stichting
mathematisch
centrum

MC

REKENAFDELING MR 128/71 DECEMBER

P.W. HEMKER

AN ALGOL 60 PROCEDURE FOR THE SOLUTION OF STIFF 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), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Contents

Abstract

In this report a concise description is given of the Adams-Moulton method together with Gear's method for the solution of stiff differential equations. An ALGOL 60-procedure is provided which selects that method which is in agreement with the behaviour of the differential equation and some prescribed accuracy parameters. A number of numerical experiments are reported.

1. Introduction

Adams-Bashforth and Adams-Moulton methods are well-known and very popular in application to high-accuracy computation of slowly varying systems of differential equations. However, difficulties occur when these methods are applied to systems in which the solution contains a rapidly varying component. These difficulties are due to the stability properties which force the user to take a small steplength. In 1952 Curtiss and Hirschfelder published linear multistep methods appropriate to these so called "stiff" systems of differential equations. Since then much research was done on the theory of linear multistep methods (e.g. Dahlquist [1956,1963], Henrici [1962]). Many methods were proposed; however, no methods were found significantly superior to those mentioned.

In 1962 Nordsieck reported a method in order to make the administration of the multistep methods free from its rigid system of grid-points. In his paper he also started an investigation of automatic steplength and order control for Adams methods. In several publications Gear [1967,1968,1971] developed the same idea for methods suitable for stiff equations.

This paper contains a short description of the Adams-Moulton (A.M.) and Curtiss-Hirschfelder (C.H.) methods. An ALGOL 60-procedure is given which makes a choice between these two types of methods and uses automatic steplength and order control. In conclusion, a number of numerical results will be reported. A description of the convergence and stability properties of the A.M. and C.H. methods is given in detail in Hemker [1971].

2. Linear multistep methods according to Adams-Moulton and Curtiss-Hirschfelder

Both Curtiss-Hirschfelder and Adams-Moulton methods are based on polynomial approximations to a function known at some grid-points. Several formulae can be used to obtain these approximations. In classical multistep theory the Lagrange formula is used. In order to give a concise description of the methods we prefer the use of the Grünert and the Newton formula.

We define two sequences of polynomials

Definition

$$\vec{G}(x) = \{1,(x-x_0),(x-x_0)(x-x_1),\ldots,(x-x_0)(x-x_1)\ \ldots\ (x-x_n)\}.$$

Definition

$$\vec{H}(x) = \{1,(x-x_0),(x-x_0)^2,\ldots,(x-x_0)^{n+1}\}.$$

Every (n+1)st degree polynomial h(x) can be written as an inproduct

$$h(x) = \vec{G}(x).\vec{a} = \vec{H}(x).\vec{p}.$$

Here the vectors $\vec{a}$ or $\vec{p}$ contain the coefficients characterizing the polynomial h(x) and $\vec{G}$ or $\vec{H}$ characterize the formula used. $\vec{G}(x).\vec{a}$ essentially give the Newton formula and $\vec{H}(x).\vec{p}$ represents the Grünert formula. A matrix A can easily be found which gives the conversion between the two types of polynomials.

Definition

$\vec{\vec{A}}$ is the matrix by which the operation $\vec{G}(x) = \vec{H}(x).\vec{\vec{A}}$ is achieved.

In Hemker [1971] a proof is given of the following statement:

Statement

The matrix $\vec{\vec{A}}$ is an uppertriangular matrix the elements of which are provided by $\vec{\vec{A}}[0,0] = 1$; $\vec{\vec{A}}[0,j] = 0$ $(j\neq 0)$ and $\vec{\vec{A}}[i,j] = (x_0-x_{j-1})\ \vec{\vec{A}}[i,j-1] + \vec{\vec{A}}[i-1,j-1]$.

This implies that A is completely determined by the distribution of the grid-points $(x_0,x_1,\ldots,x_{n+1})$ and does not depend on x.

The methods of Curtiss and Hirschfelder now are simply explained. Let y(x) be an approximate solution of the differential equation

$$y' = f(x,y),$$

known at the n+1 grid-points $x_0,x_1,\ldots,x_n$.
For typographical reasons we shall write

$$xh = x_{n+1}$$

and

$$yh = y(xh).$$

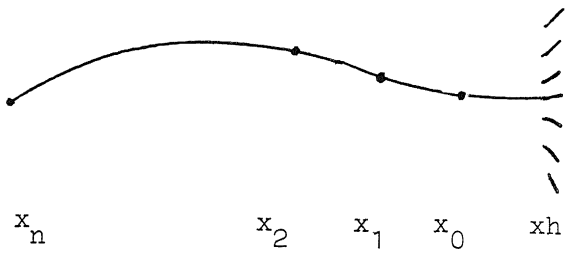

fig. 1. The method of Curtiss-Hirschfelder

We want to find a value y(xh) so that

$$y'(xh) = f(xh,y(xh)). \tag{2.1}$$

For y(x) we take a polynomial approximation by writing

$$y(x) = \vec{G}(x).\vec{a}_n + \vec{G}(x).\vec{a}_{n+1}$$

with $\vec{a}_n = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \\ 0 \end{pmatrix}$ and $\vec{a}_{n+1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{n+1} \end{pmatrix}$. (2.2)

Note that $\vec{a}_n$ is a vector known from earlier computations because y(x) is prescribed at the gridpoints $x_0,x_1,\ldots,x_n$. Furthermore, a_{n+1} is to be determined in such a way that (2.1) is satisfied. Using the definition of $\vec{\vec{A}}$ we get

$$y(x) = \vec{H}(x).\vec{\vec{A}}.\vec{a}_n + \vec{H}(x).\vec{\vec{A}}.\begin{pmatrix}0\\ \vdots\\ 0\\ 1\end{pmatrix} a_{n+1}. \qquad (2.3)$$

By defining $\vec{p}$, the Taylor coefficients of the n-th degree approximation to y(x) at the point x_0, as

$$\vec{p} = \vec{\vec{A}}\,\vec{a}_n \qquad (2.4)$$

and defining the (n+1)st column of $\vec{\vec{A}}$ as $\vec{A}_{n+1}$

$$\vec{A}_{n+1} = \vec{\vec{A}}\begin{pmatrix}0\\ \vdots\\ 0\\ 1\end{pmatrix} \qquad (2.5)$$

we may write

$$y(x) = \vec{H}(x).\vec{p} + \vec{H}(x).\vec{A}_{n+1}.a_{n+1}. \qquad (2.6)$$

Finally, by defining the differential operator D = d/dx, the next relation holds

$$D^j y(x) = D^j\vec{H}(x).\vec{p} + D^j\vec{H}(x)\vec{A}_{n+1}.a_{n+1}. \qquad (2.7)$$

We are now able to compute a <u>prediction</u> of $D^j y(xh)$, i.e.

$$D^j y(xh)^{pred} = D^j\vec{H}(xh).\vec{p}. \qquad (2.8)$$

(In fact, this is a Taylor series expansion of order n at the points x_0.) This prediction has to be followed by a <u>correction</u>, i.e.

$$\Delta D^j y(xh) = D^j\vec{H}(xh)\vec{A}_{n+1}.a_{n+1}. \qquad (2.9)$$

Here, a_{n+1} is unknown, but has to be determined so that (2.1) is satisfied. So we have the additional relation

$$\Delta Dy(xh) = f(xh,y(xh)) - Dy(xh)^{pred}. \qquad (2.10)$$

By successive substitution a direct iteration process can be obtained in which the correction of the computed value of Dy(xh) in the (m+1)-st iteration step is given by

$$\Delta_m^{m+1} Dy(xh) = f(xh,y(xh)^{'m'}) - Dy(xh)^{'m'}.$$

This relation together with (2.9) gives

$$\Delta_m^{m+1} D^j y(xh) = \frac{D^j \vec{H}(xh)\vec{A}_{n+1}}{D\vec{H}(xh)\vec{A}_{n+1}} \cdot [f(xh,y(xh)^{'m'}) - Dy(xh)^{'m'}]. \quad (2.12)$$

The first factor on the right hand-side, only depending on j, n+1 and the distribution of the grid-points, is a constant number during the iteration process. The second factor represents the difference between the derivative of the prediction and the prediction of the derivative at x = xh.

