A multi-grid method applied to a boundary value problem with variable coefficients in a rectangle

Technical Report · December 1977

CITATIONS
15

READS
164

1 author:

Wolfgang Hackbusch
Max Planck Institute for Mathematics in the Sciences
503 PUBLICATIONS **22,173** CITATIONS

SEE PROFILE

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Olshausenstr. 40
D-2300 Kiel 1

A Multi-Grid Method
Applied to a Boundary Value Problem
with Variable Coefficients in a Rectangle

Wolfgang Hackbusch

Bericht Nr 2/83
Februar 1983

This is a revised version of the author's Report 77-17 (Mathematisches Institut der Universität zu Köln) from December 1977

Summary. Multi-grid methods can be applied to very general boundary value problems. They are fast iterative methods. Thus, only few steps of the iteration are sufficient for obtaining the desired accuracy. The computational work of each iteration is proportional to the number of grid points.

Here, a multi-grid iteration is applied to a general elliptic equation of second order subject to arbitrary boundary conditions on the boundary of a rectangle. Although only linear problems are considered, the last example reported in this paper demonstrates that nonlinear boundary value problems can easily be treated by a combination of the multi-grid iteration with Newton's method.

In this revised report some printing errors of the previous Report 77-17 (Universität zu Köln, 1977) are corrected. The programmes are only very slightly changed. Recent literature is added.

The programmes named RECTC (ALGOL 6o) and RECTCF (FORTRAN) are attainable from the computer programme library of

OECD - NEA Data Bank
B.P. No. 9 (Bât. 45)
F - 9119o GIF-sur-YVETTE
France

Contents.

Subject Classifications. AMS(MOS): 65N2o, 35J25; CR: 5.17

1. Theoretical Background

1.1 The Boundary Value Problem

The differential equation

$$(1)\qquad a(x,y)\,u_{xx} + b(x,y)\,u_{xy} + c(x,y)\,u_{yy} + d(x,y)\,u_x + e(x,y)\,u_y + f(x,y)\,u = g(x,y) \qquad (4ac>b^2)$$

is to be satisfied in the rectangle $\Omega=(x_1,x_2)\times(y_1,y_2)$. The boundary Γ of Ω is the union of Γ_1: $x=x_1$, Γ_2: $y=y_1$, Γ_3: $x=x_2$, Γ_4: $y=y_2$ (cf. Fig. 1). On each side boundary conditions of the following kinds can be prescribed:

$$(2a)\qquad u(x,y) = \varphi_1(y) \qquad (x,y)\in\Gamma_1$$

$$(2b)\qquad \frac{\partial u(x,y)}{\partial n} + \alpha(y)\,u(x,y) = \varphi_2(x,y) \qquad (x,y)\in\Gamma_1$$

$$(2c)\qquad u(x_1,y) = u(x_2,y),\quad u_x(x_1,y) = u_x(x_2,y) \qquad y\in[y_1,y_2].$$

The conditions (2a-c) are formulated only for $(x,y)\in\Gamma_1$. In the case of Γ_2 and Γ_4 the function $\varphi_1(y)$ of the Dirichlet condition (2a) and α, φ_2 of the mixed boundary data (2b) are to be replaced by functions of $x\in[x_1,x_2]$. The normal derivation $\partial/\partial n$ coincides with $-\partial/\partial x$, $-\partial/\partial y$, $\partial/\partial x$, $\partial/\partial y$ for $(x,y)\in\Gamma_1$, Γ_2, Γ_3, Γ_4, respectively. If the solution is periodic with respect to the y-direction, the second equation of (2c) becomes $u_y(x,y_1)=u_y(x,y_2)$. The Neumann boundary condition corresponds to the special choice $\alpha(y)=o$ in (2b).

We allow any reasonable combination of the different kinds of boundary data on the four sides of Ω. In order to obtain a smooth solution, the conditions should not contradict each other at the corners (x_i,y_j). Optimal convergence of the applied method requires smooth coefficients in equation (1) (without g) and in (2b) (only α). In case of periodicity the coefficients are assumed to be periodic, too.

In some cases, where only Neumann or (and) periodic boundary data are given and $f(x,y)=o$ holds, the solution is determined up to a constant, provided that a integrability condition is satisfied. The program described below tests this condition and computes the unique solution which fulfils the additional requirement $u(x_1,y_1)=o$.

In Section 1.3 we shall need the assumption

(3) $c(x,y) / a(x,y)$ not too large (say ≤ 3).

This requirement can be omitted if the x-line-relaxation (cf. Section 1.3 and procedure *sm*) is replaced by a combination of x- and y-line-relaxation.

1.2 Discretization

We assume that x_2-x_1 and y_2-y_1 are even multiples of a given grid width $h > 0$. If possible choose $x_2-x_1 \geq y_2-y_1$. Furthermore, it is advantageous that (4) holds:

(4) $h_L = 2^L h$, $L \geq 0$; $(x_2-x_1)/h_L$ and $(y_2-y_1)/h_L$ be integers, one of them be a small number.

By virtue of (4) we may define the grids

$$\Omega_l = \{(x,y) \in \bar{\Omega} : (x-x_1)/h_l \text{ and } (y-y_1)/h_l \text{ integers}\}$$

corresponding to the step sizes $h_l = 2^l h$. The coarser grids Ω_l ($l \geq 1$) will be mentioned in the following section.

If $b(x,y)=0$ holds, the equation is discretized in Ω_0 by the usual five-point formula. Otherwise, the term u_{xy} is replaced either by differences of $u(x+ih,y+jh)$ $[(i,j)=(0,0), (\pm1,0), (0,\pm1), (1,1), (-1,-1)]$ or by $u(x+ih,y+jh)$ with $(i,j)=(0,0), (\pm1,0), (0,\pm1), (1,-1), (-1,1)$. The first formula is chosen if $a(x,y)\cdot b(x,y) > 0$; otherwise, the second formula is used.

In the case of (2b) the difference equation is also used at $(x_1,y) \in \Gamma_1$. The unknowns at the grid points (x_1-h,y) and $(x_1-h,y\pm h)$ are eliminated by means of the boundary condition.

The obtained system of difference equations is written as

(5) $S_0 u_0 = f_0$,

where the discrete solution u_0 is defined on Ω_0.

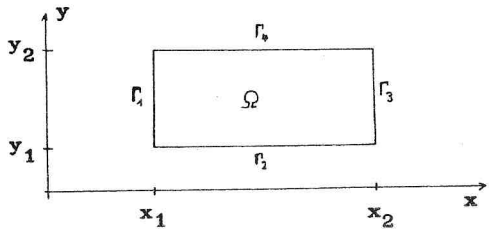

Fig. 1

1.3 The Multi-Grid Method

Multi-grid methods use recursively systems of the form

(6) $S_l u_l = f_l$ $(0 \leq l \leq L)$,

where u_l and f_l are defined on Ω_l. The system $S_L u_L = f_L$ corresponding to the coarsest grid is usually approximately solved by some steps of the line-relaxation iteration. This iteration converges well if (4) is fulfilled. Nevertheless, in some situations it is necessary to solve $S_L u_L = f_L$ exactly.

Having defined the method on the level L, we assume that the application of the multi-grid method is defined for $S_{l+1} u_{l+1} = f_{l+1}$ $(l \geq 0)$. In order to approximate the solution of $S_l u_l = f_l$, two steps are combined. First, Eq. (6) with $l+1$ instead of l is rewritten in the form

(7) $S_{l+1}\, \delta_{l+1} = f_{l+1} := r_l (S_l v_l - f_l)$,

where r_l denotes a restriction from Ω_l to Ω_{l+1}. By means of a prolongation p_l a given grid function v_l can be improved by $v_l \mapsto v_l - p_l \delta_{l+1}$ (correction step). The resulting function is smoothed by applying two steps of the line-relaxation iteration (smoothing step). The correction and the smoothing step yield one step of the multi-grid iteration on the level l.

The system (7) is approximately solved by one step of the multi-grid method starting with $v_{l+1}=0$. The matrix S_{l+1} involved in (7) is defined by

(8) $S_{l+1} = r_l S_l p_l$ $(0 \leq l \leq L-1)$.

p_l is the transposed mapping to r_l, and it describes the linear interpolation from Ω_{l+1} on Ω_l.

Starting with $u^{(0)}$ we compute the sequence $u^{(0)} \mapsto u^{(1)} \mapsto u^{(2)} \mapsto \ldots$ The most important property of the multi-grid iteration is its uniform rate of convergence:

(9) $\|u^{(\mu+1)} - u_0\|_2 \leq \varrho \|u^{(\mu)} - u_0\|_2$ (u_0: solution of (5)).

ϱ is independent of the step size h (cf. [1]) and insensitive to the coefficients of (1) and to the kind of the boundary data. Usually, $\varrho \ll 1$ holds as can be seen from the examples of Section 7 or from [2,3].

If no suitable starting value $u^{(o)}$ is known, one may choose

(1oa) $u^{(o)} = o$.

In many cases is seems to be more advantageous to use (1ob):

(1ob) $u^{(o)}$ = result of one smoothing step applied to $u^{(-1)}=o$.

The computational process is divided into three phases.

Phase A *(definition of the problem)*. The variables x_1, x_2, y_1, y_2, h are specified. The matrix S_o and the right-hand side f_o of the system (5) are determined.

Phase B *(preprocessing phase)*. The auxiliary matrices S_l are computed from (8). The computational work of this phase usually corresponds to about four steps of the multi-grid iteration.

Phase C *(iteration phase)*. A given number of steps of the multi-grid iteration is applied. Or the iteration is performed until the norm of the correction is smaller than a given error. The amount of the computational work is proportional to h^{-2}.

2. Applicability

In the following we only discuss the speed of convergence of the multi-grid iteration. We shall not study the discretization error which is proportional to h^2.

The Eq. (1) is elliptic only if $4ac>b^2$ holds on $\bar{\Omega}$. Examples reported in [2, 3] show that the speed of convergence becomes slow if b^2 approaches $4ac$. The iteration may fail to converge if $b^2 \approx 4ac$.

The iteration may also fail in case of singular perturbation problems, that means, if the convection term is much larger than the principal part: $4ac-b^2 \ll |d|+|e|$.

We abbreviate Eq. (1) by $Lu=g$ and assume without loss of generality that $a(x,y) < o$. The problem (1,2) is called *indefinite* if the eigenvalue problem $Lu = \lambda u$ (subject to homogeneous boundary conditions) has also negative solutions $\lambda < o$. In this case usual iterative methods cannot be applied. The multi-grid method do converge provided that the coarsest grid width h_L is small enough and that the corresponding systems $S_L u_L = f_L$ are solved exactly (e.g., by Gaussian elimination; cf. [2]). In Section 7.2 we shall present an example of an indefinite problem (cf. also [2,3]). The rate of convergence becomes bad (and finally exceeds the value one) if the eigenvalue with the smallest absolute value approaches zero.

Though the problem (1,2) is linear, this method can be combined with Newton's iteration to solve *nonlinear* problems, too. An application is described in Section 7.3. For applications to parabolic initial-boundary value problems compare [3].

The total storage requirement amounts to $\approx n*m/3$ (for n, m compare Section 3.1). If the system $S_L u_L = f_L$ is solved directly, in addition $[(2^{-L}n+1)\cdot(2^{-L}m+1)]^2$ places are needed.

3. Formal Parameter List

3.1 Procedure recta

The procedure *recta* performs the phase A (cf. Section 1.3). Its parameters kont, uo, fo, so, s1 are output parameters, while iso, iuo, n, eps and *problem* are input parameters.

kont The integer *kont* is used as output parameter and possibly indicates error conditions.

kont=0: successful computation of *s0* and *f0* (cf. Eq.(5)).

kont=1: n<2 or m<2 or n odd or m odd or $x_2 \leq x_1$ or $y_2 \leq y_1$ or $|(x_2-x_1)/n - (y_2-y_1)/m| > 10 \cdot eps$ or the value of iso is not correct. x_i and y_i are explained in Section 1 and in the procedure *problem*. n and m are defined below. The parameter iso is described below.

kont=2: The ratio c(x,y)/a(x,y) [cf. (3)] exceeds the value 3. The maximum is stored on s1[7,1]. The execution of *recta* is not terminated. A further call of rectb and *rectc* will not fail but the rate of convergence may become slow.

kont=3: There is a mistake in defining the kind of the boundary conditions by *problem(2,...)*.

kont=4: This value appears only if the solution is determined up to a constant. In order to obtain a solvable system, $S_o u_o = f_o$ is changed into $S_o u_o = \tilde{f}_o$, where $\varepsilon := \|f_o - \tilde{f}_o\|_2 > 10 * eps$. The computation may be continued, but note that $\tilde{f}_o$ is used. The error ε is stored on s1[6,1]. If s1[7,1] > 3 holds, also the error condition kont=2 applies.

kont<0: The index bound iuo of array uo (see below) is too small and should be defined by -kont.

uo The array uo[1:iuo] is used for the grid function u_o. The component uo[1+i+j*(n+1)] corresponds to the value of u_o at $(x,y) = (x_1+ih, y_1+jh)$, $h = (x_2-x_1)/n$, $0 \leq i \leq n$, $0 \leq j \leq m$. Therefore, iuo ≥ (n+1)*(m+1) must hold. For the possible requirement iuo ≥ 2*(n+1)*(m+1) compare the procedure *rectc* (Section 3.3). By means of *problem* (see below) a starting value can be assigned to uo.

fo The array fo[1:(n+1)*(m+1)] contains the right-hand side f_o of Eq. (5) and is defined after the call of *recta*.

so The discretization matrix S_o of Eq.(5) is stored on the array so[1:iso,1:iuo] with iso and iuo described below. The exact meaning of S_o is explained in Section 5.

iso The integer iso is the bound of the first index of array so (see above). It must be ≥5 in case of a five-point discretization (that is if b=0 in Eq. (1)) and ≥9 otherwise. It is possible to use iso=9 in all cases. If the value of iso is not compatible with the choice between five- and nine-point discretization made by procedure *problem* (*nr*=1), the error condition kont = 1 is shown.

iuo The integer iuo is the bound of the second index of array so (see above). If the requirement iuo ≥ (n+1)*(m+1) is violated, the parameter kont indicates the error (see above).

s1 The main part of the array s1[1:9,1:iu1] will be defined by the procedure *rectb* described below. Here only the components s1[3,1] and s1[6:8,1] are needed.

n The integer n (n≥2 and even) defines the step size $h = (x_2-x_1)/n$ (cf. *problem*, nr=1). The integer

$$m := \text{entier}\ (\ n*(y_2-y_1)\ /\ (x_2-x_1)\ +\ 0.5\ )$$

is assumed to satisfy m≥2, m even, $(y_2-y_1)/m \approx h$ (compare the error condition kont=1 mentioned above).

eps real eps is the machine precision.

problem The procedure problem contains all information about the differential equation and the boundary conditions. The meaning of the procedure and its heading can be seen from the following text.

```
procedure problem(nr,k,x,y,c1,c2,c3,c4,c5,c6,c7);
value nr,k,x,y; real x,y,c1,c2,c3,c4,c5,c6,c7; integer nr,k;
begin switch part:=domain, kind of boundary condition,
   coefficients, gamma1, gamma2, gamma3, gamma4,
   starting value;
   comment If nr>2 then (x,y) denotes the actual grid
   point of Ω̄ which is to be considered: x=x1+ih, y=y1+jh.
   The integer k = 1+i+j*(n+1) indicates the same grid
   point. If for example the function sin(x)*y*u(x,y) has
   to be evaluated and if the array uo is a global parameter,
   then sin(x)*y*uo[k] can be used. Compare also the
   example of Section 7.3;
goto part[nr];
domain:
   comment Here the numbers x1,y1,x2,y2 (cf. Section 1.1)
   must be assigned to the parameters c1,c2,c3,c4. c5 has
   to be defined by 5 (=five-point formula) if the
   coefficient b of Eq. (1) vanishes in Ω. Otherwise,
   c5:=9 must appear. c5 ≦ iso must hold (see parameter iso);
   goto end;
kind of boundary condition:
   comment Each of the parameters c1,c2,c3,c4 must be
   defined by o, 1 or 2. c1:=1 indicates that the Dirichlet
   condition (2a) is prescribed on Γ1, c1:=o is used for
   mixed boundary values (2b), while c1:=2 corresponds to
   the periodic data (2c). In the same way the kind of the
   boundary condition on Γ2 (y=y1), Γ3 (x=x2), and Γ4 (y=y2)
   must be defined by the parameters c2, c3, c4, respectively.
   Note that  c1=2 ⇔ c3=2 and  c2=2 ⇔ c4=2  must hold.
     The value +1 must be assigned to c5 if the solution
   is determined up to a constant. Otherwise, c5:=-1
   must be used;
   goto end;
coefficients:
   comment The coefficients a,b,c,d,e,f,g of the differential
   equation (1) are to be evaluated at (x,y) (actual
   values of the formal parameters x,y). The resulting
   values are to be assigned to the variables c1,c2,c3,
   c4,c5,c6,c7, respectively. If c5:=5 is used for nr=1,
   the value of c2 (=b(x,y)) is ignored since b=o is
   assumed;
   goto end;
```



```
gamma1:
   comment Here the coefficients of the boundary condition
   (2) on Γ1 are to be defined. In the case of (2a), the
   value φ1(y) (y: actual value of the parameter of this
   procedure) must be assigned to c1. In the case of (2b),
   c1 and c2 are to be defined by the values α(y) and
   φ2(y), respectively. In the case of (2c) nothing has
   to be done. The actual value of the parameter x is x1;
   goto end;

gamma2:
   comment Here the boundary conditions on Γ2 are to be
   defined analogously to the part gamma1. The actual
   value of the parameter y is y1. Note that φ1, φ2 and α
   depend on the parameter x;
   goto end;

gamma3:
   comment Here the boundary conditions on Γ3 are to be
   defined analogously to the part gamma1. The actual
   value of the parameter x is x2;
   goto end;
gamma4:
   comment Here the boundary conditions on Γ4 are to be
   defined analogously to the part gamma1. The actual
   value of the parameter y is y2. Note that φ1, φ2 and α
   depend on the parameter x;
   goto end;
starting value:
   comment If no starting value is to be assigned to the
   array uo, then insert an empty statement. Otherwise,
   the number assigned to c1 is used as starting value
   of uo[k], i.e. of the grid function at (x,y);

end:

end procedure problem;
```

