stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE
(DEPARTMENT OF NUMERICAL MATHEMATICS)

NN 15/77 DECEMBER

B.P. SOMMEIJER

AN ALGOL 68 IMPLEMENTATION OF TWO SPLITTING METHODS FOR SEMI-DISCRETIZED PARABOLIC DIFFERENTIAL EQUATIONS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 65-04, 65M20

An ALGOL 68 implementation of two splitting methods for semi-discretized parabolic differential equations

by

B.P. Sommeijer

ABSTRACT

This note describes an implementation of two splitting methods for semi-discretized, non-linear parabolic equations in two dimensions. The underlying formulas are described in [1]. The implementation is provided with steplength and error control. An ALGOL 68 version of the implementation is available. Numerical results of this ALGOL 68 program, applied to two semi-discretized problems, are reported.

KEY WORDS & PHRASES: *Parabolic partial differential equations, Semi-discretization, Numerical software*

CONTENTS

1. INTRODUCTION

This report has been written as a contribution to a project of the Department of Numerical Mathematics of the Mathematical Centre to develop numerical algorithms for time-dependent partial differential equations. Here we confine ourselves to semi-discretized parabolic equations in two dimensions.

From the variety of non-linear splitting methods described in [1], we choose an alternating direction method of the PEACEMAN and RACHFORD type and the linehopscotch method suggested by GOURLAY. The alternating direction method is applied to five-point coupled equations, while the linehopscotch method is applied to nine-point coupled ones. Those two methods are implemented in ALGOL 68 programs and applied to two examples.

It is emphasized that the programs are not in a final state. They should be considered as research programs and can be used for comparison. The main purpose of this note is to give some first results.

2. THE UNDERLYING FORMULAS

In this section we shortly describe the underlying formulas which are more extensively discussed in [1].

The idea of splitting is to break down a complicated multi-dimensional process into a series of one-dimensional and less complicated processes.

Here we confine our considerations to initial-boundary-value problems for parabolic partial differential equations in two space dimensions. Applying the method of lines to discretize the space variables, we obtain in many cases a system of ordinary differential equations

$$\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y}), \tag{2.1}$$

with initial condition

$$\vec{y}(t_0) = \vec{y}_0.$$

Then, integration of (2.1) can be performed by using the two-stage formula

$$\vec{y}^{(1)}_{n+1} = \vec{y}_n + \tfrac{1}{2}h_n \vec{F}(t_n+\tfrac{1}{2}h_n, \vec{y}^{(1)}_{n+1}, \vec{y}_n),$$

(2.2)

$$\vec{y}_{n+1} = \vec{y}^{(1)}_{n+1} + \tfrac{1}{2}h_n \vec{F}(t_n+\tfrac{1}{2}h_n, \vec{y}^{(1)}_{n+1}, \vec{y}_{n+1}),$$

where $\vec{F}(t,\vec{v},\vec{w})$ is a function satisfying the relation

$$\vec{f}(t,\vec{y}) = \vec{F}(t,\vec{y},\vec{y}). \tag{2.3}$$

Scheme (2.2) is second order accurate for every choice of the function $\vec{F}$ and unconditionally stable provided that the Jacobian matrices $\partial\vec{F}/\partial\vec{v}$ and $\partial\vec{F}/\partial\vec{w}$ have negative eigenvalues (cf. [1]).

It is assumed that the components of the vectors $\vec{y}$ and $\vec{f}$ can be arranged in a two-dimensional array. Each array element, denoted by y[r,k] and f[r,k], is then associated to a grid-point of the two-dimensional grid covering the region under consideration. Such a grid is not necessarily rectangular, but may be of any shape, even containing "holes". These "holes" are considered as sets of gridpoints where no differential equation is given. We shall assume that both $\vec{y}$ and $\vec{f}$ are zero at these points, or, in other words, that differential equations

$$\frac{dy[r,k]}{dt} = 0$$

and initial conditions

$$y[r,k] = 0$$

are added at these points. This is part of the semi-discretization process which has to be performed by the user.

The grid is supposed to be defined by functions n(k), s(k), e(r) and w(r) presenting the bounds on the indices r and k of $\vec{y}$ and $\vec{f}$. To be more precise, the second index of the row vector y[r,] is bounded by w(r) and e(r) and the first index of the column vector y[,k] is bounded by s(k) and n(k). An example of a grid as described above is given in fig.1. The boundary functions have to be defined by the user of the program.

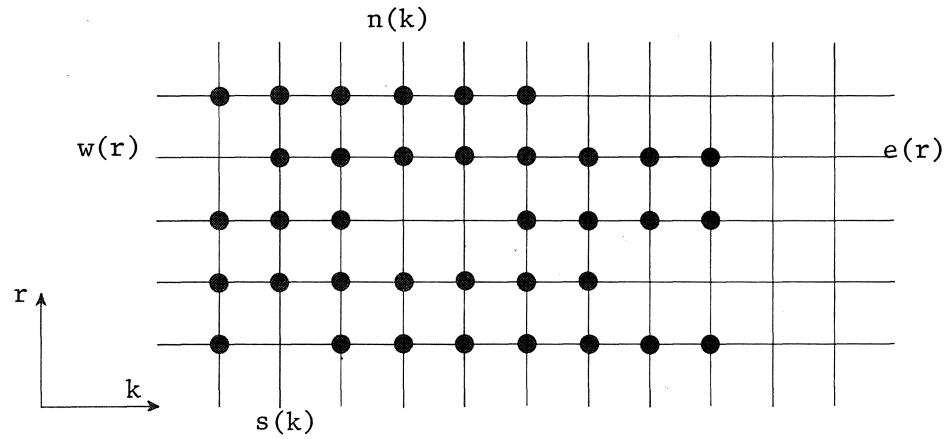

figure 1

Two-dimensional arrangement of the components of $\vec{y}$ and $\vec{f}$

We shall define a function $\vec{F}$ for five-point coupled equations and also for nine-point coupled ones. For this purpose, the set of gridpoints (see fig.1) is divided into four subsets as shown in fig.2.