Instead of the forementioned direct iteration process, we can accelerate convergence by the use of Newton's method for solving $\Delta Dy(xh)$. In this case, we replace (2.11) by

$$\Delta_m^{m+1} Dy(xh) =$$

$$= f(xh,y(xh)^{'m'}) + J.\Delta_m^{m+1} y(xh) - Dy(xh)^{'m'} = \quad (2.13)$$

$$= (f(xh,y(xh)^{'m'}) - Dy(xh)^{'m'}) + \frac{\vec{H}(xh)\vec{A}_{n+1}}{D\vec{H}(xh)\vec{A}_{n+1}} .J.\Delta_m^{m+1} Dy(xh).$$

Here, J roughly represents $\frac{df}{dy}$. In the case of a system of differential equations, where y and f are vectors, J is roughly the Jacobian matrix $\partial f_i/\partial y_j$. Now $\Delta_m^{m+1} Dy(xh)$ is computed by solving the linear system

$$\{I - \frac{\vec{H}(xh)\vec{A}_{n+1}}{D\vec{H}(xh)\vec{A}_{n+1}} J\}\Delta_m^{m+1} Dy(xh) = f(xh,y(xh)^{'m'}) - Dy(xh)^{'m'}. \quad (2.14)$$

The same comment given for the factor in (2.12), can be made for the factor multiplying the Jacobian. As soon as $\Delta D_y(xh)$ has been solved from this (nonlinear) equation or system of equations, $\Delta D^j y(xh)$ also can be computed by using (2.9).

Explicitly (2.9) yields

$$\Delta D^j y(xh) = \frac{D^j \vec{H}(xh)\vec{A}_{n+1}}{D\vec{H}(xh)\vec{A}_{n+1}} .\Delta Dy(xh). \tag{2.15}$$

The first factor on the right-hand side only depends on j,n and the distribution of the grid-points.

The starting point of the Adams-Moulton methods are only slightly different from the Curtiss-Hirschfelder methods. Let y'(x) be an approximation to the derivative of the solution of the differential equation $\eta' = f(x,\eta)$.

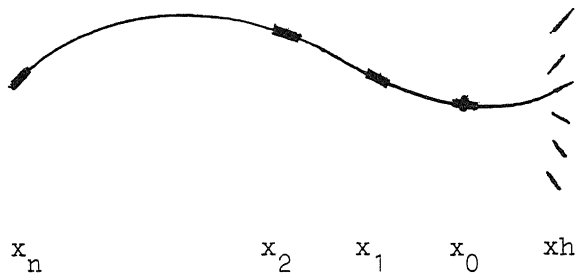

fig. 2. The method of Adams-Moulton

Let y'(x) be known at the n+1 grid-points $x_0,x_1,\ldots,x_n$. The same arguments as given for (2.2) - (2.5) yield

$$\begin{aligned} Dy(x) = y'(x) &= \vec{G}(x).\vec{a}_n + \vec{G}(x).\vec{a}_{n+1} \\ &= \vec{H}(x).\vec{p} + \vec{H}(x)\vec{A}_{n+1}a_{n+1}. \end{aligned} \tag{2.16}$$

Here, again the prediction is a Taylor-series expansion at the point x_0

$$D^j y(xh)^{pred} = D^{j-1}\vec{H}(xh).\vec{p}. \tag{2.17}$$

The correction is given by

$$\Delta D^j y(xh) = D^{j-1}\vec{H}(xh)\vec{A}_{n+1}a_{n+1}. \tag{2.18}$$

In order to compute $D^j y(xh)$ we again use (2.10) and we construct a direct iteration process analogous to (2.12)

$$\Delta_m^{m+1} D^j y(xh) = \frac{D^{j-1}\vec{H}(xh)\vec{A}_{n+1}}{\vec{H}(xh)\vec{A}_{n+1}} [f(xh,y(xh)^{'m'})-Dy(xh)^{'m'}]. \qquad (2.19)$$

Alternatively, we construct an iteration process by using an estimated Jacobian matrix as in (2.13):

$$\{I - \frac{D^{j-1}\vec{H}(xh)\vec{A}_{n+1}}{\vec{H}(xh)\vec{A}_{n+1}} J\}\Delta_m^{m+1} Dy(xh) = f(xh,y(xh)^{'m'}) - Dy(xh)^{'m'}. \qquad (2.20)$$

$\Delta y(xh)$ and $\Delta Dy(xh)$ being computed in this way, the corrections of the higher derivatives are easily obtained with the formula analogous to (2.15)

$$\Delta D^j y(xh) = \frac{D^{j-1}\vec{H}(xh)\vec{A}_{n+1}}{\vec{H}(xh)\vec{A}_{n+1}} .\Delta Dy(xh). \qquad (2.21)$$

3. Administration and realization

Multistep methods essentially carry information about the last n performed steps. This information has to be available in one form or another. According to Nordsieck's method we prefer to store at each step the numbers

$$Y(x_0) = \{y(x_0,hDy(x_0),h^2D^2y(x_0)/2,\ldots,h^nD^ny(x_0)/n!\}.$$

This vector $Y(x_0)$ corresponds to the vector $\vec{p}$ introduced in (2.4), taken into account that in Y the i-th derivative is normalized by a factor $h^i/i!$. For h we take the current steplength $h = xh-x_0$.

The prediction as described in (2.8) and (2.17) now easily becomes a multiplication of $\vec{Y}$ by a Pascal matrix (Gear [1967]). By one of the processes described in (2.11), (2.14) or (2.20), both $\Delta hDy(xh)$ and $\Delta y(xh)$

are computed. When convergence has been obtained the values of $h^i D^i y(xh)/i!$ are computed with (2.15) or (2.21).

In order to make quick reference possible we define in the case of a uniform distributed set of grid-points

$$a_{kl} = \frac{h^{l-1} D^l H(xh) A_k}{l! DH(xh) A_k}$$

in the case of C.H. methods, and

$$a_{kl} = \frac{h^{l-1} D^{l-1} H(xh) A_{k-1}}{l! H(xh) A_{k-1}} \qquad (3.1)$$

in the ease of A.M. methods.
These constant easily change the computed difference $\Delta hDy(xh)$ into the differences $\Delta h^j D^j y(xh)/j!$.

In order to change the steplength during the integration, every element $h^i D^i y(x_0)/i!$ of the vector $Y(x_0)$ is multiplied by a factor $(hnew/h)^i$. The vector $Y(x_0)$ preserves estimates of the Taylor coefficients at x_0. As these values contain all essential information available, we assume that the grid-points have a uniform distribution. Experiments by Nordsieck [1962] and Gear [1968] show that no difficulties are to be expected at this point, provided that, after each change of steplength, this steplength is fixed at least n times in succession. A method implementing a fully variable steplength is discussed in Hemker [1971].

The uniformity of the distribution of $\{x_i\}$ and the normalization of p make it possible to represent all occurences of $D^j H(xh) A_{n+1}$ by the constant numbers a_{kl}. This feature and the flexability of the steplength make it very attractive to implement the methods in this way.

