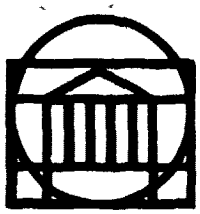
NSG-1335

RESEARCH LABORATORIES FOR THE ENGINEERING SCIENCES



SCHOOL OF ENGINEERING AND APPLIED SCIENCE

UNIVERSITY OF VIRGINIA

Charlottesville, Virginia 22901

A Report

ANALYSIS AND TESTING OF NUMERICAL FORMULAS FOR THE INITIAL VALUE PROBLEM

Submitted to:

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Submitted by:

R. Leonard Brown
Assistant Professor

Report No. UVA/528149/AMCS80/101

January 1980

A Report

ANALYSIS AND TESTING OF NUMERICAL FORMULAS FOR THE INITIAL VALUE PROBLEM

Submitted to:

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Submitted by:

R. Leonard Brown
Assistant Professor

Kurt R. Kovach

Jeffrey L. Popyack
Engineer

Department of Applied Mathematics and Computer Science
RESEARCH LABORATORIES FOR THE ENGINEERING SCIENCES
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA

Report No. UVA/528149/AMCS80/101
January 1980

Copy No. 4

FOREWORD

This work is the result of three years' effort at University of Virginia to assist NASA in choosing the numerical integrators to be used in real time simulators for aircraft and spacecraft. The first author was introduced to this subject during a visit to the Institute for Computer Applications in Science and Engineering at the NASA Langley Research Center during Summer 1975. This was followed by a grant NSG-1335 from 1976 to 1979. This report described three interrelated software systems written under this grant.

The principal investigator wishes to thank his co-authors/graduate students, also students Sandra Bollinger and Bob Athay who did some of the initial work. Dr. R. L. Bowles of NASA both got us started and provided technical guidance along the way.

Copies of the programs for CDC equipment and IBM 360 or 370 are available from the first author. For other machines, the IBM version is almost ANSI standard.

ABSTRACT

Three computer programs for the evaluation and testing of numerical integration formulas for use with fixed stepsize programs to solve initial value systems of ordinary differential equations are described. SERIES, written in PASCAL, takes as input the differential equations and produces FORTRAN subroutines for the derivatives of the system and for computing the actual solution through recursive power series techniques. Both of these are used by STAN, a FORTRAN program to interactively display a discrete analog of the Liapunov stability region of any two-dimensional subspace of the system. The derivatives may be used by CLMP, a FORTRAN program, to test the fixed stepsize formula against a good numerical result and interactively display the solutions.

TABLE OF CONTENTS

1. INTRODUCTION

Consider the numerical solution of the problem

$$y'(t) = f(y,t)$$
$$y(0) = y_0 \qquad (1)$$

for $f:R^{n+1} \rightarrow R^n$, where f is a vector valued function of independent variable t and the dependent variable y(t) in R^n, y_0 is the initial value from R^n, and y(t) is often called the state vector. This form can be used to study a wide variety of initial value problems including the following: the Method of Lines (MOL) approach to solving initial-boundary value problems in parabolic systems of partial differential equations [Ames]; higher order ordinary differential equations using a change of variable

$$y_{i+1} = d^i y/dt^i$$

so that $y'_3 = d^3y/dt^3$ [Gear]; mixed differential and nonlinear algebraic systems F(y,y',t) = 0, where F (y,y',t) is solved by an implicit numerical method using Newton's iterative method or similar [Gear]; and the equations of state, including conservation laws, of an engineering simulation. A further simplification could replace the independent variable t by a new element y_{n+1} such that $y'_{n+1} = 1$, $y_{n+1}(0) = 0$. However, this will not be used in the present analysis.

If (1) were a linear homogeneous equation

$$y' = Ay$$
$$y(0) = y_0 \qquad (2)$$

then the solution would be trivial, as the following analysis shows. For a matrix A of full rank there exists an Hermitian transformation P such that:

$$PAP^* = D$$

where D is a diagonal matrix $[\delta_{ij}\ \lambda_i]$ of the eigenvalues of A. Write (2) as

$$Py' = PAP^*\ Py \qquad (3)$$

Set $z(t) = Py(t)$ so that $z' = Dz$ is equivalent to (2) and remember that z is complex since the eigenvalues of A may be complex. Then the solution of (2) is obtained from (3) as

$$z_i(t) = \exp(\lambda_i\ t) z_i\ (0), \qquad (3)$$
$$y(t) = P^*\ z(t).$$

It has been customary to investigate the stability and accuracy of numerical solutions of (1) by describing the effect of solving (2) numerically, or more simply, by investigating the complex equation

$$y' = \lambda y,$$
$$y(0) = y_0 \qquad (4)$$

since (2) can be reduced to (4) in each component. This linear stability analysis will be described here, and its shortcomings pointed out, as a prelude to the description in Chapter 2 of the mathematical foundations of software specifically designed to analyze and test numerical methods on small ($\leq$ 20 dependent variables) subsystems of nonlinear differential equations.

