RA

STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

REKENAFDELING

Report R 743

AUTOMATIC INTEGRATION OF ORDINARY DIFFERENTIAL EQUATIONS

by

J.A. Zonneveld

May 1963

AUTOMATIC INTEGRATION OF ORDINARY DIFFERENTIAL EQUATIONS.

Introduction.

As the publication of the complete work done by the author on the above subject took more time than was expected , we thought it useful, in order to avoid unnecessary delay in the publication of some essential results, to publish in this preliminary report the formulas and some of the ALGOL60 programs that have been used for a year by the Computation Department of the Mathematical Centre and by others, among them Netherlands Railways who used a FORTRAN-transcription on the Paris IBM 7090. These programs incorporate some novel features. In order to achieve the prescribed accuracy of integration, even if derivatives tend to become large, the program at each step of integration determines the step-length and , moreover, chooses the variable to be used as independent variable.

Runge-Kutta technique.

For automatic integration it is convenient to use Runge-Kutta formulas, as a change of integration step does not interrupt the computation. The difficulty , however, is to choose the step-length, i.e. to find a criterion to test whether a step-length used was permissible and possibly to extrapolate a new step-length. To illustrate how this can be done, we must describe in some detail the basis of the Runge-Kutta technique. For this description we examine the case of a single first-order equation $y' = f(x,y)$. Let be given an initial condition $(x0,y0)$ and let us wish to find $y(x0+h)$. The RK-technique is now the following. Choose some points (x,y) , among them $(x0,y0)$ and compute the derivatives. Form a linear combination of the derivatives, which multiplied by h, gives the increment dy0 in such a way

that, if one expands y0 + dy0 and y(x0 + h) in Taylor series with respect to h, these two series agree in as many terms as possible. The coordinates of the points (x,y) and the weights in the linear combination are the unknowns, to be determined by this requirement.
For our purpose of automatic integration we extend this by forming another linear combination of derivatives which, multiplied by h and expanded with respect to h, yields a series, the first term of which is identical with the last term that was equal in the two series of y0 + dy0 and y(x0 + h). In this way we have separately the "last term taken into account" . The cost to find this last term is, except in one case, the computation of one extra derivative, which compares very favourably with the usual technique of integrating twice with step h and comparing the result with that obtained by integrating once with step 2.h .
Next we list the formulas for sets of equations of the first-order and second-order, with and without first derivative.

<u>Formulas.</u>

Two remarks: 1. by order is meant the exponent of h in the last term,

2. thidy0 is the term of h↑i in dy0.

yi'= fi(x,y1,...,yn); i = 1(1)n; yi(x0) = yi0;

<u>order 2</u>

ki0 = h.fi(x0,y10,...,yn0);

ki1 = h.fi(x0+h,y10+k10,...,yn0+kn0);

dyi0 = (ki0 + ki1)/2;

th2dyi0 = (-ki0 + ki1)/2

<u>order 3</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1  = h.fi(x0+h/3,y10+k10/3,...,yn0+kn0/3);

ki2 = h.fi(x0+h.2/3,y10+k11.2/3,...,yn0+kn1.2/3);

ki3 = h.fi(x0+h,y10+k11,...,yn0+kn1);

dyi0 = (ki0 + ki2.3)/4;

th3dyi0 = (ki0.2 - ki1.3 + ki3)/2
```

<u>order 4</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1  = h.fi(x0+h/2,y10+k10/2,...,yn0+kn0/2);

ki2 = h.fi(x0+h/2,y10+k11/2,...,yn0+kn1/2);

ki3 = h.fi(x0+h,y10+k12,...,yn0+kn2);

ki4 = h.fi(x0+h.3/4,y10+(k10.5+k11.7+k12.13-k13)/32,...,

                              yn0+(kn0.5+kn1.7+kn2.13-kn3)/32);

dyi0 = (ki0 + ki1.2 + ki2.2 + ki3)/6;

th4dyi0 = (-ki0.2 + ki1.6 + ki2.6 + ki3.6 - ki4.16)/3
```

<u>order 5</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1  = h.fi(x0+h.2/9,y10+k10.2/9,...,yn0+kn0.2/9);

ki2 = h.fi(x0+h/3,y10+(k10+k11.3)/12,...,yn0+(kn0+kn1.3)/12);

ki3 = h.fi(x0+h/2,y10+(k10+k12.3)/8,...,yn0+(kn0+kn2.3)/8);

ki4 = h.fi(x0+h.4/5,y10+(k10.53-k11.135+k12.126+k13.56)/125,...,

                         yn0+(kn0.53-kn1.135+kn2.126+kn3.56)/125);

ki5 = h.fi(x0+h,y10+(-k10.63+k11.189-k12.36-k13.112+k14.50)/28,...,

                     yn0+(-kn0.63+kn1.189-kn2.36-kn3.112+kn4.50)/28);

ki6 = h.fi(x0+h,y10+(k10.133-k11.378+k12.276+k13.112+k14.25)/168,...,

                     yn0+(kn0.133-kn1.378+kn2.276+kn3.112+kn4.25)/168);

dyi0 = (ki0.35 + ki2.162 + ki4.125 + ki5.14)/336;

th5dyi0 = (ki0.21 - ki2.162 + ki3.224 - ki4.125 + ki6.42)/14
```

```
yi"= fi(x,y1,...yn); i = 1(1)n; yi(x0) = yi0; yi'(x0) = yi0';
```

<u>order 2</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1  = h.fi(x0+h,y10+y10'.h,...,yn0+yn0'.h);

dyi0 = (yi0' + ki0/2).h;

dyi0'= (ki0 + ki1)/2;

th2dyi0 = ki0/2.h;

th2dyi0'= (-ki0 + ki1)/2
```