4. The control of steplength and order

The greater part of our steplength and order control mechanism is due to Gear [1968,1971]. The basic idea is the following: after completion of a number of steps, we look for that steplength which gives rise to a truncation error (represented by the last term of the Taylor-series expansion taken into account) equal to a prescribed local error E. This

inquiry is done for the current n-th order formula, and, in addition, for the (n-1)-st order and the (n+1)-st order formula. The resulting optimal steplength is chosen together with the corresponding order. A number of safety measures are taken in order to prevent some unwanted effects (see section 5).

The calculation of the optimal steplength for the current order n

Since the value $h^n D^n y(x_0)/n!$ also represents the predicted value of $h^n D^n y(xh)/n!$, it is easily shown that its first difference

$$\Delta h^n D^n y(xh)/n!$$

represents an approximation to

$$h^{n+1} D^{n+1} y(xh)/n! \ . \tag{4.1}$$

The truncation error of the n-th order formula is given by

$$C_{n+1} h^{n+1} D^{n+1} y(xh) \tag{4.2}$$

where, in the case of A.M. methods, C_{n+1} corresponds with $|\gamma_n^*|$ (see Henrici [1962] p. 195), and
where, in the case of C.H. methods, C_{n+1} corresponds with $\delta_{0,n+1}$ (see Henrici [1962] p. 208).

Aiming at a local error E, we obviously have to take a value of h in such a way that

$$\begin{aligned} E &\approx C_{n+1} h^{n+1} D^{n+1} y(xh) \\ &\approx C_{n+1} n! \Delta h^n D^n y(xh)/n! \\ &= C_{n+1} n! a_{nn} \Delta h D y(xh). \end{aligned} \tag{4.3}$$

Since this truncation error is proportional to h^{n+1}, we have to multiply the steplength by a factor

$$\left(\frac{E}{C_{n+1}n!a_{nn}||\Delta\ hDy(xh)||}\right)^{\frac{1}{n+1}}. \tag{4.4}$$

In the program this is realized by computing

$$\text{error} = \left|\left|\frac{hDy(xh)}{ymax}\right|\right|^2 \tag{4.5}$$

$$\text{tol} = \left(\frac{E/ymax}{C_{n+1}a_{nn}n!}\right)^2. \tag{4.6}$$

The new steplength is obtained by multiplying the old steplength by the factor ch

$$ch = (tol/error)\uparrow(0.5/(n+1)). \tag{4.7}$$

Note that E/ymax represents the relative error specified by the user.

The calculation of the optimal steplength in the case of order n-1

The stored value $h^nD^ny(xh)/n!$ itself yields an approximation to the truncation error of order n, i.e. $C_nh^nD_ny(xh)$. Aiming of the local error E, we have to take h in such a way that $E = C_nh^nD^ny(xh)$. Consequently, we have to change our steplength by a factor

$$ch = \left(\frac{E}{C_nD^ny(xh)}\right)^{1/n}. \tag{4.8}$$

The actual computation is effectuated in a way analogous to (3.5), (3.6), (3.7), i.e.

$$\text{error} = \left|\left|\frac{h^nD^ny(xh)/n!}{ymax}\right|\right|^2, \tag{4.9}$$

$$\text{tol} = \left(\frac{E/ymax}{C_nn!}\right)^2, \tag{4.10}$$

$$ch = (tol/error)\uparrow(0.5/n). \tag{4.11}$$

The calculation of the optimal steplength for order n+1

In the program an approximation to the truncation error of order n+2 is not directly available. However, it is obtained from the values $\Delta hDy(xh)$ and $\Delta hDy(x_0)$. An estimation of the truncation error can be directly derived from

$$h^n D^{n+2} y(xh) \approx \Delta h^{n+1} D^{n+1} y(xh)$$

$$\approx \Delta h^n D^n (y(xh)-y(x_0))$$

$$= a_{nn} n!(\Delta hDy(xh)-\Delta hDy(x_0)).$$

The factor by which the steplength is to be multiplied, is again computed in analogy to the expressions (3.5), (3.6) and (3.7), that is

$$\text{error} = \left|\left|\frac{\Delta hDy(xh) - \Delta hDy(x_0)}{ymax}\right|\right|^2, \qquad (4.12)$$

$$\text{tol} = \left(\frac{E/ymax}{C_{n+2} a_{nn} n!}\right)^2, \qquad (4.13)$$

$$\text{ch} = (\text{tol}/\text{error})\uparrow(0.5/(n+2)). \qquad (4.14)$$

5. The structure of the procedure MULTISTEP
A strategy for solving stiff equations

For high precision computation of solutions of differential equations A.M. methods are favourable because of their excellent accuracy properties, whereas C.H. methods have superior properties with respect to stability behaviour and are, therfore, appropriate when solving stiff equations. Unlike the choice of steplength and order which is reversible (i.e. a decision can be annulled by a next decision), the choice which type of two methods will be used, is made irreversible. When, after a call of procedure MULTISTEP with first = true, A.M. methods with a minimal steplength do not satisfy, the equation is considered to be

stiff and C.H. methods are used until a next call with first = true.

Solving the differential equation by means of an A.M. method, the decision to use Newton's method for solving equation (2.10) is made when the iteration process (2.12) turns out to be slowly convergent. Unless a call with first = true this decision will not be revoked.

A detailed description of the strategy will be given as an explanation of the ALGOL 60 procedure MULTISTEP of section 6.

In this procedure some subprocedures are declared:

(1) procedure method.
According to the Boolean value "adams" (i.e. use A.M. methods) this procedure stores the constants of (3.1), (4.6), (4.10) and (4.13):
$(a_{nj}\ \ j = 0(1)n,\ 1/(C_n n!),\ 1/(C_{n+1}.a_{nn} n!),\ 1/(C_{n+2}.a_{nn}.n!))$
in an array called "const".

(2) procedure order.
According to the integer value k (i.e. the order of the formula that is to be used), this procedure activates the values a_{kj} and calculates the values tol (4.6), (4.10) and (4.13). When Newton's method is used to solve (2.14) or (2.21), a new evaluation of the Jacobian matrix and a corresponding LU-decomposition is asked for. Inquiry into new steplength and order is delayed for k+1 steps.

(3) procedure evaluate jacobian.
This procedure evaluates the Jacobian matrix (either by numerical calculation or by evaluating the analytical expression given in one of the parameters of MULTISTEP) and then performs a LU-decomposition of the matrix $(I-a_{0k}hJ)$.

(4) procedure calculate step and order
performs the calculations described in section 4. A suggestion for a new order (k new) and a factor changing the steplength (ch) are delivered. In order to prevent unnecessary changing, some safety-margins are incorporated. Searching for a new steplength and order is delayed for 10 steps.

(5) procedure set.
In order to make rejection of a computed step possible, this procedure stores the last accepted values of the computation in array dd.

(6) procedure reset step
restarts computation from the last accepted values and changes steplength by multiplying by ch (see procedure calculate step and order).

(7) procedure begin
starts integration with minimal steplength and order 1. Values of the derivative are obtained by evaluation of the right hand side of the differential equation.

The compound tail of the procedure body is divided in some clear distinguishable parts.

1. Initialization.
If first = true the procedure starts integration with A.M. methods by calling procedure begin; otherwise computation is continued from the last accepted values.

2. Integration by steps.

2.1. prediction is performed as described in (2.8) and (2.17). See also section 3.

2.2. correction
Maximal 3 corrector iterations are taken as described in (2.11), (2.13) or (2.20). When

$$||\Delta_m^{m+1} hDy(xh)||_{sup} < \frac{E}{2.N.(k+2)}$$

where E: absolute error
N: number of equations
k: order of the method

convergence is assumed to be achieved.