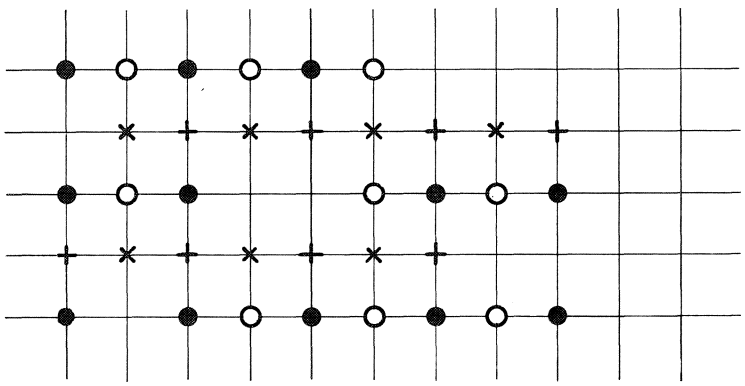

figure 2

Four subsets of gridpoints

Related to these subsets we define operators P_o, $P_\bullet$, P_+ and P_x working on vectors $\vec{v}$ which leave unchanged the components of $\vec{v}$ corresponding to the gridpoints o, •, + and x, respectively, and which sustitute a zero for all other components.

For five-point coupled equations, $\vec{F}$ is then defined as ("ADI")

$$(2.4)\qquad \begin{aligned}\vec{F}(t,\vec{v},\vec{w}) = {} & P_o\vec{f}(t,(\tfrac{1}{2}P_o+P_\bullet)\vec{v} + (\tfrac{1}{2}P_o+P_x)\vec{w}) + \\ & P_x\vec{f}(t,(\tfrac{1}{2}P_x+P_+)\vec{v} + (\tfrac{1}{2}P_x+P_o)\vec{w}) + \\ & P_\bullet\vec{f}(t,(\tfrac{1}{2}P_\bullet+P_o)\vec{v} + (\tfrac{1}{2}P_\bullet+P_+)\vec{w}) + \\ & P_+\vec{f}(t,(\tfrac{1}{2}P_+ +P_x)\vec{v} + (\tfrac{1}{2}P_+ +P_\bullet)\vec{w}).\end{aligned}$$

Here, tridiagonal systems of algebraic equations have to be solved alternatingly along the rows of o ● o and + x + points and along the columns of ● + ● and o x o points.

For a large class of five-point coupled differential equations which originate from parabolic equations it can be proved that both $\partial\vec{F}/\partial\vec{v}$ and $\partial\vec{F}/\partial\vec{w}$ have a negative spectrum [1].

For nine-point coupled equations $\vec{F}$ is defined as ("line hopscotch")

$$(2.5)\qquad \vec{F}(t,\vec{v},\vec{w}) = (P_x+P_+)\vec{f}(t,\vec{v}) + (P_o+P_\bullet)\vec{f}(t,\vec{w}).$$

By solving in the first stage firstly the o and ● components and then the + and x components and, vice versa, in the second stage, only tridiagonal implicit schemes have to be solved. Again it can be proved that both $\partial\vec{F}/\partial\vec{v}$ and $\partial\vec{F}/\partial\vec{w}$ have a negative spectrum [1].

Contrary to the usual "line hopscotch" approach, in our program the splitting direction is alternated after every complete time step. In a similar way as described above the splitting may be defined along vertical grid lines.

3. THE IMPLEMENTATION

In actual computations, one has to solve the equations (2.2) for $\vec{y}^{(1)}_{n+1}$ and $\vec{y}_{n+1}$, respectively. In order to maintain the stability properties of the partly implicit formulas, we use a Newton type process. By denoting

the approximations to $\vec{y}^{(1)}_{n+1}$ and $\vec{y}_{n+1}$ by ${}_j\vec{y}^{(1)}_{n+1}$ and ${}_j\vec{y}_{n+1}$, respectively, we thus obtain

$$ {}_{j+1}\vec{y}^{(1)}_{n+1} = {}_j\vec{y}^{(1)}_{n+1} - [I - \tfrac{1}{2}h_n J_1]^{-1}[{}_j\vec{y}^{(1)}_{n+1} - \vec{y}_n - \tfrac{1}{2}h_n \vec{F}(t_n + \tfrac{1}{2}h_n, {}_j\vec{y}^{(1)}_{n+1}, \vec{y}_n)] $$

(3.1)

$$ {}_{j+1}\vec{y}_{n+1} = {}_j\vec{y}_{n+1} - [I - \tfrac{1}{2}h_n J_2]^{-1}[{}_j\vec{y}_{n+1} - \vec{y}^{(1)}_{n+1} - \tfrac{1}{2}h_n \vec{F}(t_n + \tfrac{1}{2}h_n, \vec{y}^{(1)}_{n+1}, {}_j\vec{y}_{n+1})], $$

where J_1 and J_2 are approximations to the tridiagonal Jacobians $\partial\vec{F}/\partial\vec{v}$ and $\partial\vec{F}/\partial\vec{w}$.

From the definitions (2.4) and (2.5) it can be seen that the Newton process (3.1) can be solved for each row and each column separately.

The program is implemented with the following strategies: at most 3 Newton-iterations will be performed to solve the implicit systems. As convergence criterium we use

$$ \| {}_{j+1}\vec{y}^{(1)}_{n+1} - {}_j\vec{y}^{(1)}_{n+1} \| \leq \frac{TOL}{10} * (1 + \| {}_j\vec{y}^{(1)}_{n+1} \|) $$

(3.2) and

$$ \| {}_{j+1}\vec{y}_{n+1} - {}_j\vec{y}_{n+1} \| \leq \frac{TOL}{10} * (1 + \| {}_j\vec{y}_{n+1} \|), $$

where $\|.\|$ denotes the divided Euclidean norm and TOL is the user-specified local tolerance. If no convergence can be obtained within 3 iterations, the Jacobian of the particular row or column is re-evaluated and the Newton process is started once more. If again no convergence can be obtained within 3 iterations the steplength h is decreased by a factor 4.

In order to maintain 2^{nd} order accuracy in cases where only one iteration is performed, the initial approximations ${}_o\vec{y}^{(1)}_{n+1}$ and ${}_o\vec{y}_{n+1}$ to (3.1) are calculated by

$$ {}_o\vec{y}^{(1)}_{n+1} = (1 + h_n/2h_{n-1})\,\vec{y}_n - h_n/2h_{n-1}\,\vec{y}_{n-1}, $$

(3.3)

$$ {}_o\vec{y}_{n+1} = (1 + h_n/h_{n-1})\,\vec{y}_n - h_n/h_{n-1}\,\vec{y}_{n-1}. $$

We remark that the use of this predictor has no influence on the stability properties of the scheme.