3.2 Procedure rectb

The procedure *rectb* corresponds to the phase B (preprocessing phase; cf. Section 1.3). iu1, *s0*, *lm*, *min* and the components s1[3,1], s1[8,1] of s1 are input parameters of *rectb*. so and s1 must be defined by a foregoing call of the procedure *recta*.

s0 and the needed part of *s1* are not changed by intermediate calls of *rectb* or *rectc*. The auxiliary matrices S_1, S_2,..., S_L (cf. Section 1.3) computed by *rectb(kont,s1,s0,lm,min)* are stored on s1.

kont The integer kont possibly indicates error conditions:
kont=0: Successful computation of s1.
kont=5: Wrong choice of lm or min. The execution of *rectb* is continued with the default values lm=10 or min=1.
kont<0: The index bound iu1 is too small. It should at least be equal to -kont. See description of iu1.

s1 array s1[1:9,1:iu1], where iu1 is discussed below. s1[3,1] and s1[8,1] must be defined by a call of the procedure *recta* and are used as input parameters. After an execution of *rectb* the array s1 contains the auxiliary matrices S_l ($1 \le l \le L$).

iu1 The integer iu1 is the index bound for array s1. The minimal value of iu1 depends on the value of L (see Section 5.1). The choice $iu1 \ge (n*m-1)/3 + n + m + 8$ is always sufficient (for n and m see Section 3.1). A wrong choice of iu1 is indicated by kont<0 (see above).

so The array so[1:is0,1:iuo] is the same parameter as in the procedure *recta*. It must be defined by a call of *recta*.

lm,min lm and min > 0 of type integer are used for determining the index L mentioned in Section 1.3. L is the largest integer with $L \le abs(lm) \le 9$ satisfying $\nu_1 = n*2^{-L}$, $\nu_2 = m*2^{-L}$, where ν_1, $\nu_2 \ge min$ are integers. The treatment of the system $S_L u_L = f_L$ depends on the sign of lm.

lm > 0: In phase C the systems $S_L u_L = f_L$ will be approximately solved by some steps of the line-relaxation method.
lm ≤ 0: The systems $S_L u_L = f_L$ will be solved exactly by means of Gaussean elimination. The choice lm ≤ 0 is necessary if e.g. an indefinite problem is to be solved (cf. Sections 2 and 7.2).

Note that we use no dynamic declaration of the array nm[1:5,0:10] appearing in the procedure *rectc*, since L≤9 is sufficient in all practical applications.

3.3 Procedure rectc

The iteration phase (phase C; cf. Section 1.3) is performed by the procedure rectc(kont,uo,fo,so,u1,f1,s1,ep,itmax,outp). kont and uo are output parameters, while fo,so,s1,ep, and itmax are input parameters. outp is a procedure that can be used for observing the intermediate iterates $u^{(\mu)}$. The arrays uo,fo,so,s1 must be defined by calls of recta and rectb.

kont The integer kont indicates possible error conditions.
kont=0: Successful performance of the multi-grid iterations.
kont≥6: During a smoothing step (line-relaxation method) a tridiagonal system cannot be solved (without pivoting). The corresponding system (6) belongs to the level l=kont-6. If kont=L+6 and lm < 0 then the Gaussean elimination with pivoting fails. Use smaller step size h (cf. parameter n of recta) or a smaller value of abs(lm) (cf. Section 3.2).

uo,fo,so These arrays are the same as in recta. For the length iuo of the array uo[1:iuo] compare the description of the parameter ep. uo must contain the starting value $u^{(0)}$, for example assigned by a call of recta. *f0* and *s0* must be defined by a call of recta. The arrays *f0* and *s0* are not destroyed by a call of rectc. After the termination of the procedure rectc the array uo contains the grid function $u^{(\mu)}$ (μ determined by ep and itmax): $u^{(\mu)}(x,y) = uo[1+i+j*(n+1)]$ if $x=x_1+ih$, $y=y_1+jh$ (cf. Section 3.1, parameter uo).

u1, f1 — array u1,f1[1:iu1], where iu1 ≥ (n*m - 1)/3 + n + m + 8 (cf. Section 5.1). These arrays are used as auxiliary storage for the grid functions u_l, f_l ($1 \le l \le L$).

s1 — The array s1[1:9,1:iu1] is the same as described in Section 3.2. s1 must be defined by a call of rectb.

ep — real ep as well as the following parameter itmax control the iteration $u^{(o)} \mapsto u^{(1)} \mapsto \ldots$ The iteration is stopped if two subsequent grid functions $u^{(\mu-1)}$ and $u^{(\mu)}$ satisfy

$$\|u^{(\mu-1)}-u^{(\mu)}\|_2 := \left\{\sum_{i,j} |u_{ij}^{(\mu-1)}-u_{ij}^{(\mu)}|^2 / ((n+1)*(m+1))\right\}^{1/2} \le ep \qquad (\mu \le abs(itmax)).$$

If ep≤o or ep too close to the machine precision, then the termination of the iteration is controlled only by itmax (and possibly in an implicit manner by outp). In the case of ep > o, the array uo must be declared by uo[1:iuo] with

iuo ≥ 2*(n+1)*(m+1) (for n,m compare Section 3.1).

Otherwise, only iuo ≥ (n+1)*(m+1) is needed. Note that the *absolute* error is controlled by ep.

itmax — The sign of the integer itmax determines whether a smoothing step precedes the iteration or not.

itmax > o: The values stored on uo are used as starting value $u^{(o)}$.

itmax ≤ o: The values stored on uo are smoothed by some steps of the line-relaxation method. The result is used as starting value $u^{(o)}$.

If all entries of uo are zeros, then itmax > o and itmax ≤ o correspond to (1oa) and (1ob), respectively.

The iteration $u^{(o)} \mapsto u^{(1)} \mapsto \ldots \mapsto u^{(\mu)} \mapsto \ldots$ is terminated if the condition involving the parameter ep is fulfilled or if

μ = abs(itmax)

holds.

Note that the call of

rectc(kont,uo,fo,so,u1,f1,s1,o.o,itmax,outp)

with itmax = o is equivalent to the repeated call

for i:=1 step 1 until itmax do
rectc(kont,uo,fo,so,u1,f1,s1,o.o, 1 ,outp)

outp — After each computation of the grid function $u^{(my)}$ (my > o if itmax > o; my ≤ o, otherwise) the procedure outp is called. Its heading is

procedure outp(n,m,u,my,dif);
value n,m,my; real dif; integer n,m,my; array u;

All parameters are output parameters. The meaning of n and m is the same as used throughout this Section. my is the number of the iteration, while u contains $u^{(my)}$. The array u coincides with uo. dif equals the error

$$\|u^{(my-1)} - u^{(my)}\|_2 = \sqrt{\frac{1}{(n+1)(m+1)} \sum_{i=o}^{n} \sum_{j=o}^{m} |u^{(my-1)}(i*h,j*h) - u^{(my)}(i*h,j*h)|^2}$$

provided that my>o and ep>o. Otherwise, dif=o holds.

If u is changed by an execution of outp then these new values are used instead of $u^{(my)}$. dif may serve as input parameter. If a non-positive value is assigned to dif, the further execution of rectc will be terminated.

4. ALGOL 6o Programs

The algorithm is formulated in ALGOL 6o reference language as approved by the IFIP. Capitals are reserved to ALGOL word symbols.

```
PROCEDURE recta(kont,u0,f0,s0,is0,iu0,s1,n,eps,problem);
VALUE is0,iu0,n,eps;
INTEGER kont,is0,iu0,n; REAL eps; ARRAY u0,f0,s0,s1;
PROCEDURE problem;
BEGIN INTEGER n1,m1,n2,m2,i,j,jj,np,ii,k,is,m,i6,i7,i8;
      BOOLEAN px,py;
      REAL x1,x2,y1,y2,a,b,c,d,e,f,x,y,h,h2hq,hq,h2,g,am,ap,eps10,bm,
           bp,camax;
      PROCEDURE mix(s2,s3,s4,s5,s6,s7,s8,s9);
      REAL s2,s3,s4,s5,s6,s7,s8,s9;
      BEGIN c:=g*s2; s4:=(s4+s2)/2;
          s0[1,k]:=(s0[1,k]-c*a)/2; f0[k]:=f0[k]-c*b;
          IF is=9 THEN
          BEGIN s9:=(s9+s8)/2; c:=g*s8; s5:=s5-c*ap;
               s7:=(s7+s6)/2; d:=g*s6; s3:=s3-d*am;
               f0[k]:=f0[k]-d*bm-c*bp; s6:=s8:=0
          END;
          s5:=s5/2; s3:=s3/2; f0[k]:=f0[k]/2
      END difference scheme in the case of mixed boundary values;
      IF n<2 OR 2*entier(n/2)<n THEN
fail1:BEGIN kont:=1; GOTO end END;
      problem(1,1,0.0,0.0,x1,y1,x2,y2,a,b,b);
      is:=a; IF is NOTEQUAL 5 THEN is:=9; s1[8,1]:=is;
      IF is0<is THEN GOTO fail1;
      i8:=is-1; i7:=i8-1; i6:=i7-1;
      h:=(x2-x1)/n; eps10:=10*eps; kont:=0; s1[7,1]:=camax:=0;
      IF h<=0 OR y2<=y1 THEN GOTO fail1;
      m:=entier((y2-y1)/h+.5);
      IF m<2 OR 2*entier(m/2)<m OR abs(h-(y2-y1)/m)>eps10
      THEN GOTO fail1;
      problem(2,1,0.0,0.0,a,b,c,d,e,f,f);
      IF abs(e) NOTEQUAL 1 THEN
fail3:BEGIN kont:=3; GOTO end END;
      n1:=a; m1:=b; n2:=c; m2:=d; s1[3,1]:=e; px:=n1=2; py:=m1=2;
      IF n1<0 OR m1<0 OR n2<0 OR m2<0 OR n1>2 OR m1>2 OR
         n2>2 OR m2>2 OR NOT (px EQUIV n2=2) OR NOT (py EQUIV m2=2)
         OR e=1 AND n1*m1*n2*m2<=0 THEN GOTO fail3;
      IF px THEN n1:=n2:=0; IF py THEN m1:=m2:=0;
      n2:=n-n2; m2:=m-m2;
      y:=y1+(m1-1)*h; np:=n+1; jj:=m2*np+1;
      IF iu0<np*(m+1) THEN
      BEGIN kont:=-np*(m+1); GOTO end END;
      bm:=x1+(n1-1)*h; hq:=h*h; h2:=h/2; h2hq:=h2/hq; ap:=bp:=0;
```