<u>order 3</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1 = h.fi(x0+h.2/3,y10+(y10'.6+k10.2)/9.h,...,yn0+(yn0'.6+kn0.2)/9.h);

ki2 = h.fi(x0+h,y10+(y10'+k10/2).h,...,yn0+(yn0'+kn0/2).h);

dyi0 = (yi0' + (ki0 + ki1)/4).h;

dyi0'= (ki0 + ki1.3)/4;

th3dyi0 = (-ki0 + ki1)/4.h;

th3dyi0'= (ki0 - ki1.3 + ki2.2)/2
```

<u>order 4</u>

```
ki0 = h.fi(x0,y10,...,yn0);

ki1 = h.fi(x0+h/2,y10+(y10'.4+k10)/8.h,...,yn0+(yn0'.4+kn0)/8.h);

ki2 = h.fi(x0+h,y10+(y10'.2+k11)/2.h,...,yn0+(yn0'.2+kn1)/2.h);

ki3 = h.fi(x0+h.3/4,y10+(y10'.48+k10.7+k11.11)/64.h,...,

                                yn0+(yn0'.48+kn0.7+kn1.11)/64.h);

dyi0 = (yi0' + (ki0 + ki1.2)/6).h;

dyi0'= (ki0 + ki1.4 + ki2)/6;

th4dyi0 = (ki0.2 - ki1.6 + ki3.4)/9.h;

th4dyi0'= (-ki0.2 + ki1.12 + ki2.6 - ki3.16)/3
```

order 5

```
p = sqrt(5);

ki0 = h.fi(x0,y10,...,yn0);

ki1 = h.fi(x0+(5-p)/10.h,y10+(y10'.(10-p.2)+k10.(3-p))/20.h,...,

                                   yn0+(yn0'.(10-p.2)+kn0.(3-p))/20.h);

ki2 = h.fi(x0+(5+p)/10.h,y10+(y10'.(10+p.2)+k11.(3+p))/20.h,...,

                                   yn0+(yn0'.(10+p.2)+kn1.(3+p))/20.h);

ki3 = h.fi(x0+h,y10+(y10'.4+k10.(-1+p)+k12.(3-p))/4.h,...,

                          yn0+(yn0'.4+kn0.(-1+p)+kn2.(3-p))/4.h);

ki4 = h.fi(x0+h/2,y10+(y10'.48+k10.(7+p)+k12.(5-p))/96.h,...,

                            yn0+(yn0'.48+kn0.(7+p)+kn2.(5-p))/96.h);

ki5 = h.fi(x0+h,y10+(y10'.24+k10.(11-p.7)+k12.(-23+p.3)+k14.(6+p).4)/24.h,.

                       ..,yn0+(yn0'.24+kn0.(11-p.7)+kn2.(-23+p.3)+kn4.(6+p).4)/24.h);

dyi0 = (yi0' + (ki0.2 + ki1.(5+p) + ki2.(5-p))/24).h;

dyi0'= (ki0 + ki1.5 + ki2.5 + ki3)/12;

th5dyi0 = (-ki0.2 + ki1.(5+p) + ki2.(5-p) - ki4.8)/4.h;

th5dyi0'= ki0.2 - ki1.10 - ki2.10 + ki4.16 + ki5.2
```

```
yi"= fi(x,y1,...,yn,y1',...,yn'); i = 1(1)n; yi(x0) = yi0; yi'(x0) = yi0';
```

order 2

```
ki0 = h.fi(x0,y10,...,yn0,y10',...,yn0')

ki1 = h.fi(x0+h,y10+y10'.h,...,yn0+yn0'.h,y10'+k10,...,yn0'+kn0);

dyi0 = (yi0' + ki0/2).h;

dyi0'= (ki0 + ki1)/2;

th2dyi0 = ki0/2.h;

th2dyi0'= (-ki0 + ki1)/2
```

order 3

```
ki0 = h.fi(x0,y10,...,yn0,y10',...,yn0');

ki1 = h.fi(x0+h/3,y10+(y10'.6+k10)/18.h,...,yn0+(yn0'.6+kn0)/18.h,
                         y10'+k10/3,...,yn0'+kn0/3);

ki2 = h.fi(x0+h.2/3,y10+(y10'.6+k10.2)/9.h,...,yn0+(yn0'.6+kn0.2)/9.h,
                          y10'+k11.2/3,...,yn0'+kn1.2/3);

ki3 = h.fi(x0+h,y10+(y10'.2+k10)/2.h,...,yn0+(yn0'.2+kn0)/2.h,
                       y10'+k11,...,yn0'+kn1);

dyi0 = (yi0' + (ki0 + ki2)/4).h;

dyi0'= (ki0 + ki2.3)/4;

th3dyi0 = (-ki0 + ki1)/2.h;

th3dyi0'= (ki0.2 - ki1.3 + ki3)/2
```

<u>order 4</u>

```
ki0 = h.fi(x0,y10,...,yn0,y10',...,yn0');

ki1 = h.fi(x0+h/2,y10+(y10'.4+k10)/8.h,...,yn0+(yn0'.4+kn0)/8.h,

                       y10'+k10/2,...,yn0'+kn0/2);

ki2 = h.fi(x0+h/2,y10+(y10'.4+k10)/8.h,...,yn0+(yn0'.4+kn0)/8.h,

                       y10'+k11/2,...,yn0'+kn1/2);

ki3 = h.fi(x0+h,y10+(y10'.2+k12)/2.h,...,yn0+(yn0'.2+kn2)/2.h,

                     y10'+k12,...,yn0'+kn2);

ki4 = h.fi(x0+h.3/4,y10+(y10'.48+k10.7+k11.11)/64.h,...,yn0+(yn0'.48+

                         kn0.7+kn1.11)/64.h,y10'+(k10.5+k11.7+k12.13-k13)/32,..