Furthermore we mention the error control used. The local truncation error (LTE) is estimated by

$$\text{(3.4)} \qquad \text{LTE} = \frac{q}{1+q} \| q\vec{y}_{n+1} - (1+q)\vec{y}_n + \vec{y}_{n+1} \|,$$

where $q = h_n/h_{n-1}$ and $\|.\|$ denotes the divided Euclidean norm. The new steplength αh_n is estimated using the well-known root formula. Let $\bar{\alpha}$ be defined by

$$\text{(3.5)} \qquad \bar{\alpha} = ((\text{TOL} + \text{TOL} * \|\vec{y}_{n+1}\|)/\text{LTE})^{\frac{1}{2}}.$$

Then we put

$$\text{(3.6)} \qquad \alpha = \bar{\alpha}/\sqrt{2}.$$

The factor $\sqrt{2}$ provides a conservative estimate. In order to prevent marginal changes the steplength will not be altered when $0.85 < \alpha < 1.15$. Moreover, in order to prevent an excessive decrease or increase of the steplength, α is bounded by 0.1 and 3.0, respectively. No error control is performed after the first step of the integration process. However, if the second step fails, all results are rejected and the process is restarted with $h = h/4$.

Finally we remark that the solution in the endpoint of the integration interval is calculated by means of quadratic interpolation.

4. THE CENTRAL ALGORITHM

In order to give a description of the central part of the algorithm which resembles more or less the mathematical formulation, we need a number of variables and routines.

4.1. AUXILIARY VARIABLES AND ROUTINES

First of all there is the procedure

F(r,k,t,v,w)

which provides the (r,k)-th component of the right hand side of the differential equation in its splitted form. Furthermore, we need the procedure

dectri(b,c,d)

which performs a triangular decomposition of a tridiagonal matrix given by the vectors b(subdiagonal), c(diagonal) and d(superdiagonal) and which overwrites the elements of these vectors.

We also need the procedure

soltri(b,c,d,rhs)

which calculates the solution of a tridiagonal system of linear equations if the triangularly decomposed form as delivered by dectri is given by the vectors b, c, d and if the right hand side is given by the vector rhs. For the meaning of b, c and d see the description of dectri.

Within the program six auxiliary arrays br, cr, dr, bk, ck, and dk are used to store the matrices $[I - \frac{1}{2}h_n J_1]$ and $[I - \frac{1}{2}h_n J_2]$ in their decomposed form. The vectors br[r,], cr[r,] and dr[r,] define the tridiagonal matrix corresponding to the r-th row of the grid. In a similar way the vectors bk[,k], ck[,k] and dk[,k] define the tridiagonal matrix corresponding to the k-th column.

The values of $[I - \frac{1}{2}h_n J_1]$ corresponding to the r-th row will be calculated by the procedure

updaterowjac(r)

and temporarily stored into br[r,], cr[r,] and dr[r,]. This is done in the following way:

$$br[r,k-1] := -\tfrac{1}{2}h_n * \frac{F(r,k,t,{}^{(r,k-1)}y_{,y})-F(r,k,t,y,y)}{{}^{(r,k-1)}dy},$$

$$cr[r,k] := 1 - \tfrac{1}{2}h_n * \frac{F(r,k,t,{}^{(r,k)}y_{,y})-F(r,k,t,y,y)}{{}^{(r,k)}dy},$$

$$dr[r,k] := -\tfrac{1}{2}h_n * \frac{F(r,k,t,{}^{(r,k+1)}y_{,y})-F(r,k,t,y,y)}{{}^{(r,k+1)}dy},$$

where ${}^{(i,j)}dy = 10^{-6} * (1 + |y[i,j]|)$ and ${}^{(i,j)}y$ is defined as:

$${}^{(i,j)}y[k,\ell] = y[k,\ell] + {}^{(i,j)}dy \quad \text{if } i = k \wedge j = \ell$$

$$= y[k,\ell] \quad \text{else.}$$

In a similar way the procedure

updatecoljac(k)

calculates the matrix $[I - \tfrac{1}{2}h_n J_2]$ corresponding to the k-th column and temporarily stores the values into bk[,k], ck[,k] and dk[,k]. The procedure

rowjacobian(r)

fills the vectors br[r,], cr[r,] and dr[r,] by calling updaterowjac(r), followed by a call of dectri. Similarly, for the procedure

coljacobian(k).

Apart from the variables and procedures already introduced, the program contains several other variables and procedures which are listed below:

Variables:

rmin	: $\min_k s(k)$
rmax	: $\max_k n(k)$
kmin	: $\min_r w(r)$
kmax	: $\max_r e(r)$
t	: current variable t_n
te	: endpoint of the integration interval
y	: successive vectors $_{j+1}\vec{y}^{(1)}_{n+1}$ and $_{j+1}\vec{y}_{n+1}$ in formula (3.1)

yn : auxiliary array to store the computed solution in t_n

ynm1 : auxiliary array to store the computed solution in t_{n-1}; although a one-step scheme is used, ynm1 is necessary to compute a predictor to start the iteration process (3.1), to estimate the local truncation error and to interpolate the solution in t_e.

yhalf : auxiliary array to store the computed solution $\vec{y}_{n+1}^{(1)}$ in formula (2.2)

h : current integration step

hold : stepsize of the last integration step

hmin : minimal stepsize allowed during the integration process

hstart : the initial steplength

steprejject : boolean variable, being true if the step with stepsize h is rejected

rowrestart : boolean variable, being true if no convergence can be obtained within 3 iterations during the Newton iteration along rows, not even after updating of the Jacobian of that particular row

colrestart : similar to rowrestart but now for iteration along a column

eps : local error bound

error : estimated local truncation error

alfa : factor by which the current stepsize is multiplied to obtain the next stepsize

steps : number of integration steps performed.

Procedures: predictor : calculates the initial approximations $_o\vec{y}_{n+1}^{(1)}$ and $_o\vec{y}_{n+1}$ to start the Newton-iteration (3.1)

newtriconvergence : this boolean procedure performs the Newton-iteration (3.1); delivers true if the process did converge else false

newh	: when rowrestart or colrestart is set to true, newh divides the current stepsize by 4 but never dropping it below hmin
newmatrix	: new vectors br[r,], cr[r,], dr[r,] and bk[,k], ck[,k], dk[,k] are calculated for all r and k, when the stepsize h is changed. This is done straight forwardly without performing a new decomposition. (By this way we do not need to store the Jacobians J_1 and J_2 along all rows and columns).
localaccuracy	: delivers eps, error and alfa.
interpolate	: interpolates the solution in te.

We are now able to formulate the ADI- central algorithm applied to five-point coupled equations (a listing of the complete program is inserted in the appendix). In order to formulate the line hopscotch- central algorithm, applied to nine-point coupled equations, we can use the major part of the variables and procedures already declared.