2.3. if no convergence is obtained then
i) if a not updated Jacobian matrix was used, this matrix is updated. The step is repeated.

ii) if the steplength is not minimal, the step is repeated with steplength/4 or minimal steplength. When the Jacobian matrix can be obtained from the parameters, Newton's method will be used in order to accelerate convergence.

iii) if steplength is minimal then

iiia) if A.M. methods were used, C.H. methods will be used (in order to force the use of the Jacobian matrix).

iiib) when C.H. methods were used (and so Newton's method is used for solving (2.20)), the strong nonlinearity of the problem prevents solution with the given minimal steplength. An errormessage is given and integration is stopped. It is possible to restart with first = true or first = false and with a smaller hmin.

2.4. if the requested local error bound is exceeded, a smaller steplength is calculated and the step is repeated.

i) when recalculation of the steplength does not satisfy two times in succession, the step is restarted after a call of procedure begin (so hmin will be used).

ii) when the minimal steplength (hmin) is used, it is not possible to obtain the requested local error with this method. Two possibilities arise

iia) An A.M. method is used. In this case the error might be caused by the bad stability properties of the differential equation: the C.H. method will be used and the step will be repeated.

iib) The C.H. method is used but the discretisation error exceeds the required error bound. In this case a first order method will be used. If even this does not satisfy, strictly speaking, the procedure would have to give an errormessage and ask for a smaller hmin. However, experience has shown that this situation appears when starting stiff differential equations. Here the excessive error will be damped out in some steps and a first order C.H. method is an optimal choice. So we will give an errormessage (and an estimation of the real local error), but we will continue integration, violating our local error criterion by accepting this step.

2.5. when the step is accepted,
the vector $Y(x_0)$ (section 3) is updated.
Every ten steps a new order and steplength are calculated. Only if improvement is worthwhile the new steplength and order are used. After completion of a step the computed values are stored by procedure "set".

6. The ALGOL 60-procedure MULTISTEP

In this section we give an ALGOL 60-procedure appropriate to the integration of a system of first order differential equations. Firstly, a description of its parameters and an example of the use of the procedure will be given.

The heading of the procedure is:

```
procedure MULTISTEP (x,xend,y,hmin,hmax,ymax,eps,first,dd,fxyi,ii,
          jacij,jj,n,available,stiff);
value hmin,hmax,eps,xend,available,n;
Boolean available,stiff,first; integer ii,jj,n;
real x,xend,hmin,hmax,eps,fxyi,jacij; array y,ymax,dd;
```

The actual parameters corresponding to the formal parameters are:

x : <variable>;
the independent variable x can be used as a Jensen parameter in fxyi and jacij.
entry: the initial value x_0.

xend : <expression>;
the end value of x (xend $\geq$ x).

y : <array identifier>; array y[0:7,1:n];
the independent variable
entry: the initial values of the system of differential equations $y[0,i] := y_i(x_0)$.
exit : $y_i(xend) = y[0,i]$.

hmin,hmax : <expression>;
minimal respectively maximal steplength by which the integration is performed.

eps : <expression>;
the relative local error bound.

ymax : <array identifier>; array ymax[1:n];
entry: the absolute local error bound divided by eps;
exit : ymax[i] gives the maximal value of abs(y[0,i]) and
the entry value of ymax[i] during integration.

first : <Boolean identifier>;
if first = true then the procedure starts its strategy with the first order Adams-Moulton method and a steplength equal to hmin. Upon completion of a call first = false.
if first = false then the procedure continues the integration.

dd : <array identifier>; array dd[0:7,0:n];
in this array information is stored, which can be used in a next call;
besides some messages are delivered:
dd[0,0] = 0 A.M. method was used;
= 1 the procedure switched to C.H. method;
dd[1,0] = 0 no error message;
= 1 with the used hmin the procedure cannot handle the nonlinearity;
dd[2,0] number of times that the requested local error bound was exceeded;
dd[3,0] if dd[2,0] ≠ 0 then d[3,0] gives an estimate of the maximal local error bound.

ii,jj : <integer identifier>;
are used as Jensen parameters in fxyi and jacij.

fxyi : <expression>;
expression depending on x,y and ii, giving the value of dy_{ii}/dt.

jacij : <expression>;
(optional) expression depending on x,y,ii and jj and giving the values of $d(dy_{ii}/dt)/dy_{jj}$
(i.e. the Jacobian matrix of the system).

available : <Boolean expression>;
an expression which indicates whether expression jacij contains relevant information or not.

stiff : <Boolean expression>;
if stiff = true then the procedure directly uses Curtiss-Hirschfelder methods, skipping an attempt with Adams-Moulton methods.

n : <expression>;
the number of equations.

In procedure MULTISTEP the following library procedures are used: sum, det, sol, elmrow and elmcolvec. (see: Dekker [1968]).

Example of a call of multistep:

Consider the initial value problem

$$\begin{cases} \frac{d}{dt} x = -x(1-y) + qy \\ \varepsilon \frac{d}{dt} y = x(1-y) - py \end{cases}$$

$$x_0 = 1, \quad y_0 ; 0.$$

This can be programmed as follows:

```
p:= 1; q:= 0.99; ep:= 0.001; tend:= 25;
ymax[0,1]:= ymax[0,2]:= ymax[2]:= 1;
y[0,2]:= 0; first:= true;
MULTISTEP(t,tend,y,10-3,0.5,ymax,10-6,first,dd,
if i=1 then -(1-y[0,2])*y[0,1]+q*y[0,2]
       else ((1-y[0,2])*y[0,1]-p*y[0,2])/ep,i,
if i=1 then (if j=1 then -(1-y[0,2])    else   q+y[0,1])
       else (if j=1 then  (1-y[0,2])/ep else -(p+y[0,1])/ep),j,
2, true, false);
```

