



# Centrum voor Wiskunde en Informatica

Centre for Mathematics and Computer Science

---

P.W. Hemker, P.M. de Zeeuw

Some implementations of multigrid linear system solvers

Department of Numerical Mathematics

Report NM-R8401 January



**1984**



**Centrum voor Wiskunde en Informatica**  
Centre for Mathematics and Computer Science

---

P.W. Hemker, P.M. de Zeeuw

Some implementations of multigrid linear system solvers

Department of Numerical Mathematics

Report NM-R8401 January

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

SOME IMPLEMENTATIONS OF MULTIGRID LINEAR SYSTEM SOLVERS

P.W. HEMKER, P.M. DE ZEEUW

*Centre for Mathematics and Computer Science, Amsterdam*

In this paper portable and efficient FORTRAN implementations for the solution of linear systems by multigrid are described. They are based on ILU- or ILLU-relaxation. Scalar and vector versions are compared. Also a complete formal description of a more general multigrid algorithm is given in ALGOL 68.

1980 MATHEMATICS SUBJECT CLASSIFICATION: 65F10, 65N20

KEY WORDS & PHRASES: elliptic differential equations, solutions of linear systems, multigrid methods.

NOTE: This report will be submitted for publication elsewhere.

Report NM-R8401

Centre for Mathematics and Computer Science

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

## 1. Introduction

At the moment several implementations of multigrid methods are known for the solution of linear systems that arise from the discretization of more or less general elliptic partial differential equations [2,4,7]. Also some experiences for computations on vector machines such as the CRAY 1 or the CYBER 205 have been reported [1,3,8]. It appears that really efficient programs are now available. E.g. for the Poisson equation a code has been developed [1] for the CYBER 205, that solves the problem "up to truncation error" in 0.36  $\mu$ sec per meshpoint. It will be clear that -even with the present day computer technology- such a high speed can be obtained only when the computer code is specially tuned for the one particular problem and for the one particular machine.

In this paper we discuss the implementation of multigrid methods, not for a particular machine or problem, but for general elliptic 7-point difference equations and in a machine independent programming language. We describe two FORTRAN codes of which the purpose is to provide the user with a program that efficiently solves a large class of difference equations. A first code of this type was introduced by Wesseling in [10]. The codes are autonomous, i.e. they solve the linear systems of equations just like any standard subroutine for the solution of linear systems. The user has to specify only the matrix and the right hand side. Two versions of the codes are available -both in portable FORTRAN- one for use on scalar- the other for vector- (=pipeline) computers.

In section 2 of this paper we describe the problems to be solved. In section 3 we give an outline of the MG-algorithms used. The structure of the FORTRAN implementation is given in section 4 and in section 5 some remarks are made about computing times. In the first appendix, in an ALGOL 68 program we give a complete formal description of the flexible algorithm as mentioned in section 3. In a second appendix we give the user interfaces of the FORTRAN codes.

## 2. The difference problem

We consider the scalar linear second order elliptic PDE in two dimensions

$$(2.1.a) \quad a_{11} u_{xx} + 2 a_{12} u_{xy} + a_{22} u_{yy} + a_1 u_x + a_2 u_y + a_0 = f ,$$

on a rectangle  $\Omega \subset \mathbb{R}^2$ , with variable coefficients  $a_{ij}$ ,  $a_i$  and with boundary conditions

$$(2.1.b) \quad \begin{aligned} u_n + \alpha u_s + \beta u &= \gamma && \text{on } \Gamma_N , \\ u &= g && \text{on } \Gamma_D , \end{aligned}$$

where  $\Gamma_N \cup \Gamma_D = \delta\Omega$ . The subscripts  $n$  and  $s$  denote the derivatives normal to and along the boundary. If the equation (2.1) is discretized on a regular triangulation of the rectangle as given in figure 1, then the discretization obtained by a simple finite element method (with piecewise linear trial- and test-functions on the triangulation) will be a linear system

$$(2.2) \quad A_h u_h = f_h,$$

with a regular 7-diagonal structure. We consider codes for the solution of these linear systems. The 7-point discretization is the simplest one in which also cross-derivatives  $u_{xy}$  can be represented. It seems not worthwhile to consider more complex difference molecules because the solution of higher order discretizations can be performed by means of defect correction iteration in which only systems of the above mentioned form have to be solved.

On the rectangle  $\Omega$  equidistant computational grids  $\Omega^k$ ,  $k = 0, 1, 2, \dots, \ell$ , are defined by

$$(2.3) \quad \Omega^k = \{(x_1, x_2) \mid x_i = m_i 2^{-k}, m_i = 0, 1, \dots, N_i 2^k\}.$$

To obtain a solution  $u_h$  on  $\Omega^h$ , for the codes we consider, the user has to define the matrix  $A_h$  and the right hand side vector  $f_h$  only

for the discretization on the finest grid  $\Omega^\ell := \Omega^h$ .

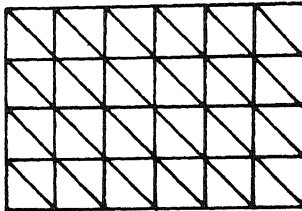


figure 1

The regular structure of the domain and the regular 7-point structure of the difference equations allows a simple structure of the data that are to be transferred to and from the programs. The solution and the right hand side can be stored in the most straightforward way in a 1- or 2- dimensional array. The coefficient matrix is stored similarly, by its diagonals.

There are many possible ways to solve the system (2.2) by multigrid. Based on previous work [5,6,7,8,9,10,11], in this paper we select two particularly efficient strategies for which FORTRAN codes have been made available and we give the description of a more general multigrid algorithm. A detailed ALGOL 68 program which

implements this more general algorithm is included in appendix 1. It can be used to experiment with the different possibilities.

### 3. The multigrid cycling algorithm.

The general multigrid algorithm for the solution of (2.2) is an iterative cycling procedure in which discretizations of (2.1) on all grids  $\Omega^k$ ,  $k = 0, 1, \dots, l$ , are used. We denote these discretizations by  $A_k u_k = f_k$ ,  $k = 0, 1, \dots, l$ ;  $k$  denotes the "level of discretization" and we take  $A_k := A_h$  and  $f_k := f_h$ .

One multigrid iteration cycle on level  $k$  is defined by the subsequent execution of

- 1.)  $p$  relaxation sweeps applied to the system  $A_k u_k = f_k$ ,
- 2.) the application of a "coarse grid correction", and
- 3.) again  $q$  relaxation sweeps for  $A_k u_k = f_k$ .

The coarse grid correction consists of: (1) the computation of

$$(3.1) \quad f_{k-1} := R_{k-1,k} ( f_k - A_k \tilde{u}_k ),$$

where  $\tilde{u}_k$  is the current approximation to the solution and  $R_{k-1,k}$  is a restriction operator which represents the current residual on the next coarser level; (2) the computation of  $\tilde{u}_{k-1}$ , an approximation to the solution of the correction equation

$$(3.2) \quad A_{k-1} u_{k-1} = f_{k-1}.$$

This approximation is obtained by application of  $s$  multigrid iteration cycles on level  $k-1$ , with a zero starting approximation; and (3) updating the current solution  $\tilde{u}$  by

$$(3.3) \quad \tilde{u}_k := \tilde{u}_k + P_{k,k-1} \tilde{u}_{k-1},$$

where the prolongation operator  $P_{k,k-1}$  denotes the interpolation from level  $k-1$  to  $k$ .

On the coarsest level another method (at choice) can be used for the computation of  $\tilde{u}_0$ .

In principle the parameters  $p$ ,  $q$  and  $s$  and the operators  $R_{k-1,k}$ ,  $A_{k-1}$ ,  $P_{k,k-1}$  are free to choose. Obvious restrictions are  $p+q \geq 1$  and  $1 \leq s \leq 3$ . A natural choice for combination with the finite element discretization (2.2) is the use of a piecewise linear interpolation over triangles in  $\Omega^{k-1}$  for  $P_{k,k-1}$ . The corresponding restriction is the transposed operator  $R_{k-1,k} = P_{k,k-1}^T$ . This prolongation and restriction are exactly the 7-point prolongation and restriction as described in [11]. With these  $P_{k,k-1}$  and  $R_{k-1,k}$  the finite element discrete operators on coarser grids are easily derived from the fine grid finite element discretization by

$$(3.4) \quad A_{k-1} = R_{k-1,k} A_k P_{k,k-1}, \quad k = \ell, \ell-1, \ell-2, \dots, 1.$$

Thus, the coarser grid discretizations are obtained by algebraic manipulation only.

An ALGOL 68 program, based on these choices for the operators is presented as a worked-out illustration in appendix 1. The multigrid cycling procedure is given in proc MG (page 9). It is imbedded in a complete solution procedure proc MGM, which also checks the consistency of the input data, which generates the coarse grid operators by (3.4) and which constructs an initial estimate by "full multigrid", i.e. first it finds an approximate solution on the coarser grid and interpolates this to the next finer ones. The parameters  $p, q, s$ , the relaxation procedure and the stopping strategy are still to be chosen. For a set of default parameters (that can be changed by the user) an autonomous procedure is given in proc SOLVE SYS (page 10). This procedure requires as data only the matrix  $A_h$ , the right hand side vector  $f_h$  and the number of levels  $\ell$ . It delivers the solution  $u_h$  without further interference by the user.

In the procedure MGM the user can select his own multigrid strategy  $(p, q, s)$  and he may select from different relaxation procedures: Point Gauss Seidel, Line Gauss Seidel or Incomplete Line LU-decomposition relaxation. V-cycles are obtained by  $s=1$ , W-cycles by  $s=2$ .

#### 4. The structure of the FORTRAN implementations.

Less flexible but more efficient implementations have been written in FORTRAN. Here we consider two versions of the general MG-algorithm. Both use  $p=0, s=q=1$  as the strategy. The first version (MGD1) uses Incomplete LU-decomposition (ILU-) relaxation as the relaxation procedure [10], the other (MGD5) uses Incomplete Line LU-decomposition (ILLU-) relaxation [9].

MGD1 is particularly efficient because of the smoothing properties of the ILU-relaxation [5,9] and the efficient residual computation. In this version on each level the 7-diagonal matrix  $A_k$  is decomposed as

$$A_k = L_k U_k - C_k,$$

where  $L_k$  is a lower-triangular matrix (with unity on the main diagonal) and  $U_k$  is an upper-triangular matrix. The requirement that  $L_k$  and  $U_k$  have non-zero diagonals only where  $A_k$  has, determines  $L_k$  and  $U_k$ . The remainder matrix  $C_k$  has only two non-zero diagonals of which the elements are easily derived from  $L_k$  and  $U_k$ .