                         .,yn0'+(kn0.5+kn1.7+kn2.13-kn3)/32);

dyi0 = (yi0' + (ki0 + ki1 + ki2)/6).h;

dyi0'= (ki0 + ki1.2 + ki2.2 + ki3)/6;

th4dyi0 = (ki0.2 - ki1.3 - ki2.3 + ki4.4)/9.h;

th4dyi0'= (-ki0.2 + ki1.6 + ki2.6 + ki3.6 - ki4.16)/3
```

<u>order 5</u>

```
ki0 = h.fi(x0,y10,...,yn0,y10',...,yn0');

ki1 = h.fi(x0+h.2/9,y10+(y10'.18+k10.2)/81.h,...,yn0+(yn0'.18+kn0.2)/81.h,

                y10'+k10.2/9,...,yn0'+kn0.2/9);

ki2 = h.fi(x0+h/3,y10+(y10'.6+k10)/18.h,...,yn0+(yn0'.6+kn0)/18.h,

               y10'+(k10+k11.3)/12,...,yn0'+(kn0+kn1.3)/12);

ki3 = h.fi(x0+h/2,y10+(y10'.8+k10+k12)/16.h,...,yn0+(yn0'.8+kn0+kn2)/16.h,

               y10'+(k10+k12.3)/8,...,yn0'+(kn0+kn2.3)/8);

ki4 = h.fi(x0+h.4/5,y10+(y10'.100+k10.12+k13.28)/125.h,...,

                yn0+(yn0'.100+kn0.12+kn3.28)/125.h,

                y10'+(k10.53-k11.135+k12.126+k13.56)/125,...,

                yn0'+(kn0.53-kn1.135+kn2.126+kn3.56)/125);

ki5 = h.fi(x0+h,y10+(y10'.56+k10.7+k12.36-k14.15)/56.h,...,

              yn0+(yn0'.56+kn0.7+kn2.36-kn4.15)/56.h,

              y10'+(-k10.63+k11.189-k12.36-k13.112+k14.50)/28,...,

              yn0'+(-kn0.63+kn1.189-kn2.36-kn3.112+kn4.50)/28);

ki6 = h.fi(x0+h,y10+(y10'.336+k10.21+k12.92+k14.55)/336.h,...,

              yn0+(yn0'.336+kn0.21+kn2.92+kn4.55)/336.h,

              y10'+(k10.133-k11.378+k12.276+k13.112+k14.25)/168,...,

              yn0'+(kn0.133-kn1.378+kn2.276+kn3.112+kn4.25)/168);

dyi0 = (yi0' + (ki0.35 + ki2.108 + ki4.25)/336).h;

dyi0'= (ki0.35 + ki2.162 + ki4.125 + ki5.14)/336;

th5dyi0 = (-ki0.21 + ki2.108 - ki3.112 + ki4.25)/56.h;

th5dyi0'= (ki0.21 - ki2.162 + ki3.224 - ki4.125 + ki6.42)/14
```

Some remarks on the above formulas.

$yi' = fi(x,y1,\ldots,yn)$.

Except for the fifth-order formula all formulas are, with respect to the integration part, identical with classical formulas. As the existing six-point fifth-order formula, due to Kutta [1] and corrected by Nystrom [2] has negative weights we constructed a new one without this defect. Although we tried, making use of the free parameters we had , to determine them in such a way that nowhere among the coordinates negative numbers occurred, we did not succeed and had to content ourselves with this solution of the 32 non-linear equations in 35 unknowns.

$yi'' = fi(x,y1,\ldots,yn)$.

The formulas for this case are, again with regard to the integration part, identical with formulas given by Nystrom , except for the fifth-order formula where we constructed a four-point integration formula without negative weights or negative coordinates ; only in this formula we needed two extra points for the computation of the last terms.

$yi'' = fi(x,y1,\ldots,yn,y1',\ldots,yn')$.

The formulas for this case are based on the formulas for first-order equations.

It may be remarked that in several formulas it is not necessary , for computation of the last term or terms to compute all points; e.g. to find the last term of the fifth-order formula for first-order equations one needs to compute six points. If the step is rejected one can skip the computation of the seventh point.

ALGOL60 procedures.

Making use of the above formulas a set of ALGOL60 procedures has been written . Two of them are reproduced here . All of them are based on the following. Let for each variable x,y1,...,yn and in the case of second order equations also for y1',...,yn' be given two positive tolerances rex, aex and reyj, aeyj and reyj', aeyj'. If we compute a step h we require that abs(thidyj) $\leq$ abs(h) $\times$ (reyj $\times$ abs(yj) + aeyj) and a similar inequality for the yj' . Here i is the order of the formula used. If one of the inequalities does not hold the step is rejected. If we denote the lefthand side of the inequalities by discr(j) and the righthand side by tol(j), in any case, rejection or not, a new hnew is computed as

hnew = (tol/discr)↑(1/(i-1)) $\times$ h

where tol/discr is the minimum of tol(j)/discr(j) for all j. In the case of rejection hnew is used to try again , otherwise to continue. The formula really used to compute hnew however is not the one given . For i = 5 we approximate the fourth-root by a rational function thereby limiting it: 1/(1 + discr/tol) + .45 . This has the advantages of faster computation and of a safety measure against too frequent rejections . If discr = tol , the ideal situation, we loose only five percent of the step, have however a twenty percent safety margin in the last term.

In some procedures for sets of first-order equations , we change according to the magnitude of the derivatives the role of the independent variable with one of the dependent ones in such a way that all derivatives with respect to the independent variable chosen are in absolute value $\leq$ 1 . Other procedures introduce as their independent variable the arc-length. Next we give one of the procedures, RK21.

