CONDITIONAL DOUBLE PRECISION IN THE NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS

A Thesis

Presented in Partial Fulfillment of the Requirements
for the Degree Master of Science

by

John William Blumberg, B.S.

The Ohio State University
1968

Approved by

Clinton R. Fouch
Adviser
Division of Computer and
Information Science

TABLE OF CONTENTS

ACKNOWLEDGMENT

The programming presented in this paper was made possible by the work and help of W.M. Lay. The guidance necessary in the development of this thesis was provided by Dr. Clinton R. Foulk. The author wishes to acknowledge the contribution made by E.S.B. in the preparation of this paper.

INTRODUCTION

Arnold Nordsieck in Mathematics of Computation[1] presents an algorithm for the numerical integration of differential equations. This algorithm is stable, self-starting, designed to set its own elementary interval size, able to reverse the order of integration, and capable of handling continous or piecewise continous functions. Using this algorithm, a user merely specifies the function to be integrated, the initial values, the length on the x-axis to integrate, and the accuracy required in the solution. The routine takes care of the step-size and automatically builds a 5^{th} degree approximating polynomial.

The routine begins by assuming that the approximating polynomial is one with coefficients of zero. This polynomial is then used and refined by the starting routine until the truncation error is within the specified limits. The routine then begins to integrate the function over the specified x-interval. As the integration proceeds the routine maintains its accuracy by changing the elementary interval size as required by the logic of the routine.

---

[1]Arnold Nordsieck, "On Numerical Integration of Ordinary Differential Equations", Mathematics of Computation, Vol. 16, No. 77, January,1962, pp. 22 - 49.

Nordsieck uses the following quadrature formula to define his method.

$$y(x+h) - y(x) = h/1440(475\, f(x+h) + 1427\, f(x) - 798\, f(x-h) + 482\, f(x-2h) - 173\, f(x-3h) + 27\, f(x-4h))$$

However, he does not use this formula directly in the integration process. He suggests the following:

$$y(x+h) - y(x) = h(f(x) + a(x) + b(x) + c(x) + d(x) + 95/288 (f(x+h) - f^p)) \quad \text{1a.}$$

$$f^p = f(x) + 2a(x) + 3b(x) + 4c(x) + 5d(x) \quad \text{1b.}$$

$$a(x+h) - a(x) = 3b(x) + 6c(x) + 10d(x) + 25/24\, (f(x+h) - f^p) \quad \text{1c.}$$

$$b(x+h) - b(x) = 4c(x) + 10d(x) + 35/72\, (f(x+h) - f^p) \quad \text{1d.}$$

$$c(x+h) - c(x) = 5d(x) + 5/48\, (f(x+h) - f^p) \quad \text{1e.}$$

$$d(x+h) - d(x) = 1/120\, (f(x+h) - f^p). \quad \text{1f.}$$

These equations constitute the set of working equations, and are formulated in this manner for ease of elementary interval size changing.

Nordsieck has defined a stable integration algorithm which will integrate a function within specified limits. To control the truncation error the step-size (h) is set to provide the proper accuracy. After each step of the integration the following two conditions are checked

$$\left| y^{(3)} - y^{(2)} \right|_{max} \leq 1/8 \left| y^{(2)} - y^{(1)} \right|_{max} \quad \text{1g.}$$

$$\left| f(x+h) - f^p \right|_{max} \leq TOL/|h|. \quad \text{1h.}$$

Where $y^{(i)}$ is the i^{th} predicted value of $y(x+h)$, and TOL is the tolerance specified by the user. Failure to pass either of these tests indicates that h is too large and is to be halved.

Test (1g) checks to insure that the iteration error is dominated by the truncation error. Test (1h) checks to insure that the truncation error is within the user specified limits. Both of these tests can be "over-satisfied", that is 2h will also satisfy the two conditions; if so the elementary interval size is to be doubled.

Nordsieck's criteria for controlling the truncation error are complete and machine or arithmetic type independent. There is, of course, one other major error which appears namely round-off error. Nordsieck's algorithm was designed for the ILLIAC, a machine which uses "fixed-point" arithmetic. The version of this algorithm under consideration is for the IBM 7094, a machine which has "floating-point" arithmetic. Nordsieck points out:

> The discussion in the present paper is limited to "fixed-point" arithmetic procedures. the question whether a "floating-point" version of the method could be made safe against loss or illusory gain of significance of the quantities in the course of a long computation, and otherwise trustworthy, is for future investigation.[2]

The source of round-off error in this routine is found in the working equations. In "fixed-point" one solves the problem by using guard digits. In formula (1a) one keeps $\log_2(|h|^{-1})$ extra digits in the calculation of $h\circ(\ )$, and one keeps $\log_2(|h|^{-1})$ extra digits in $y(x+h)$, formula (1a). The use of guard digits is a technique which cannot be done in "floating-point" since

---

[2] Ibid., p. 24.

one is not free to scale numbers at will. The only parallel is double precision; however, double precision arithmetic requires more computer time, and perhaps may not be necessary for every evaluation of the working equations.

J. H. Wilkinson[3] has developed a technique which allows one to estimate the round-off error in a single precision, "floating-point" calculation. It can be shown that (1a) of the working equations is the primary source of round-off error; an analysis of the round-off error in this calculation will give an error estimate which can be tested to provide a conditional double precision routine. One must examine formula (1a) as it is actually implemented in the program.

$$y(x+h) = y(x) + h(f(x) + a + b + c + d) + h\cdot 95/288(f(x+h)-f^{p}) \quad 1a.1$$

The last term of this expression is less than $288/95 \cdot TOL$, this is always the case because of test (1h). This term need not be considered in the round-off error as the other terms will be dominant.

As Wilkinson points out, the "floating-point" accumulation of the bracketed quantity in the second term of the above expression will be; in a double register accumulator:

$$f(x)(1+e_1) + a(1+e_2) + b(1+e_3) + c(1+e_4) + d(1+e_5) .$$

The quantity e_i is the round-off error associated with the representation of a number in a double register accumulator.

---

[3] J. H. Wilkinson, Rounding Errors in Algebraic Processes (Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963), pp. 23-25.

In the IBM 7094 there are 54 bits in the mantissa of a double register result, hence

$$|(1 + e_i)| \leq (1 + 2^{-54}) .$$

This quantity is then reduced to a single register to allow the multiplication by h. The result after the reduction will be

$$(f(x)\cdot(1+e_1) + a\cdot(1+e_2) + b\cdot(1+e_3) + c\cdot(1+e_4) + d\cdot(1+e_5))\cdot(1+e) .$$

Here $|(1+e)| \leq (1+2^{-27})$, since there are only 27 bits in the mantissa of a single register number. The factors associated with each term of this calculation will be $(1+e_i)\cdot(1+e)$ and since 2^{-54} is very small compared to 2^{-27}, each factor will be just $(1+e)$. The quantity can be reduced to

$$DEL \cdot (1+e) .$$

The next step in the evaluation of expression (1a.1) is the calculation of $y(x+h)$:

$$y(x+h) = (h\cdot DEL\cdot(1+e)\cdot(1+e_1) + y(x)\cdot(1+e_2))\cdot(1+e)$$

$$y(x+h) = h\cdot DEL\cdot(1+2e) + y(x)\cdot(1+e) .$$

If conditional double precision is to be used in the evaluation of the working equations, then y must be stored in double precision. The calculation of h·DEL yields a double precision result, even though DEL may be only a single precision number. The final addition in the calculation of $y(x+h)$ will be performed in double precision. Expression (1a.1) will become

$$y(x+h) = h\cdot DEL\cdot(1+e) + y(x).$$

The round-off error in the calculation of $y(x+h)$ will be

$$h\cdot DEL\cdot e .$$

The routine must be able to estimate the total round-off error at the end of each step, if the round-off error in the integration over the entire x-interval is to be controlled. At the end of each calculation of formula (1a) of the working equations, an estimate of the total round-off error can be obtained by

$$DX \cdot DEL \cdot e \ ,$$

where DX is the total x-axis integration interval.

This analysis leads to a test to insure that the truncation error dominates the round-off error.

$$TOL > |DX \cdot DEL \cdot e| \qquad 1i.$$

If test (1i) is not satisfied, the working equations should be calculated in double precision.

When the routine switches precision it may have to recalculate the last step. If step i is performed in single precision and test (1i) requires double precision, then step i must be redone in double precision. If step i is done in double precision and test (1i) requires only single precision, then there is no need to recalculate step i, and the routine proceeds to step i+1. With this conditional use of double precision the user need not worry about whether or not to use double precision; the routine automatically does so, if necessary. This algorithm now has automatic step-size selection and automatic precision selection.

IMPLEMENTATION

ODESA, which stands for Ordinary Differential Equation Solver Automatic, is the name of the FAP subroutine which has been coded from Nordsieck's algorithm. This subroutine has been designed for implementation on the OSU 7094 subroutine library. ODESA is a multiple entry subroutine which will integrate any system of functions of the form:

$$\begin{aligned} dy_1/dx &= f_1(x,y_1,y_2,\ldots,y_n) = f_1(x,y) \\ dy_2/dx &= f_2(x,y_1,y_2,\ldots,y_n) = f_2(x,y) \\ \cdot\cdot &= \cdots \qquad\qquad \cdot\cdot \\ dy_n/dx &= f_n(x,y_1,y_2,\ldots,y_n) = f_n(x,y), \end{aligned}$$

where the set of initial conditions is known

$$y_i(x_0) = y_{i,0} \qquad i = 1,2,3,\ldots,n .$$

To use the subroutine, the user calls tho initialization routine as follows:

GALL ODESST ,DXD,XD,Y,YPR,TOL,N,D,AUX,IFLAG

DXD - is the x-interval length the integration is to proceed, in double precision.

XD - is the value of x_0 in double precision.

Y - is a double precision array of length N, which contains the set of $y_i(x_0)$ values.

YPR - is a double precision array of length N, which will contain the f(x,y) values.

TOL - is a single precision array of length N, which contains the tolerance values supplied by the user. TOL_i is the tolerance for y_i.

N - is the number of equations in the system to be integrated.

D - is a double precision array used for working storage and has length at least 10*N.

AUX - is the name of a subroutine which calculates the YPR values.

IFLAG - a cell which contains the number of elementary steps.

The subroutine AUX must be defined to be compatible with

```
CALL        AUX,X,Y,YPR
```

The tolerance supplied by the user specifies the accuracy in the following way; a $TOL \simeq 10^{-n}$ will supply n significant digits in Y, The number of elementary steps taken will be proportional to $DXD * \log_2(1/TOL)$.

After initialization the routine will integrate the system of functions from x_0 to $x_0 + DXD$ by calling ODESA. Subsequent calls to ODESA will integrate the system from x_i to $x_i + DXD$.

```
CALL        ODESA
```

ODESA has no calling parameters. If the user wishes to revise the x-interval of integration, the subroutine ODESRV is called. A new DXD which is of opposite sign to the last one will reverse the direction of the integration. ODESRV is called as follows:

```
CALL        ODESRV,DXD
```

DXD - is the new x-interval length for integration in double precision.