One relaxation sweep of ILU-relaxation corresponds to the solution of the system

$$L_k U_k u_k^{(i+1)} = f_k + C_k u_k^{(i)} .$$

After such a relaxation sweep the residual is efficiently computed by

$$r_k^{(i+1)} := f_k - A_k u_k^{(i+1)} = C_k ( u_k^{(i+1)} - u_k^{(i)} ) .$$

The other relaxation method, ILLU-relaxation, which is due to J.A. Meyerink, is described in [9] and in more detail in Wesseling [these proceedings]. A complete description in ALGOL 68 is found in the ALGOL 68 program in the appendix 1.

The global structure of both MGD1 and MGD5 is the same. First, in a preparational phase, the sequence of coarse grid operators is constructed by a subroutine RAP, according to (3.4). Then the decomposition is performed (in DECOMP). Finally, in the cycling phase, at most MAXIT iterations of the cycling process are performed. On the basis of intermediate results -the detection of a small residual norm- the iteration can be stopped earlier. This necessitates the computation of this norm (in VL2NOR) in each cycle.

The following is an outline in quasi FORTRAN of the multigrid cycling process in MGD1. At all computational levels  $k = 1, 2, \dots, \ell$ , the matrix decomposition  $A_k = L_k U_k - C_k$  is available. At the beginning (or end) of each MG-iteration cycle,  $u_\ell$  contains the current solution and  $r_\ell$  the corresponding residual. If no initial estimate is available we take  $u_\ell \equiv \emptyset$  and  $r_\ell \equiv f$ .

## C THE MGD1 ITERATION PROCESS

DO 100 N=1,MAXIT	
CALL RESTRI (F,R,L-1)	$f_{l-1} = R_{l-1,l} \quad r_l \quad ,$
DO 10 K=L-2 ,1 ,-1	
CALL RESTRI (F,F,K)	$f_k = R_{k,k+1} \quad f_{k+1} \quad ,$
100 CONTINUE	
CALL SOLVE (U,F,1)	$u_1 = ( L_1 \quad U_1 )^{-1} \quad f_1 \quad ,$
DO 20 K=2,L-1	
CALL PROLON (U,U,K)	$u_k = P_{k,k-1} \quad u_{k-1} \quad ,$
CALL CTUPF (V,U,F,K)	$v_k = C_k \quad u_k \quad + \quad f_k \quad ,$
CALL SOLVE (U,V,K)	$u_k = ( L_k \quad U_k )^{-1} \quad v_k \quad ,$
200 CONTINUE	
CALL PROLON (R,U,L)	$r_l = P_{l,l-1} \quad u_{l-1} \quad ,$
DO 30 J=1,NF	
R(J)=R(J)+U(J)	$r_l = r_l \quad + \quad u_l \quad ,$
300 CONTINUE	
CALL CTUPF (V,R,F,L)	$v_l = C_l \quad r_l \quad + \quad f_l \quad ,$
CALL SOLVE (U,V,L)	$u_l = ( L_l \quad U_l )^{-1} \quad v_l \quad ,$
CALL CTUMV (U,R)	$r_l = C_l \quad ( u_l \quad - \quad r_l \quad ) \quad ,$
RES = VL2NOR (R)	$\ r_l\ _2 \quad .$
IF (RES .LT. TOL) GOTO 200	
1000 CONTINUE	
2000 CONTINUE	

In the actual implementation of MGD1, the matrix  $A_k$  is not kept in storage, but it is overwritten by  $L_k$  and  $U_k$ . At minimal costs, the remainder matrix  $C_k$  is recomputed each time from  $L_k$  and  $U_k$  (in the subroutines CTUMV and CTUPF).

The other program, MGD5, with ILLU-relaxation, is less efficient for problems like the Poisson equation, but it is more suitable for problems such as the convection-diffusion or the anisotropic diffusion equation, in which a small parameter multiplies the highest derivatives [6,9].

The cycling process in MGD5 is similar to the one in MGD1. In this case, however, the matrices  $A_k$  are not overwritten and the residual is computed in a straightforward way.

## C THE MGD5 ITERATION PROCESS

```

DO 100 N=1,MAXIT

CALL RESTRI(F,R,L-1)
DO 10 K=L-2, 2, -1
CALL RESTRI(F,F,K)
10 CONTINUE
CALL RESTRI(U,F,1)

CALL SMOOTH(U,F,1)

DO 20 K=2,L-1
CALL PROLON(U,U,K)

CALL SMOOTH(U,F,K)
20 CONTINUE

CALL PROLON(R,U,L)
DO 30 J=1,NF
U(J)=U(J)+R(J)
30 CONTINUE
CALL SMOOTH(U,F,L)
CALL RESIDU(R,F,U)

RES = VL2NOR(R)
IF(RES .LT. TOL) GOTO 200
100 CONTINUE
200 CONTINUE

```

$$f_{l-1} = R_{l-1,l} \quad r_l ,$$

$$f_k = R_{k,k+1} \quad f_{k+1} ,$$

$$u_1 = R_{1,2} \quad f_2 ,$$

relax on level 1,

$$u_k = P_{k,k-1} \quad u_{k-1} ,$$

relax on level k,

$$r_l = P_{l,l-1} \quad u_{l-1} ,$$

$$u_l = u_l + r_l ,$$

relax on level l,

$$r_l = f_l - A_l \quad u_l ,$$

$$\|r_l\|_2 .$$

All subroutines in the iteration processes in MGD1 or MGD5 have their own particular features that make them more or less feasible for vectorization. This will be shown in section 5.

## 5. The efficiency of the FORTRAN implementations

Both algorithms MGD1 and MGD5 have been coded in portable ANSI-FORTRAN. The codes pass the PFORT verifier, except that more complex subscript expressions appear than  $(I*M+N)$ . (These expressions, where I is variable and M and N are constants, are the only ones that are allowed for subscripting by PFORT.) In this portable FORTRAN, optimized versions for scalar- and vector- architecture have been constructed. The corresponding codes are called MGD1S, MGD1V, MGD5S and MGD5V. They are all in the form of a FORTRAN subroutine. Their user-interface is given in appendix 2. The different versions run on several machines among which the CYBER 205 and the CRAY 1.

If run on scalar architecture, after the preparational phase, the computing time for the programs is proportional to the number of iteration steps and to the number of points in the finest grid. The preparational work to generate the coarse grid operators and to form their decompositions is roughly equivalent to 3 iteration sweeps. The computing times for the scalar optimized versions on the CYBER 170 and the CYBER 205 (using scalar architecture) are given in table 5.1.

	MGD1S	MGD5S
CYBER 170	15.4	24.9
CYBER 205	8.1	11.1

Table 5.1 Computing times for  
MGD1 and MGD5 in scalar mode,  
in  $\mu$  sec/(meshpoint.cycle).

The relative time spent in the different subroutines (as defined in the previous section) is slightly different for the different machines (compilers). These times are given in table 5.2. We notice that the time to compute the prolongations, the restrictions and the norms is small as compared to the relaxation or the residual computations. Further we see e.g. that the time spent in CTUMV is 3/4 of the time spent in CTUPF, as is expected (CTUPF runs over all points, whereas CTUMV only works on points on the finest grid).

code	MGD1S	MGD1S	MGD5S	MGD5S
machine	CY 170	CY 205	CY 170	CY 205
RAP	2.32	1.50	1.40	1.10
DECOMP	0.86	1.40	0.76	1.90
PROLON	0.072	0.063	0.05	0.046
RESTRI	0.089	0.040	0.06	0.030
VL2NOR	0.040	0.044	0.025	0.032
SOLVE	0.33	0.30		
CTUMV	0.15	0.22		
CTUPF	0.22	0.29		
RESIDU			0.16	0.14
SMOOTH			0.65	0.72

Table 5.2. The time spent in the different subroutines  
in scalar mode, expressed in the time spent  
in a complete iteration cycle.

To run portable FORTRAN programs on a vector architecture we have to rely on the auto-vectorization capabilities of the available compilers. Both on the CRAY 1 and on the CYBER 205 we found it possible to vectorize all nonrecursive inner loops in this way. The length of the vectors in the experiments was  $(2^{k+1} + 1)^j$  with  $j=1$  or  $j=2$  and  $k = 1, \dots, \ell$ , where  $\ell$  denotes the finest level of discretization. Most loops run over lines in the grid ( $j=1$ ), but in a number of cases loops run over the entire net ( $j=2$ ).

Some comparisons of the CRAY 1 and the CYBER 205 have been given in [8]. There it was shown that the essential difference between both machines in these computations is the fact that the CYBER 205 is not very effective for loops with a stride unequal to 1. This is particularly important in the restriction and the prolongation, where frequently strides 2 occur. For the restriction the improvement of vector- over scalar- computing time was a factor 4.2-5.6 ( $\ell=5,6$ ) for the CRAY 1 and 1.2-2.2 ( $\ell=5,6,7$ ) for the CYBER 205.

Nevertheless, it was also shown that -although an essential part of the computation contains recursive loops- a reasonable gain of efficiency was obtained for MGD1 using the CRAY 1 or CYBER 205 vector architecture.

Since the experiences reported in [8], a new compiler for the CYBER 205 became available (FORTRAN 2.0). With this compiler it was possible to obtain in portable language a more efficient implementation of some recursive loops, whereas with the previous compiler reference had to be made to special "stacklib" routines.

With the portable FORTRAN program on the CYBER 205, an acceleration factor 3.3-4.6 is obtained for MGD1 (acceleration of MGD1V in vector mode on a two-pipe CYBER 205 over MGD1S in scalar mode on the same CYBER). The program MGD5 is less feasible for vectorization. Its acceleration factor is only 2.1-2.3. Details of the performance of the different subroutines under vector-mode computation are given in table 5.3. In this table we see the CP-times that are spent in the different subroutines of MGD1 and MGD5, when the vector version is run for one iteration cycle on the CYBER 205.

grid	65*65	129*129	257*257
RAP	20 ( 2.8)	49 ( 4.2)	143 ( 5.6)
DECOMP(MGD1)	12 ( 4.0)	43 ( 4.4)	161 ( 4.6)
DECOMP(MGD5)	29 ( 3.1)	96 ( 3.7)	352 ( 4.0)
CYCLE(MGD1)	1.1 ( 3.3)	3.3 ( 4.1)	11.6 ( 4.6)
CYCLE(MGD5)	2.3 ( 2.1)	8.2 ( 2.3)	32.0 ( 2.3)
PROLON	0.9 (2.4)	2.1 ( 4.1)	5.9 ( 5.7)
RESTRI	1.2 (1.3)	3.0 ( 1.8)	9.5 ( 2.2)
VL2NOR	0.1 (15 )	0.4 (14.8)	1.6 (15.6)
SOLVE	6.8 ( 1.6)	22.5 ( 1.8)	82.5 ( 1.9)
CTUMV	0.3 (25 )	1.3 (22.8)	5.8 (20.4)
CTUPF	0.5 (20 )	1.8 (21.6)	8.0 (19.4)
RESIDU	0.7 ( 9 )	3.1 ( 8.0)	13.2 ( 7.9)
SMOOTH	19.3 ( 1.8)	72.3 ( 1.8)	287.5 ( 1.8)

Table 5.3. The time (in m.sec.) for the different subroutines in the vector implementations MGD1V and MGD5V on the CYBER 205 ( two pipes, FORTRAN 2.0 compiler ). Between brackets the acceleration factor ( compared with the scalar versions in scalar mode ).