Common numerical formulas for integrating the system (1) are listed in Appendix A; they are either 1-step methods

$$y_n = \phi\ (y_{n-1},\ f,\ t_n;\ h)$$

or multistep methods

$$0 = L(y_n,\ y_{n-1},\ldots,y_{n-k},\ f,\ t_n;\ h)$$

where $y_n = y(0+nh) + e_n$ is the approximate solution after n equal steps of size h in the independent variable t from the initial point $(y_0,\ 0)$. The global error e_n, which depends on h, the function f(y,t), and the numerical method is described elsewhere [Gear, Stetter]. However, the stability analysis is reviewed here.

One-step methods are typified by explicit Runge-Kutta formulas

$$K_0 = hf\ (y_{n-1},\ t_{n-1}),$$

$$K_q = hf(y_{n-1} + \sum_{j=0}^{q-1} b_{qj}\ K_j,\ t_n + \sum_{j=0}^{q-1} hb_{qj}), \qquad (5)$$

$$q = 1,\ldots,s-1,$$

$$y_n = y_{n-1} + \sum_{q=0}^{s-1} g_q\ K_q,$$

which attempt to approximate the Taylor series of y(t) about t_{n-1} by using a linear combination of S stages.

Typical examples include the two-stage formula

$$K_0 = hf\ (y_{n-1},\ t_{n-1})$$

$$K_1 = hf\ (y_{n+1} + .5\ ^*\ K_0,\ t_{n-1} + h/2)$$

$$y_n = y_{n-1} + K_1.$$

This and the typical four-stage Runge-Kutta formula are given in Appendix A.

The result in the linear case is that

$$y_n = (1 + h\lambda + (h\lambda)^2/2 + \text{---} (h\lambda)^p/p!)v_{n-1} + 0(h^{p+1}) \quad (6)$$

and the formula is called of error order $p \leq s$. Even if b_{qj} and g_q are picked so that p is as large as possible, the coefficients are still not fully specified and an infinite family of coefficients, in one or more parameters, results. In the linear case, all choices of (5) result in (6), so the linear stability analysis can be simply stated:

let $e_n = y_n - z_n$ where $y_0 \neq z_0$ are initial values used in the numerical solution, then the stability region S = ($h\lambda$: $\lim_{n\to\infty} e_n$ is finite). Since $e(t_n) = \exp(nh\lambda t)(y_0 - z_0)$, S will contain values of $h\lambda$ where $h>0$ and Re (λ) < 0 in the exact case. For (6), S will contain $h\lambda$ such that

$$\left|1 + h\lambda + \text{---} + (h\lambda)^p/p!\right| \leq 1.$$

Figure 1.1 shows the stability region of Runge-Kutta formulas of various order, and [Jeltsch] has shown that even if the coefficients are not picked to maximize p but rather to maximize the radius of the largest circle in S tangent to the imaginary axis at O., this region is still bounded for finite s.

In the multistep case, predictor only and predictor-corrector combinations are used. Predictor formulas have

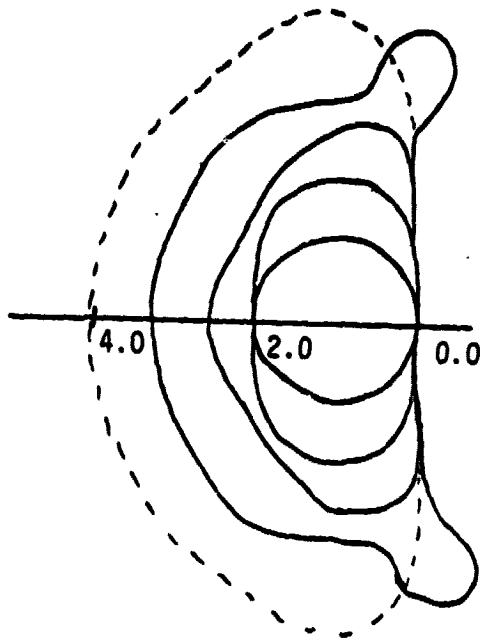


Figure 1.1 Stability regions of Runge-Kutta methods of order p=1,2,3,5, (increasing area and 7 (dotted outline).