```
procedure  RK21(x,xa,b,f,e,d,fi,pos,l,n);  value fi,pos,l,n;  integer l,n;
Boolean fi,pos; real b; array x,xa,e,d; procedure f;
begin integer i,j,j0; Boolean first,fir,rej; real h,cond1,cond2,fhm,fh,tol
   ,max,xj,xjj,s,absy; array xl,discr,y[0:n],k[0:6,0:n],el[1:2];
  procedure  RKstep (x,xa,h,j,n,first,DISCR) ;  value h,j,n;  integer j,n;
  Boolean first,DISCR; real h; array x,xa;
  begin integer i;
    procedure F(t); value t; integer t;
    begin integer i; real p;  f(x,y); p:= h/y[j]; for i:= 0 step 1 until n
      do begin if i≠j then k[t,i]:= y[i] × p end end F;
    if ¬ first  then begin for i:= 0 step 1 until n do x[i]:= xa[i];  F(0)
    end;  for i:= 0 step 1 until n do x[i]:= xa[i] + (if i = j then h else
    k[0,i])/4.5;  F(1); for i:= 0 step 1 until n do x[i]:= xa[i] + (if i=j
    then  4 × h  else  (k[0,i] + 3 × k[1,i]))/12;  F(2);  for i:= 0 step 1
    until n  do  x[i]:= xa[i] + ( if i = j then .5 × h else ( k[0,i] + 3 ×
    k[2,i])/8);  F(3);  for i:= 0 step 1 until n do x[i]:= xa[i] + (if i=j
    then  .8 × h  else ( 53 × k[0,i] - 135 × k[1,i] + 126 × k[2,i] +  56 ×
    k[3,i])/125); F(4); for i:= 0 step 1 until n do x[i]:= xa[i] + (if i=j
    then h else (- 63 × k[0,i] + 189 × k[1,i] - 36 × k[2,i] - 112 × k[3,i]
    + 50 × k[4,i])/28); F(6); if DISCR then
    begin  for i:= 0 step 1 until n do  begin if i ≠ j then x[i]:= xa[i] +
      (  133 × k[0,i] - 378 × k[1,i] + 276 × k[2,i] + 112 × k[3,i] +  25 ×
      k[4,i])/168 end; F(5); for i:= 0 step 1 until n do begin if i≠j then
      discr[i] :=  abs ( 21 × k[0,i] - 162 × k[2,i] + 224 × k[3,i] - 125 ×
      k[4,i] + 42 × k[5,i])/14 end end;
    for  i:= 0 step 1 until n do  begin if i ≠ j then x[i]:= xa[i] + (35 ×
    k[0,i] + 162 × k[2,i] + 125 × k[4,i] + 14 × k[6,i])/336 end end RKste;
    real procedure fzero;
    begin if s=xj then fzero:= cond1 else if s=xjj then fzero:= cond2 else
      begin RKstep(x,xl,s-xl[j],j,n,false,false); fzero:= b end end fzero;
    if fi then begin for i:= 0 step 1 until n do d[i]:= xa[i] ; d[n+1]:=
    d[n+2]:= 0 end ; for i:= 0 step 1 until n do x[i]:= xl[i]:= d[i] ; j:=
    d[n+2]; h:= d[n+1]; fir:= true; y[0]:= 1; goto change;
again: RKstep (x,xl,h,j,n,first,true) ; first:= rej:= false ; fhm:= 0; for
    i:= 0 step 1 until n do begin if i≠j then begin tol:= abs(h) × (e[2×i]
    × abs(x[i]) + e[2×i+1]); rej:= tol < discr[i] ∨ rej ; fh:= discr[i]/tol;
    if fh > fhm then fhm:= fh end end ; fhm:= 1/(1 + fhm) + .45; if rej then
    begin h:= h × fhm; goto again end;
accept: if fir then cond1 := b else begin cond2:= b ; if cond1 × cond2 ≤ 0
    then begin d[n+1]:= h × fhm; goto zero end ; cond1 := cond2 end; for i:=
    0 step 1 until n do d[i]:= xl[i]:= x[i] ; if ¬ fi ∧ fir then h:= d[n+1]
    else d[n+1]:= h:= h × fhm; if fir then fi:= fir:= false;
change : j0:= j ; f(x,y) ; max:= abs(y[j]) ; for i:= 0 step 1 until n do
    begin absy:= abs(y[i]); if absy > max then begin max:= absy ; j:= i end
    end; if j0 ≠ j then begin d[n+2]:= j ; d[n+1]:= h:= h × y[j]/y[j0] end;
    xj:= xl[j]; if fir then begin h:= e[2×j] × abs(xl[j]) + e[2×j+1]; if fi
    then begin if y[l]/y[j] × h < 0 ≡ pos then h:= -h end else if d[n+1] × h
    < 0 then h:= - h end ; for i:= 0 step 1 until n do begin if i ≠ j then
    k[0,i]:= h × y[i]/y[j] end; first:= true; goto again;
zero: e1[1]:= e[2×n+2]; e1[2]:= e[2×n+3]; xjj:= x[j]; xj:= ZERO(s,xj,xjj,
    fzero,e1); RKstep(x,xl,xj-xl[j],j,n,false,false); for i:= 0 step 1 until
    n do d[i]:= x[i] end RK21;
```

To describe the use of RK21 for solving a set of n first-order equations we first explain the formal parameters:

x : an array , the elements of which are x[0] , the independent variable and x[1],...,x[n] , the dependent variables. x is the output array of RK21.

xa : an array; the elements xa[0],...,xa[n] specify the initial condition for the n+1 variables.

b : a real variable or a function designator , that specifies the end of the integration interval ; the integration, started at xa, will proceed until a zero of b; b depends on x[i].

f : the identifier of a non-local procedure with two formal parameters x and y, f(x,y); f must be such, that given x[0],...,x[n] , the elements y[1],...,y[n] assume upon execution of f the values y[i] = dx[i]/dx[0].

e : the array of tolerances . e consists of the elements e[0],..., e[2×n+3]; the elements e[i], e[i+1]; i = 0(2)2×n represent the relative and absolute tolerances for the n + 1 variables. e[2×n+2] and e[2×n+3] are tolerances handed over by RK21 to a non-local procedure ZERO.

d : a non-local array with elements d[0],...,d[n+2]. The elements d[0],...,d[n] assume the value of the variables x[0],...,x[n] ; d[n+1] is the steplength used and entier(d[n+2]+.5) is the index of the variable used by RK21 as independent variable ; d is set after each step that has been accepted.

fi : a Boolean variable ; if fi = <u>true</u> then the integration is started at xa ; if fi = <u>false</u> then the integration is continued with as initial condition the contents of the array d ; in this case xa is ignored.

pos: a Boolean variable specifying the direction in which the integration is to start : if pos ≡ true the integration starts in such a way that the value of x[l] increases ; if pos ≡ false the integration starts in such a way that the value of x[l] decreases.

l : an integer denoting the variable x[l] the value of which increases or decreases according to pos. Both pos and l are used only at each call of RK21 with fi ≡ true to determine the direction in which to start the integration.

n : an integer giving the number of differential equations.

RK21 makes use of a non-local real procedure ZERO(x,a,b,fx,e) the value of which is the value of a zero of the real expression fx , such that $a \leq ZERO \leq b$ provided that for x=a and x=b, fx has different sign. The array e , with elements e[1] and e[2] is the array of tolerances. Using these tolerances ZERO determines the zero of fx in such a way that fx changes sign in the interval

abs(ZERO)-(e[1]×abs(ZERO)+e[2]) $\leq$ abs(x) $\leq$ abs(ZERO)+(e[1]×abs(ZERO)+e[2]).

ZERO, as called by RK21 makes use of e[2×n+2] as its e[1] and of e[2×n+3] as its e[2].

At each step RK21 determines which variable to use for the next step as independent variable.

Thus, starting at xa, RK21 integrates the system of n equations in the direction prescribed by pos and l, up to a zero of b, controlled by the tolerances e . The values of the n + 1 variables at the end of the integration are given by x .

Next we give a program, JAZ023, an example of the use of RK21. We integrate the van der Pol equation $y'' - mu \times (1 - y\uparrow 2) \times y' + y = 0$, starting with $x = 0$, $y = 2$, $y'= 0$, for $mu = 0$ and $mu = 10$. The integration proceeds up to the next zero of y' , continues up to the next zero and so on . Printed are x, y, y' and p, the difference of consecutive values of x. Thus we have the amplitude y and the period $2\times p$ of the relaxation oscillation.

In JAZ023 some standard procedures of the ALGOL60 system for the Electrologica X1 are used:

read	: a function designator that assumes the value of the next number on the input tape.
FIXP(m,n,E)	: a procedure that punches on paper tape the value of E in this way : sign, m decimal digits, decimal point, n decimal digits.
FLOP(m,n,E)	: a procedure that punches the value of E in floating point notation : sign, m decimal digits, $_{10}$, sign, n decimal digits.
RUNOUT	: gives a length of blank tape.
PUNLCR	: a procedure that punches the typewriter symbol NewLineCarriageReturn.
PUTEXT(s)	: a procedure that punches the string s.