In table 5.4 we show the megaflop rates for the different subroutines. These rates are defined as the number of floating point operations per second divided by  $1.0E+6$ . One can consider these numbers as a measure of how well the subroutines are suited for the hardware. For different sizes of the finest grid, the rates for the vector- and scalar-version are given for the CYBER-205 (two pipes, with autovectorization via the FORTRAN 2.0 compiler). For the 65\*65 grid also the rate for the CYBER 170-750 (with FORTRAN IV) is shown.

The CP-times used for the computation of the megaflop rate is the time spent in the subroutines on the finest and on all coarser grids. As can be expected for the vectormachine, the numbers are dependent of the vectorlengths (i.e. the number of points in the x-direction or the total number of gridpoints) and whether or not strides greater than one occur. If we compare the first column for the rates of the 129\*129 grid with the first column for the rates of the 257\*257 grid, we see both increases and decreases. The increases are explained by vectorlengths increasing from 129 to 257, the decreases are explained by vector lengths increasing from 129\*129 to 257\*257 = 66049 which makes splitting of the long vectors necessary because of the restricted number of vectoraddresses (namely 65535) on the CYBER-205.

finest grid	!	65*65	!	129*129	!	257*257
RAP (MGD1,MGD5)	!	13.7 4.9 1.8	!	21.4 5.1	!	28.7 5.1
DECOMP (MGD1)	!	8.6 2.1 1.8	!	9.4 2.1	!	9.9 2.1
DECOMP (MGD5)	!	7.1 2.3 2.6	!	8.4 2.3	!	9.0 2.3
CYCLE (MGD1)	!	15.5 4.7 2.6	!	20.3 4.9	!	23.0 5.0
CYCLE (MGD5)	!	12.1 5.8 2.6	!	13.3 5.9	!	13.6 5.9
PROLON (MGD1,MGD5)	!	11.5 4.7 2.2	!	19.0 4.7	!	26.5 4.6
RESTRI (MGD1,MGD5)	!	8.7 6.9 1.6	!	13.1 7.4	!	16.3 7.5
VL2NOR (MGD1,MGD5)	!	84.5 5.6 3.2	!	83.2 5.6	!	82.6 5.6
SOLVE (MGD1)	!	11.8 7.5 3.7	!	13.9 7.7	!	15.0 7.7
CTUMV (MGD1)	!	84.5 3.4 2.6	!	76.8 3.4	!	68.3 3.4
CTUPF (MGD1)	!	68.5 3.4 2.4	!	74.5 3.4	!	66.3 3.4
RESIDU (MGD5)	!	84.5 9.4 3.6	!	75.2 9.4	!	70.1 8.9
SMOOTH (MGD5)	!	9.8 5.5 2.8	!	10.2 5.5	!	10.1 5.5

Table 5.4. Megaflop rates for the different subroutines.

For each grid the rates for the efficient vector implementation (1st column) and the efficient scalar version (2nd column) on a two-pipe CYBER-205 (FORTRAN 2.0) are given. For the 65\*65-grid also the rate for the CYBER 170-750 (FORTRAN IV) is shown (3rd column).

## 6. Appendices

### 6.1 Appendix 1

In this appendix the text is given of an ALGOL 68 program which implements a general multigrid algorithm. The solutions and the right hand sides are represented in nets, i.e. two-dimensional arrays corresponding to the grid  $\Omega^k$ . The matrices in netmats, i.e. three-dimensional arrays; here the first 2 indices denote the equation (corresponding to a gridpoint), the 3rd index denotes the diagonal (for details, see the comments on page 4).

```

begin # solution of a linear system by multigrid #
      # a complete description #
      # not an optimal efficient implementation #

# mode declarations #

mode net = ref [, ] real ;
mode netmat = ref [,,] real ;

# elementary operators #

op zero = ( ref [] real a ) ref [] real :
  ( for i from lwb a to upb a
    do a[i]:= 0.0 od ; a );
op zero = ( net a ) net :
  ( for i from 1 lwb a to 1 upb a
    do zero a[i,] od ; a );
op zero = ( netmat a ) netmat :
  ( for i from 1 lwb a to 1 upb a
    do zero a[i,,] od ; a );
op += = ( net aa,bb ) net :
  ( int l1 = 1 lwb aa, l2 = 2 lwb aa,
    u1 = 1 upb aa, u2 = 2 upb aa;
    for i from l1 to u1 do
      for j from l2 to u2 do
        aa[i,j] += bb[i,j] od od ; aa );

# prolongation: linear interpolation #

proc lin int pol = ( net net ) net :
begin int l1 = 1 lwb net, l2 = 2 lwb net,
      b1 = 1 upb net, b2 = 2 upb net;
      heap [2*l1:2*b1,2*l2:2*b2] real fine;
      int jj; real u2,u3,u4;
      ref [] real uip= net[l1,@l2],
        upp= fine[2*l1,@2*l2];
      jj:= 2*l2; upp[jj]:= u4:= uip[l2];
      for jp from l2+1 to b2
        do u3:= u4; u4:= uip[jp];
          upp[jj+=1]:= (u3+u4)/2;
          upp[jj+=1]:= u4
        od ;
      for ip from l1+1 to b1
        do ref [] real ui = net [ip-1 ,@ l2],
          uip = net [ip ,@ l2],
          umm = fine[2*ip-1,@2*l2],
          upp = fine[2*ip ,@2*l2];
          jj:= 2*l2; u2:= ui[l2]; u4:= uip[l2];
          umm[jj]:= (u2+u4)/2; upp[jj]:= u4;
          for jp from l2+1 to b2
            do jj+= 1; u2:= ui [jp];
              u3:= u4; u4:= uip[jp];
              umm[jj] := (u2+u3)/2;
              upp[jj] := (u3+u4)/2;
              jj+= 1;
              umm[jj] := (u2+u4)/2;
              upp[jj] := u4
            od
          od ; fine
      end ;

```



```

# interpolation: quadratic on finer grids      #

proc  sqr int pol = ( net net ) net :
if    int  l1 = 1 lwb net,  l2 = 2 lwb net,
      b1 = 1 upb net,  b2 = 2 upb net;
      odd (b1-1) or odd (b2-12)
then  lin int pol (net)
else  int  l11 = 2*l1, l12 = 2*l2;
      heap [l11:2*b1,l12:2*b2] real fine;

int  jj, jp;
real x1, x2, x3, y1, y2, y3, z1, z2, z3, yy2, yy3, zz2, zz3;
ref [ ] real ui= net[ l1,@l2], fi= fine[l11,];
fi[l12]:= x1:= ui[l2]; jj:= l12+1;
for  j from l2+1 by 2 to b2-1
do   x2:= ui[j]; x3:= ui[j+1];
      fi[jj:jj+3] :=( ( 3*(x1 + 2*x2) - x3 )/8, x2,
                     ( -x1 + 3*(2*x2 + x3 ))/8, x3 );
      jj += 4; x1:= x3
od ;
for  ii from l1+1 by 2 to b1-1
do   ref [ ] real uim= net[ii-1,@l2], uii= net[ii ,@l2],
      uip= net[ii+1,@l2];
      ref [,] real finei = fine[2*ii-1:2*ii+2,@l12];

      x3:=          uim[l2] /8;
      y3:= ( yy3:= uii[l2] )/4;
      z3:= ( zz3:= uip[l2] )/8;
      finei[,l12]:= ( 3*(x3+y3) - z3, yy3, 3*(y3+z3) - x3, zz3 );

for  jj from l2+1 by 2 to b2-1
do   jp:= jj+1;      x1:= x3; y1:= y3; z1:= z3;
      x2:=          uim[jj] /4; x3:=          uim[jp] /8;
      y2:= ( yy2:= uii[jj] )/4; y3:= ( yy3:= uii[jp] )/4;
      z2:= ( zz2:= uip[jj] )/4; z3:= ( zz3:= uip[jp] )/8;

      finei[,2*jj-1:2*jj+2]:=
      ((2*(x2+y1)-z1+y2-x3,
        2*(x2+y2)-x1+y1-z1,
        3*(x3+y2)-z1,
        3*(x3+y3)-z3 ),
      (2*(y1+y2)-x1+x2-x3,  yy2,
        2*(y2+y3)-z1+z2-z3,  yy3 ),
      (3*(z1+y2)-x3,
        2*(y2+z2)-x3+y3-z3,
        2*(z2+y3)-x3+y2-z1,
        3*(z3+y3)-x3 ),
      (3*(z1+z2)-z3,  zz2,  3*(z3+z2)-z1,  zz3 ))

      od
od ;
fine
fi ;

```

```

# restriction: transposed linear interpolation #

proc lin weight = ( net ffi ) net :
begin int l1 = (1 lwb ffi) over 2, u1 = (1 upb ffi) over 2,
        l2 = (2 lwb ffi) over 2, u2 = (2 upb ffi) over 2;
  heap [l1:u1,l2:u2] real fco;
  int ti,tk,tkp;
  real ffb,ffd,ffe;

  zero fco[l1,];
  for i from l1 to u1-1
  do ti:= i+i; fco[i+1,l2]:= 0;

  for k from l2 to u2-1
  do tk:= k+k; tkp:= tk+2; ffe:= ffi[ti+1,tk+1];
    fco[i ,k+1]+:= ffe+( ffb:= ffi[ti ,tk+1] );
    fco[i+1,k ] := ffe+( ffd:= ffi[ti+1,tk ] );
    ((fco[i ,k ]+:= ffd+ffb)*:=0.5)+:= ffi[ ti, tk]
  od ;
    fco[i+1,u2] := ffd:= ffi[ti+1,tkp ];
    ((fco[i ,u2 ]+:= ffd )*:=0.5)+:= ffi[ ti,2*u2]
  od ;

  for k from l2 to u2-1
  do tk:= k+k; tkp:= tk+2;
    fco[u1,k+1]+:= ( ffb:= ffi[2*u1,tk+1] );
    ((fco[u1 ,k ]+:= ffb )*:=0.5)+:= ffi[2*u1, tk]
  od ;
    (fco[u1 ,u2]
      *:=0.5)+:= ffi[2*u1,2*u2];
    fco

end ;

# residual evaluation #

proc residual = ( netmat m, net u,f ) net :
begin int l1= 1 lwb u, l2= 2 lwb u,
        u1= 1 upb u, u2= 2 upb u;
  heap [l1:u1,l2:u2] real s;

  ref [ ] real uim:= u[l1,@l2], ui, uip:= u[l1,@l2];
  for i from l1 to u1
  do ( ui:= uip; i = u1 ! skip ! uip:= u[i+1,@l2] );
    # where the matrix does not define the netmat m, #
    # m should contain zeroes ! #
    ref [ ] real si = s[i,@l2], fi = f[i,@l2];
    ref [,] real mi = m[i,@l2,@-3];

    int jm:= l2, jj, jp:= l2;
    for j from l2 to u2
    do ( jj:= jp; j=u2 ! skip ! jp+:= 1 );

      ref [ ] real mij = mi[jj,@-3];
      si[jj]:= fi[jj] - (mij[-3]*uim[jj] + mij[-2]*uim[jp] +
        mij[-1]*ui [jm] + mij[ 0]*ui [jj] + mij[ 1]*ui [jp] +
        mij[ 2]*uip[jm] + mij[ 3]*uip[jj]);

      jm := jj
    od ; uim:= ui
  od ;s
end ;