The following results are delivered:
t=25, y[0,1] = .87755215, y[0,2] = .467675852 (correct digits are underlined), first = false, d[0,0] = 1, d[1,0] = 0, d[2,0] = 12, d[3,0] = .333.

```
procedure MULTISTEP(x,xend,y,hmin,hmax,ymax,eps, first,dd,
                fxyi,ii,jacij,jj,n,available,stiff);
value hmin,hmax,eps,xend,available,n;
boolean available,stiff,first; integer ii,jj,n;
real x,xend,hmin,hmax,eps,fxyi,jacij; array y,ymax,dd;

begin own boolean  with jacobian,adams;
      own integer kold; own real xold,hold;
      boolean evaluate,evaluated,conv;
      integer i,j,l,k,knew,maxorder,fails,same;
      real h,ch,chnew,c,tolconv,tolup,tol,toldwn,error,dfi;
      array const[1:56],a[0:7],delta,last delta,df[1:n],jac[1:n,1:n];
      integer array p[1:n];

      procedure method;
      begin with jacobian:= ¬adams;
            maxorder:= if adams then 7 else 6;
            i:= k:= 1;
            if adams then
            begin for const[i]:= 1,1,12,2,1,.5,1,.5,24,12,1,5/12,1,.75,
                  1/6,37.89,24,2,.375,1,11/12,1/3,1/24,53.33,37.89,1,
                  251/720,1,25/24,35/72,5/48,1/120,70.08,53.33,.3158,
                  95/288,1,137/120,.625,17/96,.025,1/720,87.97,70.08,
                  .07407,19087/60480,1,1.225,203/270,49/192,7/144,
                  7/1440,1/5040,106.9,87.97,.0139 do i:= i + 1
            end else
            begin for const[i]:= 1,1,3,2,1,2/3,1,1/3,6,4.5,1,6/11,1,
                  6/11,1/11,9.167,7.333,0.5,.48,1,.7,.2,.02,12.5,
                  10.42,.1667,120/274,1,225/274,85/274,15/274,1/274,
                  15.98,13.7,.04167,180/441,1,58/63,5/12,25/252,
                  3/252,1/1764,19.6,17.15,.008333 do i:= i + 1
            end
      end method;

      procedure order;
      begin if k>max order then begin dd[1,0]:= 2; goto return end;
            j:= (k-1) × (k+8) / 2 + 1;
            for i:= 0 step 1 until k do a[i]:= const[i+j];
            tolup := (eps×const[j+k+1])↑2;
            tol   := (eps×const[j+k+2])↑2;
            toldwn:= (eps×const[j+k+3])↑2;
            tolconv:= eps/(2×n×(k+2));
            evaluate:= with jacobian;
            same:= k+1
      end order;
```

```
procedure evaluate jacobian;
begin real r;
      evaluate:= false;
      if available then
      begin r:= -a[0] × h;
            for ii:= 1 step 1 until n do
            for jj:= 1 step 1 until n do jac[ii,jj]:= jacij × r;
      end else
      begin real d; array fixdy,fix y[1:n];
            for ii:=1 step 1 until n do
                  begin fix y[ii]:= y[0,ii]; fixdy[ii]:= fxyi end;
            for i:=1 step 1 until n do
            begin d:= if eps>abs(fix y[i]) then eps×eps
                        else eps×abs(fix y[i]);
                  y[0,i]:= y[0,i] + d;
                  r:= - a[0] × h/d;
                  for ii:= 1 step 1 until n do
                        jac[ii,i]:=(fxyi - fixdy[ii]) × r;
                  y[0,i]:= fix y[i]
            end
      end;
      for i:= 1 step 1 until n do jac[i,i]:= jac[i,i] + 1;
      det(jac,n,p);
      evaluated:= true
end evaluate jacobian;

procedure calculate step and order;
begin real a1,a2,a3;
      same:= 10;
      a1:= if k<1 then 0 else
           0.75×(toldwn/sum(i,1,n,(y[k,i]/ymax[i])↑2))↑(0.5/k);
      a2:= 0.80×(tol   /error ) ↑ (0.5/(k+1));
      a3:= if k>max order ∨ fails≠0 then 0 else
           0.70×(tolup /sum(i,1,n,((delta[i]-last delta[i])/
                 ymax[i])↑2))↑(0.5/(k+2));
      if a1>a2 ∧ a1>a3 then begin knew:=k-1; chnew:=a1 end else
      if a2>a3         then begin knew:=k  ; chnew:=a2 end else
                            begin knew:=k+1; chnew:=a3 end
end calculate step and order;

procedure set;
begin xold:= x; hold:= h; kold:= k; ch:= 1;
      for i:= 1 step 1 until n do
      for j:= 0 step 1 until k do dd[j,i]:= y[j,i]
end set;
```

```
procedure reset step;
begin real c;
      if ch < hmin/hold then ch:= hmin/hold else
      if ch > hmax/hold then ch:= hmax/hold;
      x:= xold; h:= hold × ch; c:= 1;
      for j:=0 step 1 until k do
      begin for i:=1 step 1 until n do y[j,i]:= dd[j,i] × c;
            c:= c × ch
      end;
      same:= k + 1
end reset step;

procedure begin;
begin fails:= 0; h:= hmin;
      for ii:= 1 step 1 until n do y[1,ii]:= fxyi × h;
      k:= 1; order; set
end begin;


if first then
begin first:= false; adams:= ¬stiff; method;
      begin; for i:= 1,2,3 do dd[i,0]:= 0
end else
begin method; k:= kold; order; ch:= 1; reset step end;

for l:= 0 while x<xend do
begin if x+h<xend then x:= x+h else
      begin ch:= (xend-x)/h; reset step; x:= xend end;

      comment prediction;
      for i:=0 step 1 until k-1 do
      for j:= k-1 step -1 until i do
      elmrow(1,n,j,j+1,y,y,1);
      for i:= 1 step 1 until n do delta[i]:= 0;
```

```
comment correction and estimation local error;
for l:=1,2,3 do
begin for ii:=1 step 1 until n do df[ii]:= fxyi×h - y[1,ii];
      if evaluate then evaluate jacobian;
      if with jacobian then sol(jac,n,p,df);

      conv:= true;
      for i:= 1 step 1 until n do
      begin dfi:= df[i];
            y[0,i]:= y[0,i] + a[0]×dfi;
            y[1,i]:= y[1,i] +      dfi;
            delta[i]:= delta[i] +  dfi;
            conv:= conv ∧ abs(dfi) < tolconv × ymax[i]
      end;
      if conv then
      begin error:= sum(i,1,n,(delta[i]/ymax[i])↑2);
            evaluated:= false; goto convergence
      end
end;

comment acceptance or rejection;
if ¬conv then no convergence:
begin if with jacobian ∧ ¬ evaluated then else
      if h>hmin×1.0001 then
      begin with jacobian:= with jacobian ∨ available;
            ch:= ch/4
      end else
      if adams then goto try curtiss else
      begin dd[1,0]:= 1; goto return end;

      evaluate:= with jacobian; reset step
end else convergence:

if error>tol then error test not ok:
begin fails:= fails + 1;
      if h>hmin×1.0001 then
      begin if fails>2 then
            begin k:= 0; reset step; begin
            end else
            begin calculate step and order;
                  if knew≠k then begin k:= knew; order end;
                  ch:= ch×chnew/fails; reset step
            end
      end else
      if adams then try curtiss:
      begin adams:= false; method; order; reset step end else
      if k≠1 then
      begin k:=1; order; reset step end else
```

```
                begin comment violate eps criterion;
                      c:= eps × sqrt( error/tol );
                      if c>dd[3,0] then dd[3,0]:= c;
                      dd[2,0]:= dd[2,0] + 1;
                      goto error test ok
                end
          end else

          error test ok:
          begin fails:= 0;
                if k>2 then begin for i:=1 step 1 until n do
                      elmcolvec(2,k,i,y,a,delta[i]) end;
                for i:= 1 step 1 until n do if abs(y[0,i])>ymax[i]
                      then ymax[i]:=abs(y[0,i]);
                same:= same - 1;
                if same=1 then begin for i:=1 step 1 until n do
                      last delta[i]:= delta[i] end else
                if same=0 then
                begin calculate step and order;
                      if chnew>1.1 then
                      begin same:= k + 1;
                            if knew≠k then
                            begin if knew>k then
                                  begin for i:=1 step 1 until n do
                                        y[knew,i]:= delta[i]×a[k]/knew
                                  end;
                                  k:= knew; order
                            end;
                            if chnew> hmax/h then chnew:= hmax/h;
                            h:= h × chnew; c:= 1;
                            for j:=1 step 1 until k do
                            begin c:= c × chnew;
                                  for i:=1 step 1 until n do
                                  y[j,i]:= y[j,i]×c
                            end
                      end
                end;
                set
          end
    end step;
return: dd[0,0]:= if adams then 0 else 1; dd[4,0]:= k
end MULTISTEP;
```

7. Numerical results

A number of differential equations are selected as testproblems. At each point x_i where the solution is asked for, the following data have been delivered:

1. used method
   A : Adams-Moulton methods
   C : Curtiss-Hirschfelder methods
   C*: Curtiss-Hirschfelder methods, with disregard of the local error bound (see section 5).
2. absolute error $|y(x_i)-\tilde{y}(x_i)|$ or
   relative error (number of correct digits: $-^{10}\log |y(x_i)-\tilde{y}(x_i)|$, where $\tilde{y}(x_i)$ is a sufficiently close approximation to the exact solution.
3. number of evaluations of the right-hand side of the differential equation (f), counted during integration from x_0 until x_i.
4. number of evaluations of the Jacobian matrix (J), counted during integration from x_0 until x_i.