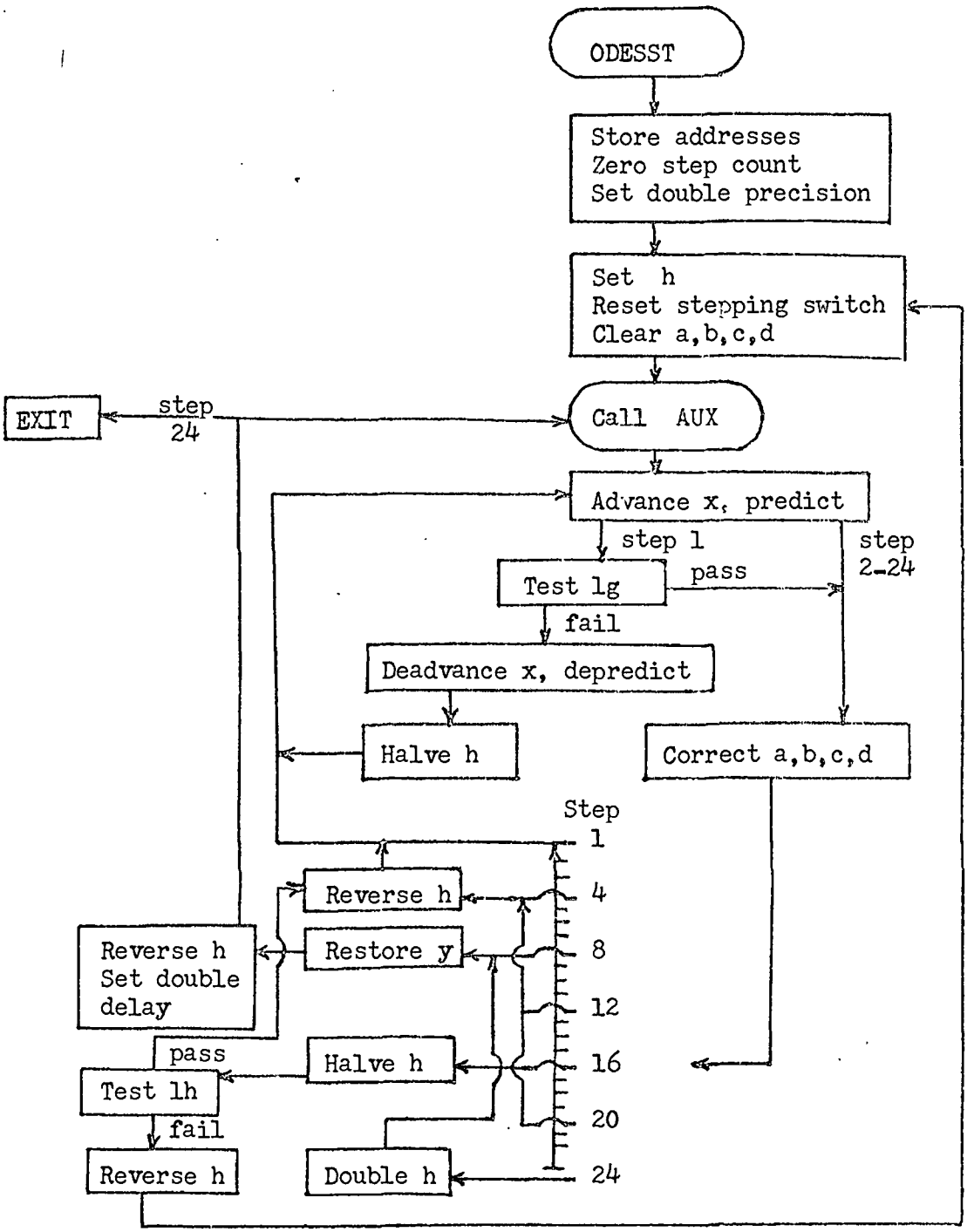
In order to cause the routine to integrate with this new DXD a call to ODESA must follow the call to ODESRV.

There is one error condition, if a call is made to ODESA or ODESRV is made without a call to ODESST being made; the job will be terminated. Any other error conditions must be supplied by the user. ODESA has conditional double precision for the evaluation of the working equations; the user does not concern himself about this and he has no explicit control over it. He has no explicit control over the elementary interval size selected by the logic of ODESA.

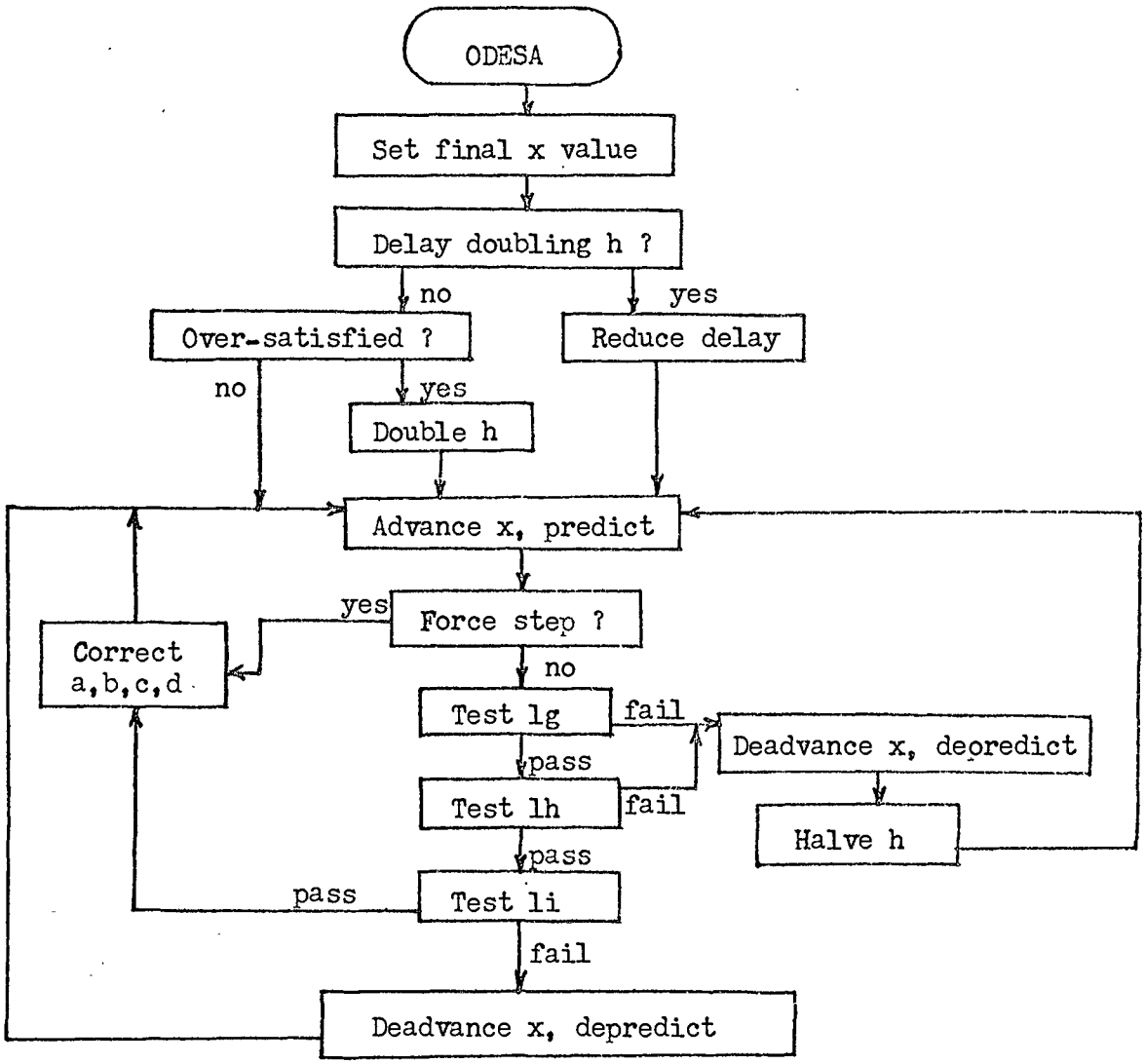
The subroutine ODESST, flowchart page 10, performs the function of suppling the remainder of the routine with the addresses of the calling parameters and performs the setup of the approximating polynomial. ODESST sets the working equation precision to double precision ; this is changed as required by calls to ODESA.

The subroutine ODESA, flowchart page 11, has the function of performing the integration of f(x,y) from x to x + DXD. ODESA takes its first elementary step in double precision and on each succeeding step checks the precision criterion and proceeds with the proper precision. If the user is integrating a system of functions, ODESA can perform the integration on each function in single or double precision dependent only on that function and its tolerance. ODESA, as well as ODESST, uses certain other internal routines to perform the operations of changing h, predicting y(x+h), correcting a,b,c,d and testing checks lg,lh, and li. These routines were written as subroutines to increase

FLOWCHART I


ODESST
Store addresses
Zero step count
Set double precision
Set h
Reset stepping switch
Clear a,b,c,d
EXIT
step 24
Call AUX
Advance x, predict
step 1
step 2-24
Test lg
pass
fail
Deadvance x, depredict
Halve h
Correct a,b,c,d
Step
1
4
8
12
16
20
24
Reverse h
Restore y
Reverse h
Set double
delay
pass
Halve h
Test lh
fail
Reverse h
Double h

FLOWCHART II



generality and make the routines available to all of ODESA without redundant coding.

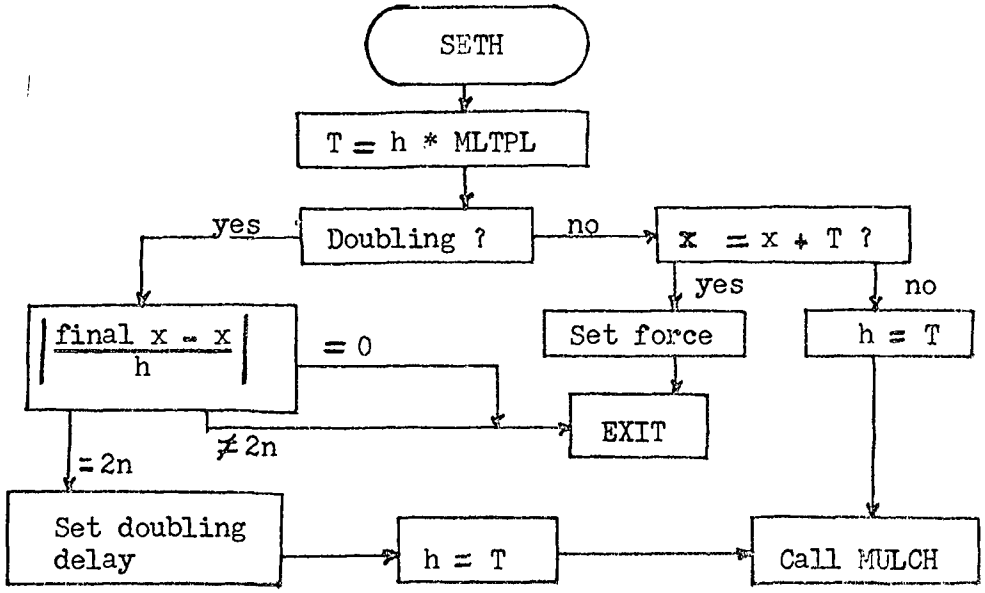
The changing of the step-size is handled by three subroutines SETH, MULCH, and REVRSE. SETH, sst h, has the function of halving and doubling h, if possible. The step-size can be halved if $h/2 + x \neq h + x$ in the high order part; if this is not true, then the old h is used and a step is forced. This is done to keep $|h| > 2^{-27}$, and set an upper bound on the number of steps. The step-size can be doubled if $2h + x \leq$ final x; that is, if there are an even number of steps left to be taken in this call to ODESA. If SETH changes h, then MULCH, multiply change in h, is called. MULCH performs the function of changing the remembered values a, b, c, d for the new step-size. This is done according to TABLE I.