```

```

# coarse grid operator construction                                     #

proc rap = ( netmat afi) netmat :
begin int l1 = (1 lwb afi) over 2, u1 = (1 upb afi) over 2,
        l2 = (2 lwb afi) over 2, u2 = (2 upb afi) over 2;
heap [l1:u1,l2:u2,-3:3] real aco;
real q= 0.25;
int ti,tip,tk,tkp;

[1:3,1:3,-3:3] real fine;
ref [] real
    a = fine[1,1,@-3], b = fine[1,2,@-3], c = fine[1,3,@-3],
    d = fine[2,1,@-3], e = fine[2,2,@-3], f = fine[2,3,@-3],
    g = fine[3,1,@-3], h = fine[3,2,@-3], j = fine[3,3,@-3];
ref real
    aa =a[ 0], ab =a[ 1], ad =a[ 3],
    ba =b[-1], bb =b[ 0], bc =b[ 1], bd =b[ 2], be =b[ 3],
    cb =c[-1], cc =c[ 0], ce =c[ 2], cf =c[ 3],
    da =d[-3], db =d[-2], dd =d[ 0], de =d[ 1], dg =d[ 3],
    eb =e[-3], ec =e[-2], ed =e[-1], ee =e[ 0],
    ef =e[ 1], eg =e[ 2], eh =e[ 3],
    fc =f[-3], fe =f[-1], ff =f[ 0], fh =f[ 2], fj =f[ 3],
    gd =g[-3], ge =g[-2], gg =g[ 0], gh =g[ 1],
    he =h[-3], hf =h[-2], hg =h[-1], hh =h[ 0], hj =h[ 1],
    jf =j[-3], jh =j[-1], jj =j[ 0];

```

# orientation:

```

aco = coarse          k-1          k
                    -----> y
!
!       fine       1       2       3
!
! i-1     1       a -- b -- c
!         ! / ! / !
!         2       d -- e -- f
!         ! / ! / !
! i       3       g -- h -- j
x   v

```

the slice [i,j,] corresponds to the coefficients in equation (i,j);  
the slice [ ,,k] corresponds to matrix diagonals as follows:

```

[ ,,-3] : n
[ ,,-2] : n-e
[ ,,-1] : w
[ ,, 0] : p      (the main diagonal)
[ ,, 1] : e
[ ,, 2] : s-w
[ ,, 3] : s

the difference star:
          -3      -2
          ! /
-1 - 0 - 1
      / !
      2      3

```

#

```

      zero aco[ l1, , ];
for i from l1 to u1-1
do ti:= i+i; tip:= ti+2;

      zero aco[i+1,l2,];
for k from l2 to u2-1
do tk:= k+k; tkp:= tk+2;
  fine[1:3,1:3,]:= afi[ti:tip,tk:tkp,];
  ref [] real a = aco[i ,k ,@-3],
              c = aco[i ,k+1,@-3],
              g = aco[i+1,k ,@-3],
              j = aco[i+1,k+1,@-3];

  #aa#((a[ 0]+:= (ab+ba+ad+da)*2+ bb+dd+bd+db )*:=q)+:=aa;
  #cc# c[ 0]+:= (ce+ec+cb+bc)*2+ ee+bb+be+eb+ef+fe;
  #gg# g[ 0]+:= (ge+eg+gd+dg)*2+ ee+dd+de+ed+eh+he;
  #jj# j[ 0] := fh+hf;
  #ac#( a[ 1]+:= (ab+bc)*2 + bb+be+db+de)*:=q;
  #ca#( c[-1]+:= (ba+cb)*2 + bb+eb+bd+ed)*:=q;
  #ag#( a[ 3]+:= (ad+dg)*2 + dd+bd+de+be)*:=q;
  #ga#( g[-3]+:= (da+gd)*2 + dd+db+ed+eb)*:=q;
  #gc#( g[-2] := (ge+ec)*2 + ee+he+de+hf+db+ef+eb)*:=q;
  #cg#( c[ 2] := (eg+ce)*2 + ee+eh+ed+fh+bd+fe+be)*:=q;
  #gj# g[ 1] := eh+hf+ef;
  #jg# j[-1] := he+fh+fe;
  #cj# c[ 3] := eh+ef+fh;
  #jc# j[-3] := he+fe+hf
od ;
  fine[1:3,1,]:= afi[ti:tip,tkp,];
  ref [] real a = aco[i ,u2,@-3],
              g = aco[i+1,u2,@-3];
  #aa#((a[ 0]+:= (ad+da)*2 + dd)*:= q)+:=aa;
  #gg# g[ 0]+:= (gd+dg)*2 + dd;
  #ga#( g[-3]+:= (gd+da)*2 + dd)*:=q;
  #ag#( a[ 3]+:= (ad+dg)*2 + dd)*:=q;
  g[-2] := g[ 1]:= 0.0
od ;

for k from l2 to u2-1
do tk:= k+k; tkp:= tk+2;
  fine[1,1:3,]:= afi[tip,tk:tkp,];
  ref [] real a = aco[u1,k ,@-3],
              c = aco[u1,k+1,@-3];
  #aa#((a[ 0]+:= (ab+ba)*2 + bb)*:= q)+:=aa;
  #cc# c[ 0]+:= (cb+bc)*2 + bb;
  #ca#( c[-1]+:= (cb+ba)*2 + bb)*:=q;
  #ac#( a[ 1]+:= (ab+bc)*2 + bb)*:=q;
  c[ 2] := c[ 3]:= 0.0
od ;
#aa#(aco[u1,u2,0]*:=q)+:=afi[2*u1,2*u2,0];
aco
end ;

```

```

# point relaxation procedure                                     #
proc pgs relax = ( ref netmat dec, netmat m, net u,f) void :
begin # point gauss seidel (pgs) #
  int l1:= 1 lwb u, u1:= 1 upb u, start1, step1, stop1,
      l2:= 2 lwb u, u2:= 2 upb u, start2, step2, stop2;

  to ( symmetric ! 2 ! 1 )
  do ( backward ! start1:= u1; step1:= -1; stop1:= l1
      ! start1:= l1; step1:= 1; stop1:= u1 );
      ( reverse ! start2:= u2; step2:= -1; stop2:= l2
      ! start2:= l2; step2:= 1; stop2:= u2 );

  for i from start1 by step1 to stop1
  do ref [] real fi= f[i,@l2], uim= u[(i>l1!i-1!i),@l2],
      ui= u[i,@l2], uip= u[(i<u1!i+1!i),@l2];
      ref [,] real mi= m[i,@l2,@-3];
      for j from start2 by step2 to stop2
      do int jm= (j>l2!j-1!j), jp= (j<u2!j+1!j);
          ref [] real mij = mi[j,@-3];

          ui[j]:= ( mij[-3]*uim[j]+mij[-2]*uim[jp]+
mij[-1]*ui [jm] - fi[j]+mij[ 1]*ui [jp]+
mij[ 2]*uip[jm]+mij[ 3]*uip[j] )/ -mij[ 0]

          od
          od ;
      ( symmetric! reverse:= not reverse; backward:= not backward)
  od
end ;

```

```

# line relaxation procedure                                     #
proc lgs relax = ( ref netmat dec, netmat m, net u,f) void :
begin # line gauss seidel (lgs) #
  int st = ( zebra ! 2 ! 1 );
  int l1:= 1 lwb u, u1:= 1 upb u, start, step, stop;

  proc line relax = ( ref [ ] real um,u,up,f,
                    ref [,] real m ) void :
  begin ref [] real b= m[ , 1], n = m[,-3], ne= m[,-2],
      a= m[ , 0], s = m[ , 3], sw= m[ , 2],
      c= m[,-1];
      #not existing matrix elements: c[l]= b[k]= 0 !!#

  int l= lwb f, k= upb f; [l:k] real aa;
  int i:=l; real g:= 0, p; aa[l]:= 1.0;

  for j from l to k
  do aa[j]:= a[j] - b[i]* ( p:= c[j]/aa[i] );
      g := f[j] - n[j]*um[j] -
          sw[j]*up[i] - s[j]*up[j] - g*p;
      ( j<k ! g -= ne[j]*um[j+1] );
      u[ j]:= g; i:= j
  od ;
  for j from k by -1 to l
  do u [j]:= g := ( u[j] - b[j]*g )/aa[j] od
end ;

```

```

for k to ( symmetric or zebra ! 2 ! 1)
do ( backward ! start := u1; step := -st; stop := l1
    ! start := l1; step := st; stop := u1 );
( zebra
! ( symmetric /= odd (k+start) ! start+:= sign step )
# ( symmetric ! even-odd ! odd-even ) half step #);

for i from start by step to stop
do line relax ( u[ (i>l1!i-1!i),], u[i,],
                u[ (i<u1!i+1!i),], f[i,], m[i,,e-3] )
od ;
( symmetric ! backward:= not backward )
od
end ;

# illu relaxation procedure #

proc illu relax = ( ref netmat dec, netmat jac, net u,f) void :
begin int l1= 1 lwb u , u1= 1 upb u, l2= 2 lwb u, u2= 2 upb u;
( netmat (dec) ::= netmat ( nil ) ! illudec (jac,dec) );
[l1:u1,l2:u2] real du,rh;

proc soll = ( int i, net r) void :
( ref [] real l = dec[i,, -1], d = dec[i,, 0],
  u = dec[i, , 1], z = r [i, ] ;
for j from l2+1 to u2 do z[j]+:= l[j]*z[j-1] od ;
for j from l2 to u2 do z[j]*:= d[j] od ;
for j from u2-1 by -1 to l2
do z[j]+:= u[j]*z[j+1] od
);

rh:= residual(jac,u,f);
soll(l1,rh);
for i from l1+1 to u1
do for j from l2 to u2
do rh[i,j]-:= jac[i,j,-3]*rh[i-1,j ] +
( j<u2 ! jac[i,j,-2]*rh[i-1,j+1] ! 0.0 )
od ;
soll(i,rh)
od ;
du[u1,]:=rh[u1,];
for i from u1-1 by -1 to l1
do for j from l2 to u2
do du[i,j] := jac[i,j, 3]*du[i+1,j ] +
( j>l2 ! jac[i,j, 2]*du[i+1,j-1] ! 0.0 )
od ;
soll(i,du);
for j from l2 to u2
do du[i,j] := rh[i,j] - du[i,j] od
od ;

for i from l1 to u1 do
for j from l2 to u2 do
u[i,j]+:= du[i,j]
od
od
end ;