The most important differences between these two algorithms are the procedures newtriconvergence, updaterowjac and updatecoljac. By using line hopscotch the space direction is fixed during both stages of one integrationstep, being the x-direction for all "odd" and the y-direction for all "even" integrationsteps. In order to formulate the line hopscotch- central algorithm we need the procedure

rowvec(r,y).

This procedure calculates the values $F(r,k,t+\frac{1}{2}h,y,y)$ for k = w[r],...,e[r] and combines these values to a vector.

For integration along columns a similar procedure

colvec(k,y)

should be declared.

4.2. THE ADI- CENTRAL ALGORITHM

```
yn := y; hold := h := hstart; steps := 0;
for r from rmin to rmax do rowjacobian(r) od;
for k from kmin to kmax do coljacobian(k) od;
rowrestart := colrestart := stepreject := false;
while t < te
do if rowrestart or colrestart or stepreject
   then y := yn
   else ynm1 := yn; yn := y
   fi;
for r from rmin to rmax
do predictor (y[r,], yn[r,], ynm1[r,], h/(2 * hold));
    if not newtriconvergence(r,y[r,], "rows")
    then if rowrestart
         then error
         else rowjacobian(r)
              if not newtriconvergence(r,y[r,], "rows")
              then newh; newmatrix; rowrestart := true;
                   goto endloop
              fi
         fi
    fi
od;
rowrestart := false;
yhalf := y;
for k from kmin to kmax
do predictor(y[,k], yn[,k], ynm1[,k], h/hold);
    if not newtriconvergence(k,y[,k],"columns")
    then if colrestart
         then error
         else coljacobian(k);
              if not newtriconvergence(k,y[,k],"columns")
              then newh; newmatrix; colrestart := true; goto endloop
              fi
```

```
        fi
    fi
od;
colrestart := false;
steps +:= 1;
if steps = 1
then alfa := 1; t +:= h; hold := h
else localaccuracy;
   if eps >= error
   then t +:= h; if t > te then interpolate; goto endloop fi;
        hold := h; stepreject := false
   else stepreject := true;
        if steps = 2
        then t -:= h; newh; yn := ynm1;
             steps := 0
        fi
   fi;
   if alfa ≠ 1
   then if steps ≠ 0
        then h *:= alfa; if h < hmin then error fi
        fi;
        newmatrix
   fi
fi;
endloop: skip
od;
```

4.3. THE LINE HOPSCOTCH- CENTRAL ALGORITHM

```
≠ initialization; ≠
≠ calculation of Jacobian-matrices; ≠
while t < te
do if rowrestart or colrestart or stepreject
   then y := yn
   else ynm1 := yn; yn := y
```

```
   fi;
if ≠ odd integrationstep ≠
then for r from rmin by 2 to rmax
     do y[r,] := yn[r,] + h/2 * rowvec(r,yn) od;
     for r from rmin + 1 by 2 to rmax
     do predictor (y[r,], yn[r,], ynm1[r,], h/ (2 * hold));
        if not newtriconvergence(r,yn,y[r,],"rows")
        then ≠ same measures will be taken as in the case of a
               five-point coupling ≠
        fi
     od;
     yhalf := y;
     for r from rmin+1 by 2 to rmax
     do y[r,] := yhalf[r,] + h/2 * rowvec(r,yhalf) od;
     for r from rmin by 2 to rmax
     do predictor(y[r,], yn[r,], ynm1[r,], h/hold);
        if not newtriconvergence(r, yhalf, y[r,], "rows")
        then ≠ same measures will be taken as in the
               case of a five-point coupling ≠
        fi
     od
else ≠ the integration process along columns will be performed
     in a similar way as described above for rows ≠
fi;
   ≠ errorcontrol; see central part for five-point coupling ≠
od;
```

5. THE PARAMETERLIST

For both algorithms a routine has been written. These routines have the same "heading" which reads:

```
proc splitmethod = (ref real t, real te, mat y, function derivative,
                    ref [ ] int n,s,e,w, ref info info) void:
```

with

```
mode mat = ref [ , ] real,
mode function = proc(int, int, real, mat)real,
mode info = struct(real, hstart, hmin, tol, int steps);
```

The meaning of the formal parameters is:

t : independent variable of the semi-discretized system of ordinary differential equations
entry: the initial value of the independent variable
exit : the last point reached in integration;
normally t is slightly greater than tend

te : entry: endpoint of integration interval at which the solution is desired

y : dependent variable
entry: the initial value of the dependent variable
exit : the solution at te

derivative : procedure delivering the right hand side component by component. The "heading" of this procedure reads:
proc derivative = (int r,k, real t, mat y) real:
derivative performs an evaluation of the right hand side of the system for the field y, at time t, in the (r,k)-th gridpoint

n,s,e,w : entry: integer arrays presenting the bounds on the indices r and k of the matrix y; the first index of the column vector y[,k] is bounded by s[k] and n[k] and the second index of the rowvector y[r,] is bounded by w[r] and e[r]

info : structured variable, containing four fields:
real hstart, real hmin, real tol, int steps.
The meaning of the field selectors is:
hstart: (entry) the initial steplength
hmin: (entry) the minimal steplength allowed during the integration process
tol : (entry) local error tolerance

steps: number of integration steps performed, i.e. accepted and rejected ones (the steps necessary to make a restart are not taken into account).

6. NUMERICAL EXAMPLES

In order to test the procedure splitmethod, it is applied to several problems. Two of these problems are discussed in this section. For both problems the semi-discretization is performed by using finite differences. We mention that for these problems only the exact solution of the *partial* differential equation is known. The relative errors in several gridpoints are given at the end of the integration interval. Both problems are integrated for three values of the tolerance parameter TOL, viz. 10^{-3}, 10^{-4}, 10^{-5}.

Problem I

The first equation we consider is a non-linear one and reads

$$u_t = u_{xx} + u_x u_y + u_{yy} - (4+4xye^{-t}+x^2+y^2)e^{-t}, \quad 0 \le x \le 2, 0 \le y \le 2$$

(6.1) with boundary conditions

$$u(t,0,y) = y^2e^{-t}, \qquad u(t,2,y) = (4+y^2)e^{-t},$$
$$u(t,x,0) = x^2e^{-t}, \qquad u(t,x,2) = (x^2+4)e^{-t}$$

and initial condition

$$u(0,x,y) = x^2+y^2.$$

The exact solution of problem (6.1) is $u(t,x,y) = (x^2+y^2)e^{-t}$. Here we do not give the semi-discretized system of equations. We choose an equidistant grid with increment 0.1 in both directions, resulting in 361 gridpoints; using central differences, semi-discretization of equation (6.1) leads to a five-point coupled function. The integration interval is

[0,1]. For several gridpoints the relative errors at t = 1.0 are listed in table 6.1.

