
    ebspl(m, i*h, k, bx);
    for j:=1 step 1 until m do b[i, j]:=bx[j];
end i;
for i1:=0 step 1 until n1 do
begin
    for nu:=1 step 1 until m do bx[nu]:=b[i1, nu];
    for i:=0 step 1 until k do
    begin
    in1:=i*n+i1;
    for mu:=1 step 1 until km do
    begin
        sum:=0;
        for nu:=1 step 1 until m do sum:=sum+c[i+nu, mu]*bx[nu];
        isum[mu]:=sum;
    end mu;
    for j1:=0 step 1 until n1 do
    begin
        for nu:=1 step 1 until m do by[nu]:=b[j1, nu];
        for j:=0 step 1 until k do
        begin
            sum:=0;
            for mu:=1 step 1 until m do sum:=sum+isum[j+mu]*by[mu];
            g[in1, j*n+j1]:=sum;
        end j;
    end j1;
    sum:=0;
    for mu:=1 step 1 until m do sum:=sum+isum[mu+k]*b[n, mu];
    g[in1, nkp1]:=sum;
    end i;
end i1;
for mu:=1 step 1 until km do
begin
    sum:=0;
    for nu:=1 step 1 until m do sum:=sum+c[k+nu, mu]*b[n, nu];
    isum[mu]:=sum;
end mu;
for j1:=0 step 1 until n do
for j:=0 step 1 until k do
begin
    sum:=0; jn1:=j*n+j1;
    for mu:=1 step 1 until m do sum:=sum+isum[mu+j]*b[j1, mu];
    g[nkp1, jn1]:=sum;
end j, j1;
end grid;
```

5.2. ALGOL Procedure for Fredholm Integral Equations

```
procedure fred(m, k, ker, eigval);
value m, k; integer m, k;
real procedure ker;
    complex array eigval;
begin
comment purpose: To compute approximate eigenvalues of the Fredholm
    integral equation (3.1), using a tensor product degenerate kernel as in Sect. 3;
comment input:
    m, an integer with m≧1, defining the order of the spline,
    k, an integer with k≧0, giving the number of equispaced knots in (0, 1),
    ker, a real procedure which defines a function ker(x, y) for all −mh≦
        k, y≦1+mh, where h=1/(k+1);
comment output:
    eigval, a complex array containing m+k approximations to m+k of the
        eigenvalues of the integral equation;
comment: The procedure can be easily modified to output the coefficients of
    the corresponding eigenfunctions:
integer up, m1, km, mkj, m21, kmp1, kp1, j1, m2, i, j, n;
real d, h;
kp1:=k+1;km:=m+k;kmp1:=km+1;m21:=m+m−1;m1:=m−1;h:=1/kp1;
begin
array c, e[1:km, 1:km], a[1:m21, 1:m], ad[1:m21];
complex array evec[1:km, 1:km];
switch s:=l1, l2, l3, l4, l5;
go to s[m];
l1: eker(m, k, ker, c);
    for i:=1 step 1 until km do
    for j:=1 step 1 until km do
        e[i, j]:=h*c[i, j];
    go to fin;
l2: d:=h/6; a[1, 1]:=2*d; a[1, 2]:=d;
    a[2, 2]:=4*d; go to next;
```

```
l3: d:=h/120; a[1,1]:=6*d; a[1,2]:=13*d;
    a[1,3]:=d; a[2,2]:=60*d; a[2,3]:=26*d;
    a[3,3]:=66*d; go to next;
l4: d:=h/5040; a[1,1]:=20*d; a[1,2]:=129*d;
    a[1,3]:=60*d; a[1,4]:=d; a[2,2]:=1208*d;
    a[2,3]:=1062*d; a[2,4]:=120*d;
    a[3,3]:=2396*d; a[3,4]:=1191*d;
    a[4,4]:=2416*d; go to next;
l5: d:=h/362880; a[1,1]:=70*d; a[1,2]:=1121*d;
    a[1,3]:=1581*d; a[1,4]:=251*d; a[1,5]:=d;
    a[2,2]:=22880*d; a[2,3]:=44117*d; a[2,4]:=13027*d;
    a[2,5]:=502*d; a[3,3]:=133310*d; a[3,4]:=87113*d;
    a[3,5]:=14608*d; a[4,4]:=156120*d; a[4,5]:=88234*d;
    a[5,5]:=156190*d; go to next;
next: for j:=1 step 1 until m do
        for i:=j step 1 until m do
          a[i,j]:=a[j,i];
        for i:=1 step 1 until m1 do
        for j:=1 step 1 until m-i do
          a[m+j,j+i]:=a[m,i];
        for i:=1 step 1 until m21 do
          ad[i]:=a[i,m];
comment: We now call eker to approximate the kernel;
    eker(m,k,ker,c);
comment: We are ready to compute e=a*c;
for i:=1 step 1 until km do
begin
    for j:=1 step 1 until m1 do
    begin
    sum:=0; up:=m1+j;
    for n:=1 step 1 until up do
    sum:=sum+c[i,n]*a[n,j];