This program as well as the FORTRAN program of Section 8 are developed on a Cyber 72 (Rechenzentrum der Universität zu Köln). The revised programs are performed on a Siemens 7.76o (Rechenzentrum der Universität Kiel). As mentioned on page 2 the programs are available from NEA Data Bank.


```
      FOR j:=m1*np+1 STEP np UNTIL jj DO
      BEGIN y:=y+h; x:=bm; FOR i:=n1 STEP 1 UNTIL n2 DO
          BEGIN k:=i+j; x:=x+h; problem(3,k,x,y,a,b,c,d,e,f,g);
               am:=c/a; IF am>camax THEN camax:=am;
               d:=d*h2; e:=e*h2;
               IF is=5 THEN b:=0 ELSE IF a*b<0 THEN
               BEGIN b:=b/2; s0[i6,k]:=s0[is,k]:=0;
                     s0[i7,k]:=s0[i8,k]:=-b; a:=a+b; c:=c+b
               END ELSE
               BEGIN s0[i6,k]:=s0[is,k]:=b:=b/2; a:=a-b; c:=c-b;
                     s0[i7,k]:=s0[i8,k]:=0
               END is=9;
               s0[1,k]:=f*hq-2*(a+b+c); s0[2,k]:=a-d;
               s0[4,k]:=a+d; s0[3,k]:=c-e; s0[5,k]:=c+e;
               f0[k]:=g*hq
      END END discretization;
      s1[7,1]:=camax; kont:=IF camax>3 THEN 2 ELSE 0;
      y:=y1; k:=0; g:=2*h;
      FOR j:=0 STEP 1 UNTIL m DO
      BEGIN x:=x1; FOR i:=0 STEP 1 UNTIL n DO
          BEGIN k:=k+1; problem(8,k,x,y,u0[k],a,a,a,a,a,a,a); x:=x+h
          END; y:=y+h
      END starting values;
      IF px THEN GOTO horizontal sides; k:=1; y:=y1;
      FOR j:=0 STEP 1 UNTIL m DO
      BEGIN IF n1>0 THEN problem(4,k,x1,y,u0[k],a,a,a,a,a,a)
          ELSE IF j<=m2 AND j>=m1 THEN
          BEGIN problem(4,k,x1,y,a,b,c,c,c,c,c,c);
                IF is=9 THEN
                BEGIN IF s0[i8,k] NOTEQUAL 0 THEN
                         problem(4,k+np,x1,y+h,ap,bp,c,c,c,c,c,c);
                      IF s0[i6,k] NOTEQUAL 0 THEN
                         problem(4,k-np,x1,y-h,am,bm,c,c,c,c,c,c)
                END;
                mix(s0[2,k],s0[3,k],s0[4,k],s0[5,k],s0[i6,k],
                    s0[i7,k],s0[i8,k],s0[is,k])
          END left boundary;
          k:=k+n;
          IF n2<n THEN problem(6,k,x2,y,u0[k],a,a,a,a,a,a) ELSE
          IF j>=m1 AND j<= m2 THEN
          BEGIN problem(6,k,x2,y,a,b,c,c,c,c,c,c);
                IF is=9 THEN
                BEGIN IF s0[is,k] NOTEQUAL 0 THEN
                         problem(6,k+np,x2,y+h,ap,bp,c,c,c,c,c,c);
                      IF s0[i7,k] NOTEQUAL 0 THEN
                         problem(6,k-np,x2,y-h,am,bm,c,c,c,c,c,c)
                END;
                mix(s0[4,k],s0[3,k],s0[2,k],s0[5,k],
                    s0[i7,k],s0[i6,k],s0[is,k],s0[i8,k])
          END right boundary;
          y:=y+h; k:=k+1
      END vertical sides;
```

```
horizontal sides:
      IF py THEN GOTO test of integrability;
      x:=x1; k:=1; jj:=m*np; j:=jj-1;
      FOR i:=0 STEP 1 UNTIL n DO
      BEGIN IF m1>0 THEN problem(5,k,x,y1,uO[k],a,a,a,a,a,a)
          ELSE IF i>=n1 AND i<=n2 THEN
          BEGIN problem(5,k,x,y1,a,b,c,c,c,c,c,c);
                IF is=9 THEN
                BEGIN IF sO[i7,k] NOTEQUAL 0 THEN
                         problem(5,k+1,x+h,y1,ap,bp,c,c,c,c,c,c);
                      IF sO[i6,k] NOTEQUAL 0 THEN
                         problem(5,k-1,x-h,y1,am,bm,c,c,c,c,c,c)
                END;
                mix(sO[3,k],sO[2,k],sO[5,k],sO[4,k],
                    sO[i6,k],sO[i8,k],sO[i7,k],sO[is,k])
          END boundary y=y1;
          k:=k+jj;
          IF m2<m THEN problem(7,k,x,y2,uO[k],a,a,a,a,a,a,a) ELSE
          IF i>=n1 AND i<=n2 THEN
          BEGIN problem(7,k,x,y2,a,b,c,c,c,c,c,c);
                IF is=9 THEN
                BEGIN IF sO[is,k] NOTEQUAL 0 THEN
                         problem(7,k+1,x+h,y2,ap,bp,c,c,c,c,c,c);
                      IF sO[i6,k] NOTEQUAL 0 THEN
                         problem(7,k-1,x-h,y2,am,bm,c,c,c,c,c,c)
                END;
                mix(sO[5,k],sO[2,k],sO[3,k],sO[4,k],
                    sO[i8,k],sO[i6,k],sO[is,k],sO[i7,k])
          END boundary y=y2;
          x:=x+h; k:=k-j
      END horizontal sides;
test of integrability:
      k:=IF px THEN 2 ELSE 1;
      IF px THEN
      BEGIN n1:=1; i:=1; FOR j:=0 STEP 1 UNTIL m DO
            BEGIN sO[1,i]:=sO[3,i]:=sO[5,i]:=fO[i]:=0; i:=i+np
      END   END px;
      IF py THEN
      BEGIN k:=k+2; m1:=1; FOR i:=1 STEP 1 UNTIL np DO
            sO[1,i]:=sO[2,i]:=sO[4,i]:=fO[i]:=0
      END py;
      s1[3,1]:=s1[3,1]*k;
      sO[2,1]:=IF n1=0 THEN n2 ELSE -n2;
      sO[3,1]:=IF m1=0 THEN m2 ELSE -m2;
      IF s1[3,1]<0 THEN GOTO end;
      d:=0; k:=m1*np; FOR j:=m1 STEP 1 UNTIL m DO
      BEGIN k:=k+n1; FOR i:=n1 STEP 1 UNTIL n DO
            BEGIN k:=k+1; d:=d+fO[k]
      END   END test if d=0 holds;
      s1[6,1]:=d:=d/(n*m); k:=0;
      FOR j:=0 STEP 1 UNTIL m DO
      BEGIN e:=IF j=0 OR j=m AND NOT py THEN d/2 ELSE d;
          FOR i:=0 STEP 1 UNTIL n DO
          BEGIN k:=k+1;
                fO[k]:=fO[k]-(IF i=0 OR i=n AND NOT px THEN e/2 ELSE e)
      END END right-hand side fO corrected;
end:
END procedure recta;
```



```
PROCEDURE igm(nm,ix,iy,iz,r,n,n1,n2,m,m1,m2,px,py,lm,s21,s31,s23);
INTEGER n,n1,n2,m,m1,m2,lm; REAL s21,s31,s23; BOOLEAN px,py;
INTEGER ARRAY nm,ix,iy,iz; ARRAY r;
COMMENT this procedure defines the values of the arrays nm[1:5,0:lm],
        ix,iy,iz[1:9], r[1:9,1:1];
BEGIN INTEGER l;
      l:=abs(s23); py:=l>2; IF py THEN l:=l-2; px:=l=2;
      n2:=s21; m2:=s31; n1:=m1:=0;
      IF n2<0 THEN BEGIN n1:=1; n2:=-n2 END; n:=-2*entier(-n2/2);
      IF m2<0 THEN BEGIN m1:=1; m2:=-m2 END; m:=-2*entier(-m2/2);
      IF lm<0 THEN GOTO ig;
      nm[1,0]:=n; nm[2,0]:=n2; nm[3,0]:=m; nm[4,0]:=m2;
      nm[5,0]:=nm[5,1]:=1;
      FOR l:=1 STEP 1 UNTIL lm DO
      BEGIN nm[1,l]:=nm[1,l-1]/2; nm[2,l]:=nm[1,l]+n2-n;
            nm[3,l]:=nm[3,l-1]/2; nm[4,l]:=nm[3,l]+m2-m;
            nm[5,l+1]:=nm[5,l]+(nm[1,l]+1)*(nm[3,l]+1)
      END;
ig:   ix[1]:=ix[3]:=ix[5]:=iy[1]:=iy[2]:=iy[4]:=iz[1]:=0;
      ix[2]:=ix[6]:=ix[8]:=iy[6]:=iy[3]:=iy[7]:=iz[2]:=-1;
      ix[4]:=ix[7]:=ix[9]:=iy[5]:=iy[8]:=iy[9]:=iz[4]:=1;
      r[1,1]:=1; r[2,1]:=r[3,1]:=r[4,1]:=r[5,1]:=.5;
      r[6,1]:=r[7,1]:=r[8,1]:=r[9,1]:=.25
END igm;
```

```
PROCEDURE rectb(kont,s1,iu1,s0,lm,min);
VALUE min,lm; INTEGER kont,iu1,lm,min; ARRAY s1,s0;
BEGIN INTEGER lmax,n2,m2,n1,m1,n,nn,m,mm,nn2,mm2,l,np,indl,ind,npp,
              i,j,kl,kp,kpi,ii,jj,l1,l2,l3,l4,k,i1,i2,j1,j2,is,i8,i7,i6,
              n11,m11,ll1,ll2,ll3,ll4;
    REAL d; INTEGER ARRAY ix,iy[1:9],nm[1:1,1:1];
    BOOLEAN px,py,defect,reg,b;
    ARRAY a1,a2[-2:2,-2:2], r[1:9,1:1];

    PROCEDURE loc(a1,a2,s,is,ist); INTEGER is,ist; ARRAY a1,a2,s;
    BEGIN defect:=ist=1; reg:=ii>0 AND ii<n AND jj>0 AND jj<m;
          b:=is=9; i8:=is-1; i7:=i8-1; i6:=i7-1;
          IF defect THEN
          BEGIN l1:=n11-ii; l2:=n2-ii; l3:=m11-jj; l4:=m2-jj;
                IF -2>l1 THEN l1:=-2; IF 2<l2 THEN l2:=2;
                IF -2>l3 THEN l3:=-2; IF 2<l4 THEN l4:=2;
                ll1:=IF l1>-1 THEN l1 ELSE -1;
                ll3:=IF l3>-1 THEN l3 ELSE -1;
                ll2:=IF l2<1 THEN l2 ELSE 1;
                ll4:=IF l4<1 THEN l4 ELSE 1
          END ELSE
          BEGIN ll1:=l1; ll2:=l2; ll3:=l3; ll4:=l4; kp:=1;
                l1:=-2*entier(-l1/2); l3:=-2*entier(-l3/2)
          END;
          FOR j1:=l3 STEP ist UNTIL l4 DO
          FOR i1:=l1 STEP ist UNTIL l2 DO
```

```
      BEGIN d:=0; IF defect THEN kp:=kpi+j1*np+i1;
            IF reg AND abs(j1)<2 AND abs(i1)<2 THEN
            BEGIN d:=s[1,kp]*a2[i1,j1]+s[2,kp]*a2[i1-1,j1]
                     +s[3,kp]*a2[i1,j1-1]+s[4,kp]*a2[i1+1,j1]
                     +s[5,kp]*a2[i1,j1+1];
                  IF b THEN d:=d+s[i6,kp]*a2[i1-1,j1-1]
                     +s[i7,kp]*a2[i1+1,j1-1]+s[i8,kp]
                     *a2[i1-1,j1+1]+s[is,kp]*a2[i1+1,j1+1]
            END ELSE
            FOR k:=1 STEP 1 UNTIL is DO
            BEGIN i2:=i1+ix[k]; j2:=j1+iy[k];
                  IF ll1<=i2 AND ll2>=i2 AND ll3<=j2 AND ll4>=j2
                  THEN d:=d+s[k,kp]*a2[i2,j2]
            END;
            a1[i1,j1]:=d
END   END loc;

is:=s1[8,1]; kont:=0; indl:=1;  lmax:=abs(lm);
IF lmax>9 THEN BEGIN kont:=5; lmax:=9 END;
IF min<=0 THEN BEGIN kont:=5; min:=1 END;
IF lm=0 THEN GOTO store;
igm(nm,ix,iy,ix,r,nn,n1,nn2,mm,m1,mm2,px,py,-1,s0[2,1],s0[3,1],s1[3,1])
IF (nn-nn2+n1=2 OR mm-mm2+m1=2) AND min=1 THEN min:=2;
n11:=IF px THEN 0 ELSE n1; m11:=IF py THEN 0 ELSE m1;
FOR l:=1 STEP 1 UNTIL lmax DO
BEGIN n:=nn; np:=n+1; m:=mm; n2:=nn2; m2:=mm2; ind:=indl;
      indl:=IF l=1 THEN 1 ELSE ind+np*(m+1);
      nn:=n/2; mm:=m/2;
      IF 2*nn NOTEQUAL n OR 2*mm NOTEQUAL m OR nn<min OR mm<min
      THEN GOTO change lmax;
      npp:=nn+1; nn2:=nn+n2-n; mm2:=mm+m2-m;
      j:=indl+npp*mm+nn;
      IF iu1<j THEN BEGIN kont:=-j; GOTO return END;
      FOR j:=0 STEP 1 UNTIL mm DO
      BEGIN jj:=j+j; kl:=indl+j*npp-1; kpi:=ind+jj*np-2;
        FOR i:=0 STEP 1 UNTIL nn DO
        BEGIN ii:=i+i; kl:=kl+1; kpi:=kpi+2; a1[0,0]:=1;
           a1[-2,-2]:=a1[-2,-1]:=a1[-2,0]:=a1[-2,1]:=
           a1[-2,2]:=a1[-1,-2]:=a1[-1,2]:=a1[0,-2]:=
           a1[0,2]:=a1[1,-2]:=a1[1,2]:=a1[2,-2]:=
           a1[2,-1]:=a1[2,0]:=a1[2,1]:=a1[2,2]:=0;
           a1[1,0]:=a1[0,1]:=a1[-1,0]:=a1[0,-1]:=0.5;
           a1[-1,-1]:=a1[1,-1]:=a1[-1,1]:=a1[1,1]:=.25;
           IF l=1 THEN loc(a2,a1,s0,is,1) ELSE loc(a2,a1,s1,9,1);
           loc(a1,a2,r,9,2);
           l1:=kl-npp; l2:=kl+npp; IF i>n11 THEN
           BEGIN IF j>m11 THEN s1[9,l1-1]:=a1[-2,-2];
                 s1[4,kl-1]:=a1[-2,0];
                 IF j<mm2 THEN s1[7,l2-1]:=a1[-2,2]
           END;
           IF j>m11 THEN s1[5,l1]:=a1[0,-2];
           s1[1,kl]:=a1[0,0];
           IF j<mm2 THEN s1[3,l2]:=a1[0,2];
           IF i<nn2 THEN
           BEGIN IF j>m11 THEN s1[8,l1+1]:=a1[2,-2];
                 s1[2,kl+1]:=a1[2,0];
                 IF j<mm2 THEN s1[6,l2+1]:=a1[2,2]
           END
        END i
      END j;
```



```
      IF py THEN
      BEGIN kl:=indl+n11; kp:=kl+mm*npp;
        FOR i:=n11 STEP 1 UNTIL nn2 DO
        BEGIN s1[5,kp]:=s1[5,kl]; s1[8,kp]:=s1[8,kl];
          s1[1,kp]:=s1[1,kp]+s1[1,kl]; s1[1,kl]:=0;
          IF i>0 THEN
          BEGIN s1[2,kp]:=s1[2,kp]+s1[2,kl]; s1[2,kl]:=0
          END i>0;
          IF i<nn THEN
          BEGIN s1[4,kp]:=s1[4,kp]+s1[4,kl]; s1[4,kl]:=0
          END i<nn;
          s1[9,kp]:=s1[9,kl]; kl:=kl+1; kp:=kp+1
        END i
      END py;
      IF px THEN
      BEGIN kl:=indl+m11*npp; kp:=kl+nn;
        FOR j:=m11 STEP 1 UNTIL mm2 DO
        BEGIN s1[4,kp]:=s1[4,kl]; s1[7,kp]:=s1[7,kl];
          s1[1,kp]:=s1[1,kp]+s1[1,kl]; s1[1,kl]:=0;
          IF j>0 THEN
          BEGIN s1[3,kp]:=s1[3,kp]+s1[3,kl]; s1[3,kl]:=0
          END j>0;
          IF j<mm THEN
          BEGIN s1[5,kp]:=s1[5,kp]+s1[5,kl]; s1[5,kl]:=0
          END j<mm;
          s1[9,kp]:=s1[9,kl]; kl:=kl+npp; kp:=kp+npp
        END j
      END px;
      IF l=lmax THEN GOTO store
    END l;
    GOTO return;
change lmax:  lmax:=l-1;
store:        s1[2,1]:=IF lm<0 THEN -lmax ELSE lmax;
return:
END procedure rectb;


PROCEDURE rectc(kont,u0,f0,s0,u1,f1,s1,ep,itmax,outp);
VALUE ep,itmax; INTEGER kont,itmax; REAL ep;
ARRAY u0,f0,s0,u1,f1,s1; PROCEDURE outp;
BEGIN INTEGER i,j,it,l,lmax,n1,m1,n,m,n2,m2,nmm,indl,ind,np,nm1,mm1,
              np2,i1j1,i1m1,i1j2,i1m2,i2j1,i2m1,nn,nn2,mm,mm2,npp,i1,
              j1,k,i2,j2,ii,jj,kp,kpi,ks,kl,kr,mmm1,nnm1,n11,m11,mnp,
              i6,i7,i8,is;
      REAL b1,b2,b3,b4,b5;
      BOOLEAN gauss,b,norm,px,py,nb,npx,npy,ep0;
      ARRAY r[1:9,1:1], a,c,e[0:2*entier((abs(s0[2,1])+1)/2)];
      COMMENT a,c,e[0:n];
      INTEGER ARRAY ix,iy,iz[1:9], nm[1:5,0:10];

PROCEDURE constants;
BEGIN n:=nm[1,l]; n2:=nm[2,l]; m:=nm[3,l]; m2:=nm[4,l];
      np:=n+1; nm1:=n-1; mm1:=m-1; np2:=2*np; ind:=nm[5,l];
      i1j1:=np+1; i1m1:=-n; i1j2:=np2+1; i1m2:=1-np2; i2j1:=np+2;
      i2m1:=2-np; iz[3]:=-np; iz[5]:=np; iz[6]:=-i1j1;
      iz[7]:=i1m1; iz[8]:=-i1m1; iz[9]:=i1j1; mnp:=m*np;
      b:=l>0 OR is=9; nb:=NOT b; i8:=8; i7:=7; i6:=6;
      k:=l+1; indl:=nm[5,k]; IF l=lmax THEN GOTO return;
      nn:=nm[1,k]; nn2:=nm[2,k]; mm:=nm[3,k]; mm2:=nm[4,k];
      npp:=nn+1; nnm1:=nn-1; mmm1:=mm-1;
return:
END constants;
```

```
PROCEDURE cor(u); ARRAY u;
BEGIN
      PROCEDURE corloc;
      BEGIN
          kpi:=ind+j*np+i; kl:=indl+(j*npp+i)/2; b2:=u1[kl]/2;
          IFj>m11 THEN u[kpi-np]:=u[kpi-np]-b2;
          IF j<m2 THEN u[kpi+np]:=u[kpi+np]-b2; b2:=b2/2;
          FOR k:=6 STEP 1 UNTIL 9 DO
          BEGIN i1:=i+ix[k]; IF i1<n11 OR i1>n2 THEN GOTO vc;
                j1:=j+iy[k]; IF j1<m11 OR j1>m2 THEN GOTO vc;
                kp:=kpi+iz[k]; u[kp]:=u[kp]-b2;
vc:       END
      END procedure corloc;
      FOR j:=1 STEP 1 UNTIL mmm1 DO
      BEGIN kp:=indl+j*npp; k:=ind+j*np2;
          FOR i:=1 STEP 1 UNTIL nnm1 DO
          BEGIN kp:=kp+1; k:=k+2; b2:=u1[kp]/2; b3:=b2/2;
                u[k-i1j1]:=u[k-i1j1]-b3; u[k+i1j1]:=u[k+i1j1]-b3;
                u[k+i1m1]:=u[k+i1m1]-b3; u[k-i1m1]:=u[k-i1m1]-b3;
                u[k+np]:=u[k+np]-b2; u[k-np]:=u[k-np]-b2
      END END;
      i:=0; IF n11=0 THEN
      FOR j:=2 STEP 2 UNTIL m2 DO corloc;
      j:=m; IF m2=m THEN FOR i:=2 STEP 2 UNTIL n2 DO corloc;
      i:=n; IF n=n2 THEN FOR j:=2*m11 STEP 2 UNTIL mm1 DO corloc;
      j:=0; IF m11=0 THEN
      FOR i:=2*n11 STEP 2 UNTIL nm1 DO corloc
END cor;

PROCEDURE sm(ia,ib,u,f,s,is);
VALUE is; INTEGER ia,ib,is; ARRAY u,f,s;
BEGIN e[n]:=1; c[n]:=a[n]:=0;
   FOR ii:=ia STEP 1 UNTIL ib DO
   BEGIN j1:=IF 2*entier(ii/2)=ii THEN 2*m1 ELSE 1;
      FOR j:=j1 STEP 2 UNTIL m2 DO
      BEGIN kp:=ind+j*np;
         IF j=m THEN GOTO last row; IF j>0 THEN GOTO mid;
         IF n1=0 THEN
         BEGIN e[0]:=s[1,kp]; a[0]:=s[4,kp];
               b1:=f[kp]-s[5,kp]*u[kp+np];
               IF b THEN b1:=b1-s[is,kp]*u[kp+i1j1];
               u[kp]:=b1
         END;
         IF n2=n THEN
         BEGIN kpi:=kp+n; e[n]:=s[1,kpi]; c[n]:=s[2,kpi];
               b1:=f[kpi]-s[5,kpi]*u[kpi+np]; IF nb THEN GOTO sm1;
               b1:=b1-s[i8,kpi]*u[kpi-i1m1]; IF npx THEN GOTO sm1;
               b1:=b1-s[is,kpi]*u[kpi+i1j1]; a[n]:=s[4,kpi];
sm1:           u[kpi]:=b1
         END;
         FOR i:=1 STEP 1 UNTIL nm1 DO
         BEGIN kpi:=kp+i; e[i]:=s[1,kpi]; c[i]:=s[2,kpi];
               a[i]:=s[4,kpi]; b1:=f[kpi]-s[5,kpi]*u[kpi+np];
               IF b THEN b1:=b1-s[i8,kpi]*u[kpi-i1m1]
                           -s[is,kpi]*u[kpi+i1j1]; u[kpi]:=b1
         END first row (j=0);
         GOTO tridiagonal system;
```



```
mid:     IF n1=0 THEN
         BEGIN e[0]:=s[1,kp]; a[0]:=s[4,kp];
               b1:=f[kp]-s[3,kp]*u[kp-np]-s[5,kp]*u[kp+np];
               IF b THEN b1:=b1-s[i7,kp]*u[kp+i1m1]
                           -s[is,kp]*u[kp+i1j1]; u[kp]:=b1
         END;
         IF n2=n THEN
         BEGIN kpi:=kp+n; e[n]:=s[1,kpi]; c[n]:=s[2,kpi];
               b1:=f[kpi]-s[3,kpi]*u[kpi-np]-s[5,kpi]*u[kpi+np];
               IF px THEN a[n]:=s[4,kpi]; IF nb THEN GOTO sm2;
               b1:=b1-s[i6,kpi]*u[kpi-i1j1]-s[i8,kpi]*u[kpi-i1m1];
               IF px THEN b1:=b1-s[is,kpi]*u[kp+i1j1]
                                -s[i7,kpi]*u[kp+i1m1];
sm2:           u[kpi]:=b1
         END;
         FOR i:=1 STEP 1 UNTIL nm1 DO
         BEGIN kpi:=kp+i;
               e[i]:=s[1,kpi]; c[i]:=s[2,kpi]; a[i]:=s[4,kpi];
               b1:=f[kpi]-s[3,kpi]*u[kpi-np]-s[5,kpi]*u[kpi+np];
               IF b THEN b1:=b1-s[i6,kpi]*u[kpi-i1j1]
                           -s[i7,kpi]*u[kpi+i1m1]
                           -s[i8,kpi]*u[kpi-i1m1]-s[is,kpi]*u[kpi+i1j1];
               u[kpi]:=b1
         END mid;
         GOTO tridiagonal system;
last row:
         IF py AND m=1 THEN mnp:=mnp+np;
         IF n1=0 THEN
         BEGIN e[0]:=s[1,kp]; a[0]:=s[4,kp];
               b1:=f[kp]-s[3,kp]*u[kp-np];
               IF b THEN b1:=b1-s[i7,kp]*u[kp+i1m1];
               IF npy THEN GOTO sm3;
               b1:=b1-s[5,kp]*u[kp-mnp+np];
               IF b THEN b1:=b1-s[is,kp]*u[kp-mnp+i1j1];
sm3:           u[kp]:=b1
         END;
         IF n2=n THEN
         BEGIN kpi:=kp+n; e[n]:=s[1,kpi]; c[n]:=s[2,kpi];
               b1:=f[kpi]-s[3,kpi]*u[kpi-np]; k:=kpi-mnp;
               IF px THEN a[n]:=s[4,kpi];
               IF py THEN b1:=b1-s[5,kpi]*u[k+np];
               IF nb THEN GOTO sm4; b1:=b1-s[i6,kpi]*u[kpi-i1j1];
               IF py THEN b1:=b1-s[i8,kpi]*u[k-i1m1];
               IF npx THEN GOTO sm4; kl:=k-n;
               IF py THEN b1:=b1-s[is,kpi]*u[kl+i1j1];
               b1:=b1-s[i7,kpi]*u[kp+i1m1];
sm4:           u[kpi]:=b1
         END;
```

```
          FOR i:=1 STEP 1 UNTIL nm1 DO
          BEGIN kpi:=kp+i;
                e[i]:=s[1,kpi]; c[i]:=s[2,kpi]; a[i]:=s[4,kpi];
                b1:=f[kpi]-s[3,kpi]*u[kpi-np];
                IF b THEN b1:=b1-s[i6,kpi]*u[kpi-i1j1]
                           -s[i7,kpi]*u[kpi+i1m1];
                IF npy THEN GOTO sm5; k:=kpi-mnp;
                b1:=b1-s[5,kpi]*u[k+np];
                IF b THEN b1:=b1-s[i8,kpi]*u[k-i1m1]
                           -s[is,kpi]*u[k+i1j1];
sm5:            u[kpi]:=b1
          END last row; mnp:=m*np;
tridiagonal system:
          IF px THEN GOTO periodic;
          b2:=e[0]; IF b2=0 THEN GOTO fail;
          u[kp]:=b1:=u[kp]/b2; a[0]:=b2:=a[0]/b2;
          FOR i:=1 STEP 1 UNTIL n DO
          BEGIN kpi:=kp+i; b3:=e[i]-c[i]*b2; IF b3=0 THEN GOTO fail;
                u[kpi]:=b1:=(u[kpi]-c[i]*b1)/b3; a[i]:=b2:=a[i]/b3
          END;
          FOR i:=nm1 STEP -1 UNTIL 0 DO
          u[kp+i]:=b1:=u[kp+i]-a[i]*b1;
          GOTO sm7;
periodic:
          IF n=1 THEN
          BEGIN b2:=e[1]+a[1]+c[1]; IF b2=0 THEN GOTO fail;
                u[kp+1]:=b5:=u[kp+1]/b2; GOTO sm6
          END n=1;
          b1:=b5:=0; b3:=1;
          FOR i:=1 STEP 1 UNTIL nm1 DO
          BEGIN b4:=c[i]; b2:=e[i]-b4*b1; IF b2=0 THEN GOTOfail;
                a[i]:=b1:=a[i]/b2; u[kp+i]:=b5:=(u[kp+i]-b4*b5)/b2;
                c[i]:=b3:=-b3*b4/b2
          END;
          b3:=c[nm1]:=b3+b1; COMMENT b1=a[nm1], b5=u[kp+nm1];
          FOR i:=n-2 STEP -1 UNTIL 1 DO
          BEGIN b1:=a[i]; c[i]:=b3:=c[i]-b1*b3;
                u[kp+i]:=b5:=u[kp+i]-b1*b5
          END;
          b4:=c[n]; b1:=a[n]; b2:=e[n]-b4*c[nm1]-b1*b3;
          IF b2=0 THEN GOTO fail;
          u[kp+n]:=b5:=(u[kp+n]-b4*u[kp+nm1]-b1*b5)/b2;
          FOR i:=1 STEP 1 UNTIL nm1 DO
          u[kp+i]:=u[kp+i]-c[i]*b5;
sm6:      u[kp]:=b5;
sm7:  END j;
      kp:=ind+n2; IF py THEN
      FOR j:=ind+n11 STEP 1 UNTIL kp DO u[j]:=u[j+mnp]
   END ii;
   IF norm AND l=0 THEN
   BEGIN b1:=u[ind];
         FOR i:=ind+mnp+n STEP -1 UNTIL ind DO u[i]:=u[i]-b1
   END normalization
END smoothing step;
```



```
PROCEDURE defect(u,f,s,is); VALUE is; INTEGER is; ARRAY u,f,s;
BEGIN
     PROCEDURE defloc;
     BEGIN ii:=2*i; jj:=2*j; b1:=0; kpi:=ind+jj*np+ii;
           FOR k:=1 STEP 1 UNTIL 4 DO
           BEGIN IF k=3 AND ii>0 AND ii<n THEN GOTO dl;
                 i1:=ii+ix[k]; IF i1<0 OR i1>n THEN GOTO dl;
                 j1:=jj+iy[k]; IF j1<0 OR j1>m THEN GOTO dl;
                 kp:=kpi+iz[k]; b2:=-f[kp];
                 FOR kl:=1 STEP 1 UNTIL is DO
                 BEGIN i2:=i1+ix[kl]; j2:=j1+iy[kl];
                       IF i2>=0 AND j2>=0 AND i2<=n AND j2<=m THEN
                       b2:=b2+s[kl,kp]*u[kp+iz[kl]]
                 END kl;
                 b1:=b1+b2*r[k,1];
dl:        END;
           f1[indl+j*npp+i]:=b1
     END defloc;
     FOR j:=1 STEP 1 UNTIL mmm1 DO
     BEGIN kpi:=indl+j*npp; k:=ind+j*np2;
         FOR i:=1 STEP 1 UNTIL nnm1 DO
         BEGIN k:=k+2; kr:=k+1; kl:=k-1; kp:=k+np; ks:=k-np;
            f1[kpi+i]:=IF b THEN
            (u[k-i2m1]*s[i8,kl]+u[k-i1m1]*(s[5,kl]+2*s[i8,k])
            +u[kp]*(2*s[5,k]+s[is,kl]+s[i8,kr])
            +u[k+i1j1]*(2*s[is,k]+s[5,kr])+u[k+i2j1]*s[is,kr]
            +u[k-2]*s[2,kl]+u[kl]*(2*s[2,k]+s[1,kl])
            +u[k]*(2*s[1,k]+s[4,kl]+s[2,kr])
            +u[kr]*(2*s[4,k]+s[1,kr])+u[k+2]*s[4,kr]
            +u[k-i2j1]*s[i6,kl]+u[k-i1j1]*(2*s[i6,k]+s[3,kl])
            +u[ks]*(2*s[3,k]+s[i7,kl]+s[i6,kr])
            +u[k+i1m1]*(2*s[i7,k]+s[3,kr])+u[k+i2m1]*s[i7,kr]
            -f[kl]-f[kr])/2-f[k]
            ELSE
            (u[k-i1m1]*s[5,kl]+u[k+i1j1]*s[5,kr]
            +u[k+i1m1]*s[3,kr]+u[k-i1j1]*s[3,kl]
            +u[kl]*(s[1,kl]+2*s[2,k])+u[k-2]*s[2,kl]
            +u[kr]*(s[1,kr]+2*s[4,k])+u[k+2]*s[4,kr]-f[kl]
            -f[kr]+u[k]*(2*s[1,k]+s[4,kl]+s[2,kr]))/2
            -f[k]+u[kp]*s[5,k]+u[ks]*s[3,k]
     END END;
     i:=0; IF n11=0 THEN FOR j:=1 STEP 1 UNTIL mm2 DO defloc;
     j:=mm; IF m2=m THEN FOR i:=1 STEP 1 UNTIL nn2 DO defloc;
     i:=nn; IF n=n2 THEN FOR j:=m11 STEP 1 UNTIL mmm1 DO defloc;
     j:=0; IF m11=0 THEN FOR i:=n11 STEP 1 UNTIL nnm1 DO defloc;
     IF px THEN
     BEGIN kr:=indl+m11*npp; k:=kr+nn;
           FOR j:=m11 STEP 1 UNTIL mm2 DO
           BEGIN f1[k]:=f1[k]+f1[kr]; f1[kr]:=0; k:=k+npp; kr:=kr+npp
     END   END px;
     IF py THEN
     BEGIN kr:=indl+n11; k:=kr+mm*npp;
           FOR i:=n11 STEP 1 UNTIL nn2 DO
           BEGIN f1[k]:=f1[k]+f1[kr]; f1[kr]:=0; k:=k+1; kr:=kr+1
END  END   END defect;
```

```
PROCEDURE direct(u,f,s,is); VALUE is; INTEGER is; ARRAY u,f,s;
BEGIN ARRAY a[1:mnp+np,1:mnp+np];
      ks:=m+1; nn:=np*ks; indl:=ind-1; k:=0;
      kr:=IF norm THEN 1+(IF px THEN n ELSE 0)
          +(IF py THEN mnp ELSE 0) ELSE 0;
      FOR j:=1 STEP 1 UNTIL nn DO
      FOR i:=1 STEP 1 UNTIL nn DO a[i,j]:=0;
      FOR j:=0 STEP 1 UNTIL m DO FOR i:=0STEP 1 UNTIL n DO
      BEGIN k:=k+1; kpi:=k+indl;
          IF k=kr OR i<n1 OR i>n2 OR j<m1 OR j>m2 THEN
          BEGIN a[k,k]:=1;
                IF k=kr THEN
                BEGIN u[kpi]:=0; GOTO dir0 END;
                IF i=0 AND px THEN a[k,k+n]:=-1;
                IF j=0 AND py THEN BEGIN a[k,k+mnp]:=-1; u[kpi]:=0 END;
dir0:     END ELSE
          FOR kp:=1 STEP 1 UNTIL is DO
          BEGIN j1:=k+iz[kp]; i1:=i+ix[kp]; u[kpi]:=f[kpi];
                IF i1<0 THEN GOTO dir1;
                IF i1>n THEN BEGIN IF npx THEN GOTO dir1; j1:=j1-n END;
                i1:=j+iy[kp]; IF i1<0 THEN GOTO dir1;
                IF i1>m THEN BEGIN IF npy THEN GOTO dir1; j1:=j1-mnp END;
                BEGIN IF npy THEN GOTO dir1; j1:=j1-mnp END;
                a[k,j1]:=a[k,j1]+s[kp,kpi];
dir1: END END;
      nn2:=nn-1; FOR i:=1 STEP 1 UNTIL nn2 DO
      BEGIN b1:=abs(a[i,i]); k:=i; i1:=i+1;
            FOR j:=i1 STEP 1 UNTIL nn DO
            BEGIN b2:=abs(a[j,i]); IF b2>b1 THEN
                  BEGIN k:=j; b1:=b2
            END   END;
            IF k>i THEN
            BEGIN b1:=u[indl+i]; u[indl+i]:=u[indl+k];
                  u[indl+k]:=b1; FOR j:=i STEP 1 UNTIL nn DO
                  BEGIN b1:=a[i,j]; a[i,j]:=a[k,j]; a[k,j]:=b1
            END   END;
            b1:=a[i,i]; IF b1=0 THEN GOTO fail; b3:=u[indl+i];
            FOR k:=i1 STEP 1 UNTIL nn DO
            BEGIN b2:=a[k,i]/b1; u[indl+k]:=u[indl+k]-b2*b3;
                  FOR j:=i1 STEP 1 UNTIL nn DO a[k,j]:=a[k,j]-b2*a[i,j]
      END   END;
      IF a[nn,nn]=0 THEN GOTO fail;
      FOR j:=nn STEP -1 UNTIL 1 DO
      BEGIN b1:=u[indl+j]:=u[indl+j]/a[j,j]; j1:=j-1;
            FOR i:=1 STEP 1 UNTIL j1 DO u[indl+i]:=u[indl+i]-a[i,j]*b1
END   END direct;
```



```
      lmax:=s1[2,1]; gauss:=lmax<=0; lmax:=abs(lmax);
      ep0:=ep>0; ep:=abs(ep); kont:=0; e[0]:=1; a[0]:=0;
      is:=s1[8,1];
      igm(nm,ix,iy,iz,r,n,n1,n2,m,m1,m2,px,py,lmax,s0[2,1],s0[3,1],s1[3,1]);
      nmm:=(n+1)*(m+1); norm:=s1[3,1]>0; npx:=NOT px; npy:=NOT py;
      n11:=IF px THEN 0 ELSE n1; m11:=IF py THEN 0 ELSE m1;
      IF ep0 THEN
      FOR i:=1 STEP 1 UNTIL nmm DO u0[i+nmm]:=u0[i];
      it:=IF itmax<0 THEN -1 ELSE 0; itmax:=abs(itmax); l:=-1;
next level:
      l:=l+1; constants;
      IF l=0 THEN
iteration:
      BEGIN IF it>=itmax THEN GOTO rtn; it:=it+1;
            IF it=0 THEN GOTO smoothing
      END l=0 ELSE
      FOR i:=indl-1 STEP -1 UNTIL ind DO u1[i]:=0;
      IF l=lmax THEN GOTO smoothing;
      IF l=0 THEN
      BEGIN sm(3,3,u0,f0,s0,is); defect(u0,f0,s0,is) END ELSE
      BEGIN sm(3,3,u1,f1,s1,9);  defect(u1,f1,s1,9)  END;
      GOTO next level;
smoothing:
      IF l>0 THEN GOTO auxiliary level;
      IF lmax=0 AND gauss THEN
      BEGIN direct(u0,f0,s0,is); GOTO rtn END;
      sm(2,4,u0,f0,s0,is);
      b1:=0; IF ep0 THEN
      BEGIN FOR i:=1 STEP 1 UNTIL nmm DO
            BEGIN b2:=u0[i]; b3:=u0[i+nmm]-b2; b1:=b1+b3*b3;
                  u0[i+nmm]:=b2
            END;
            b1:=sqrt(b1/nmm)
      END l2-norm of the difference;
      outp(n,m,u0,it,b1);
      IF ep0 AND it>0 AND b1<=ep OR b1<0 THEN GOTO rtn;
      GOTO correction;
auxiliary level:
      IF l=lmax AND gauss THEN direct(u1,f1,s1,9)
      ELSE sm(2,5,u1,f1,s1,9);
correction:
      IF l=0 THEN GOTO iteration; l:=l-1; constants;
      IF l=0 THEN cor(u0) ELSE cor(u1);
      GOTO smoothing;
fail: kont:=l+6;
rtn:
END procedure rectc;
```