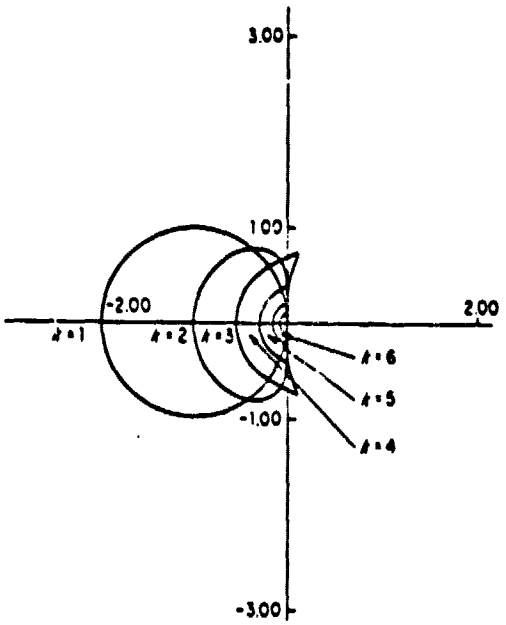


Figure 1.2. Stability regions for Adams-Bashforth methods. Method of order K is stable inside region to the left of origin.

the form

$$y_n^p = \sum_{j=1}^{k} (a_j\, y_{n-j} + hb_j\, f(y_{n-j},\, t_{n-j})) \qquad (7)$$

after applying an appropriate predictor, an implicit corrector formula may be solved to improve the solution y_n:

$$0 = \sum_{j=0}^{k} a_j^*\, y_{n-j} + hb_j^*\, f(y_{n-j},\, t_{n-j}) \qquad (8)$$

This can be rewritten (assuming $a_0^* = -1.$) as

$$y_n = s + hb_0^*\, f(y_n,\, t_n) \qquad (9)$$

where

$$s = \sum_{j=1}^{k} (a_j^*\, y_{n-j} + hb_j^*\, f\,(y_{n-j},\, t_{n-j})); \qquad (9)$$

doesn't change during the iteration. One can solve (9) iteratively by letting $y_n^{(0)} = y_n^p$ and solving

$$y_n^{(i)} = s + hb_0^*\, f(y_n^{(i-1)},\, t_n)$$

for $i = 1, 2, \ldots, Q$ where Q is either set to a constant (often 1), or chosen after some convergence criterion such as $\left| y_n^{(i)} - y_n^{(i-1)} \right| <$ some small value. Newton's method can be used on the function

$$0 = y_n - s - hb_0^*\, f(y_n,\, t_n)$$

by iterating

$$y_n^{(i)} = y_n^{(i-1)} - (I - hb_0^* J_n)^{-1}(y_n^{(i-1)} - s - hb_0^*\, f(y_n^{(i-1)},\, t_n)) \qquad (10)$$

where J_n is a close approximation to the Jacobian matrix $\partial f/\partial y\big|_{y,t}$.

The linear stability analysis often assumes that (9) is solved exactly, but in fact if Q, the number of corrector iterations, is finite, this will affect the stability. If Newton's method is used with the exact Jacobian λ, then only one iteration will yield the exact solution of (9), so this is valid in the linear case. Note that (8) is the linear, homogeneous difference equation

$$(1-b_0^* h\lambda)y_n = \sum_{j=1}^{k} (a_j^* + b_j^* h\lambda)y_{n-j}$$

and, for each particular $h\lambda$, the closed-form solution can be obtained [Gear]. This solution is of the form

$$y_n = \sum_{i=1}^{k} \left(\sum_{j=1}^{m_i} w_{ij}\, n^{j-1} \right) x_i^n \qquad (11)$$

where x_i is one of the $S \leq k$ unique roots of the polynomial equation

$$0 = \sum_{j=0}^{k} (a_j^* + h\lambda b_j^*)\, x^j,$$

m_i is the multiplicity of the i^{th} unique root, and W_{ij} depends on the k initial values $y_0, \cdots, y_{k-1}$. Note that y_n is increasing if any root $x_i > 1$, and also if $x_i = 1$ then any terms $w_{ij}\, n^{j-1}\, x_i^m$ will increase if $m_i > 1$. Therefore, two different solutions to (8) with initial values y_0, z_0 define $e_n = y_n - z_n$, and e_n will be bounded for any $h\lambda$ and any initial conditions if and only if all roots of (11) satisfy the two conditions above i.e. the stability region $S = [h\lambda:$ all roots of (11) are less than one in norm, or on

the unit circle and simple]. Figure 1.2 shows the stability regions for selected Adams-Bashforth predictors (see Appendix A). These are plotted by allowing $x = \exp(i\phi)$ for some large number of points ϕ between 0 and 2π, and solving (11) for $h\lambda$ in each case.