TABLE I

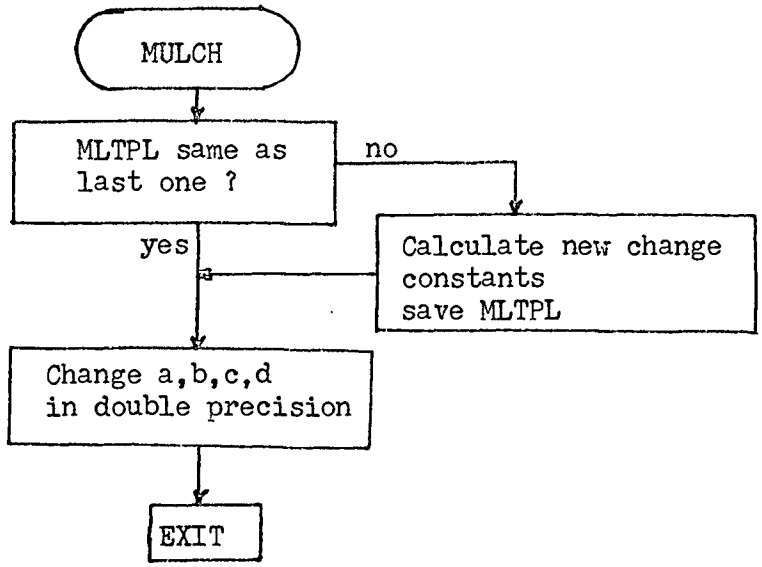
Reverse	Double	Halve	Replaces
-h	2h	h/2	h
y	y	y	y
-a	2a	a/2	a
b	4b	b/4	b
-c	8c	c/8	c
d	16d	d/16	d

These simple step changing rules allow ODESA to handle automatic step-sizing with ease. REVRSE performs the sole function of setting h to -h and calling MULCH to change the remembered quantities. The flowcharts of these three subroutines follow on pages 13 and 14.

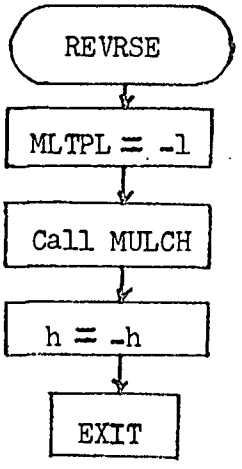
FLOWCHART III



FLOWCHART IV



FLOWCHART V

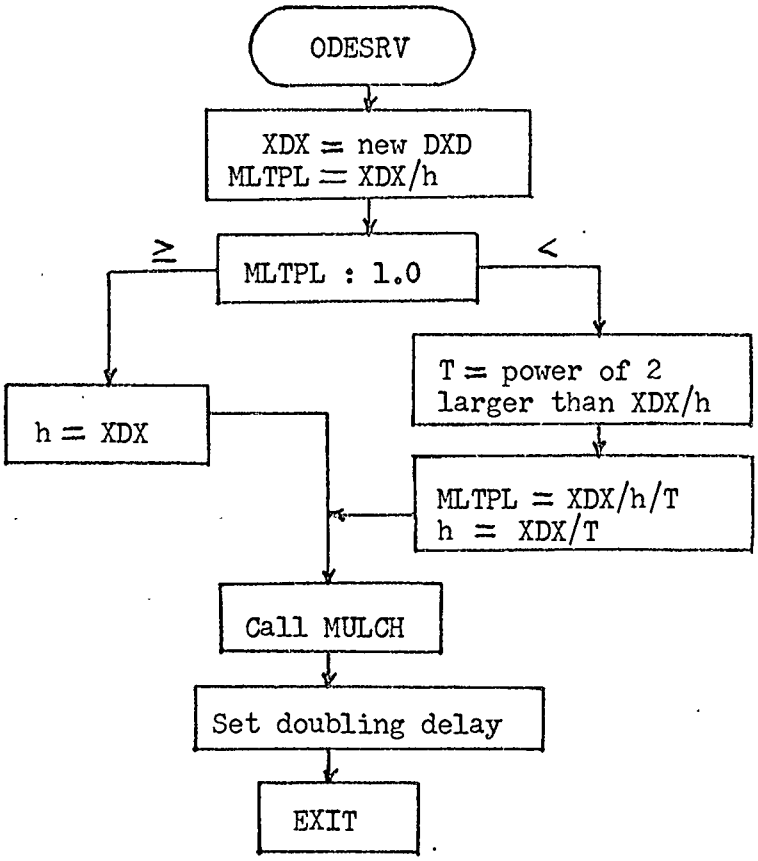


The subroutine MULCH is a routine which changes the remembered quantities a,b,c,d; this changing can be done in the working equation precision as far as round-off error in $y(x+h)$ is concerned. However, the truncation error test (1h) is very sensitive to small errors in a,b,c,d; as they are introduced in the calculation of the quantity f^p. A change in the step-size introduces a large transient in the behavior of test (1h), this transient can cause the doubling of h to be delayed indefinitely, and can cause h to be halved whon halving should not be necessary. It is for these reasons that all the arithmetic in MULCH in done in double precision, rather than conditional double precision.

The subroutine ODESRV, ODESA revising entry, has the function of changing the exit interval, DXD. The exit interval can be changed by any factor, positive or negative; the magnitude of the change factor can be greater than one or less than one.

ODESRV has to change the remembered quantities h,a,b,c,d to be consistent with the new DXD. If a reversing DXD is used, the routine must change the signs of the remembered quantities. If the new DXD is a change in magnitude, the step-size must be adjusted along with a,b,c,d. This changing is done with the aid of MULCH.

FLOWCHART VI



The advancing of x, and the prediction and correction of $y(x+h)$ are performed by the subroutine PREDC. PREDC is also the routine which causes ODESA to exit to the user when $x_i > x_0 + DXD$. This routine calculates the round-off error estimate for use in test (1i), as well as other values for use in tests (1g) and (1h). It is in this routine that formulae (1a) and (1b) of the working equations are calculated:

$$y(x+h) = y(x)+h(f(x) \quad +a+b+c+d+95/288(f(x+h) - f^p))$$

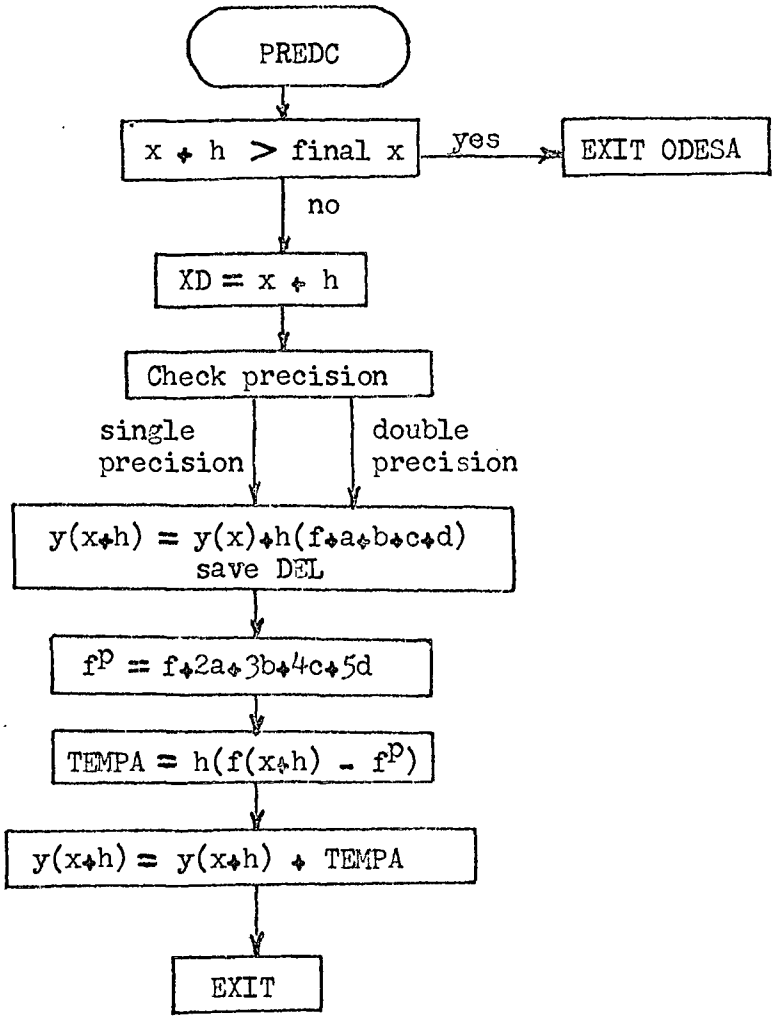
where $f^p = f(x) \quad +2a+3b+4c+5d$.

As was discussed previously, the critical section for round-off error is the calculation of $y(x) + h(f(x)+a+b+c+d)$, so it is in this calculation that the conditional double precision algorithm is applied. The partial result $h\cdot(\ ) = DEL$, is saved for use in the round-off error estimate of test (1i).

In the evaluation of the non-critical sections of this formula single precision is used; however some care is taken to prevent round-off error. For example, in the evaluation of f^p each partial sum result is saved in a double precision temporary to minimize the round-off accumulation. The flowchart of PREDC appears on page 17.

The updating of the remembered values as x_i goes to $x_i + h$ is done by a subroutine called UPDT. This routine evaluates the last four formulae of the working equations to correct a,b,c,d; UPDT also formally advances x from x_i to $x_i + h$.

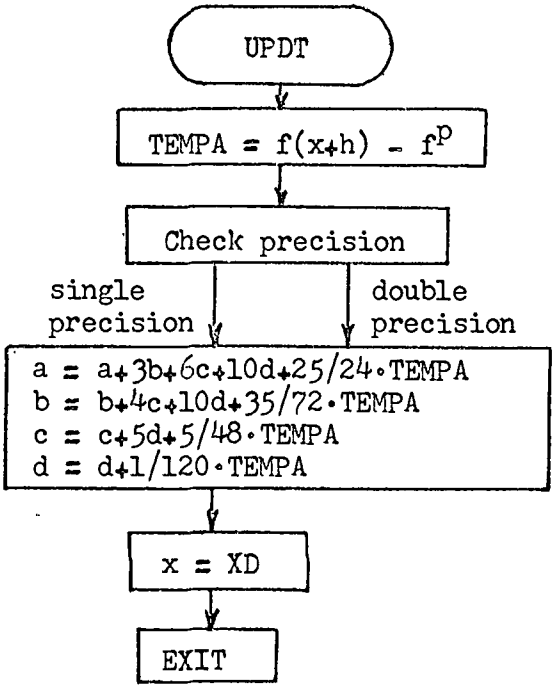
FLOWCHART VII



The two routines PREDC and UPDT form the core of the working equation evaluation section and could be fused into one subroutine, However, since a y(x+h) predicted by PREDC may be rejected by tests (1g) or (1h),and a new h may be generated by SETH; the two routines have been seperated, The remembered quantities a,b,c,d and x are updated if and only if the y(x+h) predicted passes all the criteria of tests (1g),(1h), and (1i),

UPDT has been implemented with the conditional precision algorithm; parallel single and double precision coding is provided for the updating of a,b,c,d. This is necessary since these quantities enter into the critical round-off error section of the evaluation of y(x+h).