5. Organizational and Notational Details

5.1 Organization of the Arrays uo,fo,so,u1,f1,s1

The system (5), $S_o u_o = f_o$, is represented by means of uo,fo,so. The equation of $S_o u_o = f_o$ corresponding to the grid point $(x,y) = (ih, jh)$ is written as

$$(5') \qquad \sum_{\nu=1}^{is} so[\nu,k] * uo[k+iz[\nu]] = fo[k],$$

where k=1+i+j*(n+1), is=5 or is=9 (cf. Section 3.1, problem, nr=1). The index k+iz[ν] corresponds to (x,y) + (Δi*h, Δj*h), where Δi and Δj are listed in the following tables:

ν	1	2	3	4	5	6	7	8	9
Δi	o	-1	o	1	o	-1	1	-1	1
Δj	o	o	-1	o	1	-1	-1	1	1

Table 1a: Δi, Δj depending on ν

Δj \ Δi	-1	o	1
+1	8	5	9
o	2	1	4
-1	6	3	7

Table 1b: ν depending on Δi and Δj

In the case of Dirichlet boundary values (2a) the equation (5') is omitted for $(x,y) \in \Gamma_1$. If mixed boundary data are used, then the sum of Eq. (5') is taken only over ν with $(x,y) + (\Delta i*h, \Delta j*h) \in \bar{\Omega}$. In the case of the periodic condition (2c) the components u(x,y) with $(x,y) \in \Gamma_3$ (i.e., $x=x_2$) are used as unknows, whereas u(x,y) is omitted for $(x,y) \in \Gamma_1$. Introducing the integers $n1, m1 \in \{o,1\}$, $n2 \in \{n, n-1\}$, $m2 \in \{m, m-1\}$, we can characterize all unknowns of the grid function by uo[k], k = 1+i+j*(n+1), $n1 \le i \le n2$, $m1 \le j \le m2$.