The results obtained for different values of the accuracy parameters eps and hmin are given in a table. All calculations were carried out on the EL X8 computer of the Mathematical Centre.

7.1. The differential equation

$$\begin{cases} y'(x) = -y(x) \\ y(0) = 1 \end{cases}$$

Parameters used: available = true, stiff = false.
At x = 1 the results are given in

Table 7.1.1. Correct digits of y (left) and method used (right)

eps \ hmin	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
10^{-2}	1.7 A	3.4 A	3.4 A	3.4 A	3.4 A	3.4 A
10^{-3}	1.7 C*	3.2 A	3.4 A	3.4 A	3.4 A	3.4 A
10^{-4}	1.3 C*	3.5 A	4.5 A	4.5 A	4.5 A	4.4 A
10^{-5}	1.3 C*	4.0 C*	4.7 A	4.7 A	4.7 A	4.7 A

Table 7.1.2. Evaluations of f (left) and of J (right)

19 0	18 0	20 0	20 0	20 0	20 0
24 4	22 0	25 0	25 0	25 0	25 0
24 2	35 0	32 0	32 0	32 0	32 0
24 2	46 5	43 0	43 0	43 0	43 0

Parameters used: available = true, stiff = true.
At x = 1 the results are

Table 7.1.3. Correct digits and method used

eps \ hmin	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
10^{-2}	1.8 C	4.0 C	3.0 C	2.9 C	2.9 C	2.9 C
10^{-3}	1.7 C*	4.0 C	3.0 C	2.9 C	2.9 C	2.9 C
10^{-4}	1.3 C*	2.7 C	3.3 C	3.9 C	3.9 C	3.5 C
10^{-5}	1.3 C*	4.0 C*	4.5 C	4.8 C	4.9 C	4.6 C

Table 7.1.4. Evaluations of f (left) and of J (right)

19 3	20 5	22 5	22 5	22 5	22 5
21 4	23 5	23 5	23 5	23 5	23 5
21 2	35 5	47 7	34 5	34 5	52 6
21 2	44 5	51 5	45 5	45 5	60 5

7.2. The differential equation

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1 \end{cases}$$

$$y_1(0) = 0,\ y_2(0) = 1$$

Parameters used: available = true, stiff = false.
At $x = \pi/4$ the results are:

Table 7.2.1. Correct digits of y_2 - for y_1 the results are similar - (left) and method used (right)

eps \ hmin	0.05	0.02	0.01	0.005	0.002	0.001
10^{-2}	2.7 A	3.5 A	4.4 A	3.9 A	3.8 A	3.8 A
10^{-3}	1.8 C*	3.4 A	4.2 A	5.0 A	4.5 A	4.4 A
10^{-4}	1.5 C*	3.7 C*	3.9 A	4.2 A	4.3 A	3.5 A
10^{-5}	1.5 C*	1.6 C*	4.2 C*	5.2 C*	4.9 A	5.0 A

Table 7.2.2. Evaluations of f (left) and of J (right)

18 0	18 0	18 0	18 0	18 0	18 0
26 2	24 0	24 0	22 0	22 0	22 0
36 1	41 0	44 0	44 0	27 0	28 0
36 1	84 1	54 4	53 4	50 0	50 0

Parameters used: available = true, stiff = true.
At x = $\pi/4$ the results are:

Table 7.2.3. Correct digits of y_2 and method used

eps \ hmin	0.05	0.02	0.01	0.005	0.002	0.001
10^{-2}	3.3 C	2.6 C	2.6 C	2.6 C	2.6 C	2.6 C
10^{-3}	1.8 C*	2.6 C	2.5 C	2.5 C	2.5 C	2.5 C
10^{-4}	1.5 C*	3.7 C*	3.9 C	3.7 C	3.7 C	3.6 C
10^{-5}	1.5 C*	1.6 C*	4.2 C*	5.2 C*	4.6 C	4.5 C

Table 7.2.4. Evaluations of f and J.

	18 3	18 3	18 3	18 3	18 3	18 3
	23 2	25 3	25 3	23 3	23 3	23 3
	33 1	39 3	39 3	39 3	37 3	37 3
	33 1	81 1	51 4	51 4	51 4	51 4

7.3. The differential equation

$$\begin{cases} y' = e^x \log x - e^x y + x^{-1} \\ y(0.01) = \log(0.01) \end{cases}$$

parameters used: available = true, stiff = false.
At x = 0.165 the results are:

Table 7.3.1. Correct digits (left) and method used (right)

eps \ hmin	10^{-2}	10^{-3}	10^{-4}	10^{-5}
10^{-4}	.8 C*	1.9 C*	3.1 A	3.1 A
10^{-5}	.8 C*	1.8 C*	4.7 C*	4.4 A
10^{-6}	.8 C*	1.7 C*	4.3 C*	5.2 A
10^{-7}	.8 C*	1.7 C*	3.0 C*	6.3 C*

Table 7.3.2. Evaluations of f (left) and of J (right)

36 1	80 5	59 0	57 0
46 1	126 4	113 6	79 0
51 1	214 4	147 7	105 0
51 1	316 1	321 8	202 10

At x = 2.5 the results are

Table 7.3.3. Correct digits and method used

eps \ hmin	10^{-2}	10^{-3}	10^{-4}	10^{-5}
10^{-4}	5.3 C*	5.4 C*	6.0 A	5.2 A
10^{-5}	2.4 C*	6.7 C*	6.5 C*	6.8 A
10^{-6}	5.4 C*	6.2 C*	7.0 C*	8.4 A
10^{-7}	2.5 C*	6.2 C*	7.5 C*	7.6 C*

Table 7.3.4. Evaluations of f and of J

118 7	158 9	132 0	141 2
186 9	227 13	211 13	177 1
322 9	385 13	283 15	226 4
725 13	547 13	569 22	401 20

At x = 6.5 the results are

Table 7.3.5. Correct digits and method used

eps \ hmin	10^{-2}	10^{-3}	10^{-4}	10^{-5}
10^{-4}	4.3 C*	5.3 C*	5.1 A	5.7 A
10^{-5}	6.4 C*	7.2 C*	8.2 C*	4.3 A
10^{-6}	7.8 C*	4.4 C*	7.9 C*	6.7 A
10^{-7}	8.9 C*	10.1 C*	8.7 C*	8.8 C*

Table 7.3.6. Evaluations of f and of J

	197 17	236 17	302 24	290 21
	306 21	310 21	282 20	442 28
	414 17	487 25	361 21	638 40
	941 24	693 25	703 34	590 38

7.4. The differential equation

$$\begin{cases} y_1' = -500.5\, y_1 + 499.5\, y_2 + 2 \\ y_2' = 499.5\, y_1 - 500.5\, y_2 + 2 \end{cases}$$

$$y_1(0) = -0.1, \quad y_2(0) = 0.1$$

parameters used: available = true, stiff = false.
At x = 1 the results are

Table 7.4.1. Correct digits of y_1 and y_2 (left) and method used (right)

eps \ hmin	0.1	0.05	0.02	0.01	0.005	0.002
10^{-1}	1.6 C*	1.2 C*	1.2 C*	1.2 C*	2.4 C*	1.4 C
10^{-2}	1.6 C*	2.2 C*	2.3 C*	2.3 C*	2.3 C*	2.9 C*
10^{-3}	1.6 C*	2.1 C*	2.9 C*	3.4 C*	3.8 C*	3.2 C*
10^{-4}	1.6 C*	1.8 C*	2.2 C*	3.5 C*	4.1 C*	4.1 C*
10^{-5}	1.6 C*	1.8 C*	2.6 C*	3.4 C*	4.0 C*	5.1 C*