```
begin comment JAZ023,R743; integer mu; real p; Boolean fi; array x,xa[0:2]
   ,e[0:7],d[0:4];
  procedure f(x,y); array x,y;
  begin y[1]:= x[2]; y[2]:= mu × (1 - x[1]↑2) × x[2] - x[1] end;
  real procedure ZERO(x,a,b,fx,e); value a,b; real x,a,b,fx; array e;
  begin  real c,fa,fb,fc,m,i,tol,re,ae ; re:= e[1]; ae:= e[2]; x:= a; fa:=
    fx; x:= b; fb:= fx; goto entry;
  goon: if abs(i-b) < tol then i:= b + sign(c-b) × tol; x:= if sign(i-m) =
    sign(b-i) then i else m; a:= b; fa:= fb; b:= x; fb:= fx; if sign(fc) =
    sign(fb) then
  entry : begin c:= a; fc:= fa end; if abs(fb) > abs(fc) then begin a:= b;
    fa:= fb ; b:= c ; fb:= fc ; c:= a ; fc:= fa end ; m:= (b+c)/2 ; i:= if
    fb - fa ≠ 0 then (a × fb - b × fa)/(fb - fa) else m; tol:= abs(b × re)
    + ae; if abs(m-b) > tol then goto goon; ZERO:= x:= b end ZERO;
  procedure  RK21(x,xa,b,f,e,d,fi,pos,l,n); value fi,pos,l,n; integer l,n;
  Boolean fi,pos; real b; array x,xa,e,d; procedure f;
  begin < body RK21 > end RK21;
  RUNOUT ; PUTEXT(⊀JAZ023, R743, van der Pol equation⊁) ; PUNLCR ; PUNLCR;
  e[0]:= 0; e[1] := e[2] := e[3] := e[4] := e[5]:= read; PUTEXT(⊀eps = ⊁);
  FLOP (1,2,e[1]) ; e[6]:= e[7]:= read ; PUTEXT (⊀epsZERO = ⊁) ; FLOP(1,2,
  e[6]); PUNLCR; mu:= read; PUTEXT(⊀mu = ⊁); FIXP(2,0,mu); PUNLCR; PUNLCR;
  PUTEXT(⊀      x[0]             x[1]             x[2]                p⊁); PUNLCR;
  x[0]:= xa[0]:= 0; x[1]:= xa[1]:= 2; x[2]:= xa[2]:= 0; fi:= true; goto A;
B: RK21(x,xa,x[2],f,e,d,fi,true,0,2); if fi then fi:= false;
A: FLOP(8,2,x[0]); FLOP(8,2,x[1]) ; FLOP(8,2,x[2]); if ¬ fi then FLOP(8,2,
  x[0]-p); p:= x[0]; PUNLCR; goto B end
```

Below the output of JAZ023 is printed. As the program goes on indefinitely, we stopped it each time after four lines.