The auxiliary matrices S_l and the grid functions u_l, f_l ($1 \le l \le L$, cf. Eq. (6)) are stored on s1[1:9,1:iu1], u1,f1[1:iu1], where iu1 ≥ 1 and

$$iu1 \ge n*m*(1 - 4^{-L})/3 + (n+m)*(1 - 2^{-L}) + L.$$

In the programs L is written as lmax. The system (6) consists of the equations

$$(6') \qquad \sum_{\nu=1}^{9} s1[\nu,k] * u1[k+iz[\nu]] = f1[k],$$

where $k = nm[5,l] + j*(nm[1,l]+1) + i$, $(x,y) = (i*2^l h, j*2^l h)$, $n1 \le i \le nm[2,l]$, $m1 \le j \le nm[4,l]$. The array nm[1:5,o:1o] is defined by the procedure igm.

In order to obtain short parameter lists for the procedures rectb and rectc, the discretization parameters n, m, the information about the kind of the boundary data, and the integers L and is are stored on so[2:3,1] and s1[2:3,1], s1[8,1] (note that these components are not needed in (5') or (6'), since $(x,y) + (\Delta i*h, \Delta j*h) \notin \bar{\Omega}$). Let n1,n2,m1,m2 be the variables defined above. These values can be computed from

$$so[2,1] := n2*sign(\tfrac{1}{2} - n1), \quad so[3,1] := m2*sign(\tfrac{1}{2} - m1).$$

Furthermore, n:=2*entier((n2+1)/2) and m:=2*entier((m2+1)/2) can be obtained. Note that n and m are assumed to be even integers. The sign of

$$s1[2,1] := \pm\, lmax \qquad (lmax = L)$$

indicates if $S_L u_L = f_L$ is to be solved directly (-) or if it is to be approximated by the line-relaxation method (+). The Boolean variables px and py take the value true if the boundary condition is periodic with respect to x or y, respectively. The condition $u(x_1,y_1)=o$ (i.e., uo[1]=o) is added if and only if norm = true holds. These values are contained in

```
s1[3,1] := ( if norm then 1 else -1)
*((if px then 2 else 1) + (if py then 2 else o)).
```

The integer is mentioned in Eq. (5') is stored on s1[8,1]. In Section 3.1 we mentioned that the component d = s1[6,1] has a special meaning in case of norm = true. In order to obtain an integrable system (5), the right-hand side fo is changed into fo[k]+d (if k corresponds to grid points belonging to Γ_i, possibly, d is replaced by d/2 or d/4; cf. Section 4.).

The execution of the procedures rectb and rectc will fail if one of the components so[2:3,1] or s1[2:3,1] or s1[8,1] is destroyed. The only exception is s1[2,1]. Its value ±L may be replaced by $\varepsilon\tilde{L}$ if $|\varepsilon| = 1$ and $o \le \tilde{L} \le L$.

5.2 Procedure recta

The coefficients of Eq. (5') are defined by

$$so[1,k]:=-2(a+c)+h^2f;\quad so[2,k]:=a-\tfrac{h}{2}d;\quad so[4,k]:=a+\tfrac{h}{2}d;$$
$$so[3,k]:=c-\tfrac{h}{2}e;\quad so[5,k]:=c+\tfrac{h}{2}e$$

if we consider only interior grid points and only the case of b = o. Here a,...,f are the coefficients of Eq. (1). h denotes $(x_2-x_1)/n = (y_2-y_1)/m$.

If *so* is defined by a call of recta and if a new right-hand side g(x,y) of Eq. (1) is considered, a further call of recta redefines only the array fo. In the case of Dirichlet or periodic boundary data, the redefinition of fo can be achieved more simply by

```
for j:=1 step 1 until m2 do
begin y:=y1 + j*h; k:=1 + j*(n+1); x:=x1;
      for i:=1 step 1 until n2 do
      begin x:=x+h; k:=k+1; fo[k]:=g(x,y)*h*h
end  end  for h,n,m2,n2 compare Sections 5.1, 5.2;
```

By a similar loop new starting values may be assigned to uo. New Dirichlet boundary data can be assigned to uo by

```
np:=n+1; mnp:=m*np; for i:=1 step 1 until n-1 do
begin k:=1+i+j*n1; uo[k]:=boundary(x1+i*h,y1);
      uo[k+mnp]:=boundary(x1+i*h,y2)
end;
for j:=o step 1 until m do
begin uo[1+j*np]:=boundary(x1,y1+j*h);
      uo[1+m+j*np]:=boundary(x2,y1+j*h)
end;
```

5.3 Procedure rectb

By rectb the auxiliary matrices S_l ($1 \le l \le L$) are computed according to Eq.(8), $S_{l+1}=r_lS_lp_l$. The components of r_l are stored on r[1:9,1:1]. In the case of variable coefficients the computational work of the phase B is proportional to h^{-2}. We emphazise that this work can be reduced to $O(h^{-1})$ if the coefficients of Eq. (1) are constant (cf. [2]). During the performance of $S_{l+1}=r_lS_lp_l$ the variables n,m,n2,m2,np(=n+1),... correspond to the level l, whereas nn,mm,nn2,mm2,nnp,... are the respective values of the level l+1.

5.4 Procedure rectc

The variables n,m,...,nn,mm,... have the same meaning as those of rectb. The recursive iteration is programmed in an explicit manner.

The procedure *cor* performs the correction $u_l \leftarrow u_l - p_l u_{l+1}$. The irregular grid points are treated in *corloc*.

ib-ia+1 steps of the line-relaxation method of level l are applied to u_l if *sm* (=smoothing step) is called.

By a call of *defect* the restriction $f_{l+1}:=r_l(S_lu_l-f_l)$ of the defect $S_lu_l-f_l$ is computed and stored on f1.

The Gaussian elimination applied to $S_Lu_L=f_L$ is performed by the procedure *direct*. Since $nn:=(n*2^{-L}+1)*(m*2^{-L}+1)$ (= number of components of u_L) is assumed to be an small number, we use a very simple form of the algorithm. Therefore, a matrix of the size a[1:nn,1:nn] is declared. The user may replace this procedure by a more skilful program (e.g. reduction to a matrix with a small band width).

6. Discussion of Numerical Properties

Usually, i.e. in all uncritical cases, the rate of convergence of the iteration performed by rectc is about 1/30 - 1/15 (cf. Section 7 or [2,3]). The rate does not depend on the step size h or on the ratio $(x_2-x_1)/(y_2-y_1)$. Furthermore, the speed of convergence is insensitive to the kind of the boundary data and to the behavior of the coefficients of Eq. (1).

The rate of convergence becomes **bad** if

a) the ratio c(x,y)/a(x,y) is too large [cf. (3) or Section 3.1, kont=2],

b) $4b^2(x,y) / (a(x,y)\cdot c(x,y))$ approximates the value one from below,

c) the boundary value problem (1,2) is nearly singular, i.e., if the eigenvalue problem $Lu = \lambda u$ with homogeneous boundary data has a solution $\lambda \approx 0$ (for the operator L compare Section 2).

d) the convection term du_x+eu_y is very large (cf. Section 2).

In extreme cases the rate of convergence can exceeed the value one, and the method fails.

Furthermore, the speed of convergence depends on the regularity of the differential operator L (but not on the choice of the right-hand side g). An example with discontinuous coefficients is given in [3].

Note that for indefinite problems the parameter lm of rectb must be negative (cf. Sections 2, 7.2). Otherwise, the iteration diverges.

7. Test Results and Examples of the Use

The following examples were performed on the Cyber 72/76 of the Rechenzentrum der Universität zu Köln. Numerous further examples are reported in [2,3].

7.1 First Example

The equation

$$(11)\quad x\,u_{xx} + \frac{1}{x}u_{xy} + u_{yy} + (1+\sin(2\pi y))u_x + x\cdot\sin(2\pi y)\cdot u_y - \frac{\sin(2\pi y)}{x}u = 1 \qquad (1\le x\le 3,\ 0\le y\le 1)$$

has the solution u(x,y)=x if the following boundary data are chosen:

$$(12a)\quad u(1,y) = 1 \qquad (y\in[0,1],\ \text{i.e. } (x,y)\in\Gamma_1),$$

$$(12b)\quad u(x,0) = u(x,1),\quad u_y(x,0) = u_y(x,1) \qquad (x\in[1,3]),$$

$$(12c)\quad u_x(3,y) + u(3,y) = 4 \qquad (y\in[0,1],\ \text{i.e. } (x,y)\in\Gamma_3).$$

The problem (11,12) corresponds to the following procedure:

```
PROCEDURE first example(nr,k,x,y,c1,c2,c3,c4,c5,c6,c7);
VALUE nr,k,x,y; REAL x,y,c1,c2,c3,c4,c5,c6,c7; INTEGER nr,k;
BEGIN SWITCH part:=domain,kind,coeff,g1,end,g3,end,start;
      GOTO part[nr];
domain: c1:=c4:=1; c2:=0; c3:=3; c5:=9; GOTO end;
kind:   c1:=1; c2:=c4:=2; c3:=0; c5:=-1; GOTO end;
coeff:  c1:=x; c2:=1/x; c3:=c7:=1; c6:=sin(6.28318530718*y);
        c4:=1+c6; c5:=x*c6; c6:=-c6/x; GOTO end;
g1:     c1:=1; GOTO end;
g3:     c1:=1; c2:=4; GOTO end;
start:  c1:=1; COMMENT the labels g1 and start may be replaced by kind;
end:
END first example;
```

The program

```
BEGIN INTEGER kont;
ARRAY u0[1:45],f0[1:45],s0[1:9,1:45],u1,f1[1:21],s1[1:9,1:21];

PROCEDURE outp1(n,m,u,my,dif);
VALUE n,m,my; REAL dif; INTEGER n,m,my; ARRAY u;
BEGIN REAL d,x; INTEGER i,j;
      d:=0; FOR i:=0 STEP 1 UNTIL n DO
      BEGIN x:=1+2*i/n; FOR j:=0 STEP 1 UNTIL m DO
            d:=d+(u[1+j*(n+1)+i]-x)**2
      END;
      output(10,"("/"("my=")"zdb,"("error=")"+d.6d"+dd")",
            my,sqrt(d/((n+1)*(m+1))))
END outp1;

      recta(kont,u0,f0,s0,9,45,s1,8,"-12,first example);
      IF kont NOTEQUAL 0 THEN GOTO fail;
      rectb(kont,s1,21,s0,2,1);
      IF kont NOTEQUAL 0 THEN GOTO fail;
      rectc(kont,u0,f0,s0,u1,f1,s1,-"-8,8,outp1);
fail: output(10,"("/"("kont=")"-4zd")",kont)
END

my= 1 error=+1.073599e-01
my= 2 error=+7.108683e-03
my= 3 error=+4.580484e-04
my= 4 error=+2.905746e-05
my= 5 error=+1.836416e-06
my= 6 error=+1.159151e-07
my= 7 error=+7.314158e-09
my= 8 error=+4.617135e-10
kont=     0
```

yields the following table:

μ	1	2	3	4	5	6	7	8
$\|u^{(\mu)} - u_o\|_2$	$1.o74_{10}$-1	7.1_{10}-3	4.6_{10}-4	2.9_{10}-5	1.8_{10}-6	1.2_{10}-7	7.3_{10}-9	4.6_{10}-1o

Table 2: Errors of the first example ($u_o = x$)

The next tables contain the array $u^{(1)}$ and the auxiliary functions u_1 and u_2 obtained from $u^{(o)}=1$ (u_1 is printed out in the procedure cor before u_{l-1} is corrected by u_l). These numbers may be used for testing. The entries of the Tables 3-5 correspond to the grid points $(x,y) = (1+i\cdot 2^{l}/4, j\cdot 2^{l}/4)$ with $1 \le i \le 8\cdot 2^{-l}$, $1 \le j \le 4\cdot 2^{-l}$.

i \ j	1	2
1	1.14o16	1.27515

Table 3: array u_2

i \ j	1	2	3	4
1	-.589243	-1.1o657	-1.53962	-1.718o8
2	-.622577	-1.15936	-1.59651	-1.75546

Table 4: array u_1

i \ j	1	2	3	4	5	6	7	8
1	1.29o856	1.572366	1.848165	2.119668	2.387915	2.652149	2.9o48o3	3.13545o
2	1.288436	1.57oo67	1.845749	2.116256	2.381181	2.638219	2.882998	3.111981
3	1.286272	1.569o54	1.846o35	2.118683	2.3882o1	2.654161	2.9o8866	3.14o473
4	1.288741	1.569866	1.845463	2.115957	2.38o873	2.637926	2.88279o	3.111898

Table 5: array $u^{(1)}$ (result of one iteration)

The listed arrays appear during the calculation sketched below:

$$u^{(o)}=1 \xrightarrow[\text{defect}]{\text{sm,}} f_1 \; u_1:=o \xrightarrow{\text{sm,}\ \text{defect}} f_2 \; u_2:=o \xrightarrow{\text{sm}} u_2 \xrightarrow[\text{sm}]{\text{cor,}} u_1 \xrightarrow[\text{sm}]{\text{cor,}} u^{(1)}$$

level o level 1 level 2 level 1 level o

7.2 Second Example: Indefinite Problem

The problem

$$L\,u := u_{xx} + u_{yy} - \sin(x)\,u_x + \cos(x)\,u = \lambda u \qquad (o \le x,y \le \pi),$$
$$u(x,\,y) = o \quad \text{if } x \in \{o,\pi\} \text{ or } y \in \{o,\pi\}$$

has the eigenvalue $\lambda = o$. In spite of that we shall solve the indefinite boundary value problem

$$(13)\quad \begin{aligned} &u_{xx} + u_{yy} - \sin(x)\,u_x + \left[3 + \cos(x)\right] u = 1 \qquad (o \le x,y \le \pi),\\ &u(x,y) = o \quad \text{if } x \in \{o,\pi\} \text{ or } y \in \{o,\pi\}. \end{aligned}$$

Eq. (13) is decoded by

```
procedure second example(nr,k,x,y,c1,c2,c3,c4,c5,c6,c7);
value nr,k,x,y; real x,y,c1,c2,c3,c4,c5,c6,c7; integer nr,k;
begin switch part:=zero,one,coeff,zero,zero,zero,zero,zero;
goto part[nr];
zero: c1:=c2:=o; c3:=c4:=3.1415926535898; c5:=5; goto end;
one:  c1:=c2:=c3:=c4:=1; c5:=-1; goto end;
coeff: c1:=c3:=c7:=1; c2:=c5:=o; c4:=-sin(x); c6:=3+cos(x);
end:
end second example;
```

The program

```
begin integer kont;
    array uo[1:1250], fo[1:625], so[1:5,1:625],
          u1,f1[1:234], s1[1:9,1:234];
    procedure outp2(n,m,u,my,dif);
    value n,m,my; real dif; integer n,m,my; array u;
    print(my,dif);
    recta(kont,uo,fo,so,5,625,s1,24,7_10-15,second example);
    if kont ≠ o then goto fail;
    rectb(kont,s1,234,so,-3,3); if kont≠o then goto fail;
    rectc(kont,uo,fo,so,u1,f1,s1,_40-8,-8,outp2);
fail: print(kont,uo)
end
```

yields the following table:

μ	o	1	2	3	4	5	6	7	8
$\lVert u^{(\mu)} - u^{(\mu-1)} \rVert$	1.9_{10}-2	9.6_{10}-1	1.5_{10}-1	3.9_{10}-2	7.0_{10}-3	1.5_{10}-3	3.1_{10}-4	6.3_{10}-5	1.3_{10}-5

Table 6: indefinite problem (13)