Table 7.4.2. Evaluations of f (left) and of J (right)

	15 2	18 3	22 2	26 3	29 4	28 5
	24 1	32 2	27 3	28 3	30 3	35 4
	24 1	36 3	43 3	45 3	39 3	48 4
	24 1	44 1	53 4	55 4	56 4	66 6
	24 1	44 1	80 3	85 4	86 5	81 6

At x = 2 the results are

Table 7.4.3. Correct digits of y_1 and y_2 (left) and method used (right)

eps \ hmin	0.1	0.05	0.02	0.01	0.005	0.002
10^{-1}	2.0 C*	2.3 C*	1.7 C*	1.4 C*	1.5 C*	2.4 C
10^{-2}	2.3 C*	2.6 C*	2.7 C*	2.2 C*	3.9 C*	3.4 C*
10^{-3}	2.1 C*	2.3 C*	2.9 C*	3.1 C*	3.9 C*	3.8 C*
10^{-4}	1.8 C*	2.4 C*	2.8 C*	3.7 C*	6.2 C*	5.1 C*
10^{-5}	1.8 C*	4.2 C*	3.2 C*	4.0 C*	4.6 C*	6.2 C*

Table 7.4.4. Evaluations of f and of J

21 4	28 5	28 4	35 5	37 6	34 6
34 3	42 4	38 5	39 6	38 4	45 6
40 4	53 5	60 5	61 5	51 5	58 6
44 2	72 5	72 7	71 7	71 5	89 9
44 2	86 2	106 5	106 6	111 8	106 9

At x = 10 the results are

Table 7.4.5. Correct digits of y_1 and y_2 (left) and method used (right)

eps \ hmin	0.1	0.05	0.02	0.01	0.005	0.002
10^{-1}	2.7 C*	3.4 C*	3.0 C*	3.2 C*	3.2 C*	3.5 C
10^{-2}	4.0 C*	3.9 C*	3.2 C*	3.3 C*	6.0 C*	3.8 C*
10^{-3}	4.3 C*	3.9 C*	3.6 C*	3.6 C*	4.4 C*	4.7 C*
10^{-4}	5.5 C*	5.1 C*	5.6 C*	5.0 C*	4.9 C*	5.6 C*
10^{-5}	4.9 C*	7.6 C*	5.9 C*	6.5 C*	6.5 C*	5.8 C*

Table 7.4.6. Evaluations of f and J

41 7	48 8	46 7	52 7	57 10	63 9
57 7	67 7	70 9	94 12	59 7	68 10
83 7	94 8	108 12	109 12	91 7	105 9
110 6	133 7	135 11	128 11	123 9	155 13
159 6	214 6	173 8	176 11	180 13	179 13

7.5 The differential equation

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ y_3' = -10^6 y_1 - 1001\,10^3 y_2 - 1001 y_3 \end{cases}$$

$$y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 1$$

Parameters used: available = true, stiff = false.
At x = 0.5 the results are

Table 7.5.1. Correct digits of y_1, y_2 and y_3 (left) and method used (right)

eps \ hmin	0.05	0.02	0.01	0.005	0.002	0.001
10^{-2}	2.7 A	3.9 A	2.3 A	3.5 A	2.2 A	2.2 A
10^{-3}	2.1 C*	3.3 A	2.3 A	1.4 A	3.6 A	3.6 A
10^{-4}	1.9 C*	3.4 C*	2.1 A	3.9 A	4.1 A	1.5 A
10^{-5}	1.9 C*	2.3 C*	4.1 C*	4.9 C*	1.3 A	2.1 A

Table 7.5.2. Evaluations of f (left) and of J (right)

17 1	20 2	22 2	23 2	20 3	19 3
18 3	22 3	23 3	35 4	22 3	22 3
24 1	28 3	31 3	32 3	28 3	40 5
24 1	54 1	53 4	53 4	46 5	43 5

At x = 1.0 the results are

Table 7.5.3. Correct digits of y_1, y_2 and y_3 (left) and method used (right)

eps \ hmin	0.05	0.02	0.01	0.005	0.002	0.001
10^{-2}	2.6 C	2.7 A	3.6 C	2.2 A	2.2 A	2.2 A
10^{-3}	2.1 C*	2.6 A	2.3 C	1.1 A	2.6 A	3.5 A
10^{-4}	1.6 C*	1.9 C*	2.1 C	3.4 C*	3.8 A	1.5 A
10^{-5}	1.6 C*	2.1 C*	4.0 C*	4.5 C*	1.3 C*	2.1 A

Table 7.5.4. Evaluations of f and of J

31 3	39 5	40 5	42 5	39 6	38 6
29 4	50 7	42 6	76 9	248 33	303 42
44 2	42 5	65 7	175 20	73 9	158 20
44 2	97 4	67 6	73 7	235 30	306 41

7.6. The differential equation

$$\begin{cases} y_1' = -(1-y_2)y_1 + 0.99\,y_2 \\ y_2' = 10^3 * ((1-y_2)y_1 - y_2) \\ y_1(0) = 1, \quad y_2(0) = 0 \end{cases}$$

parameters used: available = true, stiff = false.
At x = 50 the results are

Table 7.6.1. Correct digits of y_1 and y_2 (left) and method used (right)

eps \ hmin	0.01	0.05	0.002	0.001	0.0005
1	2.6 C*	2.9 C*	2.6 C	2.5 C	2.5 C
10^{-1}	2.7 C*	2.7 C*	2.5 C*	2.6 C*	2.6 C*
10^{-2}	2.7 C*	2.7 C*	3.0 C*	2.6 C*	2.6 C*
10^{-3}	*)	3.0 C*	3.0 C*	3.0 C*	3.0 C*
10^{-4}	*)	*)	3.7 C*	3.7 C*	3.7 C*

*): error message d[0,1] = 1

Table 7.6.2. Evaluations of f (left) and of J (right)

32 4	55 9	34 5	38 5	35 5
30 3	30 3	34 3	35 4	44 4
33 3	33 3	36 4	41 4	49 5
7 1	39 3	45 4	51 5	60 6
7 1	7 1	57 6	65 7	76 5

7.7. The differential equation

$$\begin{cases} y' = -200(y-10+(10+x)e^{-x}) + (9+x)e^{-x} \\ y(0) = 10 \end{cases}$$

solution:

$$y(x) = 10 - (10+x)e^{-x} + 10e^{-200x}$$

parameters used: available = true, stiff = false.