```
JAZ023, R743, van der Pol equation


eps = +.1⏨- 2 epsZERO = +.1⏨- 9

mu = + 0


     x[0]           x[1]           x[2]           p

+           0    +.20000000⏨+ 1 +       0

+.31415930⏨+ 1 -.19999978⏨+ 1 +       0     +.31415930⏨+ 1

+.62831863⏨+ 1 +.19999938⏨+ 1 +       0     +.31415933⏨+ 1

+.94247796⏨+ 1 -.19999896⏨+ 1 +       0     +.31415933⏨+ 1

+.12566373⏨+ 2 +.19999855⏨+ 1 +       0     +.31415933⏨+ 1


JAZ023, R743, van der Pol equation


eps = +.1⏨- 3 epsZERO = +.1⏨- 9

mu = + 0


     x[0]           x[1]           x[2]           p

+           0    +.20000000⏨+ 1 +       0

+.31415927⏨+ 1 -.19999998⏨+ 1 +       0     +.31415927⏨+ 1

+.62831854⏨+ 1 +.19999995⏨+ 1 +       0     +.31415927⏨+ 1

+.94247780⏨+ 1 -.19999992⏨+ 1 +       0     +.31415927⏨+ 1

+.12566371⏨+ 2 +.19999989⏨+ 1 +       0     +.31415927⏨+ 1
```

```
JAZ023, R743, van der Pol equation


eps = +.1₁₀- 2 epsZERO = +.1₁₀- 9

mu = +10

      x[0]             x[1]            x[2]            p

+          0      +.20000000₁₀+ 1 +          0

+.93238577₁₀+ 1  -.20142855₁₀+ 1 +          0     +.93238577₁₀+ 1

+.18863034₁₀+ 2  +.20142858₁₀+ 1 +          0     +.95391759₁₀+ 1

+.28402217₁₀+ 2  -.20142857₁₀+ 1 +          0     +.95391836₁₀+ 1

+.37941396₁₀+ 2  +.20142859₁₀+ 1 +          0     +.95391785₁₀+ 1


JAZ023, R743, van der Pol equation


eps = +.1₁₀- 3 epsZERO = +.1₁₀- 9

mu = +10


      x[0]             x[1]            x[2]            p

+          0      +.20000000₁₀+ 1 +          0

+.93238651₁₀+ 1  -.20142854₁₀+ 1 +          0     +.93238651₁₀+ 1

+.18863050₁₀+ 2  +.20142854₁₀+ 1 +          0     +.95391851₁₀+ 1

+.28402235₁₀+ 2  -.20142854₁₀+ 1 +          0     +.95391847₁₀+ 1

+.37941420₁₀+ 2  +.20142854₁₀+ 1 +          0     +.95391848₁₀+ 1
```

Next we have procedure RK22, suitable for solving equations of the form dx[j]/dx[0] = fj(x[0],...,x[n])/f0(x[0],...,x[n]) ; j = 1(1)n , where none of the fj , j = 0(1)n becomes infinite. The procedure introduces an independent variable s such that dx[j]/ds = fj/sqrt(SUM(k,0,n,fk↑2)); j=0(1)n. s represents the arc-length.

The formal parameters of RK22 are similar to those of RK21 with the following exceptions:

f : the identifier of a procedure with two parameters, f(x,y) . f must be such that given x[0],...,x[n], the elements y[j], j=0(1)n assume upon execution of f the values fj(x[0],...,x[n]) .

d : a non-local array with elements d[0],...,d[n+2] . The elements d[0],...,d[n] assume as in RK21 the value of the variables x[0],...,x[n]; d[n+1] again is the step-length used, this time in the independent variable s; abs(d[n+2]) is equal to s; the array d is, as in RK21 set after each step that has been accepted.

The real procedure SUM used is a standard procedure in the X1 ALGOL60 system. The declaration would be:

```
real procedure SUM(i,a,b,c); value b; integer i,a,b; real c;
begin real s;  s:= 0; for i:= a step 1 until b do s:= s + c; SUM:= s end .
```

As an example of the use of RK22 we give JAZ024.