```

```

# illu decomposition procedure                                     #

proc illudec = ( netmat jac, ref netmat decomp ) void :
begin int l1= 1 lwb jac, u1= 1 upb jac,
      l2= 2 lwb jac, u2= 2 upb jac;
      int ip;
      real dd,l1,ii,1 dinv u;
      [l2:u2,-1:+1] real d;
      [l2:u2,-2:+2] real dinv;
      [l2:u2,-1:+2] real l dinv;
      heap [l1:u1,l2:u2,-1:+1] real dec;

      d[l2:u2,-1:+1]:= jac[l1,l2:u2,-1:+1];
      dd:= dec[l1,l2,0]:= 1.0/d[l2,0];
      for j from l2 to u2-1
      do dec[l1,j ,+1]:= -d[j ,+1]*dd;
        dec[l1,j+1,-1]:= l1:=-d[j+1,-1]*dd;
        dec[l1,j+1, 0]:= dd:= 1.0/( d[j+1, 0] + d[j,1]*l1 )
      od ;

      for i from l1 to u1-1
      do ip:= i+1;
        dinv[u2,0]:= ii:= dec[i,u2,0];
        for j from u2-1 by -1 to l2
        do dinv[ j,0]:= ii:= dec[i, j,0] +
          ii * dec[i,j,1]*dec[i,j+1,-1]
        od ;

        for k to 2 do
        for j from u2 by -1 to l2+k do
        dinv[ j , -k]:= dinv[ j , 1-k]*dec[i,j-k+1,-1];
        dinv[j-k, k]:= dinv[j-k+1,k-1]*dec[i,j-k ,+1]
        od od ;

        for k from -1 to 2 do
        for j from l2+(k=-1!1!0) to u2-(k=2!2!1)
        do l dinv[ j , k]:= jac[ip,j , -3]*dinv[ j , k ] +
          jac[ip,j , -2]*dinv[j+1,k-1]
        od ;

        l dinv[u2,k]:= jac[ip,u2,-3]*dinv[u2 ,k ] ( k<1 ! )
        od ;

        for k from -1 to 1 do
        for j from l2+(k=-1!1!0) to u2-(k=1!1!0)
        do l dinv u := l dinv[ j , k ]*jac[i,j+k ,3];
          l dinv u+:= l dinv[ j , k+1]*jac[i,j+k+1,2] ( j+k<u2 ! );
          d[j,k] := jac[ip,j,k] - l dinv u
        od od ;

      dd:= dec[ip,l2,0]:= 1.0/d[l2,0];
      for j from l2 to u2-1
      do dec[ip,j ,+1]:= -d[j ,+1]*dd;
        dec[ip,j+1,-1]:= l1:=-d[j+1,-1]*dd;
        dec[ip,j+1, 0]:= dd:= 1.0/( d[j+1, 0] + d[j,1]*l1 )
      od od ;
      decomp:= dec
end ;

```

```

# linear algebra solution procedure #

proc mgm = ( ref [] netmat lh, ref [] net uh,fh,
            int itmax,p,q,s,t,
            proc ( ref netmat , netmat , net , net ) void relax,
            ref [] netmat decomp, ref int itused,
            proc ( int , netmat , net , net ) bool goon mgm,
            proc ( int , string ) void fail) void :
begin int l= upb uh, r = s;
      ref [] netmat lhdec =
          ( decomp ::= ref [] netmat ( nil )
            ! loc [0:l] netmat ! decomp );

proc mg = ( int l) void :
# one multigrid cycle on level l #
if l = 0
then relax(lhdec[0],lh[0],uh[0],fh[0])
else # pre-relaxation #
to p do relax(lhdec[l],lh[l],uh[l],fh[l]) od ;

# coarse grid correction #
fh[l-1]:= lin weight( residual (lh[l],uh[l],fh[l]) );
zero uh[l-1];
to (l-1!t:s) do mg (l-1) od ;
uh[l] += lin int pol ( uh[l-1] );

# post-relaxation #
to q do relax(lhdec[l],lh[l],uh[l],fh[l]) od
fi ;

int err = # check consistency data #
( lwb uh /= 0 or lwb fh /= 0 or lwb lh /= 0
  or upb fh /= 1 or upb lh /= 1 ! 1
!: netmat l1 = lh[l];
  3 lwb l1 /=-3 or 3 upb l1 /= 3 ! 2
!: net ff = fh[l];
  int l1 := 1 lwb ff, u1 := 1 upb ff,
  l2 := 2 lwb ff, u2 := 2 upb ff;
  l1 /= 1 lwb l1 or u1 /= 1 upb l1 or
  l2 /= 2 lwb l1 or u2 /= 2 upb l1 ! 3
!: int tpl = 2**l;
  l1 mod tpl /=0 or u1 mod tpl/=0 or
  l2 mod tpl /=0 or u2 mod tpl/=0 ! 4
!: l1:= l1 over tpl; u1:= u1 over tpl;
  l2:= l2 over tpl; u2:= u2 over tpl;
  ( itused <= 0
  ! uh[0]:= zero heap [l1:u1,l2:u2] real
  ); s <= 0 or s > 3 or t <= 0 ! 5
!: itmax<0 or p<0 or q<0 ! 6
!: lwb lhdec /= 0 or upb lhdec /=1 ! 7
! 0 );
( err>0 ! fail ( err," mgm "));

if itused < 0 # no coarse operators available #
then # create galerkin approximations #
for i from l by -1 to 1
do lh[i-1]:= rap(lh[i]);
  fh[i-1]:= lin weight(fh[i])
od ; itused:= 0
fi ;

```



```

    if itused = 0      # no initial estimate available #
    then for i from 0 to l
        do lhdec[i]:= nil od ;

        # apply full multigrid #
        to t do mg(0) od ;
        for k to l-1
            do uh[k]:= sqr int pol (uh[k-1]);
              to r do mg (k) od
            od ; uh[l]:= sqr int pol (uh[l-1]);
        goon mgm (itused,lh[l],uh[l],fh[l])
    fi ;

    to itmax          # multigrid iteration          #
    while mg (l); itused += 1;
        goon mgm (itused, lh[l], uh[l], fh[l])
    do skip od
end ;

# black box solution procedure          #

proc solve sys =( int l, ref netmat lh, ref net uh, fh) void :
# solves the linear system lh*uh = fh          #
([0:l] netmat matrix; [0:l] net rhs,solution;
matrix[l]:= lh; rhs[l]:= fh;
mgm(matrix,solution,rhs,mgitmax,mgp,mgq,mgq,mgq,mgrelax,
nil , loc int := -1, mgm goon, fail);
uh:= solution[l] );

# default global parameters          #

bool symmetric:= false , backward:= false ,
reverse := false , zebra := false ;
int mgitmax := 8,
mgp:= 0 , mgq:= 1,
mgs:= 1 , mgt:= 1;
proc ( ref netmat , netmat , net , net ) void
mgrelax := illu relax;
proc mgm goon:= ( int itnum, netmat lh, net uh, fh) bool :
true ;
proc fail := ( int n, [] char text) void :
( print((newline,text,n,newline)); stop);

#example program #

int l:= 4;

netmat matrix := loc [0:2**1,0:2**1,-3:3] real ;
net solution, rhs := loc [0:2**1,0:2**1 ] real ;

read((matrix,rhs));
solve sys (l,matrix,solution,rhs);
print(solution)

end

```

## 6.2 Appendix 2

In this second appendix we give the user interfaces of the FORTRAN subroutines MGD1V (or MGD1S) and MGD5V (or MGD5S). We include also examples of a calling program. A tape with the complete programs can be obtained from the authors.

```

SUBROUTINE MGDIV(A,U,RHS,UB,US,TEMP,LEVELS,NXC,NYC,NXF,NYF,NF,NM,
.ISTART,MAXIT,TOL,IOUT,RESNO)
COMMON /POI/ NGP(12),NGRIDX(12),NGRIDY(12)
COMMON /CPU/ CP(9)
DIMENSION A(NM,7),U(NM),UB(NF),RHS(NM),US(NM),TEMP(NXF),IOUT(5)

```

```

C-----
C
C PURPOSE
C-----
C
C THIS PROGRAM SOLVES A USER PROVIDED 7-POINT DIFFERENCE
C EQUATION ON A RECTANGULAR GRID.
C
C MATHEMATICAL METHOD
C-----
C
C SAWTOOTH MULTIGRID CYCLING
C (I.E. ONE SMOOTHING-SWEEP AFTER EACH COARSE GRID CORRECTION)
C WITH SMOOTHING BY INCOMPLETE CROUT-DECOMPOSITION,
C 7-POINT PROLONGATION AND RESTRICTION,
C GALERKIN APPROXIMATION OF COARSE GRID MATRICES.
C
C*****
C
C          **** PARAMETERS ****
C
C*****
C
C ---
C (INPUT DATA - SIZE OF PROBLEM)
C LEVELS      NUMBER OF LEVELS IN MULTIGRID METHOD
C              SHOULD BE .GE.2 AND .LE.12
C NXC,NYC     NUMBER OF VERTICAL, HORIZONTAL GRID-LINES
C              ON COARSEST GRID
C NXF,NYF     NUMBER OF VERTICAL, HORIZONTAL GRID-LINES
C              ON FINEST GRID
C NF          NUMBER OF GRID-POINTS OF FINEST GRID
C NM          NUMBER OF GRID-POINTS ON ALL GRIDS TOGETHER
C
C NOTE THAT THE FOLLOWING RELATIONS SHOULD HOLD,
C-----
C          NF=NXF*NYF
C          NXF=(NXC-1)*(2**(LEVELS-1))+1
C          NYF=(NYC-1)*(2**(LEVELS-1))+1
C
C THE PROGRAM CHECKS THE CONSISTENCY OF THESE DATA
C
C EXAMPLES
C-----
C
C          LEVELS =   2   3   4   5   6   7
C          NXC    =   3   3   3   3   3   3
C          NYC    =   3   3   3   3   3   3
C          NXF    =   5   9  17  33  65 129
C          NYF    =   5   9  17  33  65 129
C          NF     =  25  81 289 1089 4225 16641
C          NM     =  34 115 404 1493 5718 22359
C
C          LEVELS =   2   3   4   5   6   7
C          NXC    =   5   5   5   5   5   5
C          NYC    =   5   5   5   5   5   5
C          NXF    =   9  17  33  65  129  257
C          NYF    =   9  17  33  65  129  257
C          NF     =  81 289 1089 4225 16641 66049
C          NM     = 106 395 1484 5709 22350 88399
C
C