7.3 Third Example: A Nonlinear Problem

The nonlinear example

$$(14)\quad \begin{aligned} &-u_{xx} - u_{yy} = e^{u} \qquad (o \le x,y \le 1),\\ &u(x,y) = o \qquad (x \in \{o,1\} \text{ or } y \in \{o,1\}) \end{aligned}$$

can be reduced to

$$(14')\quad \begin{aligned} &-u_{xx} - u_{yy} = e^{u} \qquad (o \le x,y \le \tfrac{1}{2}),\\ &u(x,y) = o \qquad (x=o \text{ or } y=o),\\ &u_x(\tfrac{1}{2},y) = o,\ u_y(x,\tfrac{1}{2}) = o \quad (o \le x,y \le \tfrac{1}{2}) \end{aligned}$$

by virtue of its symmetries. We apply Newton's iteration with the starting value $u^{(o)} = o$. The arising linear system is approximated by only one step of the multi-grid iteration. Table 7 is obtained by the following program.

```
begin integer kont,k,it; real v;
array u,vo,fo[1:289],so[1:5,1:289],v1,f1[1:119],s1[1:9,1:119];
procedure linearized problem (nr,k,x,y,c1,c2,c3,c4,c5,c6,c7);
value nr,k,x,y; real x,y,c1,c2,c3,c4,c5,c6,c7; integer nr,k;
begin switch part:=domain,kind,coeff,zero,zero,zero,zero,zero;
        goto part[nr];
domain: c3:=c4:=.5; c5:=5;
zero:   c1:=c2:=o; goto end;
kind:   c1:=c2:=1; c3:=c4:=o; c5:=-1; goto end;
coeff:  c1:=c3:=-1; c2:=c4:=c5:=o; c6:=-exp(u[k]);
        comment The four neighbouring points correspond to
                k+1, k-1, k+17, k-17, since n+1=m+1=17. If
                x=1/2, k+1 must be replaced by k-1. Analogously
                k+17 becomes k-17, if y=1/2. Note that the
                coefficients of so are +4 and -1 for interior
                grid points. c7 denotes the defect of u. The
                factor 1o24 corresponds to h^-2;
        c7:=c6+1o24*(4*u[k]-u[k-1]-u[k-17]
        -u[if x>.499 then k-1 else k+1]
        -u[if y>.499 then k-17 else k+17]);
end:
end linearized problem;
procedure outp3(n,m,u,my,dif); value n,m,my;
integer n,m,my; real dif; array u; ; comment empty statement;
```

```
for k:=1 step 1 until 289 do u[k]:=o;
for it:=1 step 1 until 8 do
begin recta(kont,vo,fo,so,5,289,s1,16,10-13,linearized problem);
      if kont≠o then goto fail;
      if it=1 then rectb(kont,s1,119,so,5,1);
      comment The auxiliary matrices are determined from u(o)=o;
      if kont>o then goto fail;
      rectc(kont,vo,fo,so,v1,f1,s1,o.o,1,outp3);
      comment The correction vo to u is approximated by
              only one step (itmax=1);
      if kont≠o then goto fail;
      v:=o; for k:=1 step 1 until 289 do
      begin v:=v+vo[k]↑2; u[k]:=u[k]-vo[k]
      end u(it-1) corrected by vo resulting in u(it);
      print(it,sqrt(v/289))
end 8 steps of Newton's iteration;
fail: print(kont,u[289])
end third example
```

it	1	2	3	4	5	6	7	8
$\|u^{(it)}-u^{(it-1)}\|$	$4.4_{10}-2$	$1.1_{10}-3$	$2.9_{10}-5$	$7.1_{10}-7$	$1.9_{10}-8$	$4.5_{10}-10$	$1.4_{10}-11$	$3.9_{10}-13$

Table 7: Convergence of the nonlinear problem (14')

The value of u[289] (i.e. $u(\frac{1}{2},\frac{1}{2})$) is o.o78o44o62956. For the computation of the second solution of (14) compare [3].

8. Equivalent FORTRAN Programs *

The FORTRAN IV subroutines correspondung to the ALGOL procedures recta, rectb, rectc are named RECTAF, RECTBF, RECTCF. They call the following subroutines:

RECTAF: MIX
RECTBF: IGM, LOC
RECTCF: IGM, CONST, SM, DEFECT, DEFLOC, COR, CORLOC, DIRECT.

The parameter list of RECTAF is the same as for the ALGOL procedure recta:

RECTAF(KONT,Uo,Fo,So,ISo,IUo,S1,N,EPS,PR)

RECTBF and RECTCF contain an additional parameter ISo:

RECTBF(KONT,S1,IU1,So,ISo,LM,MIN)
RECTCF(KONT,Uo,Fo,So,ISo,U1,F1,S1,EP,ITMAX,OUTP)

The meaning of ISo is the same as for RECTAF and recta.

```
      SUBROUTINE RECTAF (KONT,UO,FO,SO,ISO,IUO,S1,N,EPS,PR)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAA/ A,B,C,D,G,AP,BP,AM,BM,IS
      DIMENSION UO(IUO),FO(IUO),SO(ISO,IUO),S1(9,1)
      LOGICAL PX,PY
      KONT=1
      ZO=0.
      IF (N.LT.2.OR.MOD(N,2).NE.0) RETURN
      CALL PR(1,1,ZO,ZO,X1,Y1,X2,Y2,E,F,F)
      IS=E
      IF(ISO.LT.IS) RETURN
      IF (IS.NE.5) IS=9
      S1(8,1)=IS
      H=(X2-X1)/N
      EPSOO=10.*EPS
      S1(7,1)=0.
      CAMAX=0.
      IF (H.LE.0..OR.Y2.LE.Y1) RETURN
      M=(Y2-Y1)/H
      IF (M.LT.2.OR.MOD(M,2).NE.0.OR.ABS(H-(Y2-Y1)/M).GT.EPSOO) RETURN
      CALL PR(2,1,ZO,ZO,A,B,C,D,E,F,F)
      KONT=3
      IF (ABS(E).NE.1.) RETURN
      S1(3,1)=E
      N1=A
      PX=N1.EQ.2
      M1=B
      PY=M1.EQ.2
      N2=C
      M2=D
      IF (N1.LT.0.OR.M1.LT.0.OR.N2.LT.0.OR.M2.LT.0.OR.N1.GT.2.OR.
     1   M1.GT.2.OR.N2.GT.2.OR.M2.GT.2.OR.PX.AND.N2.NE.2.OR.
     2   .NOT.PX.AND.N2.EQ.2.OR.PY.AND.M2.NE.2.OR..NOT.PY.AND.M2.EQ.2
     3   .OR.E.EQ.1..AND.N1*M1*N2*M2.LE.0) RETURN
```