```
procedure RK22(x,xa,b,f,e,d,fi,pos,l,n) ; value fi,pos,l,n ; integer l,n ;
Boolean fi, pos; real b; array x,xa,e,d; procedure f;
begin integer i ; Boolean fir, rej ; real s,sz,h,cond1,cond2,fh,fhm,tol,v;
  array xl,discr,y[0:n],k[0:6,0:n],el[1:2];
  procedure RKstep(x,xa,h,n,DISCR) ; value n ; integer n ; Boolean DISCR ;
  real h; array x,xa;
  begin integer i;
    procedure F(t); value t; integer t;
    begin integer i; real p ; f(x,y) ; p:= h/sqrt(SUM(i,0,n,y[i]↑2)) ; for
      i:= 0 step 1 until n do k[t,i]:= y[i] × p end F;
    for i:= 0 step 1 until n do x[i]:= xa[i]; F(0); for i:= 0 step 1 until
    n  do  x[i]:= xa[i] + k[0,i]/4.5 ; F(1) ; for i:= 0 step 1 until n  do
    x[i]:= xa[i] + (k[0,i] + 3 × k[1,i])/12 ; F(2); for i:= 0 step 1 until
    n do  x[i]:= xa[i] + (k[0,i] + 3 × k[2,i])/8 ; F(3) ; for i:= 0 step 1
    until n do x[i]:= xa[i] + (53 × k[0,i] - 135 × k[1,i] + 126 × k[2,i] +
    56 × k[3,i])/125 ; F(4) ; for i:= 0 step 1 until n  do  x[i]:= xa[i] +
    ( - 63 × k[0,i] +  189 × k[1,i] -  36 × k[2,i] -  112 × k[3,i] +  50 ×
    k[4,i])/28; F(6); if DISCR then
    begin for i:= 0 step 1 until n do x[i]:= xa[i] + (133 × k[0,i] - 378 ×
      k[1,i] + 276 × k[2,i] + 112 × k[3,i] + 25 × k[4,i])/168 ; F(5) ; for
      i:= 0 step 1 until n do discr[i]:= abs( 21 × k[0,i] - 162 × k[2,i] +
      224 × k[3,i] - 125 × k[4,i] + 42 × k[5,i])/14 end;
    for i:= 0 step 1 until n do x[i]:= xa[i] + (35 × k[0,i] + 162 × k[2,i]
    + 125 × k[4,i] + 14 × k[6,i])/336 end RKstep;
  real procedure fzero;
  begin if  v=sz  then fzero:= cond1 else if  v=s  then fzero:= cond2 else
    begin RKstep(x,xl,v-sz,n,false); fzero:= b end end fzero;
  if fi then  begin  for i:= 0 step 1 until n do d[i]:= xa[i] ; d[n+2]:= 0
  end ; for i:= 0 step 1 until n do x[i]:= xl[i]:= d[i] ; sz:= s:= d[n+2];
  h:= sqrt(SUM(i,0,n,(e[2×i]×abs(d[i]) + e[2×i+1])↑2)) ; fir:= true; if fi
  then  begin f(x,y) ; if y[1] × h < 0 = pos then h:= - h end  else if h ×
  d[n+1] < 0 then h:= - h;
again: RKstep(x,xl,h,n,true); rej:= false; fhm:= 0; for i:= 0 step 1 until
  n do  begin tol:= abs(h) × (e[2×i] × abs(x[i]) + e[2×i+1]) ; rej:= tol <
  discr[i] ∨ rej; fh:= discr[i]/tol ; if fh > fhm then fhm:= fh end; fhm:=
  1/(1 + fhm) + .45; if rej then begin h:= h × fhm; goto again end;
accept: s:= s + h; if fir then cond1:= b else  begin cond2:= b; if cond1 ×
  cond2 ≤ 0 then begin d[n+1]:= h × fhm; goto zero end; cond1:= cond2 end;
  for  i:= 0 step 1 until n do  d[i]:= xl[i]:= x[i] ; d[n+2]:= sz:= s ; if
  ¬ fi ∧ fir then h:= d[n+1] else d[n+1]:= h:= h × fhm ; if fir then fi :=
  fir:= false; goto again;
zero :  e1[1]:= e[2×n+2] ; e1[2]:= e[2×n+3] ; s:= ZERO (v,sz,s,fzero,e1) ;
  RKstep (x,xl,s-sz,n,false) ; for i:= 0 step 1 until  n  do  d[i]:= x[i];
  d[n+2]:= s end RK22;
```

JAZ024 gives a solution of

dx[1]/dx[0] = (mu × (1 - x[0]↑2) × x[1] - x[0])/x[1]

with initial condition x[0] = 2, x[1] = 0 up to a zero of x[1] and so on. In the output the arc-length s is given also. Again we stopped the program after four lines.