```

```

C ---
C ISTART      (INPUT)
C             =1 IF THE USER PROVIDES AN INITIAL ESTIMATE
C             OF THE SOLUTION IN UB
C             =0 IF NO INITIAL ESTIMATE IS PROVIDED IN UB
C ---
C MAXIT       (INPUT)
C             MAXIMUM NUMBER OF MULTIGRID ITERATIONS
C ---
C TOL         (INPUT)
C             TOLERANCE DESIRED BY THE USER, TOL IS A BOUND OF THE
C             L2-NORM OF THE RESIDUAL
C             REMARK IF EITHER MAXIT ITERATIONS OR THE TOLERANCE HAVE
C             ----- BEEN ACHIEVED, THEN MULTIGRID CYCLING IS STOPPED.
C ---
C IOUT        (INPUT)
C             INTEGER ARRAY DIMENSIONED AS IOUT(5) THAT CONTROLS
C             THE AMOUNT OF OUTPUT DESIRED BY THE USER.
C             SMALLER IOUT-VALUES MEAN LESS OUTPUT,
C             POSSIBLE VALUES ARE ,
C             IOUT(1)=1 CONFIRMATION OF INPUT DATA
C                     0 NONE
C             IOUT(2)=2 MATRICES AND RIGHT-HAND SIDES ON ALL LEVELS
C                     1 MATRIX AND RIGHT-HAND SIDE ON HIGHEST LEVEL
C                     0 NONE
C             IOUT(3)=2 MATRIX-DECOMPOSITIONS ON ALL LEVELS
C                     1 MATRIX-DECOMPOSITION ON HIGHEST LEVEL
C                     0 NONE
C             IOUT(4)=3 NORMS OF RESIDUALS, REDUCTION FACTORS,
C                     FINAL RESIDUAL, FINAL SOLUTION
C                     2 NORMS OF RESIDUALS, REDUCTION FACTORS,
C                     FINAL RESIDUAL
C                     1 NORMS OF RESIDUALS, REDUCTION FACTORS
C                     0 NONE
C             IOUT(5)=1 THE TIME SPENT IN VARIOUS SUBROUTINES
C                     0 NONE
C                     REMARK CLOCK ROUTINES ARE NOT STANDARD
C                     ----- FORTRAN. TO OBTAIN TIMINGS THE USER
C                     SHOULD ADAPT THE SUBROUTINE TIMING,
C                     IT SHOULD DELIVER THE CPU-TIME ELAPSED.
C ---
C A           (INPUT)
C             REAL ARRAY DIMENSIONED AS A(NM,7)
C             THE USER HAS TO INITIALIZE A( 1,1), ..., A( 1,7)
C                                     .
C                                     A( K,1)   A( K,7)
C                                     .
C                                     A(NF,1), ..., A(NF,7)
C             WITH THE MATRIX CORRESPONDING TO THE FINEST GRID.
C             THE ORDERING OF THE POINTS IN THE GRID IS AS FOLLOWS
C             THE SUBSCRIPT K=(J-1)*NXF+I CORRESPONDS TO THE POINT
C
C             (X,Y) = ( I*H , J*H )
C                       X       Y
C                       I=1, ..., NXF  J=1, ..., NYF

```

THE 7-POINT DIFFERENCE MOLECULE AT THE POINT WITH  
SUBSCRIPT  $K=(J-1)*NXF+I$  IS POSITIONED IN THE X,Y-PLANE  
AS FOLLOWS

```

      Y, J
      +
      +
      +   A(K,6)   A(K,7)
      +   .       .
      +   A(K,3)   A(K,4)   A(K,5)
      +   .       .       .
      +           A(K,1)   A(K,2)
      +           .       .
      +
      O+ + + + + + + + + + + X, I

```

IMPORTANT THE USER HAS TO PROVIDE THE MATRIX A ONLY ON THE FINEST  
----- GRID.

IMPORTANT THE USER HAS TO TAKE CARE THAT PARTS OF THE MOLECULES  
----- OUTSIDE THE DOMAIN ARE INITIALIZED TO ZERO, OTHERWISE

-----  
WRONG RESULTS ARE PRODUCED.

IMPORTANT THE COEFFICIENT MATRIX A IS OVERWRITTEN BY THE PROGRAM.  
----- AFTER A CALL OF MGDIV (DECOMP), A CONTAINS THE INCOMPLETE  
CROUT DECOMPOSITIONS.

---  
RHS (INPUT)  
REAL ARRAY DIMENSIONED AS RHS(NM)  
THE USER HAS TO INITIALIZE RHS(1),...,RHS(NF) WITH  
THE RIGHT-HAND SIDE OF THE EQUATION.  
THE ORDERING IS THE SAME AS INDICATED FOR ARRAY A.  
IMPORTANT THE USER HAS TO PROVIDE THE RIGHT-HAND SIDE OF THE  
----- DISCRETIZED EQUATION ONLY ON THE FINEST GRID

---  
U (OUTPUT)  
REAL ARRAY DIMENSIONED AS U(NM)  
CONTAINS THE (APPROXIMATE) NUMERICAL SOLUTION AFTER A  
CALL OF MGDIV.

---  
UB (WORKSPACE/INPUT)  
REAL ARRAY DIMENSIONED AS UB(NF)  
IS USED AS A SCRATCH ARRAY. IF ISTART=1 THEN UB(1),....  
..,UB(NF) SHOULD CONTAIN AN INITIAL ESTIMATE OF THE  
SOLUTION PROVIDED BY THE USER.  
AFTER A CALL OF MGDIV, UB CONTAINS THE RESIDUAL OF THE  
THE NUMERICAL SOLUTION.

---  
US (WORKSPACE)  
REAL ARRAY DIMENSIONED AS US(NM)  
IS USED AS A SCRATCH ARRAY

---  
TEMP (WORKSPACE)  
REAL ARRAY DIMENSIONED AS TEMP(NXF)  
IS USED AS A (SMALL) SCRATCH ARRAY.  
IF THE SCALAR VERSION OF SUBROUTINE SOLVE (DENOTED BY  
COMMENT CARDS BEGINNING WITH CSC) IS USED THEN IT IS  
SUFFICIENT TO DIMENSION TEMP AS TEMP(1).

---  
RESNO (OUTPUT)  
THIS VARIABLE CONTAINS THE L2-NORM OF THE RESIDUAL AT  
THE END OF EXECUTION OF MGDIV.

```

C-----
C   THIS IS AN EXAMPLE OF A MAIN PROGRAM USING MGDIV
C-----
C
C   ACTUAL USER PROVIDED DIMENSION STATEMENTS,
C
C   DIMENSION A(88399,7),RHS(88399),U(88399),US(88399),UB(66049),
C   .TEMP(257),IOUT(5)
C
C   USER DATA STATEMENTS,
C
C   DATA NXC, NYC, NXF, NYF/5,5,257,257/
C   DATA LEVELS, NM, NF/7,88399,66049/
C   DATA MAXIT, ISTART/10,0/
C   DATA IOUT(1), IOUT(2), IOUT(3), IOUT(4), IOUT(5)/1,0,0,1,1/
C
C   PROBLEM SET UP
C
C   CALL MATRHS(A,RHS,NM,NXF,NYF)
C*****
C   MATRHS IS A SUBROUTINE WHICH FILLS THE MATRIX AND THE RIGHT-HAND
C   SIDE, IT DOES NOT BELONG TO THE PACKAGE AND IS ONLY AN EXAMPLE.
C*****
C
C   SOLUTION OF THE LINEAR SYSTEM
C
C   CALL MGDIV(A,U,RHS,UB,US,TEMP,LEVELS,NXC, NYC, NXF, NYF, NF, NM,
C   . ISTART, MAXIT, 0.0, IOUT, RESNO)
C
C   POSSIBLE REFINEMENT OF THE SOLUTION, 5 MORE ITERATIONS
C
C   CALL CYCLES(A,U,RHS,UB,US,TEMP,LEVELS,NXF,NF,NM,1,5,0.0,IOUT,
C   .RESNO)
C
C   POSSIBLE REFINEMENT UNTIL RESIDUAL NORM .LT. 1.0E-12
C
C   CALL CYCLES(A,U,RHS,UB,US,TEMP,LEVELS,NXF,NF,NM,1,30,1.0E-12,IOUT,
C   .RESNO)
C
C   STOP
C   END

```

```

SUBROUTINE MGD5V(A,V,RHS,VB,LDU,WORK,LEVELS,NXC, NYC,NXF,NYF,
.      NE,NM, ISTART,MAXIT,TOL,IOUT,RESNO)
COMMON /POI/  NGP(12),NGRIDX(12),NGRIDY(12)
COMMON /CPU/  CP(10)
REAL LDU
DIMENSION A(NM,7),V(NM),VB(NM),RHS(NM),LDU(NM,3),
.      WORK(NXF,9),IOUT(5)

```

---

```

C
C
C PURPOSE
C -----
C

```

```

C THIS PROGRAM SOLVES A USER PROVIDED 7-POINT DIFFERENCE
C EQUATION ON A RECTANGULAR GRID.
C

```

```

C MATHEMATICAL METHOD
C -----
C

```

```

C SAWTOOTH MULTIGRID CYCLING
C (I.E. ONE SMOOTHING-SWEEP AFTER EACH COARSE GRID CORRECTION)
C WITH SMOOTHING BY INCOMPLETE LINE LU-DECOMPOSITION,
C 7-POINT PROLONGATION AND RESTRICTION,
C GALERKIN APPROXIMATION OF COARSE GRID MATRICES.
C

```

---

```

C*****
C

```

```

C      ****   PARAMETERS   ****
C

```

---

```

C*****
C

```

```

C ---
C      (INPUT DATA - SIZE OF PROBLEM)
C  LEVELS           NUMBER OF LEVELS IN MULTIGRID METHOD
C                   SHOULD BE .GE.3 AND .LE.12
C  NXC,NYC          NUMBER OF VERTICAL, HORIZONTAL GRID-LINES
C                   ON COARSEST GRID, NXC SHOULD BE .GE.5
C                                     AND NYC SHOULD BE .GE.3
C  NXF,NYF          NUMBER OF VERTICAL, HORIZONTAL GRID-LINES
C                   ON FINEST GRID
C  NF               NUMBER OF GRID-POINTS OF FINEST GRID
C  NM               NUMBER OF GRID-POINTS ON ALL GRIDS TOGETHER
C
C                   SEE COMMENTS IN MGD1V FOR FURTHER DETAILS.
C -----
C