* compare footnote on page 16

```
      IF (PX) N1=0
      IF (PY) M1=0
      IF (PX) N2=0
      IF (PY) M2=0
      M2=M-M2
      N2=N-N2
      Y=Y1+H*(M1-1)
      NP=N+1
      JJ=M2*NP
      II=N2+1
      MP=M+1
      KONT=-NP*MP
      IF(IUO.LT.-KONT) RETURN
      HQ=H*H
      H2=H/2.
      H2HQ=X1+H*(N1-1)
      JJJ=M1*NP+1
      JJJJ=JJ+1
      III=N1+1
      DO 10 J=JJJ,JJJJ,NP
      Y=Y+H
      X=H2HQ
      DO 10 I=III,II
      K=I+J-1
      X=X+H
      CALL PR(3,K,X,Y,A,B,C,D,E,F,G)
      IF(C/A.GT.CAMAX) CAMAX=C/A
      D=D*H2
      E=E*H2
      IF (IS.EQ.9) GOTO 5
      B=0.
      GOTO 9
5     IF (A*B.GT.0.) GOTO 7
      B=B/2.
      SO(6,K)=0.
      SO(9,K)=0.
      SO(7,K)=-B
      SO(8,K)=-B
      A=A+B
      C=C+B
      GOTO 9
7     B=B/2.
      SO(6,K)=B
      SO(9,K)=B
      SO(7,K)=0.
      SO(8,K)=0.
      A=A-B
      C=C-B
9     SO(1,K)=F*HQ-2.*(A+B+C)
      SO(2,K)=A-D
      SO(3,K)=C-E
      SO(4,K)=A+D
      SO(5,K)=C+E
10    FO(K)=G*HQ
      KONT=0
      IF (CAMAX.GT.3.) KONT=2
      S1(7,1)=CAMAX
      Y=Y1
      K=0
```



```
      DO 30 JJ=1,MP
      X=X1
      DO 20 II=1,NP
      K=K+1
      CALL PR(8,K,X,Y,B,A,A,A,A,A,A)
      UO(K)=B
20    X=X+H
30    Y=Y+H
      AM=0.
      BM=0.
      AP=0.
      BP=0.
      G=2.*H
      H2HQ=H2/HQ
      IF (PX) GOTO 85
      K=1
      Y=Y1
      DO 80 JJJ=1,MP
      J=JJJ-1
      IF (N1.LE.0) GOTO 40
      CALL PR(4,K,X1,Y,B,A,A,A,A,A,A)
      UO(K)=B
      GOTO 50
40    IF (J.GT.M2.OR.J.LT.M1) GOTO 50
      CALL PR(4,K,X1,Y,A,B,C,C,C,C,C)
      IF (IS.EQ.5) GOTO 45
      IF (SO(8,K).NE.0.) CALL PR(4,K+NP,X1,Y+H,AP,BP,C,C,C,C,C)
      IF (SO(6,K).NE.0.) CALL PR(4,K-NP,X1,Y-H,AM,BM,C,C,C,C,C)
45    CALL MIX(SO(1,K),SO(2,K),SO(3,K),SO(4,K),SO(5,K),FO(K),
     1     SO(6,K),SO(7,K),SO(8,K),SO(9,K))
50    K=K+N
      IF (N2.GE.N) GOTO 60
      CALL PR(6,K,X2,Y,B,A,A,A,A,A,A)
      UO(K)=B
      GOTO 70
60    IF (J.LT.M1.OR.J.GT.M2) GOTO 70
      CALL PR(6,K,X2,Y,A,B,C,C,C,C,C)
      IF (IS.EQ.5) GOTO 65
      IF (SO(9,K).NE.0.) CALL PR(6,K+NP,X2,Y+H,AP,BP,C,C,C,C,C)
      IF (SO(7,K).NE.0.) CALL PR(6,K-NP,X2,Y-H,AM,BM,C,C,C,C,C)
65    CALL MIX(SO(1,K),SO(4,K),SO(3,K),SO(2,K),SO(5,K),FO(K),
     1     SO(7,K),SO(6,K),SO(9,K),SO(8,K))
70    Y=Y+H
80    K=K+1
85    IF (PY) GOTO 135
      X=X1
      K=1
      MNP=M*NP
      J=MNP-1
      DO 130 II=1,NP
      I=II-1
      IF (M1.LE.0) GOTO 90
      CALL PR(5,K,X,Y1,B,A,A,A,A,A,A)
      UO(K)=B
      GOTO 100
90    IF (I.LT.N1.OR.I.GT.N2) GOTO 100
      CALL PR(5,K,X,Y1,A,B,C,C,C,C,C)
      IF (IS.EQ.5) GOTO 95
      IF (SO(7,K).NE.0.) CALL PR(5,K+1,X+H,Y1,AP,BP,C,C,C,C,C)
      IF (SO(6,K).NE.0.) CALL PR(5,K-1,X-H,Y1,AM,BM,C,C,C,C,C)
95    CALL MIX(SO(1,K),SO(3,K),SO(2,K),SO(5,K),SO(4,K),FO(K),
     1          SO(6,K),SO(8,K),SO(7,K),SO(9,K))
```

```
100   K=K+MNP
      IF (M2.GE.M) GOTO 110
      CALL PR(7,K,X,Y2,B,A,A,A,A,A,A)
      UO(K)=B
      GOTO 120
110   IF (I.LT.N1.OR.I.GT.N2) GOTO 120
      CALL PR(7,K,X,Y2,A,B,C,C,C,C,C)
      IF (IS.EQ.5) GOTO 115
      IF (SO(9,K).NE.0.) CALL PR(7,K+1,X+H,Y2,AP,BP,C,C,C,C,C)
      IF (SO(6,K).NE.0.) CALL PR(7,K-1,X-H,Y2,AM,BM,C,C,C,C,C)
115   CALL MIX(SO(1,K),SO(5,K),SO(2,K),SO(3,K),SO(4,K),FO(K),
     1         SO(8,K),SO(6,K),SO(9,K),SO(7,K))
120   X=X+H
130   K=K-J
135   K=1
      IF (.NOT.PX) GOTO 138
      K=2
      N1=1
      I=1
      DO 136 J=1,MP
      SO(1,I)=0.
      SO(5,I)=0.
      SO(3,I)=0.
      FO(I)=0.
136   I=I+NP
138   IF (.NOT.PY) GOTO 145
      K=K+2
      M1=1
      DO 142 I=1,NP
      SO(1,I)=0.
      SO(2,I)=0.
      FO(I)=0.
142   SO(4,I)=0.
145   S1(3,1)=S1(3,1)*K
      SO(2,1)=N2
      IF (N1.NE.0) SO(2,1)=-N2
      SO(3,1)=M2
      IF (M1.NE.0) SO(3,1)=-M2
      IF (S1(3,1).LE.0.) RETURN
      D=0.
      JJJ=M1+1
      II=N1+1
      K=M1*NP
      DO 150 J=JJJ,MP
      K=K+N1
      DO 150 I=II,NP
      K=K+1
150   D=D+FO(K)
      D=D/(N*M)
      S1(6,1)=D
      K=0
      DO 160 J=1,MP
      E=D
      IF (J.EQ.1.OR.J.EQ.MP.AND..NOT.PY) E=D*.5
      DO 160 I=1,NP
      F=E
      IF (I.EQ.1.OR.I.EQ.NP.AND..NOT.PX) F=F*.5
      K=K+1
160   FO(K)=FO(K)-F
      RETURN
      END
```



```
      SUBROUTINE MIX(S1,S2,S3,S4,S5,FF,S6,S7,S8,S9)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAA/ A,B,C,D,G,AP,BP,AM,BM,IS
      C=G*S2
      S4=(S4+S2)/2.
      S1=(S1-C*A)/2.
      FF=FF-C*B
      IF (IS.EQ.5) GOTO 10
      S9=(S9+S8)/2.
      C=G*S8
      S5=S5-C*AP
      S8=0.
      S7=(S7+S6)/2.
      D=G*S6
      S3=S3-D*AM
      FF=FF-D*BM-C*BP
      S6=0.
10    S5=S5/2.
      S3=S3/2.
      FF=FF/2.
      RETURN
      END


      SUBROUTINE LOC(A1,A2,S,ISO,IS,IST)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      LOGICAL DEFECT,REG,B
      DIMENSION A1(5,5),A2(5,5),S(ISO,1)
      COMMON /VAB/ N2,M2,N,M,NP,KPI,II,JJ,N11,M11,IX(9),IY(9)
      B=IS.EQ.9
      DEFECT=IST.EQ.1
      REG=II.GT.0.AND.II.LT.N.AND.JJ.GT.0.AND.JJ.LT.M
      IF (.NOT.DEFECT) GOTO 10
      L1=N11-II
      L2=N2-II
      L3=M11-JJ
      L4=M2-JJ
      IF (2.LT.-L1) L1=-2
      IF (2.LT.L2) L2=2
      IF (-2.GT.L3) L3=-2
      IF (2.LT.L4) L4=2
      LLL1=3+MAX0(L1,-1)
      LLL3=3+MAX0(L3,-1)
      LLL2=3+MIN0(L2,1)
      LLL4=3+MIN0(L4,1)
      GOTO 20
10    LLL1=LL1
      LLL2=LL2
      LLL3=LL3
      LLL4=LL4
      KP=1
      L1=L1+IABS(MOD(L1,2))
      L3=L3+IABS(MOD(L3,2))
20    LL1=L1+3
      LL2=L2+3
      LL3=L3+3
      LL4=L4+3
```

```
      DO 80 JJ1=LL3,LL4,IST
      J1=JJ1-3
      DO 80 II1=LL1,LL2,IST
      I1=II1-3
      D=0.
      IF (DEFECT) KP=KPI+J1*NP+I1
      IF (REG.AND.IABS(J1).LE.1.AND.IABS(I1).LE.1) GOTO 60
      DO 50 K=1,IS
      I2=II1+IX(K)
      J2=JJ1+IY(K)
      IF (LLL1.LE.I2.AND.LLL2.GE.I2.AND.LLL3.LE.J2.AND.LLL4.GE.J2)
     1   D=D+S(K,KP)*A2(I2,J2)
50    CONTINUE
      GOTO 80
60    D=S(1,KP)*A2(II1,JJ1)+S(2,KP)*A2(II1-1,JJ1)+S(3,KP)*A2(II1,JJ1-1)
     1   +S(4,KP)*A2(II1+1,JJ1)+S(5,KP)*A2(II1,JJ1+1)
      IF (B) D=D+S(6,KP)
     2   *A2(II1-1,JJ1-1)+S(7,KP)*A2(II1+1,JJ1-1)
     3   +S(8,KP)*A2(II1-1,JJ1+1)+S(9,KP)*A2(II1+1,JJ1+1)
80    A1(II1,JJ1)=D
      RETURN
      END



      SUBROUTINE RECTBF (KONT,S1,IU1,S0,IS0,LM,MINIM)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAB/ N2,M2,N,M,NP,KPI,II,JJ,N11,M11,IX(9),IY(9)
      DIMENSION A1(5,5),A2(5,5),R(9),NDUM(5,1)
      DIMENSION S0(IS0,1),S1(9,IU1)
      LOGICAL PX,PY
      DATA R /1.,4*.5,4*.25/
      IS=S1(8,1)
      KONT=0
      CALL IGM(NDUM,IX,IY,IX,NN,N1,NN2,MM,M1,MM2,PX,PY,-1,
     1   S0(2,1),S0(3,1),S1(3,1))
      LMAX=IABS(LM)
      IF (LMAX.LE.9) GOTO 5
      KONT=5
      LMAX=9
5     MINI=MINIM
      IF (MINI.GT.0) GOTO 6
      MINI=1
      KONT=5
6     INDL=1
      LMAX1=LMAX+1
      IF (MINI.EQ.1.AND.(NN-NN2+N1.EQ.2.OR.MM-MM2+M1.EQ.2))MINI=2
      N11=N1
      IF (PX) N11=0
      M11=M1
      IF (PY) M11=0
      DO 300 L=2,LMAX1
      N=NN
      NP=N+1
      M=MM
      N2=NN2
      M2=MM2
      IND=INDL
      IF (L.NE.2) INDL=IND+NP*(M+1)
      NN=N/2
      MM=M/2
      IF (2*NN.NE.N.OR.2*MM.NE.M.OR.NN.LT.MINI.OR.MM.LT.MINI) GOTO 100
```



```
      NPP=NN+1
      NN2=NN+N2-N
      MM2=MM+M2-M
      J=INDL+NPP*MM+NN
      IF(IU1.GE.J) GOTO 7
      KONT=-J
      RETURN
7     MPP=MM+1
      DO 280 JJJ=1,MPP
      J=JJJ-1
      JJ=2*J
      KL=INDL+J*NPP-1
      KPI=IND+JJ*NP-2
      DO 280 III=1,NPP
      I=III-1
      II=2*I
      KL=KL+1
      KPI=KPI+2
      A1(3,3)=1.
      A1(1,1)=0.
      A1(1,2)=0.
      A1(1,3)=0.
      A1(1,4)=0.
      A1(1,5)=0.
      A1(2,1)=0.
      A1(2,5)=0.
      A1(3,1)=0.
      A1(3,5)=0.
      A1(4,1)=0.
      A1(4,5)=0.
      A1(5,1)=0.
      A1(5,2)=0.
      A1(5,3)=0.
      A1(5,5)=0.
      A1(5,4)=0.
      A1(3,4)=.5
      A1(4,3)=.5
      A1(2,3)=.5
      A1(3,2)=.5
      A1(2,2)=.25
      A1(4,2)=.25
      A1(2,4)=.25
      A1(4,4)=.25
      IF (L.EQ.2) GOTO 240
      CALL LOC(A2,A1,S1,9,9,1)
      GOTO 250
240   CALL LOC(A2,A1,S0,IS0,IS,1)
250   CALL LOC(A1,A2,R,9,9,2)
      L1=KL-NPP
      L2=KL+NPP
      IF (I.LE.N11) GOTO 260
      IF (J.GT.M11) S1(9,L1-1)=A1(1,1)
      S1(4,KL-1)=A1(1,3)
      IF (J.LT.MM2) S1(7,L2-1)=A1(1,5)
260   IF (J.GT.M11) S1(5,L1)=A1(3,1)
      S1(1,KL)=A1(3,3)
      IF (J.LT.MM2) S1(3,L2)=A1(3,5)
      IF (I.GE.NN2) GOTO 280
      IF (J.GT.M11) S1(8,L1+1)=A1(5,1)
      S1(2,KL+1)=A1(5,3)
      IF (J.LT.MM2) S1(6,L2+1)=A1(5,5)
280   CONTINUE
      IF (.NOT.PY) GOTO 286
      KL=INDL+N11
      KP=KL+MM*NPP
      II=N11+1
      III=NN2+1
      DO 285 I=II,III
      S1(5,KP)=S1(5,KL)
      S1(8,KP)=S1(8,KL)
      S1(1,KP)=S1(1,KP)+S1(1,KL)
      S1(1,KL)=0.
      IF (I.EQ.1) GOTO 284
      S1(2,KP)=S1(2,KP)+S1(2,KL)
      S1(2,KL)=0.
284   IF (I.EQ.NPP) GOTO 2841
      S1(4,KP)=S1(4,KP)+S1(4,KL)
      S1(4,KL)=0.
2841  S1(9,KP)=S1(9,KL)
      KL=KL+1
285   KP=KP+1
286   IF (.NOT.PX) GOTO 288
      KL=INDL+M11*NPP
      KP=KL+NN
      JJ=M11+1
      JJJ=MM2+1
      DO 287 J=JJ,JJJ
      S1(4,KP)=S1(4,KL)
      S1(7,KP)=S1(7,KL)
      S1(1,KP)=S1(1,KP)+S1(1,KL)
      S1(1,KL)=0.
      IF (J.EQ.1) GOTO 2861
      S1(3,KP)=S1(3,KP)+S1(3,KL)
      S1(3,KL)=0.
2861  IF (J.EQ.MPP) GOTO 2862
      S1(5,KP)=S1(5,KP)+S1(5,KL)
      S1(5,KL)=0.
2862  S1(9,KP)=S1(9,KL)
      KL=KL+NPP
287   KP=KP+NPP
288   IF (L.EQ.LMAX1) GOTO 200
300   CONTINUE
100   LMAX1=L-1
200   LMAX=LMAX1-1
      S1(2,1)=-LMAX
      IF (LM.GT.0) S1(2,1)=LMAX
      RETURN
      END
```

```
      SUBROUTINE IGM(NM,IX,IY,IZ,N,N1,N2,M,M1,M2,PX,PY,LM,
     1   S21,S31,S23)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      LOGICAL PX,PY
      DIMENSION NM(5,1),IX(9),IY(9),IZ(9)
      L=S23
      L=IABS(L)
      PY=L.GT.2
      IF (PY) L=L-2
      PX=L.EQ.2
      N2=S21
      M2=S31
      N1=0
      M1=0
      IF (N2.GE.0) GOTO 10
      N1=1
      N2=-N2
10    IF (M2.GE.0) GOTO 20
      M1=1
      M2=-M2
20    N=N2+MOD(N2,2)
      M=M2+MOD(M2,2)
      IF (LM.LT.0) GOTO 40
      NM(1,1)=N
      NM(2,1)=N2
      NM(3,1)=M
      NM(4,1)=M2
      NM(5,1)=1
      IF (LM.EQ.0) GOTO 40
      NM(5,2)=1
      DO 30 LL=1,LM
      L=LL+1
      NM(1,L)=NM(1,LL)/2
      NM(2,L)=NM(1,L)+N2-N
      NM(3,L)=NM(3,LL)/2
      NM(4,L)=NM(3,L)+M2-M
30    NM(5,L+1)=NM(5,L)+(NM(1,L)+1)*(NM(3,L)+1)
40    IX(1)=0
      IX(3)=0
      IX(5)=0
      IY(1)=0
      IY(2)=0
      IY(4)=0
      IZ(1)=0
      IX(2)=-1
      IX(6)=-1
      IX(8)=-1
      IY(6)=-1
      IY(3)=-1
      IY(7)=-1
      IZ(2)=-1
      IX(4)=1
      IX(7)=1
      IX(9)=1
      IY(5)=1
      IY(8)=1
      IY(9)=1
      IZ(4)=1
      RETURN
      END
```



```
      SUBROUTINE RECTCF (KONT,UO,FO,SO,ISO,U1,F1,S1,EP,ITMAXI,OUTP)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      DIMENSION UO(1),FO(1),SO(ISO,2),U1(1),F1(1),S1(9,1),R(9)
      DIMENSION NM(5,11)
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      DATA R /1.,4*.5,4*.25/
      LMAX=S1(2,1)
      GAUSS=LMAX.LE.0
      LMAX=IABS(LMAX)+1
      EPS=EP
      KONT=0
      CALL IGM(NM,IX,IY,IZ,N,N1,N2,M,M1,M2,PX,PY,LMAX-1,
     1   SO(2,1),SO(3,1),S1(3,1))
      NORM=S1(3,1).GT.0.
      IS=S1(8,1)
      NPX=.NOT.PX
      NPY=.NOT.PY
      ITMAX=ITMAXI
      NMM=(N+1)*(M+1)
      IF (EPS.LE.0.) GOTO 20
      DO 10 I=1,NMM
10    UO(I+NMM)=UO(I)
20    L=0
      IT=0
      IF (ITMAX.LT.0) IT=-1
      ITMAX=IABS(ITMAX)
      N11=N1
      IF (PX) N11=0
      M11=M1
      IF (PY) M11=0
30    L=L+1
      CALL CONST(IS,NM)
      IF (L.NE.1) GOTO 50
40    IF (IT.GE.ITMAX) RETURN
      IT=IT+1
      IF (IT.EQ.0) GOTO 100
      GOTO 70
50    J=INDL-1
      DO 60 I=IND,J
60    U1(I)=0.
70    IF (L.EQ.LMAX) GOTO 100
      IF (L.GT.1) GOTO 80
      CALL SM(3,3,UO,FO,SO,ISO,IS,KONT)
      IF (KONT.GT.0) RETURN
      CALL DEFECT(UO,FO,SO,ISO,F1,R,IS)
      GOTO 30
80    CALL SM(3,3,U1,F1,S1,9,9,KONT)
      IF (KONT.GT.0) RETURN
      CALL DEFECT(U1,F1,S1,9,F1,R,9)
      GOTO 30
100   IF (L.GT.1) GOTO 120
      IF (LMAX.EQ.1.AND.GAUSS) GOTO 125
      CALL SM(2,4,UO,FO,SO,ISO,KONT)
      IF (KONT.GT.0) RETURN
      B1=0.
      IF (EPS.LE.0.) GOTO 115
```

```
      DO 110 I=1,NMM
      B5=UO(I)
      B3=UO(I+NMM)-B5
      B1=B1+B3*B3
110   UO(I+NMM)=B5
      B1=SQRT(B1/NMM)
115   CALL OUTP(N,M,UO,IT,B1)
      IF (B1.LE.EPS.AND.EPS.GT.0..AND.IT.GT.0) RETURN
      GOTO 140
120   IF (L.EQ.LMAX.AND.GAUSS) GOTO 130
      CALL SM(2,5,U1,F1,S1,9,9,KONT)
      IF (KONT.GT.0) RETURN
      GOTO 140
125   CALL DIRECT(KONT,UO,FO,SO,ISO,IS)
      RETURN
130   CALL DIRECT(KONT,U1,F1,S1,9,9)
      IF (KONT.GT.0) RETURN
140   IF (L.EQ.1) GOTO 40
      L=L-1
      CALL CONST(IS,NM)
      IF (L.GT.1) GOTO 150
      CALL COR2(UO,U1)
      GOTO 100
150   CALL COR2(U1,U1)
      GOTO 100
      END
```

```
      SUBROUTINE CORLOC(U,U1,I,J)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION U(1),U1(1)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      KPI=IND+J*NP+I
      KL=INDL+(J*NPP+I)/2
      B2=U1(KL)*.5
      IF (J.GT.M11) U(KPI-NP)=U(KPI-NP)-B2
      IF (J.LT.M2) U(KPI+NP)=U(KPI+NP)-B2
      B2=B2*.5
      DO 10 K=6,9
      I1=I+IX(K)
      IF (I1.LT.N11.OR.I1.GT.N2) GOTO 10
      J1=J+IY(K)
      IF (J1.LT.M11.OR.J1.GT.M2) GOTO 10
      KP=KPI+IZ(K)
      U(KP)=U(KP)-B2
10    CONTINUE
      RETURN
      END
```



```
      SUBROUTINE DEFLOC(U,F,S,ISO,F1,R,IS,I,J)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION U(1),F(1),S(ISO,1),F1(1),R(9)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      II=2*I
      JJ=2*J
      B1=0.
      KPI=IND+JJ*NP+II
      DO 20 K=1,4
      IF (K.EQ.3.AND.II.GT.0.AND.II.LT.N) GOTO 20
      I1=II+IX(K)
      IF (I1.LT.0.OR.I1.GT.N) GOTO 20
      J1=JJ+IY(K)
      IF (J1.LT.0.OR.J1.GT.M) GOTO 20
      KP=KPI+IZ(K)
      B2=-F(KP)
      DO 10 KL=1,IS
      I2=I1+IX(KL)
      J2=J1+IY(KL)
      IF (I2.LT.0.OR.J2.LT.0.OR.I2.GT.N.OR.J2.GT.M) GOTO 10
      KR=KP+IZ(KL)
      B2=B2+S(KL,KP)*U(KR)
10    CONTINUE
      B1=B1+B2*R(K)
20    CONTINUE
      KR=INDL+J*NPP+I
      F1(KR)=B1
      RETURN
      END
```

```
      SUBROUTINE DEFECT(U,F,S,ISO,F1,R,IS)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION U(1),F(1),S(ISO,1),F1(1),R(9)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      MMNPP=MM*NPP
      IF (MMM1.LE.0.OR.NNM1.LE.0) GOTO 15
      DO 10 J=1,MMM1
      KPI=INDL+J*NPP
      K=IND+J*NP2
      DO 10 I=1,NNM1
      K=K+2
      KR=K+1
      KL=K-1
      KP=K+NP
      KS=K-NP
```

```
      IF (B) GOTO 5
      F1(KPI+I)=(U(K-I1M1)*S(5,KL)+U(K+I1J1)*S(5,KR)+U(K+I1M1)*S(3,KR)
     1    +U(K-I1J1)*S(3,KL)+U(KL)*(S(1,KL)+2.*S(2,K))+U(K-2)*S(2,KL)
     2    +U(KR)*(S(1,KR)+2.*S(4,K))+U(K+2)*S(4,KR)-F(KL)-F(KR)
     3    +U(K)*(2.*S(1,K)+S(4,KL)+S(2,KR)))*.5-F(K)+U(KP)*S(5,K)
     4    +U(KS)*S(3,K)
      GOTO 10
5     F1(KPI+I)=(U(K-I2M1)*S(8,KL)+U(K-I1M1)*(S(5,KL)+2.*S(8,K))+
     1    U(KP)*(2.*S(5,K)+S(9,KL)+S(8,KR))+U(K+I1J1)*(2.*S(9,K)+S(5,KR)
     2    )+U(K+I2J1)*S(9,KR)+U(K-2)*S(2,KL)+U(KL)*(2.*S(2,K)+S(1,KL))
     3    +U(K)*(2.*S(1,K)+S(4,KL)+S(2,KR))+U(KR)*(2.*S(4,K)+S(1,KR))
     4    +U(K+2)*S(4,KR)+U(K-I2J1)*S(6,KL)+U(K-I1J1)*(2.*S(6,K)+S(3,KL)
     5    )+U(KS)*(2.*S(3,K)+S(7,KL)+S(6,KR))+U(K+I1M1)*(2.*S(7,K)+S(3,
     6    KR))+U(K+I2M1)*S(7,KR)-F(KL)-F(KR))*.5-F(K)
10    CONTINUE
15    I=0
      IF (N11.GT.0.OR.MM2.EQ.0)GOTO 30
      DO 20 J=1,MM2
20    CALL DEFLOC(U,F,S,ISO,F1,R,IS,I,J)
30    J=MM
      IF (M2.LT.M.OR.NN2.EQ.0) GOTO 50
      DO 40 I=1,NN2
40    CALL DEFLOC(U,F,S,ISO,F1,R,IS,I,J)
50    I=NN
      IF (N.GT.N2.OR.MMM1.LT.M11) GOTO 70
      III=M11+1
      DO 60 JJJ=III,MM
      J=JJJ-1
60    CALL DEFLOC(U,F,S,ISO,F1,R,IS,I,J)
70    J=0
      IF (M11.GT.0.OR.NNM1.LT.N11) GOTO 90
      JJJ=N11+1
      DO 80 III=JJJ,NN
      I=III-1
80    CALL DEFLOC(U,F,S,ISO,F1,R,IS,I,J)
90    IF (NPX) GOTO 110
      II=M11+1
      JJ=MM2+1
      K=INDL+M11*NPP+NN
      DO 100 JJJ=II,JJ
      F1(K)=F1(K)+F1(K-NN)
      F1(K-NN)=0.
100   K=K+NPP
110   IF (NPY) RETURN
      JJ=N11+1
      II=NN2+1
      K=INDL+N11+MMNPP
      DO 120 III=JJ,II
      F1(K)=F1(K)+F1(K-MMNPP)
      F1(K-MMNPP)=0.
120   K=K+1
      RETURN
      END
```



```
      SUBROUTINE COR2(U,U1)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION U(1),U1(1)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      KP=INDL+NPP*(MM+1) -1
      IF (MMM1.LE.0.OR.NNM1.LE.0) GOTO 15
      DO 10 J=1,MMM1
      KP=INDL+J*NPP
      K=IND+J*NP2
      DO 10 I=1,NNM1
      KP=KP+1
      K=K+2
      B2=U1(KP)*.5
      B3=B2*.5
      U(K+NP)=U(K+NP)-B2
      U(K-NP)=U(K-NP)-B2
      U(K-I1J1)=U(K-I1J1)-B3
      U(K+I1J1)=U(K+I1J1)-B3
      U(K+I1M1)=U(K+I1M1)-B3
10    U(K-I1M1)=U(K-I1M1)-B3
15    I=0
      IF (N11.GT.0.OR.M2.LT.2) GOTO 30
      DO 20 J=2,M2,2
20    CALL CORLOC(U,U1,I,J)
30    J=M
      IF (M2.LT.M.OR.N2.LT.2) GOTO 50
      DO 40 I=2,N2,2
40    CALL CORLOC(U,U1,I,J)
50    I=N
      IF (N.GT.N2) GOTO 70
      II=2*M11+1
      KS=MM1+1
      IF (KS.LT.II) GOTO 70
      DO 60 JJ=II,KS,2
      J=JJ-1
60    CALL CORLOC(U,U1,I,J)
70    J=0
      IF (M11.GT.0) RETURN
      JJ=2*N11+1
      IF (N.LT.JJ) RETURN
      DO 80 II=JJ,N,2
      I=II-1
80    CALL CORLOC(U,U1,I,J)
      RETURN
      END


      SUBROUTINE SM (IA,IB,U,F,S,ISO,IS,KONT)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION U(1),F(1),S(ISO,1)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      DIMENSION A(129), E(129), C(129)
      IF (NP.LE.129) GOTO 5
      KONT=21
      RETURN
C     CHANGE THE LENGTH OF A,C,E IF NECESSARY
```

```
 5    E(1)=1.
      E(NP)=1.
      A(1)=0.
      C(NP)=0.
      A(NP)=0.
      DO 350 II=IA,IB
      J1=MOD(II,2)+1
      IF (J1.EQ.1) J1=2*M1+1
      KS=M2+1
      IF (J1.GT.KS) GOTO 350
      DO 300 JJ=J1,KS,2
      J=JJ-1
      KP=IND+J*NP
      IF (J.EQ.M) GOTO 80
      IF (J.GT.0) GOTO 40
      IF (N1.EQ.1) GOTO 10
      E(1)=S(1,KP)
      A(1)=S(4,KP)
      B1=F(KP)-S(5,KP)*U(KP+NP)
      IF (B) B1=B1-S(9,KP)*U(KP+I1J1)
      U(KP)=B1
10    IF (N2.LT.N) GOTO 20
      KPI=KP+N
      E(NP)=S(1,KPI)
      C(NP)=S(2,KPI)
      B1=F(KPI)-S(5,KPI)*U(KPI+NP)
      IF (NB) GOTO 15
      B1=B1-S(8,KPI)*U(KPI-I1M1)
      IF (NPX) GOTO 15
      B1=B1-S(9,KPI)*U(KPI+I1J1)
      A(NP)=S(4,KPI)
15    U(KPI)=B1
20    IF (NM1.LE.0) GOTO 200
      DO 30 I=1,NM1
      KPI=KP+I
      I1=I+1
      E(I1)=S(1,KPI)
      C(I1)=S(2,KPI)
      A(I1)=S(4,KPI)
      B1=F(KPI)-S(5,KPI)*U(KPI+NP)
      IF (B) B1=B1-S(8,KPI)*U(KPI-I1M1)-S(9,KPI)*U(KPI+I1J1)
30    U(KPI)=B1
      GOTO 200
40    IF (N1.EQ.1) GOTO 50
      E(1)=S(1,KP)
      A(1)=S(4,KP)
      B1=F(KP)-S(3,KP)*U(KP-NP)-S(5,KP)*U(KP+NP)
      IF (B) B1=B1-S(7,KP)*U(KP+I1M1)-S(9,KP)*U(KP+I1J1)
      U(KP)=B1
50    IF (N2.LT.N) GOTO 60
      KPI=KP+N
      E(NP)=S(1,KPI)
      C(NP)=S(2,KPI)
      B1=F(KPI)-S(3,KPI)*U(KPI-NP)-S(5,KPI)*U(KPI+NP)
      IF (PX) A(NP)=S(4,KPI)
      IF (NB) GOTO 55
      B1=B1-S(6,KPI)*U(KPI-I1J1)-S(8,KPI)*U(KPI-I1M1)
      IF (NPX) GOTO 55
      B1=B1-S(9,KPI)*U(KP+I1J1)-S(7,KPI)*U(KP+I1M1)
55    U(KPI)=B1
```



```
60    IF (NM1.LE.0) GOTO 200
      DO 70 I=1,NM1
      KPI=KP+I
      I1=I+1
      E(I1)=S(1,KPI)
      C(I1)=S(2,KPI)
      A(I1)=S(4,KPI)
      B1=F(KPI)-S(3,KPI)*U(KPI-NP)-S(5,KPI)*U(KPI+NP)
      IF (B) B1=B1-S(6,KPI)*U(KPI-I1J1)-S(7,KPI)*U(KPI+I1M1)
     1   -S(8,KPI)*U(KPI-I1M1)-S(9,KPI)*U(KPI+I1J1)
70    U(KPI)=B1
      GOTO 200
80    IF (PY.AND.M.EQ.1) MNP=MNP+NP
      IF (N1.EQ.1) GOTO 90
      E(1)=S(1,KP)
      A(1)=S(4,KP)
      B1=F(KP)-S(3,KP)*U(KP-NP)
      IF (B) B1=B1-S(7,KP)*U(KP+I1M1)
      IF (NPY) GOTO 85
      K=KP-MNP
      B1=B1-S(5,KP)*U(K+NP)
      IF (B) B1=B1-S(9,KP)*U(K+I1J1)
85    U(KP)=B1
90    IF (N2.LT.N) GOTO 100
      KPI=KP+N
      E(NP)=S(1,KPI)
      C(NP)=S(2,KPI)
      B1=F(KPI)-S(3,KPI)*U(KPI-NP)
      K=KPI-MNP
      IF(PX) A(NP)=S(4,KPI)
      IF (PY) B1=B1-S(5,KPI)*U(K+NP)
      IF (NB) GOTO 95
      B1=B1-S(6,KPI)*U(KPI-I1J1)
      IF (PY) B1=B1-S(8,KPI)*U(K-I1M1)
      IF (NPX) GOTO 95
      KL=K-N
      IF (PY) B1=B1-S(9,KPI)*U(KL+I1J1)
      B1=B1-S(7,KPI)*U(KP+I1M1)
95    U(KPI)=B1
100   IF (NM1.LE.0) GOTO 120
      DO 110 I=1,NM1
      KPI=KP+I
      I1=I+1
      E(I1)=S(1,KPI)
      C(I1)=S(2,KPI)
      A(I1)=S(4,KPI)
      B1=F(KPI)-S(3,KPI)*U(KPI-NP)
      IF (B) B1=B1-S(6,KPI)*U(KPI-I1J1)-S(7,KPI)*U(KPI+I1M1)
      IF (NPY) GOTO 110
      K=KPI-MNP
      B1=B1-S(5,KPI)*U(K+NP)
      IF (B) B1=B1-S(8,KPI)*U(K-I1M1)-S(9,KPI)*U(K+I1J1)
110   U(KPI)=B1
120   MNP=M*NP
200   IF (PX) GOTO 230
      IF (E(1).EQ.0.) GOTO 400
      B1=U(KP)/E(1)
      U(KP)=B1
```

```
      B2=A(1)/E(1)
      A(1)=B2
      KP=KP-1
      DO 210I=2,NP
      KPI=KP+I
      B3=E(I)-C(I)*B2
      IF (B3.EQ.0.) GOTO 400
      B1=(U(KPI)-C(I)*B1)/B3
      U(KPI)=B1
      B2=A(I)/B3
      A(I)=B2
210   CONTINUE
      KPI=KP+NP
      I=NP
      B1=U(KPI)
      DO 220 I1=1,N
      I=I-1
      KPI=KPI-1
      B1=U(KPI)-A(I)*B1
220   U(KPI)=B1
      GOTO 300
230   KPI=KP-1
      IF (N.GT.1) GOTO 240
      B2=E(2)+A(2)+C(2)
      IF (B2.EQ.0.) GOTO 400
      B5=U(KPI+2)/B2
      U(KPI+2)=B5
      GOTO 290
240   B3=1.
      B1=0.
      B5=0.
      DO 250 I=2,N
      B4=C(I)
      B2=E(I)-B4*B1
      IF (B2.EQ.0.) GOTO 400
      B1=A(I)/B2
      A(I)=B1
      B5=(U(KPI+I)-B4*B5)/B2
      U(KPI+I)=B5
      B3=-B3*B4/B2
      C(I)=B3
250   CONTINUE
      B3=B3+B1
      C(N)=B3
      IF (N.EQ.2) GOTO 270
      DO 260 I1=2,NM1
      I=NP-I1
      B1=A(I)
      B3=C(I)-B1*B3
      C(I)=B3
      B5=U(KPI+I)-B1*B5
      U(KPI+I)=B5
260   CONTINUE
270   B4=C(NP)
      B1=A(NP)
      B2=E(NP)-B4*C(N)-B1*B3
      IF (B2.EQ.0.) GOTO 400
      B5=(U(KPI+NP)-B4*U(KPI+N)-B1*B5)/B2
      U(KPI+NP)=B5
      DO 280 I=2,N
280   U(KPI+I)=U(KPI+I)-C(I)*B5
290   U(KP)=B5
300   CONTINUE
      IF (NPY) GOTO 350
      K=IND+N2
      KL=IND+N11
      DO 340 J=KL,K
340   U(J)=U(J+MNP)
350   CONTINUE
      IF (.NOT.NORM.OR.L.NE.1) RETURN
      B1=U(IND)
      II=IND+MNP+N
      DO 370 I=IND,II
370   U(I)=U(I)-B1
      RETURN
400   KONT=5+L
      RETURN
      END
```



```
      SUBROUTINE DIRECT(KONT,U1,F1,S1,ISO,IS)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      DIMENSION A(81,81), U1(1), F1(1), S1(ISO,1)
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      KS=M+1
      NN=NP*KS
      IF (NN.LE.81) GOTO 10
      KONT=20
      RETURN
C     CHANGE THE DIMENSION (81,81) OF A IF NECESSARY
10    KR=1
      IF (PX) KR=KR+N
      IF (PY) KR=KR+MNP
      IF(.NOT.NORM) KR=0
      INDL=IND-1
      K=0
      DO 20 J=1,NN
      DO 20 I=1,NN
20    A(I,J)=0.
      DO 50 JJ=1,KS
      J=JJ-1
      DO 50 II=1,NP
      I=II-1
      K=K+1
      KPI=K+INDL
      IF (K.EQ.KR) GOTO 25
      IF (I.GE.N1.AND.I.LE.N2.AND.J.GE.M1.AND.J.LE.M2) GOTO 30
25    A(K,K)=1.
      IF (K.EQ.KR) GOTO 50
      IF (I.EQ.0.AND.PX) A(K,K+N)=-1.
      IF (J.GT.0.OR.NPY) GOTO 50
      A(K,K+MNP)=-1.
      U1(KPI)=0.
      GOTO 50
30    DO 40 KP=1,IS
      U1(KPI)=F1(KPI)
      J1=K+IZ(KP)
      I1=I+IX(KP)
      IF (I1.LT.0) GOTO 40
      IF (I1.LE.N) GOTO 32
      IF (NPX) GOTO 40
      J1=J1-N
32    I1=J+IY(KP)
      IF (I1.LT.0) GOTO 40
      IF (I1.LE.M) GOTO 34
      IF (NPY) GOTO 40
      J1=J1-MNP
34    A(K,J1)=S1(KP,KPI)+A(K,J1)
40    CONTINUE
50    CONTINUE
      NN2=NN-1
      DO 100 I=1,NN2
      B1=ABS(A(I,I))
      K=I
      I1=I+1
      DO 60 J=I1,NN
      B2=ABS(A(J,I))
```

```
      IF (B2.LE.B1) GOTO 60
      K=J
      B1=B2
60    CONTINUE
      IF (K.LE.I) GOTO 80
      B1=U1(INDL+I)
      U1(INDL+I)=U1(INDL+K)
      U1(INDL+K)=B1
      DO 70 J=I,NN
      B1=A(I,J)
      A(I,J)=A(K,J)
70    A(K,J)=B1
80    B1=A(I,I)
      IF (B1.EQ.0.) GOTO 105
      B3=U1(INDL+I)
      DO 100 K=I1,NN
      B2=A(K,I)/B1
      U1(INDL+K)=U1(INDL+K)-B2*B3
      DO 100 J=I1,NN
100   A(K,J)=A(K,J)-B2*A(I,J)
      IF (A(NN,NN).NE.0.) GOTO 106
105   KONT=5+L
      RETURN
106   II=NN+1
      DO 120 JJ=1,NN
      J=II-JJ
      B1=U1(INDL+J)/A(J,J)
      U1(INDL+J)=B1
      J1=J-1
      IF (J1.LE.0) GOTO 120
      DO 110 I=1,J1
110   U1(INDL+I)=U1(INDL+I)-A(I,J)*B1
120   CONTINUE
      RETURN
      END


      SUBROUTINE CONST(IS,NM)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /VAC/ L,LMAX,N1,M1,N,M,N2,M2,NMM,IND,INDL,NP,
     1   NM1,MM1,NP2,I1J1,I1M1,I1J2,I1M2,I2J1,I2M1,NN,NN2,MM,
     2   MM2,NPP,MMM1,NNM1,GAUSS,B,NORM,
     3   IX(9),IY(9),IZ(9),PX,PY,NB,NPX,NPY,N11,M11,MNP
      DIMENSION NM(5,11)
      LOGICAL GAUSS,NORM,PX,PY,NPX,NPY,B,NB
      N=NM(1,L)
      N2=NM(2,L)
      M=NM(3,L)
      M2=NM(4,L)
      NP=N+1
      NM1=N-1
      MNP=M*NP
      B=L.GT.1 .OR. IS.EQ.9
      NB=.NOT.B
      MM1=M-1
      NP2=2*NP
      IND=NM(5,L)
      I1J1=NP+1
      I1M1=-N
      I1J2=NP2+1
      I1M2=1-NP2
      I2J1=NP+2
      I2M1=2-NP
      IZ(3)=-NP
      IZ(5)=NP
      IZ(6)=-I1J1
      IZ(7)=I1M1
      IZ(8)=-I1M1
      IZ(9)=I1J1
      K=L+1
      INDL=NM(5,K)
      IF (L.EQ.LMAX) RETURN
      NN=NM(1,K)
      NN2=NM(2,K)
      MM=NM(3,K)
      MM2=NM(4,K)
      NPP=NN+1
      NNM1=NN-1
      MMM1=MM-1
      RETURN
      END
```