```
begin comment JAZ024,R743; integer mu; Boolean fi; array x,xa[0:1],e[0:5],
 d[0:4];
 procedure f(x,y); array x,y;
 begin y[0]:= x[1]; y[1]:= mu × (1 - x[0]↑2) × x[1] - x[0] end f;
 real procedure ZERO(x,a,b,fx,e); value a,b; real x,a,b,fx; array e;
 begin  real c,fa,fb,fc,m,i,tol,re,ae ; re:= e[1]; ae:= e[2]; x:= a; fa:=
   fx; x:= b; fb:= fx; goto entry;
 goon: if abs(i-b) < tol then i:= b + sign(c-b) × tol; x:= if sign(i-m) =
   sign(b-i) then i else m; a:= b; fa:= fb; b:= x; fb:= fx; if sign(fc) =
   sign(fb) then
 entry : begin c:= a; fc:= fa end; if abs(fb) > abs(fc) then begin a:= b;
   fa:= fb ; b:= c ; fb:= fc ; c:= a ; fc:= fa end ; m:= (b+c)/2 ; i:= if
   fb - fa ≠ 0 then (a × fb - b × fa)/(fb - fa) else m; tol:= abs(b × re)
   + ae; if abs(m-b) > tol then goto goon; ZERO:= x:= b end ZERO;
 procedure  RK22(x,xa,b,f,e,d,fi,pos,l,n); value fi,pos,l,n; integer l,n;
 Boolean fi,pos; real b; array x,xa,e,d; procedure f;
 begin < body RK22 > end RK22;
 RUNOUT ; PUTEXT(⪡JAZ024, R743, van der Pol equation⪢ ); PUNLCR ; PUNLCR;
 e[0] := e[1] := e[2] := e[3] := read; PUTEXT(⪡eps = ⪢) ; FLOP(1,2,e[0]);
 e[4] := e[5] := read; PUTEXT(⪡epsZERO = ⪢); FLOP(1,2,e[4]); PUNLCR; mu:=
 read; PUTEXT(⪡mu = ⪢); FIXP(2,0,mu); PUNLCR; PUNLCR;
 PUTEXT(⪡      x[0]             x[1]                s⪢); PUNLCR ; x[0]:= xa[0]:=
 2; x[1]:= xa[1]:= 0; fi:= true; goto A;
B: RK22(x,xa,x[1],f,e,d,fi,false,1,1); if fi then fi:= false;
A: FLOP(8,2,x[0]) ; FLOP(8,2,x[1]) ; FLOP(8,2, if ¬ fi then abs(d[3]) else
 0); PUNLCR; goto B end
```

```
JAZ024, R743, van der Pol equation


eps = +.1₁₀- 2 epsZERO = +.1₁₀- 9

mu = + 0


     x[0]            x[1]             s

+.20000000₁₀+ 1 +          0      +          0

-.20000598₁₀+ 1 -.43423043₁₀-11 +.62831707₁₀+ 1

+.20002225₁₀+ 1 -.23478997₁₀-10 +.12566572₁₀+ 2

-.20004011₁₀+ 1 +.20761082₁₀- 9 +.18850509₁₀+ 2

+.20005809₁₀+ 1 -.21321966₁₀- 9 +.25135007₁₀+ 2


JAZ024, R743, van der Pol equation


eps = +.1₁₀- 3 epsZERO = +.1₁₀- 9

mu = + 0


     x[0]            x[1]             s

+.20000000₁₀+ 1 +          0      +          0

-.20000056₁₀+ 1 +.35511594₁₀-11 +.62831884₁₀+ 1

+.20000138₁₀+ 1 +.41282533₁₀-11 +.12566400₁₀+ 2

-.20000219₁₀+ 1 +.11281642₁₀-10 +.18849636₁₀+ 2

+.20000300₁₀+ 1 +.91220365₁₀-11 +.25132899₁₀+ 2
```

```
JAZ024, R743, van der Pol equation

eps = +.1⏨- 2 epsZERO = +.1⏨- 9
mu = +10

      x[0]              x[1]              s
+.20000000⏨+ 1 +           0       +          0
-.20142887⏨+ 1 -.99252828⏨-11 +.29387303⏨+ 2
+.20143118⏨+ 1 +.10179330⏨-10 +.58788341⏨+ 2
-.20142871⏨+ 1 +.20951352⏨-10 +.88189384⏨+ 2
+.20143068⏨+ 1 -.18346102⏨-10 +.11759042⏨+ 3
```

```
JAZ024, R743, van der Pol equation

eps = +.1⏨- 3 epsZERO = +.1⏨- 9
mu = +10

      x[0]              x[1]              s
+.20000000⏨+ 1 +           0       +          0
-.20142873⏨+ 1 +.13167939⏨-10 +.29387382⏨+ 2
+.20142880⏨+ 1 -.16203483⏨-10 +.58788433⏨+ 2
-.20142870⏨+ 1 -.43686242⏨-10 +.88189484⏨+ 2
+.20142882⏨+ 1 +.19910494⏨- 8 +.11759053⏨+ 3
```

References.

[1] Kutta W., Beitrag zur naeherungsweisen Integration totaler Differentialgleichungen, Zeitschr.f.Math.u.Phys. 46(1901)435-453.

[2] Nystrom E.J., Ueber die numerische Integration von Differentialgleichungen, Act.Soc.Sci.Fennicae Tom L, No.13(1925).

[3] Ceschino F., Evaluation de l'erreur par pas dans les problemes differentiels, Chiffres 5(1962)223-229.
During the write up of this report this paper appeared. In it a fourth-order formula for first-order equations, similar to one of our formulas is given. One of the weights appearing in this formula is, however, negative.