	(x,y)							
TOL	(0.1,0.1)	(0.1,1.0)	(0.1,1.9)	(1.0,0.5)	(1.0,1.5)	(1.9,0.1)	(1.9,1.0)	(1.9,1.9)
10^{-3}	$3.3\,10^{-2}$	$4.7\,10^{-4}$	$1.0\,10^{-2}$	$4.4\,10^{-4}$	$3.0\,10^{-4}$	$1.0\,10^{-2}$	$3.7\,10^{-4}$	$1.3\,10^{-2}$
10^{-4}	$4.2\,10^{-3}$	$6.4\,10^{-5}$	$1.3\,10^{-3}$	$4.3\,10^{-5}$	$2.8\,10^{-5}$	$1.3\,10^{-3}$	$4.8\,10^{-5}$	$1.6\,10^{-3}$
10^{-5}	$4.3\,10^{-4}$	$6.4\,10^{-6}$	$1.3\,10^{-4}$	$4.3\,10^{-6}$	$2.9\,10^{-6}$	$1.3\,10^{-4}$	$4.8\,10^{-6}$	$1.6\,10^{-4}$

table 6.1

Problem II

The second problem we consider is a linear one. Again we only state the partial differential equation:

$$u_t = 0.1\,u_{xx} + 0.1\,u_{xy} + 0.15\,u_{yy}, \quad 0 \le x \le 2,\ 0 \le y \le 2 \tag{6.2}$$

with boundary conditions

$$u(t,0,y) = \exp(-0.35t)\sin y, \quad u(t,2,y) = \exp(-0.35t)\sin(2+y),$$
$$u(t,x,0) = \exp(-0.35t)\sin x, \quad u(t,x,2) = \exp(-0.35t)\sin(x+2)$$

and initial condition

$$u(0,x,y) = \sin(x+y).$$

This problem has the exact solution: $u(t,x,y) = \exp(-0.35t)\sin(x+y)$. We use the same grid as described in problem I. This time, discretization of the right hand side of (6.2) using central differences leads to a nine-point coupled function. Again the integration interval is [0,1]. For several gridpoints the relative errors at t = 1.0 are listed in table 6.2.

	(x,y)							
TOL	(0.1,0.1)	(0.1,1.0)	(0.1,1.9)	(1.0,0.5)	(1.0,1.5)	(1.9,0.1)	(1.9,1.0)	(1.9,1.9)
10^{-3}	$2.0\ 10^{-3}$	$3.8\ 10^{-3}$	$1.7\ 10^{-3}$	$1.7\ 10^{-3}$	$5.5\ 10^{-4}$	$1.7\ 10^{-3}$	$2.3\ 10^{-3}$	$2.0\ 10^{-3}$
10^{-4}	$3.9\ 10^{-4}$	$7.1\ 10^{-4}$	$3.9\ 10^{-4}$	$1.8\ 10^{-4}$	$2.4\ 10^{-4}$	$3.9\ 10^{-4}$	$2.9\ 10^{-4}$	$5.0\ 10^{-4}$
10^{-5}	$1.1\ 10^{-4}$	$4.4\ 10^{-5}$	$4.7\ 10^{-5}$	$3.3\ 10^{-4}$	$4.9\ 10^{-4}$	$4.8\ 10^{-5}$	$2.6\ 10^{-4}$	$8.5\ 10^{-5}$

table 6.2

7. REFERENCE

[1] P.J. VAN DER HOUWEN & J.G. VERWER, *Non-linear splitting methods for semi-discretized parabolic differential equations*, Report NW 51/77, Mathematical Centre, Amsterdam, 1977.