FLOWCHART VIII

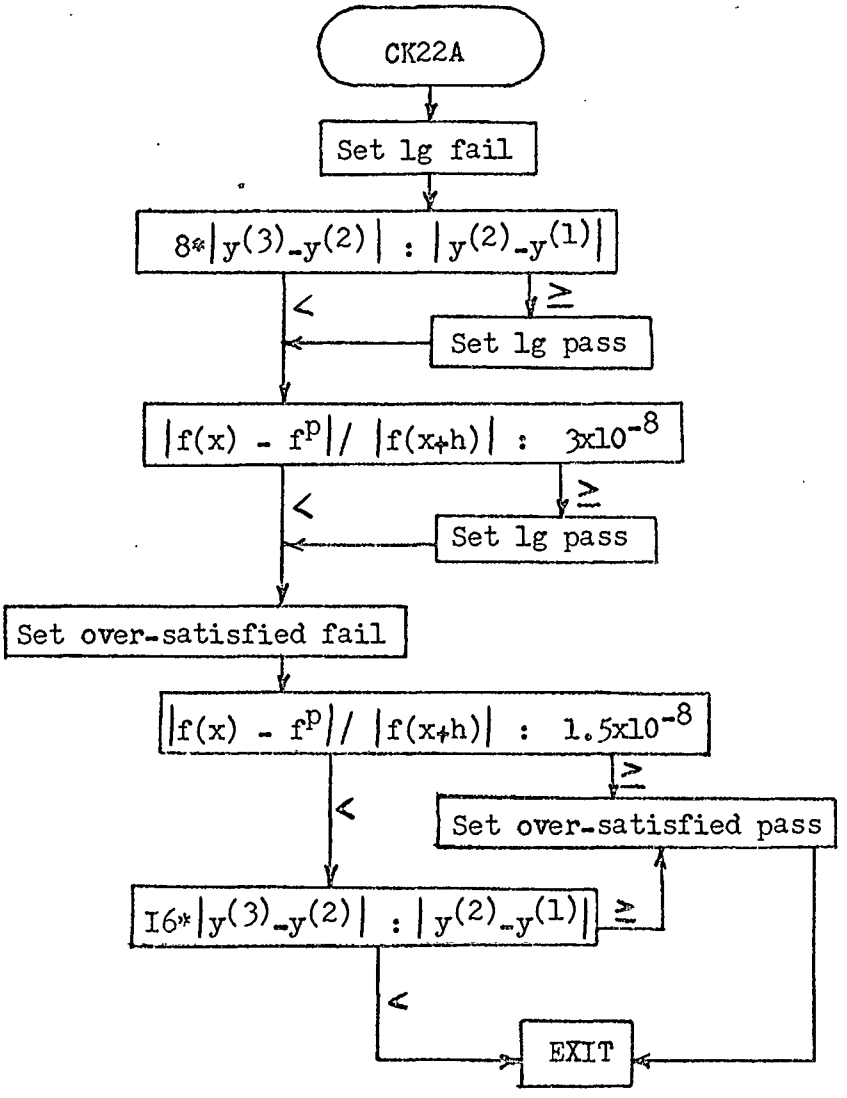


The tests which maintain a check on the truncation error are in subroutines CK22A and CK22B. If either of these tests fail, then the step-size must be halved. If both pass such that 2*h will also pass, then the tests are "over-satisfied" and the step-size should be doubled if possible. The subroutine SPDP sets the precision needed by that step, and sets test (1i) to flag whether or not that step must be redone. A step is to be recalculated if it was done in single precision and double precision was required.

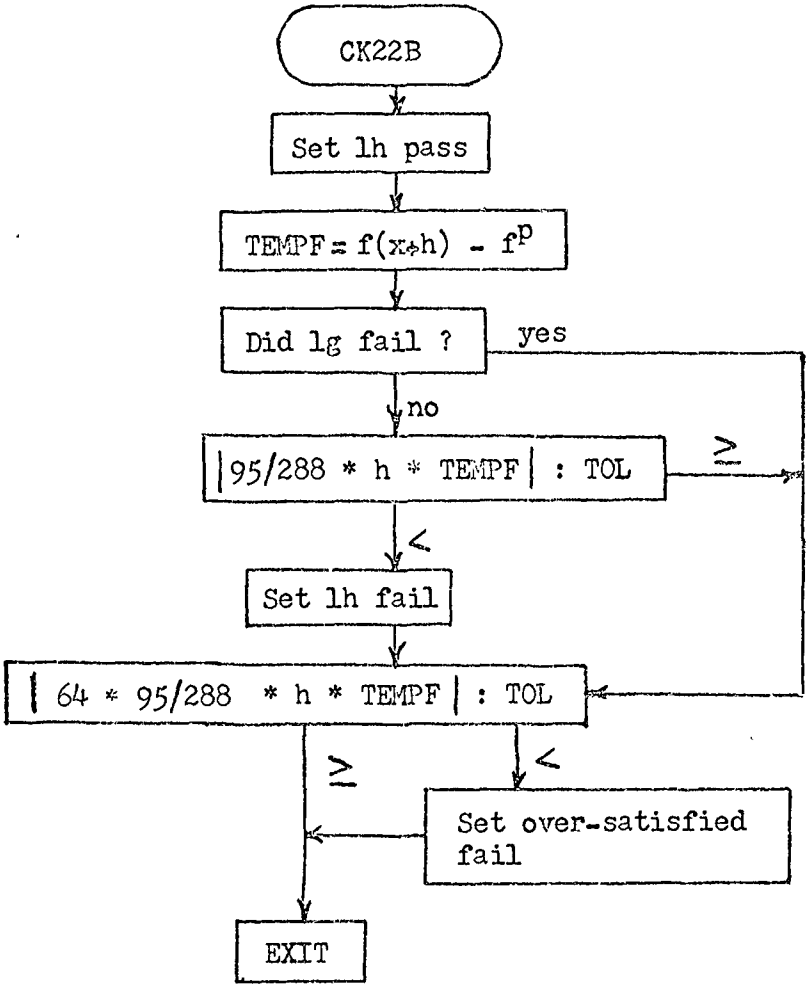
SPDP and the conditional double precision algorithm is designed in such a way that each function in the system to be integrated will be integrated in single or double precision as required. The precision flags are stored in the working storage (D), provided by the user; in the low order part of D(9*N) through D(10*N - 1). A flag of zero indicates single precision, and a non-zero flag indicates double precision. The user must supply the values f(x,y) in double registers. However, if each f(x,y) is independent of the others, the user can write his own conditional double precision routine to evaluate the f(x,y)'s. This can be done since f(x,y) need only be carried in the precision in which the working equations are to be calculated. A user writing the subroutine AUX in SCATRAN or other algebraic language may find this suggestion quite difficult to implement.

The flowcharts of these three subroutines follow.

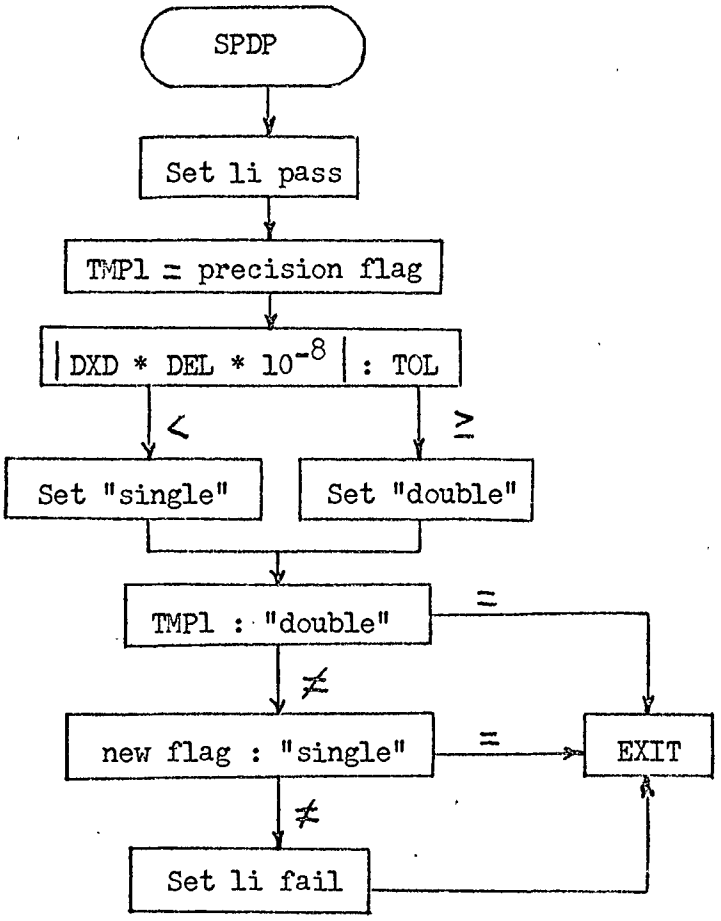
FLOWCHART IX



FLOWCHART X


CK22B
Set lh pass
TEMPF = f(x+h) - f^p
Did lg fail ?
yes
no
|95/288 * h * TEMPF| : TOL
≥
<
Set lh fail
| 64 * 95/288 * h * TEMPF | : TOL
≥
<
Set over-satisfied fail
EXIT

FLOWCHART XI



RESULTS

In order to test the method, functions must be selected which contribute no round-off error in the evaluation of f(x,y), as this analysis assumes no round-off in the subroutine AUX. Such a function is

$$dy/dx = y \quad \text{where} \quad y(0) = 1. \tag{3a.}$$

This differential equation is well behaved and one with a well known solution

$$y(x) = e^{x} \ . \tag{3b.}$$

This equation was integrated from $x = 0$ to $x = 10$; the tolerance was set to 10^{-3}, 10^{-7}, and 10^{-9}.

To demonstrate the performance of ODESA, it was run in three different versions; one that evaluates the working equations in single precision, one that was double precision, and one that uses the conditional double preeision algorithm. In these test runs information was obtained about the solution, Y, the number of elementary steps in single and double precision, and the computer time used. A listing of the conditional precision version of ODESA is included in the appendix.

TABLE II shows the results of a test run made with single precision arithmetic used throughout, even in the subroutine MULCH. As was discussed, the elementary interval size control logic is very sensitive to small errors in the remembered quantities, and these results show what effect transients have on the step-size control.

TABLE II

$dy/dx = y$ $\quad y(0) = 1$ Single Precision

$TOL = 10^{-7}$

X	Y	STEPS	TIME/MINUTES
1.0	$.271828179 \times 10^{1}$	159	.006
4.0	$.545981476 \times 10^{2}$	543	.020
7.0	$.109663307 \times 10^{4}$	927	.034
10.0	$.220264634 \times 10^{5}$	4453	.159

The solution, Y, in each case is within the tolerance, however, the number of elementary steps taken to get these solutions is quite excessive. The mathematical truncation error in this 5^{th} degree method is

$$(863/12) \cdot h^7/7! \cdot y^{(VII)} \quad . \qquad 3c.$$

At $x = 10$ this expression becomes $280 \cdot h^7$, which should be equal to the tolerance. Solving this, the number of steps should be on the order of 200 rather than 4453; of course, one can not expect to predict exactly the number of steps from expression (3c). This demonstrates that the number of elementary steps taken is at least an order of magnitude too large.