    e[i,j]:=sum; sum:=0; mkj:=kmp1-j;
    for n:=1 step 1 until up do
    sum:=sum+c[i,kmp1-n]*a[n,j];
    e[i,mkj]:=sum;
end j;
for j:=m step 1 until kp1 do
begin
    sum:=0; j1:=j-m;
    for n:=1 step 1 until m21 do
    sum:=sum+c[i,j1+n]*ad[n];
    e[i,j]:=sum;
end j;
end i;
fin:;
```

```
comment: At this point one must insert a procedure call which computes the
   km complex eigenvalues of the km by km matrix e. These should be output
   in the array eigval;
end; end fred;
```

6. Numerical Examples

In this section we give two numerical examples to illustrate the performance of the method for computing eigenvalues of Fredholm integral equations. The computations were done on the DEC-10 at The University of Texas at Austin and on the TR440 and CD-Cyber 175 at the Leibniz-Rechenzentrum, Munich.

Example 6.1. Let $K(s,t)=\sin\left(\frac{\pi}{2}(s+t)\right)$ (cf. [3]).

Discussion. This kernel is in $C(U)$. It is clearly symmetric, so its eigenvalues are real. The largest true eigenvalue of the homogeneous problem (3.1) is given by $\frac{1}{2}+\frac{1}{\pi}=0.81830989$. The absolute error in approximating this first eigenvalue using the spline method with $m=1,\dots,5$, and $k=3,6,12,24$ is shown in Table 7. In addition, estimates of the rate of convergence of the method for these choices of m are shown in Table 8. These were computed by finding the ratio $\ln(e_{k+3}/e_k)/\ln(1/(k+4)/1/(k+1))$ for $k=3,6,\dots,27$, where e_k denotes the error using k equally spaced knots.

We observe that for $m=2$ and 4, the algorithm performed as expected, giving quadratic and quartic convergence. What is remarkable, however, is that for odd m, the convergence was observed to be of order $m+1$ rather than order m as expected (at least for $m=1,3,5$). Because of this unexpected super-convergence for the odd splines, we also computed discrete L_1- and L_2-error bounds for the error in the approximation of the kernel itself. The rates of convergence agreed with those in Table 8. The reason for this super-convergence – which does not contradict our estimates – is given in the paper [7] by E. Schäfer. What is also remarkable is that the absolute errors for $m=1$ seem to be better than those for $m=2$ with the same k. The same holds for $m=3$ as compared with $m=4$.

Example 6.2. Let $K(s,t)=\begin{cases} s(1-t), & s\leqq t \\ t(1-s), & t\leqq s. \end{cases}$

Discussion. This is the well-known Green's function associated with the string problem. It is in $C(U)$, and C^∞ in each of the triangular parts defined by the diagonal $s=t$. This kernel is also symmetric, and the largest eigenvalue of the problem (3.1) is known to be approximately 0.10132118. The absolute error in approximating this eigenvalue using the spline method with $m=1,\dots,4$ and $k=3,6,12,24$ is shown in Table 9. Estimates of the rate of convergence for these values of m are shown in Table 10. For $m=2,3,4$ the convergence was quadratic as predicted by Table 6. Again, for $m=1$, we observed super-convergence, obtaining quadratic convergence rather than the expected linear. ■

Table 7. Errors in the first eigenvalue for Example 6.1

m \ k	3	6	12	24
1	2.0@−3	6.2@−4	1.8@−4	4.7@−5
2	5.1@−2	1.7@−2	5.0@−3	1.3@−3
3	6.9@−4	7.5@−5	6.3@−6	4.2@−7
4	2.8@−4	3.0@−4	2.6@−5	1.8@−6
5	4.7@−5	1.6@−6	2.2@−8	3.7@−8

Table 8. Rates of convergence in Example 6.1

m \ k	3	6	9	12	15	18	21	24	27
1	2.1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
2	1.97	1.99	2.0	2.0	2.0	2.0	2.0	2.0	2.0
3	3.97	3.99	4.0	4.03	3.99	3.80	5.0	4.67	3.98
4	3.96	3.99	3.99	4.02	3.99	3.98	4.21	4.41	3.85