```

```

C  ISTART           (INPUT)
C  MAXIT            (INPUT)
C  TOL              (INPUT)
C  IOUT             (INPUT)
C  A                (INPUT)
C  RHS              (INPUT)
C

```

```

C                   THESE INPUT PARAMETERS HAVE THE SAME MEANING AS IN MGD1V
C -----
C                   THE ONLY DIFFERENCE IS THAT THE ARRAY A WILL NEVER BE
C                   OVERWRITTEN BY MGD5V.
C

```

```

C ---
C  LDU              (OUTPUT)
C                   REAL ARRAY DIMENSIONED AS LDU(NM,3)
C                   LDU CONTAINS DECOMPOSITIONS OF ALL TRIDIAGONAL BLOCKS D
C

```

```

C ---
C   V          (INPUT/OUTPUT)
C             REAL ARRAY DIMENSIONED AS V(NM)
C             IF ISTART=1 THEN V(1),...,V(NF) SHOULD CONTAIN AN
C             INITIAL ESTIMATE OF THE SOLUTION PROVIDED BY THE USER.
C             IF ISTART=0 THEN V IS INITIALIZED TO ZERO.(SUBR. PREPAR)
C             AFTER A CALL OF MGD5V, V CONTAINS THE (APPROXIMATE)
C             NUMERICAL SOLUTION.
C ---
C   VB         (WORKSPACE/OUTPUT)
C             REAL ARRAY DIMENSIONED AS VB(NF)
C             AFTER A CALL OF MGD5V, VB CONTAINS THE RESIDUAL OF THE
C             NUMERICAL SOLUTION V.
C ---
C   WORK       (WORKSPACE)
C             REAL ARRAY DIMENSIONED AS WORK(NXF,9)
C             IS USED AS A (SMALL) SCRATCH ARRAY
C ---
C   RESNO      (OUTPUT)
C             THIS VARIABLE CONTAINS THE L2-NORM OF THE RESIDUAL AT
C             THE END OF EXECUTION OF MGD5V.
C -----

```

```

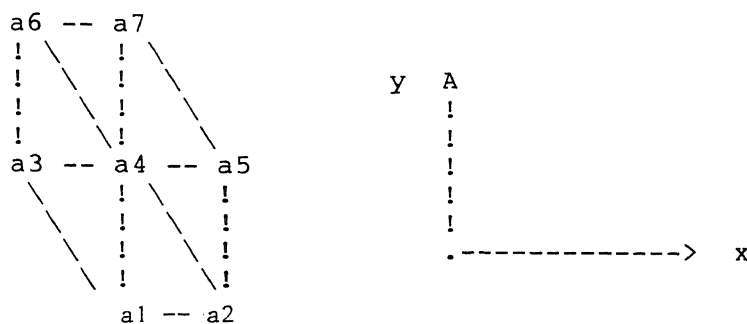
C -----
C   THIS IS AN EXAMPLE OF A MAIN PROGRAM USING MGD5V
C -----
C   ACTUAL USER PROVIDED DIMENSION STATEMENTS,
C
C   REAL LDU
C   DIMENSION A(88399,7),RHS(88399),V(88399),VB(88399),
C   .LDU(88399,3),WORK(257,9),IOUT(5)
C
C   USER DATA STATEMENTS,
C
C   DATA NXC, NYC, NXF, NYF/5,5,257,257/
C   DATA LEVELS, NM, NF/7,88399,66049/
C   DATA MAXIT, ISTART/10,0/
C   DATA IOUT(1), IOUT(2), IOUT(3), IOUT(4), IOUT(5)/1,0,0,1,1/
C
C   PROBLEM SET UP
C
C   CALL MATRHS(A,RHS,NM,NXF,NYF)
C *****
C   MATRHS IS A SUBROUTINE WHICH FILLS THE MATRIX AND THE RIGHT-HAND
C   SIDE, IT DOES NOT BELONG TO THE PACKAGE AND IS ONLY AN EXAMPLE.
C *****
C
C   SOLUTION OF THE LINEAR SYSTEM
C
C   CALL MGD5V(A,V,RHS,VB,LDU,WORK,LEVELS,
C   .           NXC, NYC, NXF, NYF, NF, NM, ISTART, MAXIT, 0.0, IOUT, RESNO)
C
C   POSSIBLE REFINEMENT OF THE SOLUTION, 5 MORE ITERATIONS
C
C   CALL CYCLES(A,V,RHS,VB,LDU,WORK,LEVELS,NXF,NF,NM,
C   .           1,5,0.0,IOUT,RESNO)
C   POSSIBLE REFINEMENT UNTIL RESIDUAL NORM .LT. 1.0E-12
C
C   CALL CYCLES(A,V,RHS,VB,LDU,WORK,LEVELS,NXF,NF,NM,
C   .           1,30,1.0E-12,IOUT,RESNO)
C
C   STOP
C   END

```



## 6.3 Appendix 3

In this appendix we give a full description in FORTRAN of our implementation of the ILLU-decomposition. First we give a brief description of that decomposition and the corresponding relaxation sweep. Let the seven diagonal matrix A correspond with the following molecule:



Let the matrix A be decomposed in block tridiagonal form:

$$A = L + D + U = \begin{pmatrix} D_1 & U_1 & & & & & \\ L_2 & D_2 & U_2 & & & & \\ & L_3 & D_3 & U_3 & & & \\ & & & & L_i & D_i & U_i \\ & & & & & L_n & D_n \end{pmatrix}$$

$L_i$   $i = 2$  (1)  $n$  corresponds with  $a_1$  and  $a_2$ ,  
 $D_i$   $i = 1$  (1)  $n$  corresponds with  $a_3$ ,  $a_4$  and  $a_5$ ,  
 $U_i$   $i = 1$  (1)  $n-1$  corresponds with  $a_6$  and  $a_7$ .

Then the ILLU-decomposition is defined by  $L_j$ ,  $\bar{D}_j$ ,  $U_j$  with  
 $\bar{D}_1 = D_1$ ,

$$\bar{D}_j = D_j - \text{tridiag} \left( L_j, \bar{D}_{j-1}^{-1}, U_{j-1} \right), \text{ for } j = 2(1)n.$$

The tridiagonal matrix  $\bar{D}_j$  is stored by means of its exact decomposition  $L_j, D_j, U_j$ . ( $L_j$  and  $U_j$  are bidiagonal,  $D_j$  is a main diagonal, the main diagonals of  $L_j$  and  $U_j$  are equal to one).

Let  $u^{(i)}$  be an approximate solution of  $Au=f$ , then an ILLU-relaxation sweep reads:

step1: compute  $r := f - A u^{(i)}$ ;  
 step2: solve  $(L + \bar{D}) \bar{D}^{-1} (D + U) v = r$ ;  
 step3:  $u^{(i+1)} = u^{(i)} + v$ .

SUBROUTINE DECOMP(A1,A2,A3,A4,A5,A6,A7,N,M,NM)

```

C-----
C   INCOMPLETE CROUT-DECOMPOSITION (ILU-DECOMPOSITION) OF THE SEVENDIA
C   GONAL MATRIX A REPRESENTED BY A1,A2,A3,A4,A5,A6,A7.
C   A IS OVERWRITTEN BY ITS DECOMPOSITION.
C   THE MAIN DIAGONAL OF L IS ONE EVERYWHERE, THE OTHER DIAGONALS OF L
C   ARE STORED IN A1, A2, A3.
C   THE DIAGONALS OF U ARE STORED IN  A4, A5, A6, A7.
C   M IS THE NUMBER OF GRIDPOINTS IN THE X-DIRECTION,
C   N IS THE NUMBER OF GRIDPOINTS IN THE Y-DIRECTION,
C   NM=N*M.
C
C   NOTE  THE LOOPS 6, 10, 20, 30, 40, 50, 60, 400 ARE AUTOMATICALLY
C   ----  VECTORIZED.
C         THE LOOPS 5 AND 55 ARE RECURSIVE AND WILL THEREFORE NOT BE
C         VECTORIZED.

```

```

C-----
      DIMENSION A1(NM),A2(NM),A3(NM),A4(NM),A5(NM),A6(NM),A7(NM)
      A4J=A4(1)
      DO 5 J=2,M
        A3(J)=A3(J)/A4J
        A4(J)=A4(J)-A3(J)*A5(J-1)
        A4J=A4(J)
5     CONTINUE
      DO 6 J=2,M
        A6(J)=A6(J)-A3(J)*A7(J-1)
6     CONTINUE
      M1=M-1
      JB=1
      JE=M
      DO 100 K=2,N
        JB=JB+M
        JE=JE+M
        DO 10 J=JB,JE
          A1(J)=A1(J)/A4(J-M)
10    CONTINUE
        DO 20 J=JB,JE
          A2(J)=(A2(J)-A1(J)*A5(J-M))/A4(J-M1)
20    CONTINUE
        DO 30 J=JB,JE
          A3(J)=A3(J)-A1(J)*A6(J-M)
30    CONTINUE
        DO 40 J=JB,JE
          A4(J)=A4(J)-A2(J)*A6(J-M1)-A1(J)*A7(J-M)
40    CONTINUE
        DO 50 J=JB,JE
          A5(J)=A5(J)-A2(J)*A7(J-M1)
50    CONTINUE
        A4J=A4(JB-1)
        DO 55 J=JB,JE
          A3(J)=A3(J)/A4J
          A4(J)=A4(J)-A3(J)*A5(J-1)
          A4J=A4(J)
55    CONTINUE
        DO 60 J=JB,JE
          A6(J)=A6(J)-A3(J)*A7(J-1)
60    CONTINUE
100   CONTINUE

```

```

C-----
C   FOR ILU-RELAXATION THE RECIPROCAL OF A4 IS NEEDED, NOT A4 ITSELF.
C-----

```

```

      DO 400 JJ=1,NM,65535
        JJE=(JJ-1)+MIN0(65535,NM-(JJ-1))
        DO 400 J=JJ,JJE
          A4(J)=1.0/A4(J)
400   CONTINUE
      RETURN
      END

```

```

SUBROUTINE ILLUDC (A,DIMA,L,D,U,NX,NY,NXY,WORK)

```

```

-----
C
C   INCOMPLETE LINE LU (ILLU-DECOMPOSITION) OF THE SEVENDIAGONAL
C   MATRIX A. A REMAINS INTACT, L D AND U ARE FILLED IN WITH THE
C   DECOMPOSITIONS OF

```

$$D_{JJ} = 1 \quad (J = 1(1)NY)$$

```

C
C   NX IS THE NUMBER OF GRIDPOINTS IN THE X-DIRECTION,
C   NY IS THE NUMBER OF GRIDPOINTS IN THE Y-DIRECTION,
C   NXY=NX*NY
C
-----