If the corrector is computed by Q iterations, then (11) becomes instead

$$y_{(0)} = y^p = \sum_{j=1}^{k} ((a_j + h\lambda b_j)\, x^{k-j}) \qquad (12)$$

$$y^{(s)} = \sum_{j=1}^{k} ((a_j^* + h\lambda b_j^*)\, x^{k-j}) + h\lambda b_0^*\, y^{(s-1)},$$

$s = 1,---,Q$ and for each $x = \exp(i\phi)$, there will be Q values of $h\lambda$ since $y^{(Q)}$ is a Q degree polynomial in $h\lambda$. However, the $\lim_{(Q\to\infty)} y^{(Q)} = y_n$, the exact solution to (11), if (12) converges at all.

However, problems of interest are not linear homogeneous, and therefore the linear stability analysis gives information only about the local behavior of the related differential equation

$$z' = J_n(z-y(t)) + f(y(t), t)$$

$$z(t_n) = y_n$$

This behavior could change drastically for even small changes in y_n, and in many cases the eigenvalues of J_n are not known unless the equations were artificially linearized before solving. A more useful stability analysis would try to match the stability characteristics of the nonlinear

function f(y,t) by proper choice of numerical method which most closely approximates those characteristics in important areas of the domain of f(y,t). This has been attempted in the work reported here. Software has been written which will accept a system of up to 20 nonlinear differential equations, specified by input equations similar to simulation languages such as DARE-P [Korn et. al.]. This will output two subroutines. One of these, DIFFUN (T,Y,DY), will return in the array DY(*), *=1,--,m, the derivatives evaluate at independent variable t=T, dependent variable Y(*). The other subroutine, SOL(T,YO,YNEW,INO) will return, in YNEW(*), a series solution of the equation at t = T > 0 given the initial conditions y_0 in YO(*), if possible. This is described in Chapter 3.

These routines are then used in the interactive graphics software described in Chapter 2 which, given M-2 initial values, searches the remaining 2-dimensional plane about an approximate equilibrium point for a connected region of initial values having a particular property related to stability. These regions can be graphically displayed for both the exact and various numerical solutions. The routine DIFFUN can also be used by the testing routine CLMP described in Chapter 4. This routine, given DIFFUN, the initial values, and possibly an inhomogeneous input u(t), will display the Fourier amplitude for the first 20

harmonics for both an "ideal" solution generated by state of the art software, and also by a chosen fixed step size numerical method. These amplitudes can be compared graphically using a bar graph, two solutions can be graphed, and the ratios of the Fourier harmonics to the input can be displayed. In summary, this software can be used to choose the most appropriate of available numerical methods by comparing stability domains for numerical solutions to the same domains for the exact solution. This will insure that the numerical solution is stable. Then it is possible to verify that decision by testing for accuracy by observing appropriate results of a numerical simulation. Each of the following chapters outlines one stage in this sequence; the first section of each chapter will give some relevant theoretical considerations, the second section will comprise a user's manual for that segment of the software, with examples.

2. STABILITY ANALYSIS OF THE NONLINEAR INITIAL VALUE PROBLEM

The standard linear stability analysis is formula specific and describes the behavior of a numerical formula applied to the complex test equation

$$y' = \lambda y,$$

$$y(t_0) = y_0.$$

Let $E_n = [e(t_{n-k+1}),\ldots,e(t_{n-1}), e(t_n)]$ be the difference sequence for $y(t_n)-z(t_n)$ where each solves the test equation for a different initial value y_0, z_0. Then $e(t_n) = (y_0 - z_0)\exp(\lambda t)$ is non-increasing in norm for Real $(\lambda) \leq 0$. Such a condition is called stability. It is desirable for the numerical solution y_n to be stable if the true solution is stable, so for a given stepsize h, one finds all complex λ such that any numerical sequence $E_n^* = [e_{n-k+1},\ldots,e_{n-1}, e_n]$ has the property $|e_{i+1}| \leq |e_i|$ for all i, where $e_i = y_i - z_i$, the difference between the numerical solution sequences with initial values y_0, z_0.