TABLE III, page 25, shows the results when this function is integrated with the working equations in single precision, and the subroutine MULCH done in double precision. These results show that the wild behavior of the elementary interval size can be controlled. At $TOL = 10^{-7}$, the number of steps is 480 rather than 4453, this behavior is much closer to the predicted

TABLE III

$dy/dx = y$ $\quad y(0) = 1$ Single Precision

$TOL = 10^{-3}$

X	Y	ERROR	STEPS	TIME/MINUTES
1.0	$.27182885 \times 10^{1}$	$.67 \times 10^{-6}$	28	.001
4.0	$.54598125 \times 10^{2}$	$.25 \times 10^{-6}$	44	.002
7.0	$.10966326 \times 10^{4}$	$.51 \times 10^{-7}$	69	.004
10.0	$.22026455 \times 10^{5}$	$.10 \times 10^{-5}$	116	.006

$TOL = 10^{-7}$

X	Y	ERROR	STEPS	TIME/MINUTES
1.0	$.27182817 \times 10^{1}$	$.11 \times 10^{-7}$	38	.001
4.0	$.54598145 \times 10^{2}$	$.51 \times 10^{-7}$	119	.004
7.0	$.10966330 \times 10^{4}$	$.11 \times 10^{-7}$	244	.009
10.0	$.22026462 \times 10^{5}$	$.31 \times 10^{-7}$	480	.018

$TOL = 10^{-9}$

X	Y	ERROR	STEPS	TIME/MINUTES
1.0	$.2718281795 \times 10^{1}$	$.33 \times 10^{-8}$	67	.002
4.0	$.5459814746 \times 10^{2}$	$.26 \times 10^{-7}$	254	.009
7.0	$.I096633069 \times I0^{4}$	$.89 \times 10^{-8}$	484	.017
10.0	$.2202646326 \times 10^{5}$	$.25 \times 10^{-7}$	733	.026

TABLE IV

$dy/dx = y$ $\quad y(0) = 1$ Double Precision

TOL = 10^{-3}

X	Y	ERROR	STEPS	TIME/MINUTES
I.0	$.27182885 \times 10^{1}$	$.67 \times 10^{-6}$	28	.003
4.0	$.54598127 \times 10^{2}$	$.23 \times 10^{-6}$	44	.004
7.0	$.10966327 \times 10^{4}$	$.42 \times 10^{-6}$	69	.006
10.0	$.22026457 \times 10^{5}$	$.80 \times 10^{-7}$	116	.008

TOL = 10^{-7}

X	Y	ERROR	STEPS	TIME/MINUTES
1.0	$.27182817 \times 10^{1}$	$.11 \times 10^{-7}$	38	.001
4.0	$.54598148 \times 10^{2}$	$.17 \times 10^{-7}$	118	.005
7.0	$.10966331 \times 10^{4}$	$.30 \times 10^{-8}$	243	.009
10.0	$.22026465 \times 10^{5}$	$.70 \times 10^{-8}$	456	.018

TOL = 10^{-9}

X	Y	ERROR	STEPS	TIME/MINUTES
1.0	$.2718281828 \times 10^{1}$	$.10 \times 10^{-10}$	67	.003
4.0	$.5459814999 \times 10^{2}$	$.16 \times 10^{-9}$	247	.009
7.0	$.1096633156 \times 10^{4}$	$.18 \times 10^{-9}$	483	.019
10.0	$.2202646574 \times 10^{5}$	$.52 \times 10^{-9}$	723	.028

behavior, and is acceptable.

At tolerances of 10^{-3} and 10^{-7}, it can be seen that the error in the solutions are within the required tolerance. With a tolerance of 10^{-7} the solutions are satisfactory, but the error is quite near the tolerance. With TOL set to 10^{-9}, the single precision calculation fails to yield acceptable solutions, the round-off error in the working equations is destroying the solutions. This is predictable, since a TOL of 10^{-9} requires more than one register of accuracy.

This function was then integrated in double precision; the results are presented in TABLE 1V, page 26. These results show that in double precision the routine can provide solutions accurate to nine digits, where as single precision can not. The number of elementary steps is compatible with the single precision results. The computer time used to integrate the function in double precision is approximately 50% greater at a tolerance of 10^{-3}.

Since at large tolerances ODESA can give satisfactory solutions with single precision arithmetic and at small tolerances ODESA must use double precision arithmetic, the conditional precision algorithm seems to be in order. TABLE V, page 28, shows the results from the integration of function (3a) in conditional precision. With a tolerance of 10^{-3}, the routine proceeds in single precison over the entire x-interval, after starting in double precision. The solutions at this tolerance

TABLE V

$dy/dx = y$ $\quad y(0) = 1$ Conditional Precision

$TOL = 10^{-3}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
1.0	$.27182885 \times 10^{1}$	$.67 \times 10^{-6}$	4	24	.003
4.0	$.54598125 \times 10^{2}$	$.25 \times 10^{-6}$	20	24	.004
7.0	$.10966326 \times 10^{4}$	$.51 \times 10^{-7}$	45	24	.006
10.0	$.22026455 \times 10^{5}$	$.10 \times 10^{-5}$	92	24	.008

$TOL = 10^{-7}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
1.0	$.27182817 \times 10^{1}$	$.11 \times 10^{-7}$	14	24	.001
4.0	$.54598147 \times 10^{2}$	$.30 \times 10^{-7}$	41	77	.005
7.0	$.10966330 \times 10^{4}$	$.11 \times 10^{-7}$	41	199	.009
10.0	$.22026464 \times 10^{5}$	$.12 \times 10^{-7}$	41	431	.019

$TOL = 10^{-9}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
1.0	$.2718281828 \times 10^{1}$	$.10 \times 10^{-10}$	0	67	.002
4.0	$.5459814999 \times 10^{2}$	$.16 \times 10^{-9}$	0	247	.010
7.0	$.1096633156 \times 10^{4}$	$.18 \times 10^{-9}$	0	483	.019
10.0	$.2202646574 \times 10^{5}$	$.52 \times 10^{-9}$	0	723	.029

are quite acceptable, and the number of steps taken is compatible with the two other versions. At TOL = 10^{-7}, the routine starts in single precision and then switches to double precision, even though the single precision solutions have been shown to be accurate to seven digits. This behavior is attributed to the fact that the round-off error bound developed through the Wilkinson analysis is an upper bound, and in this case the actual round-off error is below this bound. The solutions obtained with TOL = 10^{-9} are obtained by the use of double precision over the entire x-interval. Single precision arithmetic has been shown to fail at TOL = 10^{-9}, and test (1i) in ODESA caused double precision to be used. The solutions obtained are accurate to within the tolerance.

As a check on ODESA's ability to handle a differential equation where the elementary interval size should vary, the following equation was used:

$dy/dx = 1 \qquad x < 4.5 \text{ or } x > 6.5$

$dy/dx = 100 \qquad 4.5 \leq x \leq 6.5 \qquad \text{where } y(0) = 0.$

The solution is:

$y(x) = x \qquad 0 \leq x < 4.5$

$y(x) = 100 \cdot (x-4.5) + 4.5 \qquad 4.5 \leq x \leq 6.5$

$y(x) = (x-6.5) + 204.5 \qquad 6.5 < x \quad .$

The integration is to proceed over the interval x = 0 to x = 25. The tolerance was set to 10^{-3}, 10^{-7}, and 10^{-9}. TABLE VT,

page 31, shows the results of this integration using ODESA in conditional double precision. Of interest in these results will be the behavior of the step-size control as the routine integrates the function. There are two areas of interest; one where the derivative is constant, and one where the derivative is changing rapidly.

At each tolerance the routine begins by taking 4 steps from x = 0 to x = 4, after the 24 initialization steps. This is the expected behavior, since the derivative is constant. To step from 4 to 5 the routine takes 74,122, and 337 steps at tolerances of 10^{-3}, 10^{-7}, and 10^{-9} respectively. This is due to the change in the derivative at x = 4.5; the error at this point is large due to the discontinuity. In the interval x = 5 to x = 6, the routine takes 6,6, and 454 steps at the three tolerances. In the interval x = 6 to x = 7 the derivative again changes, the routine takes 72,97, and 338 steps at the three tolerances. This stepping behavior is almost identical to the behavior at the first discontinuity. The derivative again becomes constant and the step-size begins to increase. From x = 7 to x = 10, ODESA takes 11, 11, and 14 steps. Over the remainder of the interval from x = 10 to x = 25, the routine takes 1 step per unit of integration at the tolerances 10^{-3} and 10^{-7}. At TOL = 10^{-9}, four steps are taken per unit of integration.

The precision used at a tolerance of 10^{-3} is single precision over the entire interval. A tolerance of 10^{-9} requires

TABLE VI

$dy/dx = 1$ $\quad x < 4.5$ or $x > 6.5$ $\quad y(0) = 0$
$dy/dx = 100$ $\quad 4.5 \leq x \leq 6.5$ $\quad$ Conditional Precision

$TOL = 10^{-3}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
4.0	$.40000000 \times 10^{1}$	$.00 \times 10^{0}$	4	24	.003
5.0	$.54501224 \times 10^{2}$	$.12 \times 10^{-4}$	78	24	.006
6.0	$.15450122 \times 10^{3}$	$.12 \times 10^{-5}$	84	24	.007
7.0	$.20500302 \times 10^{3}$	$.30 \times 10^{-5}$	156	24	.010
10.0	$.20800302 \times 10^{3}$	$.30 \times 10^{-5}$	167	24	.011
15.0	$.21300302 \times 10^{3}$	$.30 \times 10^{-5}$	172	24	.013
20.0	$.21800302 \times 10^{3}$	$.30 \times 10^{-5}$	177	24	.014
25.0	$.22300302 \times 10^{3}$	$.30 \times 10^{-5}$	182	24	.015

$TOL = 10^{-7}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
4.0	$.4000000000 \times 10^{1}$	$.00 \times 10^{0}$	4	24	.002
5.0	$.5450000079 \times 10^{2}$	$.79 \times 10^{-8}$	34	116	.007
6.0	$.1545000008 \times 10^{3}$	$.87 \times 10^{-9}$	34	122	.008
7.0	$.2050000016 \times 10^{3}$	$.16 \times 10^{-8}$	124	129	.012
10.0	$.2080000016 \times 10^{3}$	$.16 \times 10^{-8}$	135	129	.013
15.0	$.2130000016 \times 10^{3}$	$.16 \times 10^{-8}$	140	129	.014
20.0	$.2180000016 \times 10^{3}$	$.16 \times 10^{-8}$	145	129	.016
25.0	$.2230000016 \times 10^{3}$	$.16 \times 10^{-8}$	150	129	.017

$TOL = 10^{-9}$

X	Y	ERROR	STEPS SP	STEPS DP	TIME MINUTES
4.0	$.4000000000 \times 10^{1}$	$.00 \times 10^{0}$	0	28	.002
5.0	$.5450000079 \times 10^{2}$	$.79 \times 10^{-8}$	0	365	.016
6.0	$.1545000008 \times 10^{3}$	$.89 \times 10^{-9}$	0	819	.036
7.0	$.2050000016 \times 10^{3}$	$.16 \times 10^{-8}$	0	1157	.051
10.0	$.2080000016 \times 10^{3}$	$.16 \times 10^{-8}$	0	1171	.052
15.0	$.2130000016 \times 10^{3}$	$.16 \times 10^{-8}$	0	1189	.054
20.0	$.2180000016 \times 10^{3}$	$.16 \times 10^{-8}$	0	1209	.056
25.0	$.2230000017 \times 10^{3}$	$.17 \times 10^{-8}$	0	1228	.058

double precision over the entire interval. With the tolerance set to 10^{-7}, the precision switches from single to double when the derivative increases at $x = 4.5$. The precision switches back to single precision, after the derivative decreases at $x = 6.5$.