```

```

C
C   INTEGER DIMA
C   REAL L
C   DIMENSION A(DIMA,7),L(NXY),D(NXY),U(NXY),WORK(NX,9)
C   CALL TRIDEC(A(1,3),A(1,4),A(1,5),L,D,U,NX)
C   NPOLD=1
C   DO 100 J=2,NY
C   NPNEW=NPOLD+NX
C   CALL BLOCKS(A(NPOLD,1),A(NPNEW,1),DIMA,
C   .           L(NPOLD),D(NPOLD),U(NPOLD),
C   .           L(NPNEW),D(NPNEW),U(NPNEW),NX,
C   .           WORK(1,1),WORK(1,2),WORK(1,3),WORK(1,4),WORK(1,5),
C   .           WORK(1,6))
C   NPOLD=NPNEW
100 CONTINUE
RETURN
END
SUBROUTINE TRIDEC(DM,DZ,DP,LJ,DJ,UJ,NX)

```

```

-----
C
C   PERFORMS DECOMPOSITION OF A TRIDIAGONAL MATRIX REPRESENTED BY DM,
C   DZ, DP.
C   THE DECOMPOSITION CONSISTS OF A LOWER TRIANGULAR BIDIAGONAL MATRIX
C   LJ, AN UPPER TRIANGULAR BIDIAGONAL MATRIX UJ AND AN ONE DIAGONAL
C   MATRIX DJ, THE MAIN DIAGONALS OF LJ AND UJ EQUAL ONE.
C   NX IS THE NUMBER OF POINTS IN THE X-DIRECTION.
C

```

```

C   NOTE LOOP 20 IS AUTOMATICALLY VECTORIZED.
C   ---- LOOP 10 IS RECURSIVE AND WILL THEREFORE NOT BE VECTORIZED.
C
-----

```

```

C
C   REAL LJ
C   DIMENSION DM(NX),DZ(NX),DP(NX),LJ(NX),DJ(NX),UJ(NX)
C   DJ(1)=1.0/DZ(1)
C   DJIM1=DJ(1)
C   DO 10 I=2,NX
C   LJ(I)=-DM(I)*DJIM1
C   DJ(I)=1.0/(DZ(I)+LJ(I)*DP(I-1))
C   DJIM1=DJ(I)
10 CONTINUE
C   NX1=NX-1
C   DO 20 I=1,NX1
C   UJ(I)=-DP(I)*DJ(I)
20 CONTINUE
RETURN
END

```

```

SUBROUTINE BLOCKS(AJML,AJ,DIMA, LJML,DJML,UJML, LJ,DJ,UJ,NX,
      QM2,QM1,QZE,QP1,QP2, LD)
C-----
C   INCOMPLETE LINE LU DECOMPOSITION (ILLU-DECOMPOSITION) OF J-TH ROW
C   OF BLOCKS OF THE SEVENDIAGONAL MATRIX A.
C   AJ IS J TH ROW OF BLOCKS OF A,
C   AJML IS (J-1) TH ROW OF BLOCKS OF A,
C   LJML, DJML, UJML ARE (J-1) TH ROWS OF L, D, U WHICH REPRESENT
C   BIDIAGONAL MATRICES (MAIN DIAGONALS EQUAL ONE) WHICH PRODUCT IS
C   -
C   D
C   (J-1)
C   LJ, DJ, UJ BECOME THE J TH ROWS OF L, D, U AFTER A CALL OF BLOCKS.
C   NX IS THE NUMBER OF GRIDPOINTS IN THE X-DIRECTION.
C   QM2,QM1,QZE,QP1,QP2,LD ARE WORK ARRAYS.
C
C   NOTE THE LOOPS 10, 30, 40, 51, 52, 53, 54, 60, 70, 80 ARE AUTOMA-
C   ---- TICALLY VECTORIZED.
C   LOOP 20 IS RECURSIVE AND WILL THEREFORE NOT BE VECTORIZED.
C-----
C   INTEGER DIMA
C   REAL LJML,LJ,LD
C   DIMENSION AJML(DIMA,7),AJ(DIMA,7),LJML(NX),DJML(NX),UJML(NX),
C   . LJ(NX),DJ(NX),UJ(NX),
C   . QM2(NX),QM1(NX),QZE(NX),QP1(NX),QP2(NX),
C   . LD(NX,4)
C-----
C                                     - -1
C   FIRST STEP - COMPUTATION OF 5-DIAG( D      ),
C                                     J-1
C   RESULTING DIAGONALS ARE QM2, QM1, QZE, QP1, QP2
C-----
NX1=NX-1
NX2=NX-2
DO 10 I=1,NX1
QZE(I)=UJML(I)*LJML(I+1)
10 CONTINUE
QZE(NX)=DJML(NX)
QZEIPL=QZE(NX)
DO 20 II=1,NX1
I=NX-II
QZE(I)=DJML(I)+QZE(I)*QZEIPL
QZEIPL=QZE(I)
20 CONTINUE
DO 30 I=2,NX1
QM1(I)=LJML(I)*QZE(I)
QP1(I)=UJML(I)*QZE(I+1)
30 CONTINUE
QP1(1)=UJML(1)*QZE(2)
QM1(NX)=LJML(NX)*QZE(NX)
DO 40 I=3,NX2
QM2(I)=LJML(I-1)*QM1(I)
QP2(I)=UJML(I)*QP1(I+1)
40 CONTINUE
QP2(1)=UJML(1)*QP1(2)
QP2(2)=UJML(2)*QP1(3)
QM2(NX1)=LJML(NX2)*QM1(NX1)
QM2(NX)=LJML(NX1)*QM1(NX)

```

```

C-----
C
C          SECOND STEP - COMPUTATION OF 4 DIAGONALS OF  $\begin{matrix} & & - & -1 \\ L & D & & \\ J & J-1 & & \end{matrix}$ 
C-----
C
C      QM1(1)=0.0
C      QM2(2)=0.0
C      QP2(NX1)=0.0
C      QP1(NX)=0.0
C      DO 51 I=1,NX1
C      LD(I,1)=AJ(I,1)*QM1(I)+AJ(I,2)*QM2(I+1)
51 CONTINUE
C      DO 52 I=1,NX1
C      LD(I,2)=AJ(I,1)*QZE(I)+AJ(I,2)*QM1(I+1)
52 CONTINUE
C      DO 53 I=1,NX1
C      LD(I,3)=AJ(I,1)*QP1(I)+AJ(I,2)*QZE(I+1)
53 CONTINUE
C      DO 54 I=1,NX1
C      LD(I,4)=AJ(I,1)*QP2(I)+AJ(I,2)*QP1(I+1)
54 CONTINUE
C      LD(NX,1)=AJ(NX,1)*QM1(NX)
C      LD(NX,2)=AJ(NX,1)*QZE(NX)
C-----
C
C          THIRD AND FOURTH STEP - COMPUTATION OF  $\begin{matrix} & & - & & - & -1 \\ D & = & D & - & 3\text{-DIAG} & ( L & D & & U & ) \\ J & & J & & & & J & J-1 & & J-1 \end{matrix}$ 
C
C          -
C      D IS REPRESENTED BY QM1, QZE, QP1
C          J
C-----
C
C      DO 60 I=2,NX
C      QM1(I)=AJ(I,3)-LD(I,1)*AJM1(I-1,7)-LD(I,2)*AJM1(I,6)
60 CONTINUE
C      DO 70 I=1,NX1
C      QZE(I)=AJ(I,4)-LD(I,2)*AJM1(I,7)-LD(I,3)*AJM1(I+1,6)
70 CONTINUE
C      DO 80 I=1,NX2
C      QP1(I)=AJ(I,5)-LD(I,3)*AJM1(I+1,7)-LD(I,4)*AJM1(I+2,6)
80 CONTINUE
C      QZE(NX)=AJ(NX,4)-LD(NX,2)*AJM1(NX,7)
C      QP1(NX1)=AJ(NX1,5)-LD(NX1,3)*AJM1(NX,7)
C-----
C
C          FIFTH STEP - COMPUTATION OF DECOMPOSITION  $\begin{matrix} & & & & - \\ L & , & D & , & U & \text{ OF } & D \\ J & & J & & J & & J \end{matrix}$ 
C-----
C
C      CALL TRIDEC(QM1,QZE,QP1,LJ,DJ,UJ,NX)
C      RETURN
C      END

```

## 7. References

1. Barkai, D. & Brandt, A., Vectorized Multigrid Poisson Solver for the CDC CYBER 205 In: Procs. Int. Multigrid Conference, Copper Mountain, Colorado, April 1983, to appear.
2. Dendy, J.E.Jr., Black Box multigrid. J.Comp.Phys. 48 (1982) 366-386
3. Dendy, J.E.Jr., Black Box multigrid for non-symmetric problems. In: Procs. Int. Multigrid Conference, Copper Mountain, Colorado, April 1983, to appear.
4. Foerster, H. & Witsch, K. Multigrid software for the solution of elliptic problems on rectangular domains: MG00. In: Multigrid methods (W.Hackbusch & U.Trottenberg eds) pp.427-460, Springer LNM 960, Springer Verlag, 1982.
5. Hemker, P.W., On the comparison of Line-Gauss Seidel and ILU relaxation in multigrid algorithms. In: Computational and asymptotic methods for boundary and interior layers. ( J.J.H. Miller ed.) pp.269-277, Boole Press, Dublin, 1982.
6. Hemker, P.W., Multigrid methods for problems with a small parameter. In: Numerical Analysis (D.F. Griffiths ed.), Procs. Dundee Conference 1983, Springer LNM 1..., Springer Verlag, (to appear 1984).
7. Hemker, P.W., Kettler, R., Wesseling, P. & De Zeeuw, P.M., Multigrid Methods: development of fast solvers. In: Procs. Int. Multigrid Conference, Copper Mountain, Colorado, April 1983, to appear.
8. Hemker, P.W., Wesseling, P. & De Zeeuw, P.M., A portable vector code for autonomous multigrid modules In: PDE Software: Modules, Interfaces and Systems. (B.Engquist ed.) pp. - , Procs. IFIP WG 2.5 Working Conference, North Holland Publ. Comp., (to appear 1984).
9. Kettler, R., Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. In: Multigrid methods (W.Hackbusch & U.Trottenberg eds) pp.502-534, Springer LNM 960, Springer Verlag, 1982.
10. Wesseling, P., A robust and efficient multigrid method. In: Multigrid methods (W.Hackbusch & U.Trottenberg eds) pp.614-630, Springer LNM 960, Springer Verlag, 1982.
11. Wesseling, P., Theoretical and practical aspects of a multigrid method. SIAM J.Sci. Stat. Comp. 3 (1982) 387-407.