For Euler's formula, $y_n = (1 + h\lambda)\, y_{n-1}$, hence the numerical solution is stable for $|1 + h\lambda| \leq 1$. For multistep formulas, linear stability is characterized by the generating polynomials

$$r(x) = \sum_{i=0}^{k} a_i\, x^{k-i}$$

$$s(x) = \sum_{i=1}^{k} b_i\, x^{k-i} \qquad (1)$$

where r and s have no common divisors. The region of linear stability is all $h\lambda$ such that $r(x) + h\lambda\, s(x)$ has all roots inside the unit circle, or on the unit circle and simple. For Euler's formula, the $r(x) + h\lambda\, s(x)$ is $(-x+1)+ h\lambda$. Since this analysis is formula specific, to investigate the formula's effect on an actual $f(y,t)$ one considers all the eigenvalues λ_i of the Jacobian matrix $\partial f(y,t)/\partial y$; if all $h\lambda_i$ are inside the stability region for all y_n, t_n of interest, then the numerical solution will be stable. Insuring that such a condition holds is usually not desirable and often not possible.

To develop a stability analysis for nonlinear $f(y,t)$ let $f(y,t)$ have the property

$$\text{Real} \langle y-z,\ f(y\ t)-f(z,t)\rangle \leq \mu \left\| y-z\right\|^2 \qquad (2)$$

for all t,y, and z of interest. Here $\langle u,v\rangle = y^t Qv$ for some positive definite Hermitian matrix Q, and $\left\|u\right\|^2 = \langle u,u\rangle$. Then for any two solutions $y(t)$, $z(t) = y(t)-e(t)$, $e(t)$ satisfies

$$de(t)/dt = f(y(t),t) - f(z(t),t)$$

and (2) implies that

$d \left\|e(t)\right\|^2/dt = 2\ \text{Real} \langle e(t),de(t)/dt\rangle \leq 2\mu \left\|e(t)\right\|^2$ and thus

$$e(t) \leq \exp(\mu t)\ e(t_0)$$

which is non-increasing for $u \leq 0$.

However, (2) is again a condition that cannot be easily verified, so a concept relating the true solution sequence

$Y(t_n)$ to the computed solution sequence Y_n is presented here. The following definitions and theorem are helpful. They occur in [Dahlquist, 1978] and the theorem proof is presented in [Brown, 1979a] and is reproduced in Appendix B.

Definition - a linear k step formula satisfies

$$0 = \sum_{j=0}^{k} a_j \bar{y}_{n-j} + hb_j f(\bar{y}_{n-j}, t_{n-j}) \qquad (3)$$

Definition - a one leg k step formula corresponding to (3) satisfies

$$0 = \sum_{j=0}^{k} a_j y_{n-j} + hsf(1/s \sum_{j=0}^{k} b_j y_{n-j}, 1/s \sum_{j=0}^{k} b_j t_{n-j}) \qquad (4)$$

where $s = s(1)$ and without loss of generality can be set to 1 by proper scaling of the coefficients.

Theorem 1 - Let Y_n be a sequence which satisfies (4), and let $\bar{Y}_n = \{\bar{y}_n\}$ be such that

$$\bar{y}_n = \sum_{j=0}^{k} b_j y_{n-j} = s(E)y_n \qquad (5)$$

where E denotes the back shifting operator. Then $\bar{Y}_n$ satisfies (3). Conversely, if $\bar{Y}_n$ satisfies (3) then there exists a sequence Y_n such that $\bar{y}_n = S(E)\, y_n$, and Y_n satisfies (4).

This shows that Y_n given by the one-leg k step formula will have similar stability properties to its corresponding linear k-step sequence $\bar{Y}$. In [Dahlquist, 1975] there is described a discrete Liapunov function $V_{G,k,h}$ which, applied

to a sequence Y_n, characterizes the stability of that sequence generated by a nonlinear system $y' = f(y,t)$.

Let

$$V_{G,k,h,\ (Y_n)} = \sum_{i=1}^{k} \sum_{j=1}^{k} g_{ij} \langle y_{n+1-i},\ y_{n+1-j} \rangle$$

where G is a positive definite, k by k symmetric matrix. The structure of G assures that $V_{G,k,h}$ is positive definite for $Y_n \neq \{0\}$.

Definition - The (G, k, h) domain of attraction of (the numerical solution to) the system is all z_0 such that

$$\Delta V_{G,k,h}(Z_0) = V_{G,k,h}\ (Z_1) - V_{G,k,h}\ (Z_0) \leq 0,$$

where

$$Z_0 = \{z((1-k)h),\ldots,z(-h),z(0)\},$$

$$Z_1 = \{z((1-k)h),\ldots,z_0,z_1\}$$

in the numerical case and

$$Z_1 = \{z((1-k)h),\ldots,z_0,z(h)\}$$

for the exact case.

Definition - The (G,k,h) stability region of (the numerical solution to) the nonlinear system is all z_0 such that

$$V_{G,k,h}\ (Z_0) \leq \inf_{(Z_0 \varepsilon \partial D)} V_{G,k,h}\ (Z_0)$$

where ∂D is the boundary of the (G,k,h)-domain of attraction.