The total error at tolerances of 10^{-3} and 10^{-7} is within the specified tolerances. At TOL = 10^{-9} the error is not within the tolerance, at $x = 5$. This large amount of error is due to the sharp discontinuity at $x = 4.5$. The solution at $x = 6$ is within the tolerance, since the derivative is constant from 5 to 6. After the second discontinuity at $x = 6.5$, the error returns to a more reasonable level.

CONCLUSION

The conditional precision feature of ODESA gives the user freedom from all worry about how to obtain solutions to differential equations accurate to at least one full register. Conditional precision should be the most efficient method of obtaining solutions accurate to any tolerance, since faster less accurate arithmetic will be used when less accuracy is required, and slower more accurate arithmetic will be used only when very accurate solutions are required or when round-off error could spoil the solutions. However, as the results show the user must pay for the automatic feature. The routine checks its precision criterion at each step, this checking increases the overhead time. ODESA is not, in a limited sense, the most efficient differential equation solving algorithm. The most efficient scheme would be one without automatic starting, automatic interval control, and automatic precision selection. The user would be responsible for the analysis of his problem, and would predetermine the step-size and precision. Through the use of ODESA, a user can make the most efficient use of computer time, in an over-all sense. Nordsieck estimates that the saving in computer time can be by a factor of 10 or greater, if the elementary interval size should vary; since ODESA controls the step-size and evaluates the derivatives only twice per step.

The conditional precision technique presented here repre-

sents a method of controlling the accumulation of round-off error in a "floating-point" numerical method. The application of conditional precision can be seen where greater precision is available. There exist routines which deliver up to 15 registers of precision. These routines are slow, since they are subroutines and not hardware routines like double precision arithmetic. The conditional precision test could be used to determine any precision between single and 15^{th}, and result in large savings in computer time. This use of conditional precision remains for future investigation.