APPENDIX

As mentioned before, the programs for five-point coupled equations and for nine-point coupled ones bear a close resemblance. Here, as an example, we list the complete program in the case of a five-point coupled equation.

```
SPLIT 5:

'BEGIN'

    'MODE' 'VEC' = 'REF'[ ]'REAL',
           'MAT' = 'REF'[,]'REAL';
    'MODE' 'FUNCTION' = 'PROC'('INT','INT','REAL','MAT')'REAL',
           'SPLITFUNCTION' = 'PROC'('INT','INT','REAL','MAT','MAT')'REAL',
           'INFO' = 'STRUCT'('REAL'HSTART,HMIN,TOL,'INT'STEPS),
           'TRIDIAMAT' = 'STRUCT'('VEC'SUB,DIAG,SUP);


    'OP' - = ('VEC'Y1,Y2)'VEC':
     ('INT'MAX='UPB'Y1, MIN='LWB'Y1;
      'HEAP'[MIN:MAX]'REAL'Y;
      'FOR' I 'FROM' MIN 'TO' MAX 'DO' Y[I]:=Y1[I]-Y2[I] 'OD';
      Y );


    'OP' * = ('REAL'R,'VEC'Y)'VEC':
     ('INT'MIN='LWB'Y, MAX='UPB'Y; 'HEAP'[MIN:MAX]'REAL'V;
      'FOR' I 'FROM' MIN 'TO' MAX 'DO' V[I]:=Y[I]*R 'OD'; V);


    'OP' 'NORM' = ('VEC'Y)'REAL':
     ('INT'MIN='LWB'Y, MAX='UPB'Y;'REAL'S:=0;
      'FOR' I 'FROM' MIN 'TO' MAX
      'DO' S+:=('REAL' YI=Y[I]; YI*YI) 'OD';
      SQRT(S/(MAX-MIN+1)));


   'OP' + = ('MAT'Y1,Y2)'MAT':
    ( 'INT'N1= 'LWB'Y1, N2= 'UPB'Y1,
         M1=2'LWB'Y1, M2=2'UPB'Y1;
      'HEAP'[N1:N2,M1:M2]'REAL'Y;
      'FOR' I 'FROM' N1 'TO' N2
      'DO' 'FOR' J 'FROM' M1 'TO' M2
           'DO' Y[I,J]:=Y1[I,J]+Y2[I,J]
           'OD'
      'OD';
      Y );


   'OP' - = ('MAT'Y1,Y2)'MAT':
    ( 'INT'N1= 'LWB'Y1, N2= 'UPB'Y1,
         M1=2'LWB'Y1, M2=2'UPB'Y1;
      'HEAP'[N1:N2,M1:M2]'REAL'Y;
      'FOR' I 'FROM' N1 'TO' N2
      'DO' 'FOR' J 'FROM' M1 'TO' M2
           'DO' Y[I,J]:=Y1[I,J]-Y2[I,J]
           'OD'
      'OD';
      Y );
```

```
'OP' * = ('REAL'R,'MAT'Y)'MAT':
 ( 'INT'N1= 'LWB'Y, N2= 'UPB'Y,
        M1=2'LWB'Y, M2=2'UPB'Y;
   'HEAP'[N1:N2,M1:M2]'REAL'Z;
   'FOR' I 'FROM' N1 'TO' N2
   'DO' 'FOR' J 'FROM' M1 'TO' M2
        'DO' Z[I,J]:=Y[I,J]*R
        'OD'
   'OD';
   Z );


'OP' / = ('MAT'Y,'REAL'R)'MAT':
  'IF' R /= Ø
  'THEN' 1.Ø/R * Y
  'ELSE' ERROR; 'NIL'
  'FI';


 'PROC' MIN = ('REF'[ ]'INT'Z)'INT':
   'BEGIN' 'INT'L='LWB'Z; 'INT'M:=Z[L];
     'FOR' K 'FROM' L+1 'TO' 'UPB'Z
     'DO' 'IF' Z[K]<M 'THEN' M:=Z[K] 'FI' 'OD';
     M
   'END';


 'PROC' MAX = ('REF'[ ]'INT'Z)'INT':
   'BEGIN' 'INT'L='LWB'Z; 'INT'M:=Z[L];
     'FOR' K 'FROM' L+1 'TO' 'UPB'Z
     'DO' 'IF' Z[K]>M 'THEN' M:=Z[K] 'FI' 'OD';
     M
   'END';


 'PROC' ZEROVEC = ('VEC'V)'VOID':
   'FOR' I 'FROM' 'LWB'V 'TO' 'UPB'V 'DO' V[I]:=Ø.Ø 'OD';


 'PROC' ZEROMAT = ('MAT'Z)'VOID':
   'FOR' R 'FROM' 'LWB'Z 'TO' 'UPB'Z
   'DO' ZEROVEC(Z[R, ]) 'OD';
```

```
'PROC' DECTRI = ('INT'MIN,MAX,'TRIDIAMAT'MAT)'VOID':
  'BEGIN' 'VEC' SUB = SUB 'OF' MAT,
              DIAG =DIAG 'OF' MAT,
               SUP = SUP 'OF' MAT;

        'PROC' TESTD='VOID':
          'IF' 'ABS'D<=NORM1*1.E-8
          'THEN' PRINT((NEWLINE,"ERROR IN LU-DECOMPOSITION"));
                 ERROR
          'FI';

    'REAL'S,U,NORM,NORM1,D:=DIAG[MIN],R:=SUP[MIN];
    NORM:=NORM1:='ABS'D+'ABS'R;
    TESTD;
    U:=SUP[MIN]:=R/D; S:=SUB[MIN];
    'FOR' I 'FROM' MIN+1 'TO' MAX-1
    'DO' D:=DIAG[I]; R:=SUP[I];
         NORM1:='ABS'D+'ABS'R+'ABS'S;
         DIAG[I]:=D-:=U*S;
         TESTD;
         U:=SUP[I]:=R/D; S:=SUB[I];
         'IF' NORM1>NORM 'THEN' NORM:=NORM1 'FI'
    'OD';
    D:=DIAG[MAX]; NORM1:='ABS'D+'ABS'S;
    DIAG[MAX]:=D-:=U*S;
    TESTD
  'END' #DECTRI#;


'PROC' SOLTRI = ('INT'MIN,MAX,'TRIDIAMAT'MAT,'VEC'RHS)'VEC':
  'BEGIN' 'VEC' SUB = SUB 'OF' MAT,
              DIAG =DIAG 'OF' MAT,
               SUP = SUP 'OF' MAT;

          'REAL'R:=RHS[MIN]/:=DIAG[MIN];
        'FOR' I 'FROM' MIN+1 'TO' MAX
        'DO' R:=RHS[I]:=(RHS[I]-SUB[I-1]*R)/DIAG[I] 'OD';
        'FOR' I 'FROM' MAX-1 'BY' -1 'TO' MIN
        'DO' R:=RHS[I]-:=SUP[I]*R 'OD';
       RHS
  'END' #SOLTRI#;
```

```
'PROC' SPLITMETHOD = ('REF''REAL'T,'REAL'TE,'MAT'Y,'FUNCTION' DERIVATIVE,
                      'REF'[ ]'INT'N,S,E,W,'REF''INFO'INFO)'VOID':

  'BEGIN'     'REAL' HSTART = HSTART 'OF' INFO,
                     HMIN   = HMIN   'OF' INFO,
                     TOL    = TOL    'OF' INFO,
              'REF' 'INT' STEPS  = STEPS  'OF' INFO;

   'OP' 'NORM' = ('MAT'Y)'REAL':