This has the following application. Rather than requiring that Real $\langle y-z,\ f(t,y)-f(t,z) \rangle \leq 0$, a connected subset of initial values y_0 is found such that y(h) will be

in that subset if y_0 was. This is the stability region. This insures that the difference y(h)-z(h) is bounded since both y(h) and z(h) are in the stability region if y_0 and z_0 were. If f(y,t) is autonomous, $y(t_n)$ will remain in the region as $n \to \infty$. For most well-behaved functions f(y,t), the boundary of the region around a stable point can be approximated computationally. Once the analytic stability region is known, the numerical stability region can be calculated using the one-leg k-step method for the same sequence $[y((1-k)h), \ldots, y(h), y_0]$ to get y_1. The two regions can then be compared.

Analytically, it is possible to form a particular G, based on the coefficients of a one-leg k step method, such that all numerical sequences based on f(y,t) that satisfy (2) will have a stable solution. It was shown in [Liniger and Odeh] how to pick G for second order two step formulas, second order three-step formulas, and third order three-step formulas.

It is shown below that even an arbitrary choice of the positive definite Hermitian matrix G will generate some usable results, and theorem 2 demonstrates that using some G for a one-leg k step solution y_n will generate the same stability region for the related solution $\bar{y}_n$ of the linear k step formula for a modified $\bar{G}$. The proof appears in [Brown 1979a] and in appendix B.

Theorem 2 - If $V_{G,k,h}(Y_n) = c$ for the symmetric positive definite matrix G, then there exists a symmetric, positive definite matrix $\bar{G}$, dependent only on G and s(x), such that $V_{G,k,h}(\bar{Y}_n) = c$, where $\bar{y}_n = S(E)y_n$ are the elements of $\bar{Y}_n$ and Y_n.

Sufficient conditions can be developed for the existence of the (G,k,h) stability regions based on known techniques such as Liapunov's direct method [Lenigk] and property (2). An important concept in the development is the equilibrium $y^*(t)$ of the differential function f(y,t). While some references define it for an initial value

$$y^*(t_0) = 0$$

in the space of the dependent variables, this is accomplished by an unnecessary change of variables that could be confusing. The important point is that $y^*(t)$ satisfies

$$f(y^*(t_0), t_0) \text{ be } 0 \text{ at } t_0 = 0,$$

so that, if f is autonomous, then y(t) is constant, and otherwise, the Taylor series about t_0 is given by $y^*(t_0) + t\text{-}t_0)^2 f'(z)/2$ and is thus slowly varying and nearly constant for t near t_0.

Definition - The solution $y^*(t)$ of y' = f(y,t) for $y(t_0) = \bar{y}_0^*$, such that $f(y_0^*, t_0) = 0$, is called the <u>equilibrium of f(y,t)</u>.

Definition - The equilibrium of f(y,t) is said to be <u>asymptotically stable</u> if there exists a t_1 in (t_0, ∞) such

that for every $\varepsilon>0$ there exists $d_1 = d_1(\varepsilon,t_1) > 0$ such that if $|y_0 - y_0^*| < d_1$, then

$$|y(t)-y^*(t)| < \varepsilon$$

in $[t_1, \infty]$, and there exists a t_2 in (t_0,∞) and $d_2(t_2)$ such that if $|y_0 - y_0^*| < d_2(t_2)$ then

$$\lim(t\to\infty) |y(t) - y^*(t)| = 0,$$

where $y(t)$ satisfies

$$y'(t) = f(y,t),$$

$$y(t_0) = y_0.$$

Theorem 3 [Lenikg] - The equilibrium of $f(y,t)$ is asymptotically stable if there exists a function $v(y,t)$ which is positive definite in some region D about $y^*(t_0)$ and $\lim v(y,t) = 0$ uniformly in t as $\|y-y^*\| \to 0$ there, and whose total derivative is negative definite on D.

With this background, it can be shown that a well-behaved $f(y,t)$ which has a not necessarily unique Liapunov function $v(y,t)$ with region D implies the existence of a (I,k,h) domain of attraction D' and stability region D'' of both the exact solution and of a one-leg k step solution based on a stable multi-step formula. These two theorems are stated and proved in Appendix B and in [Brown, 1979b].