Table 9. Errors in the first eigenvalue for Example 6.2

m \ k	3	6	12	24
1	1.3@−2	4.5@−3	1.3@−3	3.7@−4
2	1.9@−2	6.6@−3	2.0@−3	5.3@−4
3	4.1@−3	1.6@−3	4.8@−4	1.3@−4
4	1.5@−3	1.2@−3	4.4@−4	1.3@−4

Table 10. Rates of convergence in Example 6.2

m \ k	3	6	9	12	15	18	21	24	27
1	1.89	1.96	1.98	1.99	1.99	1.99	2.0	2.0	2.0
2	1.92	1.96	1.98	1.99	1.99	1.99	2.0	2.0	2.0
3	1.72	1.89	1.94	1.96	1.97	1.98	1.98	1.99	2.0
4	0.39	1.51	1.75	1.84	1.89	1.92	1.94	1.95	1.96

The method was also tested on a number of other kernels. These included the symmetric kernels $\sin(5\pi(s+t))$, $\sin(9\pi(s+t))$, $1+\exp(-s-t)$, and the unsymmetric kernels $1+t\exp(s+t)$, $1+\sin 3t\exp(s+\sin 3t)$, $\sin(2\pi s)-s\sin(2\pi t)$. It was further tested on the Green's kernel

$$K(s,t)=\begin{cases} s^3(t^3-1), & s\leqq t,\\ t^3(s^3-1), & t\leqq s,\end{cases}$$

and on the Green's functions of the differential operators

$$L\varphi := \frac{d^2\varphi}{dt^2} - \varphi \quad \text{with } \varphi(0) = \varphi(1) = 0$$

and

$$L\varphi := \frac{d^4\varphi}{dt^4} - 2\frac{d^2\varphi}{dt^2} + \varphi \quad \text{with } \varphi(0) = \varphi'(0) = \varphi(1) = \varphi'(1) = 0,$$

the latter ones possessing discontinuities in the first and third derivative, respectively.

For even m the rates of convergence agreed with those given in Tables 5 and 6, while for odd m we often but not always observed one higher-order convergence than expected from the lower estimates for the order of convergence given in the Theorems 2.1 and 2.2, combined with (3.5), when the kernel was smooth enough to allow it.

7. Remarks

1. It is also possible to construct simple procedures to find the (partial) derivatives of the tensor-product spline (2.4) at any point (x, y) in the unit square (cf. deBoor [1] for the one-dimensional case).

2. If it is desired to integrate the kernel against some other function, it will probably be most efficient to make use of the fact that it is a piecewise polynomial. Thus, a sufficiently accurate Gauss quadrature formula may be useful. The procedure grid can be modified easily to produce the values of the kernel at Gauss points in each subrectangle.

Acknowledgements. We would like to thank Ms. Ellin Wilson, Austin, and Mr. W. Lückemann, München, for programming assistance; also H.J. Oberle, Technische Universität München, and W. Busch, Universität Oldenburg, for performing additional test runs.
This work was supported in part by AFOSR grant 77-3150.

References

1. deBoor, C.: On calculating with *B*-splines, J. Approximation Theory **6**, 50-62 (1972)
2. deBoor, C., Lyche, T., Schumaker, L.L.: On calculating with *B*-splines, II. Integration. In: Numerische Methoden der Approximations-Theorie, ISNM Vol. 30, pp. 123-146. Basel: Birkhauser 1976
3. Hämmerlin, G.: Ein Ersatzkernverfahren zur numerischen Behandlung von Integralgleichungen 2. Art, Z. Angew. Math. Mech. **42**, 439-463 (1962)
4. Hämmerlin, G.: Zur numerischen Behandlung von homogenen Fredholmschen Integralgleichungen 2. Art mit Splines. In: Spline Functions Karlsruhe 1975, Lecture Notes in Mathematics 501, pp. 92-98. Berlin Heidelberg New York: Springer 1976
5. Hämmerlin, G., Schumaker, L.L.: Error bounds for the approximation of Green's kernels by splines. Numer. Math. **33**, 17-22 (1979)
6. Lyche, T., Schumaker, L.L.: Local spline approximation methods, J. Approximation Theory **15**, 294-325 (1975)
7. Schäfer, E.: Fehlerabschätzungen für Eigenwertnäherungen nach der Ersatzkernmethode bei Integralgleichungen. Numer. Math. **32**, 281-290 (1979)

Received February 1, 1978