APPENDIX

```
$FAP
         COUNT   600
         ENTRY   ODESST          STARTING ENTRY POINT
         ENTRY   ODESA           REGULAR ENTRY POINT
         ENTRY   ODESRV          ENTRY FOR REVERSING THE STEP
ODESST   STZ     GO             RESET GO SWITCH
         STZ     BSATC          ALWAYS OK.
          STL    STARTD         SET SWITCH FOR INITIALIZATION
         STL     START          SET START SWITCH
         STL     FORCE          SET FORCE SWITCH
         TRA     SAVEMC         GO SAVE MACHINE CONDITIONS
         SPACE   2
GETCAL   DLD*    1,4            GET INTERVAL LENGTH
         DST     XDX            STORE IN XDX
         CAL     2,4            GET CURRENT XD LOCATON
         STA     XD3
         STA     XD4
         STA     XD5
         STA     Y-1
         STA     XG4-2
         STA     STILIN+1
         STA     DBLH+3
         CLA*    6,4            GET NUMBER OF EQUATIONS
         STO     N              SAVE IN N
         ADD     N                        FORM N*2.
         STO     NTN                      **
         CAL     3,4            GET A(Y)
         STA     Y
         ACL     NTN               NEED BES ADDRESS.
         STA     DEPRED+2
         STA     RSTOR+2
         STA     CLEAR-4
         STA     S8+2
         STA     SAVYYP+2
         STA     LAST4
         STA     FIXY+2
         STA     FIXY+4
```

```
CAL     4,4             GET A(YPR)
STA     YPR
ACL     NTN                     NEED BES ADDRESS.
STA     LAS
STA     RSTOR+4
STA     DEPRED+4
STA     SAVYYP
STA     LAST
STA     YPR1
STA     YPR2+2
STA     FIXY
STA     YPR3
STA     D6
CAL     5,4             GET A(TOL)
ACL     N               NEED BES ADDRESS
STA     NEG3
STA     SP11
CAL     7,4             GET A(D)
ACL     NTN                 NEED BES ADDRESS.
STA     CLEAR-3
STA     S8+1
ACL     NTN                 NEED BES ADDRESS.
STA     SNGLE+1
STA     CLEAR+1
STA     LAST-4
ACL     NTN                 NEED BES ADDRESS.
STA     SNGLE+2
STA     CLEAR+2
STA     LAST-3
ACL     NTN                 NEED BES ADDRESS.
STA     SNGLE+3
STA     CLEAR+3
STA     LAST-2
ACL     NTN                 NEED BES ADDRESS.
STA     SNGLE+4
STA     CLEAR+4
STA     LAST-1
ACL     NTN                 NEED BES ADDRESS.
STA     DEPRED+1
STA     RSTOR+1
STA     SAVYYP+3
STA     LAST2
```

```
        ACL     NTN             NEED BES ADDRESS.
        STA     YPR12
        STA     GETA
        STA     NEQ1
        STA     D6+1
        ACL     NTN             NEED BES ADDRESS.
        STA     YPR2
        STA     FIXY+1
        ACL     NTN             NEED BES ADDRESS.
        STA     SAVYYP+1
        STA     RSTOR+3
        STA     DEPRED+3
        ACL     NTN                     GET AD(D9).
        STA     LAST1
        STA     LAS1
        STA     SP1+1
        ACL     =1
        STA     SP6
        STA     DP1
        STA     SP11+3
        STA     SP11+5
        STA     YVA1
        STA     UPD1
        CAL     8,4             GET ADDR OF AUX SUBROUTINE
        STA     AUX+1
        CAL     9,4
        STA     ZEROSP          SETUP AND ZERO.
        STA     PKTSP           STEP COUNTER.
        STA     PKTSP+2
ZEROSP  STZ     **
        ACL     =1
        STA     ZERODP
        STA     PKTDP
        STA     PKTDP+2
ZERODP  STZ     **
        SXA     DP1+2,4
        LXA     NTN,4
 DP1    STL     **,4            START DOUBLE PRECISION.
        TIX     DP1,4,2
        AXT     **,4
```

```
*       STARTING INITIALIZATION PROCEDURE
*
XD3     DLD     XD              GET CURRENT XD
        DST     XTD             STORE IN THE LAST GOOD STEP CELL
        DLD     XDX             GET THE INTERVAL LENGTH
        DST      HD             THIS IS FIRST INCREMENT
XD4     DFAD    XD              ADD IN THE CURRENT X
        DST     FXD             STORE AS ABSOLUTE VALUE
        LXA     NTN,1                   GET NO. EQNS.
        DLD     **,1                    GET INITIAL Y.
        DST     **,1                    SAVE D0.
        TIX     *-2,1,2                 ALL EONS.
        DLD     Z00                     GET ZERO.
 CLEAR  LXA     NIN,1                   GET NO. EONS.
        DST     **,1                    ZERO D1
        DST     **,1                    ZERO D2
        DST     **,1                    ZERO D3
        DST     **,1                    ZERO D4
        TIX     CLEAR+1,1,2             ALL EONS.
        AXT     0,6             SET STEP COUNTER
*
        AXT     6,1             SIX 4 STEP INTER FOR STARTING
*
        STZ     FIRST           SET FIRST INITIAL ENTRY SWITCH
        AXT     4,2                     FOUR STEPS NECESSARY.
        TSX     AUX,4           GET YPR(Y,XD)
*
```

```
*                MAIN PROGRAM SECTION
*
PREDCT  TSX      PREDC,4          ADVANCE,PREDICT,ITERATE
        NZT      FORCE            IS THIS A FORCING STEP
        TRA      CORCT            YES, BLINDLY ORRECT
        NZT      START            IS THIS INITIALIZATION
        TRA      ADM22A           NO,KEEP IN NORMAL ENTRY STREAM
        ZET      FIRST            IS THIS THE FIRST STEP
        TRA      CORCT            NO, GO CORRECT SOLUTION
ADM22A  TSX      CK22A,4          CHECK 22A CRITERIA
        NZT      BSATA            DID THIS STEP PASS
        TRA      DEPRED           NO, GO DEPREDICT AND DEADVANCE
        ZET      START            IS THIS INITIALIZATION
        TRA      CORCT            YES, GO CORRECT
        TSX      CK22B,4          CHECK 22B CRITERIA
        NZT      BSATB
        TRA      DEPRED
        TSX      SPDP,4           TEST FOR PROPER PRECISION,
        NZT      BSATC
        TRA      CORCT
 RSTOR  LXA      NTN,4
        DLD      **,4
        DST      **,4
        DLD      **,4
        DST      **,4
        TIX      RSTOR+1,4,2
        TRA      PREDCT
DEPRED  LXA      NTN,4                  GET NO. EQNS.
        DLD      **,4                   GET OLD Y IN D5.
        DST      **,4                   RESTORE IN Y.
        DLD      **,4                   GET OLD YPR IN D8.
        DST      **,4                   STORE IN YPR.
        TIX      *-4,4,2                GET ALL EQNS.
        DLD      HALF                 GET HALF .5
        DST      MLTPL                 HALVE VALUES.
        TSX      SETH,4             HD (STEP SIZE)
        TRA      PREDCT           GO BACK AND PREDICT
```

```
CORCT   TSX     UPDT,4          CORRECT REMEMBERED VALUES
        STL     FORCE           RESET FORCE SWITCH
        STL     FIRST           SET FIRST ENTRY SWITCH
        NZT     START           IS THIS INITIALIZATION
        TRA     DUBLCK          NO,TRA TO CHECK DOUBLING
        TIX     PREDCT,2,1       PREDICT FOR INITIALIZATION
        AXT     4,2             RESET STEP OUNTER
        TXI     *+1,1,-1        DECREASE INTERVAL COUNTER
        TRA*    STEP,1          TRANSFER TO PROPER STEP
*               STARTING PROCEDURES
*
        TRA     S41220          AFTER  STEP 4
        TRA     S8              -FTER  STEP 8
        TRA     S41220          -FTER  STEP 12
        TRA     S16             -FTER  STEP 16
        TRA     S41220          -FTER  STEP 20
STEP    TRA     S24             -FTER  STEP 24
 S8     LXA     NTN,4                     GET NO. EQNS.
        DLD     **,4                      GET Y IN DO.
        DST     **,4                      RESTORE IN Y.
        TIX     *-2,4,2                   ALL EONS.
REVIN   TSX     AUX,4           GET YPR OF ORIGINAL
        AXT     4,3             SET DOUBLING COUNT
S41220  TSX     REVRSE,4        REVERSE DIRECTIONS
        NZT     STARTD           FINISHED STARTING
        TRA     EXIT             YES,EXIT
        TRA     PREDCT          GO BACK AND PREDICT
 S16    DLD     HALF                     GET HALF .5
        DST     MLTPL                     HALVE VALUES.
        TSX     SETH,4                  VALUES
        TSX     CK22B,4         CHECK THE TILERANCES
        ZET     BSATB           DOES IT PASS
        TRA     S8              YES, GO BACK AND RESET IC
        TSX     REVRSE,4        NO, REVERSE DIRECTIONS
        TRA     CLEAR-1         GO BACK AND START NEW H
 S24    DLD     TWO                      GET TWO 2.0
        DST     MLTPL
        TSX     SETH,4                 SIZE
        STZ     STARTD          INITIALIZATION COMPLETED
        STZ     START
        TRA     S8
```

```
*       NORMAL ENTRY
*
ODESA   NZT     START           HAS PROCEDURE BEEN STARTED
        TRA     ABORT           NO, GO PRONT MESSAGE N KILL
        STL     GO              SET RUN SWITCH
        STZ     START           RESET START SWITCH
SAVEMC  LMTM
        SXA     XR1+1,1
        SXA     XR1+2,2
        SXA     XR1+3,3
        SXA     XR1+4,4
        SXA     XR1+5,5
        SXA     XR1+6,6
        NZT     GO              IS THIS A GO RUN
        TRA     GETCAL          NO, GO GET CALLING SEQUENCE
        LXA     TEMP,3          RESTORE DOUBLING COUNT
XD5     DLD     XD              GET CURRENT XD
        DST     XTD             SET LEFT HAND SIDE
        DFAD    XDX             ADD IN THE INTERVAL LENGTH
        DST     FXD             SAVE LAST INTERVAL AS ABSVAL
DUBLCK  TIX     PREDCT,3,1      CHECK DOUBLE COUNT AND PREDICT
        NZT     BOVSAT          ARE CONDITIONS SOVERSATIFIED
        TRA     PREDCT          NO, GO PREDICT
        DLD     TWO                     YES,DOUBLE STEP
        DST     MLTPL
        TSX     SETH,4          ..
        TRA     PREDCT
```

```
*                ENTRY WHICH REVISES THE EXIT INTERVAL
*
ODESRV  SXA      HERE,4          SAVE MACHINE CONITIONS
        NZT      START            HAS THIS BEEN STARTED
        TRA      ABORT            NO, KILL
        DLD*     1,4              GET NEW XDX
        DST      XDX              STORE
        DFDP     HD               DIIDE BY OLD HD
        DST      MLTPL           SAVE FOR CHANGING VALUES
        SSP                       MAKE SURE ITS POSITIVE
        LRS      0                ALL OF IT
        CAS      =1.              IS XDX NEW GREATER THAN OLD HD
        TRA      GOAHED           YES, MUST DEFINE NEW HD
        TRA      *+1              NO,
        DLD      XDX              GET NEW XDX
        DST      HD               XDX GOOD ENOUGH FOR HD
        TRA      CHANGT           GO CHANGE REMEMBERED VALUES
GOAHED  ANA      =0377000000000  FIND POWER OF 2 LARGER
        ADD      =0001400000000       THAN XDX/HD AND
        STO      TEMP                  SAVE IN TEMP
        CLA      MLTPL            GET XDX/HD
        FDP      TEMP            DIVIDE BY NEXT POWER OF 2
        STO      MLTPL           THIS IS THE CHANGE OF VALUES
        STZ      TEMP+1
        DLD      XDX              GET NEW XDX
        DFDP     TEMP             DIVIDE BY LARGEST POWER OF 2
        DST      HD               THIS IS THE NEW HD
CHANGT  TSX      MULCH,4          GO CHANGE REMEMBERED VALUES
        AXT      4,3                     RESET DOUBLING COUNT.
        SXA      TEMP,3           SAVED IN TEMP
HERE    AXT      **,4
        TRA      2,4              RETURN
*
```

```
*      ROUTINE WHICH DETERMINES THE PRECISION TO PROCEED WITH.
 SPDP  SXA     SP2,4                      SAVE XR4.
       SXA     SP2+1,3                    SAVE XR3.
       STZ     BSATC
       LXA     NTN,4
       LXA     N,3
 SP5   CLA*    SPI1+3
       STO     TMP1
       ZAC
 SP1   LDO     XDX            GET DISTANCE MOVED.
       FMP     D9,4
       XCA
       FMP     =1.E-8
       SSP                    ABS.(DIST*(.)).
 SP11  CAS     TOL,3                      COMPARE WITH TOL.
       TRA     *+4            SET DOUBLE.
       TRA     *+3            SET DOUBLE
       STZ     D9+1,4         FORCE SINGLE PRECISION.
       TRA     *+2
       STL     D9+1,4         FORCE DOUBLE PRECISION
       CLA     TMP1
       TNZ     SP4
 SP6   CLA     D9+1,4
       TZE     SP4
       STL     BSATC
       TRA     SP2
       TXI     *+1,3,-1
 SP4   TIX     SP5,4,2
 SP2   AXT     **,4
       AXT     **,3
       TRA     1,4                        RETURN.
```

```
*                ROUTINE DETERMINES WHETHER A STEP CAN BE MADE
SETH    SXA      REG4,4
        DLD      HD              GET CURRENT INCREMENT
        DFMP     MLTPL           CHANGE BY MULTIPLE
        DST      TEMP            SAVE
        CLA      MLTPL           IS THIS
        CAS      =2.                  A DOUBLING
        TRA      *+2                    PROCEDURE
        TRA      DBLH            YES, GO HCECK DOUBLING
        DLD      TEMP            NO,CHECK FOR
        DFAD     XDX                TOO SMALL
        CAS      XDX                   AN INCREMENTP
        TRA      *+2             OK
        TRA      TOSML           INCPEMENT TOO SMALL
HOK     DLD      TEMP            MAKE CHANGE
        DST      HD              ..
        TSX      MULCH,4         ..
REG4    AXT      **,4            AND
        TRA      1,4               RETURN
DBLH    ZAC                      ZERO AC
        LDO      =0              ZERO MG
        DFAM     FXD             GET ABS OF LAST X
        DFSM     XD              SUB ABS OF THE PRESENT X
        DFDP     HD              DIVIDE BY OLD INCREMENT
        SSP                      USE ABSOLUTE VALUE
        FAD      =.5             ADD ONE HALF FOR SURE
        UFA      =0233000000000  GET INTEGER PART
        ANA      =0000777777777     CLEAR THE INTEGER.
        TZE      REG4                    ZEDRO DONT DOUBLE HD.
        LBT                      IS IT EVEN
        TRA      STCT            YES, ALRIGHT TO DOUBLE
        TRA      REG4            NO, USE PRESENT HD
TOSML   STZ      FORCE           MUST FORCE THIS STEP
*
        TRA      REG4
STCT    AXT      4,3              RESET DOUBLING COUNT
        TRA      HOK                     GO CHANGE VALUES.
```

```
*       COMPUTES NEW Y VALUE Y(XTD + HD)
*
PREDC   SXA     XR4,4           SAVE MACHINE CONDITIONS
        SXA     XR4+1,2
        SXA     XR4+2,1
*
*
*               DEFINE XD AND EXIT TO CALLER WHEN NECESSARY