In [Dahlquist, 1978] and [Leveque et al] an algorithm is presented for calculating a G-stability matrix given any A-stable linear multistep method. G is guaranteed to be positive definite and symmetric. This code is included in

STAN. Some notes on theoretical considerations concerning its use are presented here. Since G can only be generated for A-stable formulas which include the entire left complex half-plane in their stability region, and yet only certain implicit multistep methods of order no greater than two can be A-stable, the method is not immediately useful for any explicit and most implicit formulas. However, by considering the modified method

$$r^*(x) = ar(x) + bs(x)$$
$$s^*(x) = cr(x) + ds(x)$$

then

$$0 = r^*(E)y_n + qs^*(E)y_n$$

for

$$q = (a\lambda + b)/(c\lambda + d)$$

may be A-stable, even though

$$0 = r(E)y_n + \lambda s(E)y_n$$

is not, for proper choice of a, b, c, and d.

A further extension looks at

$$r^{**}(x) = r^*(\phi x) - m(\phi)s^*(\phi x)$$
$$s^{**}(x) = s^*(\phi x)$$

where

$$m(\phi) = \min\ (x = \phi)\ \text{Real}\ (r^*(x)/s^*(x)).$$

The coefficients a, b, c, d, and ϕ have the following applications in the work here. If the multistep formula is stable at infinity, then there exists some point m^1 such that the linear stability region includes $h\lambda$ such that

Real ($h\lambda$) < m_1. Whether or not the formula is stable at infinity, it will contain a region in the left half-plane adjacent to 0., and we are interested in the largest stable disk, with diameter m_2, tangent to the imaginary axis at 0.. These cases are handled by setting $\phi = 1$, and (a, b, c, d) = (1,0,0,1) and finding $m_1 = m(1)$ for the stable at infinity case for r^{**}, s^{**}. For the disk case, set $\phi = 1$, (a,b,c,d) = (0,1,1,0), and the diameter is $-1/m(\phi)$. Figure 2.1 illustrates this for the 5^{th} order BDF formula.

After obtaining G by these techniques, it is guaranteed that if Y_n is generated using (4) from

$$y' = f(y,t)$$

then

$$V_{G,k,h}(Y_{n+1}) \leq \phi' \cdot V_{G,k,h}(Y_n)$$

where

$$\phi' = \begin{cases} \phi \text{ if } \mu h \leq m(\phi) \\ \phi((1+b(\phi)(\mu h - m(\phi))/(1-b\phi)(\mu h - m(\phi)))^{1/2} \\ \text{if } m(\phi) \leq \mu h \leq m(\phi) + 1/h(\phi) \text{ if } \mu h \geq m(\phi) \end{cases}$$

where

$$\text{Real} < ahf(u(y)) + bu(y) - (ahf(v(y)) + bv(y)), u(y) - v(y) > \leq \mu \| u(y) - v(y) \|^2$$

and u and v satisfy

$$chf(u(y)) + d(u(y)) = y.$$

The term $b(\phi)$ depends on s^{**}.