The first example of Section 7.1 is executed by the following program yielding the results of Tab. 2.

```
      PROGRAM RECF
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION    UO(8450), FO(4225), SO(9,4225),
     1             U1(1497), F1(1497), S1(9,1497)
      COMMON /OUTPUT/IOUT
      EXTERNAL OUTP,FIRST
      ISO=9
      IOUT=11
      EPS=7.E-15
      NEND=45
      EPSO=1.E-11
      CALL RECTAF (KONT,UO,FO,SO,ISO,8450,S1,8,EPS,FIRST)
      IF(KONT.NE.0) GO TO 100
      WRITE(IOUT,990)
  990 FORMAT(' RECTAF AUSGEFUEHRT')
      CALL RECTBF (KONT,S1,4225,SO,ISO,2,1)
      IF(KONT.NE.0) GO TO 100
      WRITE(IOUT,991)
  991 FORMAT(' RECTBF AUSGEFUEHRT')
      CALL RECTCF (KONT,UO,FO,SO,ISO,U1,F1,S1,EPSO,8,OUTP)
  100 CONTINUE
      WRITE (IOUT,77) KONT
77    FORMAT(' KONT=',I2)
      WRITE (IOUT,30)(UO(KK),KK=1,45)
30    FORMAT(' ',5E20.12)
      STOP
      END

      SUBROUTINE OUTP(N,M,U,IT,D)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /OUTPUT/IOUT
      DIMENSION U(1)
      X=1.
      DD=0.
      DO 10 I=1,45
      DD=DD+(U(I)-X)**2
      X=X+0.25
      IF (X.GT.3.1) X=1.
10    CONTINUE
      DD=DD/((N+1)*(M+1))
      DD=SQRT(DD)
      WRITE (IOUT,1) IT,D,DD
1     FORMAT(' IT=',I2,'    D = ',E10.4,'      DD = ',E10.4)
      WRITE (IOUT,2) (U(I),I=1,45)
2     FORMAT(' ',5E14.7)
      RETURN
      END


      SUBROUTINE FIRST(NR,IXY,X,Y,C1,C2,C3,C4,C5,C6,C7)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      GOTO (10,20,30,40,100,60,100,80), NR
10    C1=1.
      C2=0.
      C3=3.
      C4=1.
      C5=9.
      RETURN
20    C1=1.
      C2=2.
      C3=0.
      C4=2.
      C5=-1.
      RETURN
30    C1=X
      C2=1./X
      C3=1.
      C7=1.
      C6=SIN(6.28318530718*Y)
      C4=1.+C6
      C5=X*C6
      C6=-C6/X
      RETURN
40    C1=1.
      RETURN
60    C1=1.
      C2=4.
      RETURN
80    C1=1.
      RETURN
100   RETURN
      END
```

Acknowledgement. The author wishes to thank Mr. G. Hofmann for improving the FORTRAN programs.

References

[1] Hackbusch, W.: On the convergence of multi-grid iterations. Beiträge zur Numerischen Mathematik 9 (1981) 213 - 239

[2] Hackbusch, W.: On the multi-grid method applied to difference equations. Computing 2o (1978) 291 - 3o6

[3] Hackbusch, W.: A fast numerical method for elliptic boundary value problems with variable coefficients. In: Proceedings of the Second GAMM Conference on Numerical Methods in Fluid Mechanics (eds: E.H.Hirschel and W. Geller). DFVLR, Köln 1977

Additional references for the revised report:

[4] Stüben, K.and U. Trottenberg: Multigrid methods: Fundamental algorithms, model problem analysis and applications. In: [9] 1 - 176

[5] Hackbusch, W.: Multi-grid convergence theory. In: [9] 177-219

[6] Brandt, A.: Guide to multigrid development. In: [9] 22o - 312

[7] Foerster, H. and K.Witsch: Multigrid software for the solution of elliptic problems on rectangular domains: MGoo (release 1). In: [9] 427 - 46o

[8] Wesseling, P.: A robust and efficient multigrid method. In: [9] 614 - 63o

[9] Hackbusch, W. and U. Trottenberg (eds): Multigrid Methods. Proceedings, Köln-Porz, Nov. 1981. Lecture Notes in Mathematics 96o. Springer, Berlin 1982