        DLD     HD              GET CURRENT INCREMENT
        AXT     1,2             SET NEGATIVE SWITCH
        TPL     *+2             IF H POSITIVE OK
        AXT     0,2             SET POSITIVE SWITCH
        DFAD    XTD             ADD EFT MOST VALUE
        ZET     STARTD          IS THIS DURING INITIALIZATION
        TRA     STILIN+1        YES, FORGET ABOUT EXIT
        DST     TEMP            SAVE
        DFSB    FXD             SUBTRACT FINAL X
        TZE     STILIN          IF IS ZERO EVEYONE IS STILL IN
        TPL*    OUT,2           DECIDE WHERE TO GO
        TRA*    OUT+1,2         ..
        TRA     EXIT            EXIT
OUT     TRA     STILIN          OR STAY
        TRA     EXIT            EXIT
STILIN  DLD     TEMP            STAY, DEFINE
        DST     XD              XD
*               SAVE YPR(XTD) AND Y(XTD)
        LXA     NTN,2                       GET NO. EQNS.
SAVYYP  DLD     **,2                        GET NEXT YPR.
        DST     **,2                        SAVE D8.
        DLD     **,2                        GET NEXT Y.
        DST     **,2                        SAVE D5.
        TIX     SAVYYP,2,2                  DO ALL.
*
*               PREDICT NEXT VALUE OF Y
        LXA     NTN,2                       GET NO. EQNS.
 YVA1   CLA     **,2                        GET SPDP.
        TZE     SNGLE                       DO IT IN SINGLE PRE.
 YVAL   DLD     ZOO                         ZERO.ACMO.
        DFAD    **,2                        D1
        DFAD    **,2                        D2
        DFAD    **,2                        D3
        DFAD    **,2                        D4
 LAST   DFAD    **,2                        ADD YPR
 LAST1  STO     D9,2                        SAVE FACTOR.
        DFMP    HD                  MULTIPLY BY CURRENT X VALUE
 LAST2  DFAD    **,2                        ADD Y IN D5.
 LAST4  DST     **,2                        STORE Y.
```

```
*                COMPUTE THE NEXT VALUE FOR F (D-4)
         STZ     TEMP
         STZ     TEMP+1
         DLD     TWO                          ..
         DST     T1                           SAVE CONST.
         AXT     4,1                FOUR D VALUES TO CHANGE
NEXD     CLA     LAST,1
         STA     *+1
         LDQ     **,2               GET NEXT D.
         FMP     T1                 * T1.
         DFAD    TEMP
         DST     TEMP
         CLA     T1                   INCREASE
         FAD     =1.                  CONSTANT
         STO     T1                   BY ONE
         TIX     NEXD,1,1
         DLD     TEMP
 YPR1    DFAD    **,2                         ADD YPR,
 YPR12   DST     **,2                         STORE D6.
         T1X     YVA1,2,2
*
*                FIND CORRECTION FACTOR AND DETERMINE LARGEST
*                CORRECTION AND DIFFERENCE
         STZ     FMAXB                ZERO OUT ELEMENT
         AXT     2,1                  WISH TO CORRECT TWICE
AUXX     TSX     AUX,4                CALL AUX SUBROUTINE FOR YPR
         LXA     NTN,2                        GET ALL EONS.
GETA     DLD     **,2                         GET APRX YPR.
         TXH     *+2,1,1            IF FIRST AC HAS PROPER VALUE
YPR2     DLD     **,2                         GET 2ND YPR.
         STO     TEMPB                SAVE APPROX
         CLA     **,2                 GET NEW YPR
         STO     T2                   SAVE IT
         FSB     TEMPB                FORM DIFFERENCE YPR - APPROX
         STO     TEMPG                STORE
         XCA
         FMP     HD                 TEMPA=.3298611111*HD*TEMPG
         XCA
         FMP     COR
         DST     TEMPA
```

```
*                IS THIS THE LARGEST TEMPA
         SSP                      MAKE SURE IT IS POSITIVE
         CAS     FMAXB
         TRA     *+3                NO, GO SET LARGEST VALUE
         TRA     FIXY               YES, CO CORRECT Y
         TRA     FIXY               YES, CO CORRECT Y
         STO     FMAXB            THIS IS THE LARGEST TEMPA ABS
         CLA     TEMPG              SAVE DIFFERENCE
         SLW     FMAXC              SAVED AS ABS VAL
         CLA     T2                 SAVE YPR VALUE
         SLW     TEMPC              SAVED ABS VAL
 FIXY    DLD     **,2
         DST     **,2
         DLD     **,2
         DFAD    TEMPA
         DST     **,2
         TIX     GETA,2,2
         TXL     FINISH,1,1
         CLA     FMAXB
         STO     FMAXA
         STZ     FMAXB
FINISH   TIX     AUXX,1,1
XR4      AXT     **,4
         AXT     **,2
         AXT     **,1
         TRA     1,4                RETURN
EXIT     SXA     TEMP,3           SVE DOUBLING COUNT
XRI      SXA     TEMP+1,6         SAVE STEP COUNTER
         AXT     **,1
         AXT     **,2
         AXT     **,3
         AXT     **,4
         AXT     **,5
         AXT     **,6
         STL     START
         ZET     GO
         TRA     1,4
         TRA     10,4
 SNGLE   ZAC                                ZERO AC.
         FAD     **,2                       D1
         FAD     **,2                       D2
         FAD     **,2                       D3
         FAD     **,2                       D4
 LAS     FAD     **,2                       ADD YPR.
 LAS1    STO     D9,2                       SAVE (..).
         XCA
         FMP     HD                         * CURRENT X.
         TRA     LAST2
```

```
*                DETERMINES IF 22A SATISFIED OR OVERSATISFIED
*
*        8.*FMAXB.LE.FMAXA.OR.FMAXC/TEMPC.LE.3.*X.-8
*
 CK22A  STZ      BSATA          SET TO FALSE
        LDO      FMAXB            GET FMAXB
        FMP      =8.
*       FORM 8*FMAXB
        CAS      FMAXA            COMPARE WITH FMAXA
        TRA      NEXTES           AC GREATER. TRY NEXT TEST
        TRA      *+1              EQUAL  OK
        STL      BSATA            LESS THAN    SET TRUE
NEXTES  CLA      FMAXC          GET FMAXC
        FDP      TEMPC          FORM FMAXC/TEMPC
        XCA
        CAS      =3.E-8           COMPARE WITH CONSTANT
        TRA      OVERSA           FMAXC/TEMP .GT. 3*E-8
        TRA      *+1
        STL      BSATA            SET TRUE
*
OVERSA  STZ      BOVSAT           SET FLASE
        CAS      =1.5E-8        IS FMAXC/TEMPC .LE. 1.5E-8
        TRA      NEXTRY           NO
        TRA      *+1              YES
SETB    STL      BOVSAT           YES
        TRA      1,4              RETURN
NEXTRY  LDO      FMAXB            GET FMAXB
        FMP      =16.
        CAS      FMAXA            COMPARE WITH FMAXA
        TRA      1,4              NO GOOD RETURN
        TRA      SETB             OK SET TRUE
        TRA      SETB             OK SET TRUE
```

```
*                CHECK CONDITION 2 AND OVERSATISFICATION
*
CK22B   SXA      ENDIT,2
        SXA      ENDIT+1,4
        STL      BSATB              SET CONDITION 2 TRUE
NEO     LXA      N,2                N EQUATIONS
        LXA      NTN,4
 NEO1   CLA      **,4                        GET LAST YPR IN D6
        STO      T1                 SAVE T1
 NEO3   CLA      **,2                  GET TOLERANCE.
        STO      T3                 SAVE IT
 YPR3   CLA      **,4                        GET LAST YPR.
        STO      T2                 SAVE IT
        FSB      T1                 FORM YPR - D6
        SSP                        MAKE SURE IT IS POSITIVE
        STO      TEMPF              STORE THE ABSOLUTE VALUE
        NZT      BSATA                IS BSATA           FALSE
        TRA      NXTS               YES, GO TO NEXT TEST
        XCA
        FMP      HD
        SLW      T1                 SAVE ABS HD*TEMPF
        XCA
        FMP      COR                FORM CORRECTION TERM
        CAS      T3                 COMPARE WITH TOLERANCE
        TRA      *+3                GREATER
        TRA      NXTS               EQUAL
        TRA      NXTS               LESS
        STZ      BSATB              MAKE FALSE
        PXA      0,2                GET INDEX OF FALSE TOLERANCE
        STO      TOLPOS             SAVE IT
NXTS    LDQ      TEMPF              GET TEMPF
        FMP      HD                 MULTIPLY BY CURRENT H
        XCA
        FMP      =21.4409722
        SSP                        MAKE SURE IT IS POSITIVE
        CAS      T3                   COMPARE WITH TOLERANCE
        STZ      BOVSAT               GREATER SET FALSE
        TRA      *+1
        CLA      BOVSAT               CHECK TO SEE
        ADD      BSATB                IF PREMATURELY
        TZE      ENDIT                DONE
        TXI      *+1,4,-2
        TIX      NEO+2,2,1
ENDIT   AXT      **,2
        AXT      **,4
        TRA      1,4
```

```
*               PREPARES REMEMBERED VALUES
*               FOR A STEP SIZE CHANGE OF MLTPL
*
MULCH   SXA     X2,2
        SXA     X2+1,1
        SXA     X2+2,3
        LDQ     =0
        CLA     MLTPL           GET CHANGE MULTIPLE
        CAS     NUM-8                   SAME AS LAST CHANGE
        TRA     *+2             NO
        TRA     CONOK           YES, GO AHEAD AND CHANGE
        DST     NUM-8                   SAVE IN DP
        AXT     6,1
CONST   XCA
        FMP     MLTPL           FORM NEXT MULTIPLE
        DST     NUM,1                   SAVE IT IN DP.
        TIX     CONST,1,2
 CONOK  LXA     NTN,2                   ALL SETS.
        AXT     4,1             4 IN EACH SET
        AXT     8,3
 MLDP   CLA     LAST,I        GET PROPER D ADDRESS.
        STA     *+2
        STA     *+3
        DLD     **,2                    GET NEXT D.
        DFMP    NUM,3
        DST     **,2                    STORE NEW D.
        TXI     *+1,3,-2
        TIX     MLDP,1,1
        TIX     CONOK+1,2,2
X2      AXT     **,2
        AXT     **,1
        AXT     **,3
        TRA     1,4
```

```
*               HAVING ACCEPTED STEP REMEMBERED
*               VALUES AND LEFT MKST XTD VALUE
*
UPDT    SXA     XG4,4
        SXA     XG4+1,3
        SXA     XG4+2,2
        SXA     XG4+3,1
        LXA     NTN,2                       CHANGE N SETS.
D6      DLD     **,2                        GET YPR.
        DFSB    **,2                        SUB YPR D6.
        DST     TEMPA
        AXT     12,3
        AXT     8,4
 UPD1   CLA     D9+1,2                      CHECK FLAG.
        TZE     SPRE            UPDT IN SINGLE.
1NDCH   PXA     0,4
        ARS     1                           XR4/2=XR1.
        PAX     0,1             XR1 HAS NO. OF TERMS / EQUATION
        DLD     TEMPA
        DFMP    COR,4
        DFAD*   LAST,1                      OLD FACTOR.
        DST     TEMP
        TNX     AGN1,1,1
 POS2   DLD*    LAST,1                      GET NEXT FACTOR.
        DFMP    MUL,3
        DFAD    TEMP
        DST     TEMP
        TXI     *+1,3,-2
        TIX     POS2,1,1          FINISH THIS UPDATING
 AGN1   SXA     *+6,4
        PXA     0,4
        ARS     1
        PAX     0,4
        DLD     TEMP
        DST*    LAST,4
        AXT     **,4
        TIX     INDCH,4,2
```

```
PKTDP  CLA     **          BUMP DP COUNTER.
       ADD     =1
       STO     **
XG41   TIX     D6.2.2
       DLD     XD            GET CURRENT X
       DST     XTD           STORE IN LEFT MOST X CELL
XG4    AXT     **.4
       AXT     **.3
       AXT     **.2
       AXT     **.1
       TRA     1.4
SPRE   PXA     0.4
       ARS     1
       PAX     0.1
       LDO     TEMPA
       FMP     COR.4
       FAD*    LAST.1
       DST     TEMP
       TNX     AGN2.1.1
POS3   LDQ*    LAST.1
       FMP     MUL.3
       DFAD    TEMP
       DST     TEMP
       TX1     *+1.3.-2
       TIX     POS3.1.1
AGN2   SXA     *+6.4
       PXA     0.4
       ARS     1
       PAX     0.4
       DLD     TEMP
       DST*    LAST.4
       AXT     **.4
       TIX     SPRE.4.2
PKTSP  CLA     **          BUMP SP COUNTER.
       ADD     =1
       STO     **
       TRA     XG41
```

```
*                UTILITY ROUTINES
*
*
*                CALLS SUBROUTINE AUX. FOR NEW VALUES OF YPR
*
AUX      SXA     X41,4
         TSX     **,4
         PZE     XD
Y        PZE
YPR      PZE
X41      AXT     **,4
         TRA     1,4                RETURN
*
*                ROUTINE TO CHANGE THE DIRECTIONS OF THE STEP

REVRSE   SXA     REGIS4,4
         CLA     =-1.           SET MULTIPLE TO MINUS ONE
         STO     MLTPL               SAVE -1 STEP FACTOR.
         TSX     MULCH,4
         DLD     HD             GET CURRENT INCREMENT
         CHS
         LRS     0
         DST     HD             MAKE IT INCREMENT NEGATIVE
         AXT     4,3            RESET DOUBLEING COUNT
REGIS4   AXT     **,4
         TRA     1,4
         SPACE   2
ABORT    CALL    JOBOFF          FORCE OFF USER
```

```
*                INTERNAL STORAGE
         EVEN
  TMP1   DEC     0.,0.
  HALF   DEC     0.5,0.0
  T1     DEC     0.,0.
  TEMPA  DEC     0.,0.
  ZOO    DEC     0,0                          ZERO CELLS.
  TWO    DEC     2.,0.                        DP 2.
XTD      PZE
         PZE
XD       PZE
         PZE
HD       PZE
         PZE
  XDX    PZE
         PZE
FXD      PZE
         PZE
MLTPL    PZE
         PZE
  NUM    BES     8
TEMP     PZE
         PZE
         DEC     1.041666674327,0.,.486111114743,0.
         DEC     .1041666674327,0.0
         DEC     .8333339541E-2.0.
   COR   DEC     .32986111111,0.
         DEC     3.,0.,6.,0.,10.,0.,4.,0.,10.,0.,5.,0.
MUL      PZE
N        PZE
  NTN    PZE                                  N+N.
FMAXB    PZE
TEMPB    PZE
TEMPG    PZE
FMAXC    PZE
T2       PZE
TEMPC    PZE
  BSATC  PZE
BSATA    PZE
BSATB    PZE
BOVSAT   PZE
T3       PZE
TOLPOS   PZE
FMAXA    PZE
TEMPF    PZE
FLAG     PZE
```

```
GO      PZE
START   PZE
STARTD  PZE
FORCE   PZE
FIRST   PZE
 TOL    EQU     1
 D9     EQU     1
        END
```

BIBLIOGRAPHY

Nordsieck, Arnold. "On Numerical Integration of Ordinary Differential Equations". Mathematics of Computation. Vol. 16, No. 77, January, 1962.

Wilkinson, J.H. . Rounding Errors in Algebraic Processes. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1963.