    ( 'REAL' S:=0.0;
      'FOR' I 'FROM' 'LWB'Y 'TO' 'UPB'Y
      'DO' 'FOR' J 'FROM' 2'LWB'Y 'TO' 2'UPB'Y
           'DO' S+:=('REAL' YIJ=Y[I,J]; YIJ*YIJ)
           'OD'
      'OD';
      SQRT(S/NM) );


  'PROC' NUMBER OF GRIDPOINTS = 'INT':
    'BEGIN' 'INT' N:=0;
            'FOR' R 'FROM' RMIN 'TO' RMAX
            'DO' N+:=E[R]-W[R]+1 'OD';
           N
    'END';


  'PROC' ROWVEC = ('INT'R,'SPLITFUNCTION'F)'VEC':
    'BEGIN' 'HEAP'[KMIN:KMAX]'REAL'B; ZEROVEC(B);
       'FOR' K 'FROM' W[R] 'TO' E[R]
       'DO' B[K]:=F(R,K,T+H/2,Y,YN) 'OD';
       B
    'END';


  'PROC' COLVEC = ('INT'K,'SPLITFUNCTION'F)'VEC':
    'BEGIN' 'HEAP'[RMIN:RMAX]'REAL'B; ZEROVEC(B);
       'FOR' R 'FROM' S[K] 'TO' N[K]
       'DO' B[R]:=F(R,K,T+H/2,YHALF,Y) 'OD';
       B
    'END';


  'PROC' F = ('INT'R,K,'REAL'T,'MAT'V,W)'REAL':
    'BEGIN' [R-1:R+1,K-1:K+1]'REAL' YSPLIT;
      YSPLIT['AT'1,'AT'1]:=
((       0.0          ,(R=RMIN!0.0!W[R-1,K]),        0.0          ),
((K=KMIN!0.0!V[R,K-1]), (V[R,K]+W[R,K])/2.0 ,(K=KMAX!0.0!V[R,K+1])),
 (       0.0          ,(R=RMAX!0.0!W[R+1,K]),        0.0          ));
      DERIVATIVE(R,K,T,YSPLIT)
    'END' #F#;
```

```
'PROC' UPDATEROWJAC = ('INT'R)'VOID':
  'BEGIN' 'INT'WR=W[R], ER=E[R];'REAL'FU; [WR:ER]'REAL'DY;
          'PROC' ADD=('INT'K,KK)'MAT':
            'BEGIN' 'HEAP'[R-1:R+1,K-1:K+1]'REAL' YPLUSDY;
             'FOR' I 'FROM'(R=RMIN!RMIN!R-1) 'TO'(R=RMAX!RMAX!R+1)
             'DO' 'FOR'J'FROM'(K=KMIN!KMIN!K-1)'TO'(K=KMAX!KMAX!K+1)
                  'DO' YPLUSDY[I,J]:=YN[I,J] 'OD'
             'OD';
             YPLUSDY[R,KK]+:=DY[KK];
             YPLUSDY
            'END' #ADD#;
      'FOR' K 'FROM' WR 'TO' ER
      'DO' DY[K]:=1.E-6*(1+'ABS'YN[R,K]) 'OD';
      FU:=F(R,WR,T,YN,YN);
      CR[R,WR]:=1-H/2*(F(R,WR,T,ADD(WR,WR),YN)-FU)/DY[WR];
      DR[R,WR]:= -H/2*(F(R,WR,T,ADD(WR,WR+1),YN)-FU)/DY[WR+1];
      'FOR' K 'FROM' WR+1 'TO' ER-1
      'DO' FU:=F(R,K,T,YN,YN);
        BR[R,K-1]:= -H/2*(F(R,K,T,ADD(K,K-1),YN)-FU)/DY[K-1];
        CR[R,K]  :=1-H/2*(F(R,K,T,ADD(K,K),YN)-FU)/DY[K];
        DR[R,K]  := -H/2*(F(R,K,T,ADD(K,K+1),YN)-FU)/DY[K+1]
      'OD';
      FU:=F(R,ER,T,YN,YN);
    BR[R,ER-1]:= -H/2*(F(R,ER,T,ADD(ER,ER-1),YN)-FU)/DY[ER-1];
    CR[R,ER]  :=1-H/2*(F(R,ER,T,ADD(ER,ER),YN)-FU)/DY[ER]
  'END' #UPDATEROWJAC#;


'PROC' UPDATECOLJAC = ('INT'K)'VOID':
  'BEGIN' 'INT'SK=S[K], NK=N[K];'REAL'FU; [SK:NK]'REAL'DY;
          'PROC' ADD=('INT'R,RR)'MAT':
            'BEGIN' 'HEAP'[R-1:R+1,K-1:K+1]'REAL' YPLUSDY;
             'FOR' I 'FROM'(R=RMIN!RMIN!R-1) 'TO'(R=RMAX!RMAX!R+1)
             'DO' 'FOR'J'FROM'(K=KMIN!KMIN!K-1)'TO'(K=KMAX!KMAX!K+1)
                  'DO' YPLUSDY[I,J]:=YN[I,J] 'OD'
             'OD';
             YPLUSDY[RR,K]+:=DY[RR];
             YPLUSDY
            'END' #ADD#;
      'FOR' R 'FROM' SK 'TO' NK
      'DO' DY[R]:=1.E-6*(1+'ABS'YN[R,K]) 'OD';
      FU:=F(SK,K,T,YN,YN);
      CK[SK,K]:=1-H/2*(F(SK,K,T,YN,ADD(SK,SK))-FU)/DY[SK];
      DK[SK,K]:= -H/2*(F(SK,K,T,YN,ADD(SK,SK+1))-FU)/DY[SK+1];
      'FOR' R 'FROM' SK+1 'TO' NK-1
      'DO' FU:=F(R,K,T,YN,YN);
        BK[R-1,K]:= -H/2*(F(R,K,T,YN,ADD(R,R-1))-FU)/DY[R-1];
        CK[R,K]  :=1-H/2*(F(R,K,T,YN,ADD(R,R))-FU)/DY[R];
        DK[R,K]  := -H/2*(F(R,K,T,YN,ADD(R,R+1))-FU)/DY[R+1]
      'OD';
      FU:=F(NK,K,T,YN,YN);
    BK[NK-1,K]:= -H/2*(F(NK,K,T,YN,ADD(NK,NK-1))-FU)/DY[NK-1];
    CK[NK,K]  :=1-H/2*(F(NK,K,T,YN,ADD(NK,NK))-FU)/DY[NK]
  'END' #UPDATECOLJAC#;
```

```
'PROC' NEWTRICONVERGENCE = ('INT'I,'VEC'RHS,'STRING'TEXT)'BOOL':
  'BEGIN' ['LWB' RHS : 'UPB' RHS]'REAL'CORR; 'BOOL'CONVERGENCE;
      'TO' 3
      'WHILE' CORR:='IF' TEXT="ROWS"
                    'THEN' SOLTRI(W[I],E[I],(BR[I, ],CR[I, ],DR[I, ])
                                  ,RHS-YN[I, ]-H/2*ROWVEC(I,F))
                    'ELSE' SOLTRI(S[I],N[I],(BK[ ,I],CK[ ,I],DK[ ,I])
                                  ,RHS-YHALF[ ,I]-H/2*COLVEC(I,F))
                    'FI';
                   CONVERGENCE:='NORM'CORR<TOL/10.0*(1.0+'NORM'RHS);
                   RHS:=RHS-CORR;
            'NOT' CONVERGENCE
      'DO' 'SKIP'
      'OD';
     CONVERGENCE
  'END' #NEWTRICONVERGENCE#;


'PROC' NEWH = 'VOID':