At x = 0.4 the results are

Table 7.7.1. Correct digits. Method used: C*

eps \ hmin	0.2	0.1	0.05	0.02	0.01	0.005
10^{-1}	3.0	3.3	3.6	4.0	2.5	2.5
10^{-2}	3.0	3.3	3.6	4.0	3.2	3.0
10^{-3}	3.0	3.3	3.6	3.3	4.5	4.3
10^{-4}	3.0	3.3	3.6	4.9	6.1	2.6
10^{-5}	3.0	3.3	3.6	4.0	2.4	8.3

Table 7.7.2. Evaluations of f (left) and evaluations of J (right)

	8 1	11 1	15 1	21 2	25 2	27 2
	8 1	12 1	16 1	22 2	30 2	37 3
	8 1	12 1	20 1	30 2	38 3	58 5
	8 1	12 1	20 1	35 2	40 3	65 7
	8 1	12 1	20 1	72 13	88 9	65 3

At x = 10 the results are

Table 7.7.3. Correct digits

	0.2	0.1	0.05	0.02	0.01	0.005
10^{-1}	5.4	6.9	6.6	6.9	4.5	6.0
10^{-2}	4.6	4.2	4.6	4.9	7.1	5.5
10^{-3}	5.5	6.0	5.2	4.3	5.1	4.4
10^{-4}	5.2	7.0	5.6	7.0	6.1	6.3
10^{-5}	7.3	5.9	6.5	7.0	7.1	7.2

Table 7.7.4. Evaluations of f and of J

33 6	31 6	35 6	51 8	49 4	51 7
47 5	53 7	55 7	61 7	71 8	82 10
74 7	82 8	81 8	93 10	104 10	130 14
88 4	141 8	198 33	110 9	110 10	153 15
100 4	176 6	293 32	499 105	234 21	163 11

7.8. The differential equation

$$\begin{cases} y_1' = 0.1\, y_1 - 49.9\, y_2 \\ y_2' = -50\, y_2 \\ y_3' = 70\, y_2 - 120\, y_3 \end{cases}$$

$$y_1(0) = 2,\ y_2(0) = 1,\ y_3(0) = 2.$$

Parameters used: available = true, stiff = false.
At x = 0.4 the results are

Table 7.8.1. Correct digits of y_1 (left) and method used (right)

eps \ hmin	0.1	0.05	0.02	0.01	0.005
10^{-1}	3.0 C*	3.8 C*	1.6 C*	2.8 C*	2.1 C
10^{-2}	3.0 C*	3.8 C*	4.0 C*	2.4 C*	2.0 C*
10^{-3}	3.0 C*	3.8 C*	4.0 C*	4.1 C*	4.6 C*
10^{-4}	3.0 C*	3.8 C*	4.4 C*	3.4 C*	5.3 C*

Table 7.8.2. Absolute error of y_2 (y_3 gives similar results)

.77e-3	.44e-4	.11e-4	.14e-2	.79e-2
.77e-3	.44e-4	.60e-5	.14e-4	.17e-2
.77e-3	.44e-4	.22e-4	.62e-4	.24e-4
.77e-3	.44e-4	.95e-6	.11e-5	.50e-6

Table 7.8.3. Evaluations of f (left) and of J (right)

12 1	17 1	27 2	32 2	39 6
12 1	19 1	32 2	47 4	55 6
12 1	20 1	38 2	54 2	70 5
12 1	20 1	41 1	66 3	90 5

At x = 10 the results are

Table 7.8.4. Correct digits of y_1 (left) and method used (right)

eps \ hmin	0.1	0.05	0.02	0.01	0.005
10^{-1}	.9 C*	1.1 C*	.9 C*	.9 C*	1.2 C
10^{-2}	1.4 C*	2.0 C*	2.0 C*	2.3 C*	1.8 C*
10^{-3}	2.8 C*	2.5 C*	2.6 C*	2.6 C*	3.1 C*
10^{-4}	2.2 C*	3.3 C*	3.4 C*	3.6 C*	3.4 C*

Table 7.8.5. Evaluations of f and of J

27 4	28 4	36 4	47 5	93 15
34 5	34 4	45 5	60 7	81 10
39 5	50 6	61 6	75 6	109 10
59 5	59 7	74 6	103 7	142 11

Appendix. Computation of the constants

In the program the following constants are used:

a_{kl}	$l=0(1)k$
$(C_k\ k!)^{-1}$	
$(C_{k+1}\ a_{kk}\ k!)^{-1}$	
$(C_{k+2}\ a_{kk}\ k!)^{-1}$	

$k = 1,2,\ldots,6$ (or 7 in the case of A.M. methods).

The error constants C_k are well-known from classical multistep theory, see Henrici [1962] p.195 and p.208.

The constants a_{kj} will be calculated as an illustration of the theory discussed in section 2.

$D^j\vec{H}(xh)[m]$

j \ m	0	1	2	3	4
-1	h	$1/2\ h^2$	$1/3\ h^3$	$1/4\ h^4$	$1/5\ h^5$
0	1	h	h^2	h^3	h^4
1	0	1	$2h$	$3h^2$	$4h^2$
2	0	0	2	$6h$	$12h^2$
3	0	0	0	6	$24h$
4	0	0	0	0	24

$\vec{A}_n[m]$

m \ n	0	1	2	3	4
0	1	0	0	0	0
1	0	1	h	$2h^2$	$6h^3$
2	0	0	1	$3h^2$	$11h^2$
3	0	0	0	1	6h
4	0	0	0	0	1

$D^j\vec{H}(xh)\vec{A}_n$

j \ n	0	1	2	3	4
-1	h	$1/2\ h^2$	$5/6\ h^3$	$9/4\ h^4$	$251/30\ h^5$
0	1	h	$2h^2$	$6h^3$	$24h^4$
1	0	1	3h	$11h^2$	$50h^3$
2	0	0	2	12h	$70h^2$
3	0	0	0	6	60h
4	0	0	0	0	24

$\dfrac{h^{j-1}D^jH(xh)A_n}{DH(xh)A_n}$

j \ n	0	1	2	3	4
-1					
0		1	2/3	6/11	24/50
1		1	1	1	1
2		0	2/3	12/11	70/50
3		0	0	6/11	60/50
4		0	0	0	24/50

$\dfrac{h^{l-1}D^lH(xk)A_k}{l!DH(xh)A_k} =$

$= a_{kl}$ (C.H.)

l \ k	1	2	3	4
0	1	2/3	6/11	24/50
1	1	1	1	1
2	0	1/3	6/11	35/50
3	0	0	1/11	10/50
4	0	0	0	1/50

$$\frac{h^j D^j H(xh)A_k}{H(xh)A_k}$$

j \ n	0	1	2	3	4
-1	1	1/2	5/12	9/24	251/720
0	1	1	1	1	1
1	0	1	3/2	11/6	50/24
2	0	0	1	12/6	70/24
3	0	0	0	1	60/24
4	0	0	0	0	1

$$\frac{h^{l-1} D^{l-1} H(xh)A_{k-1}}{l!H(xh)A_{k-1}} = a_{kl} \quad \text{(A.M.)}$$

l \ k	1	2	3	4	5
0	1	1/2	5/12	3/8	251/720
1	1	1	1	1	1
2	0	1/2	3/4	11/12	25/24
3	0	0	1/6	1/3	35/72
4	0	0	0	1/24	5/48
5	0	0	0	0	1/120

References

Curtiss, C.F. and Hirschfelder, J.O.
Integration of stiff equations.
Proc. Nat. Acad. Sci. U.S. 38 (1952) 235.

Dahlquist, G.
Convergence and stability in numerical integration of ordinary differential equations.
Math. Scand. 4 (1956) 33.

Dahlquist, G.
A special stability problem for linear multistep methods.
BIT 3 (1963) 27.

Dekker, T.J.
ALGOL 60 procedures in numerical algebra. Part 1.
Mathematical Centre Tracts 22 (1968).

Gear, C.W.
The numerical integration of ordinary differential equations.
Math. Comp. 21 (1967) 146.

Gear, C.W.
The automatic integration of stiff ordinary differential equations.
Proc. IFIP Congr. 1968, p.187.

Gear, C.W.
The automatic integration of ordinary differential equations.
C. ACM 14 (1971) 176.

Gear, C.W.
Algorithm 407. DIFSUB for the solution of ordinary differential equations.
C. ACM 14 (1971) 185.

Hemker, P.W.
Linear multistep methods with variable steplength.
NR 15, Mathematisch Centrum, Amsterdam (1971).

Henrici, P.

Discrete variable methods in ordinary differential equations. John Wiley (1962).

Nordsieck, A.

On numerical integration of ordinary differential equations. Math. Comp. 16 (1962) 22.