  'IF' H=HMIN
  'THEN' ERROR
  'ELSE' H/:=4; ALFA/:=4;
         ( H<HMIN ! ALFA*:=HMIN/H; H:=HMIN)
  'FI';


'PROC' PREDICTOR = ('VEC' Y,YN,YNM1,'REAL'Q )'VOID':
   Y:=(Q + 1.0) * YN - Q * YNM1;


'PROC' LOCALACCURACY = 'VOID':
  'BEGIN' 'REAL' Q=H/HOLD;
     EPS:=TOL*(1.0+'NORM'Y);
     ERROR:=Q/(1.0+Q)*'NORM'(Q*YNM1-(1.0+Q)*YN+Y);
     ALFA:=SQRT(EPS/(2*ERROR));
     ( ALFA > 0.85 ! ( ALFA < 1.15 ! ALFA:=1.0 ));
     ( ALFA > 3.0 ! ALFA:=3.0 );
     ( ALFA < 0.1 ! ALFA:=0.1 )
  'END';


'PROC' INTERPOLATE = 'VOID':
  'BEGIN' 'REAL'A=(T-TE)/H, B=HOLD/H; 'REAL'C=1-A+B;
     Y:=(B*C*(1-A)*Y+A*C*(1+B)*YN-A*(1-A)*YNM1)/(B*(1+B))
  'END';
```

```
'PROC' ROWJACOBIAN = ('INT'R)'VOID':
  'BEGIN' UPDATEROWJAC(R);
          DECTRI(W[R],E[R],(BR[R, ],CR[R, ],DR[R, ]))
  'END';


'PROC' COLJACOBIAN = ('INT'K)'VOID':
  'BEGIN' UPDATECOLJAC(K);
          DECTRI(S[K],N[K],(BK[ ,K],CK[ ,K],DK[ ,K]))
  'END';


'PROC' NEWMATRIX = 'VOID':
  'BEGIN'
      'FOR' R 'FROM' RMIN 'TO' RMAX
      'DO' NEWLU(W[R],E[R],(BR[R, ],CR[R, ],DR[R, ])) 'OD';
      'FOR' K 'FROM' KMIN 'TO' KMAX
      'DO' NEWLU(S[K],N[K],(BK[ ,K],CK[ ,K],DK[ ,K])) 'OD'
  'END';


'PROC' NEWLU = ('INT'MIN,MAX,'TRIDIAMAT' MAT)'VOID':
  'BEGIN' 'VEC'  SUB = SUB  'OF' MAT,
                DIAG = DIAG 'OF' MAT,
                 SUP = SUP  'OF' MAT;

          'REAL' U,V,W;
     U:=DIAG[MIN]; DIAG[MIN]:=1.0-ALFA*(1.0-U);
     V:=SUP[MIN]; SUP[MIN]:=ALFA*V*U/DIAG[MIN];
     W:=SUB[MIN]; SUB[MIN]*:=ALFA;
     'FOR' I 'FROM' MIN+1 'TO' MAX-1
     'DO'U:=DIAG[I];DIAG[I]:=1.0-ALFA*(1.0-U-W*V)-SUP[I-1]*SUB[I-1];
         V:=SUP[I]; SUP[I]*:=ALFA*U/DIAG[I];
         W:=SUB[I]; SUB[I]*:=ALFA
     'OD';
     DIAG[MAX]:=1.0-ALFA*(1.0-DIAG[MAX]-W*V)-SUP[MAX-1]*SUB[MAX-1]
  'END' #NEWLU#;


 'INT' RMIN=MIN(S),
       KMIN=MIN(W),
       RMAX=MAX(N),
       KMAX=MAX(E);
[RMIN:RMAX,KMIN:KMAX]'REAL'BR,CR,DR,BK,CK,DK,YN,YHALF,YNM1;
'REAL' H, ALFA, HOLD, EPS, ERROR ;
'INT' NM = NUMBER OF GRIDPOINTS;

ZEROMAT(BR); ZEROMAT(CR); ZEROMAT(DR);
ZEROMAT(BK); ZEROMAT(CK); ZEROMAT(DK);
```

```
                                        # START OF THE CENTRAL ALGORITHM #

YN:=Y;
HOLD:=H:=HSTART; STEPS:=Ø;

'FOR' R 'FROM' RMIN 'TO' RMAX 'DO' ROWJACOBIAN(R) 'OD';
'FOR' K 'FROM' KMIN 'TO' KMAX 'DO' COLJACOBIAN(K) 'OD';

'BOOL' ROWRESTART:='FALSE',COLRESTART:='FALSE',STEPREJECT:='FALSE';
'WHILE' T<TE
'DO' 'IF' ROWRESTART 'OR' COLRESTART 'OR' STEPREJECT
     'THEN' Y:=YN
     'ELSE' YNM1:=YN; YN:=Y
     'FI';
   'FOR' R 'FROM' RMIN 'TO' RMAX
   'DO' PREDICTOR(Y[R, ],YN[R, ],YNM1[R, ],H/(2.0*HOLD));
        'IF' 'NOT'NEWTRICONVERGENCE(R,Y[R, ],"ROWS")
        'THEN' 'IF' ROWRESTART
               'THEN' ERROR
               'ELSE' ROWJACOBIAN(R);
                    'IF' 'NOT' NEWTRICONVERGENCE(R,Y[R, ],"ROWS")
                    'THEN' NEWH; NEWMATRIX; ROWRESTART:='TRUE';
                           ENDLOOP
                    'FI'
               'FI'
        'FI'
   'OD';
   ROWRESTART:='FALSE';
   YHALF:=Y;
   'FOR' K 'FROM' KMIN 'TO' KMAX
   'DO' PREDICTOR(Y[ ,K],YN[ ,K],YNM1[ ,K],H/HOLD);
        'IF' 'NOT'NEWTRICONVERGENCE(K,Y[ ,K],"COLUMNS")
        'THEN' 'IF' COLRESTART
               'THEN' ERROR
               'ELSE' COLJACOBIAN(K);
                    'IF' 'NOT' NEWTRICONVERGENCE(K,Y[ ,K],"COLUMNS")
                    'THEN' NEWH; NEWMATRIX; COLRESTART:='TRUE';
                           ENDLOOP
                    'FI'
               'FI'
        'FI'
   'OD';
   COLRESTART:='FALSE';
```

```
        STEPS +:=1;
        'IF' STEPS = 1
        'THEN' ALFA:=1.0; T+:=H; HOLD:=H
        'ELSE' LOCALACCURACY;
               'IF' EPS >= ERROR
               'THEN' T+:=H; (T > TE ! INTERPOLATE; ENDLOOP ); HOLD:=H;
                      STEPREJECT:='FALSE'
               'ELSE' STEPREJECT:='TRUE';
                      ( STEPS = 2 ! T-:=H; NEWH; YN:=YNM1; STEPS:=0 )
               'FI';
               (ALFA /=1.0 ! (STEPS /= 0 ! H*:=ALFA; (H < HMIN ! ERROR));
                             NEWMATRIX)
        'FI';
      ENDLOOP:'SKIP'
    'OD'

                                              # END OF THE CENTRAL ALGORITHM #

    'END' #SPLITMETHOD#;


    'PR' PROG 'PR'

    'SKIP'
'END'
```