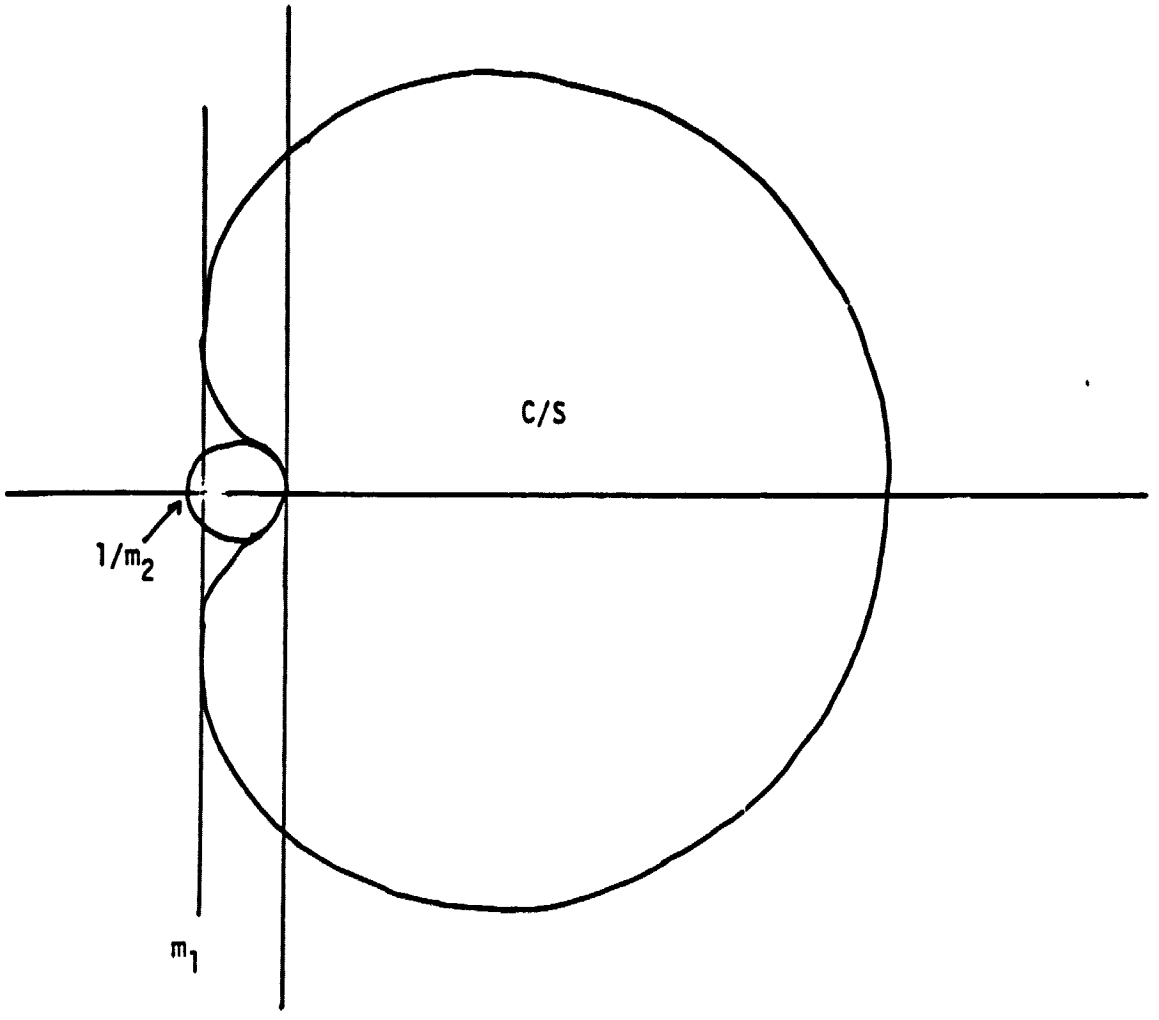


Figure 2.1. Interpretation of $m(\phi)$ for methods stable at infinity (m_1) and not necessarily stable at infinity (m_2).

However, we are interested in those initial values corresponding to Real < f(y)-f(z), y-z> ≤ 0, but computing where

$$V_{G,k,h}(y_1) \leq V_{G,k,h}(y_0)$$

G chosen as above, will show the particular numerical method to its best advantage. In the analytic case, G is chosen so that

$$G = \{g_{kk} = 1,\ g_{ij} = 0 \text{ otherwise}\}.$$

Since $V(Y)_{G,k,h}$ depends on the cross-product

$$\langle y_i, y_j \rangle = y_i^t Q y_j$$

it is helpful to compute Q that is appropriate to the two-dimensional subspace of the problem being solved. This is done by computing the positive definite, symmetric matrix that would be used as the Liapunov stability function v(y) if only those two variables were involved. This is done by decomposing

$$f(y,t) = By + g(y,t)$$

where B is the numerically differenced Jacobian around t = 0, y = y* the approximate equilibrium. The Q can be computed so that

$$B^t Q + QB = I$$

for ^{t} the transpose operator.

STAN USER MANUAL

An interactive graphics program called STability ANalysis (STAN) is stored in object form on disk and can be linked with two subroutines SOL (T,YO,YNEW,IND) and DIFFUN (T,Y,DY). These subroutines can be written by the user or be produced by the PASCAL program SERIES described in Chapter 3. DIFFUN returns in the N-dimensioned array DY the derivative f(y,t) given t = T, Y = y(t) an array of dependent variable values. SOL returns in the N-dimensional array YNEW the solution to y'=f(y,t) at T=t, given YO = y(0) an N-dimensioned array of initial values at t=0. SERIES uses recursive power series techniques for this and sets IND > 0 if the radius of convergence is not (-1,1); if the analytic solution is known it can be programmed by the user. STAN can also work with only a dummy routine SOL (T,YO, YNEW,IND) which assigns YNEW(I) = YO(I), I=1,...,N, if the analytic stability properties are never requested.

The program uses prompts to guide the user through its execution. When the possible commands, usually strings of no more than 10 characters, are not obvious, the command HELP will list them. Only enough of the command to establish its unique identity need be given. Many of the commands are not needed for simple jobs because the execution of the GO command will automatically prompt the user to

provide the necessary information. The commands are available to allow the user to reset his system for more complicated jobs. Default values are provided for almost every program parameter. A typical load under the NOS operating system, given the object code of STAN and the subroutines SOL and DIFFUN from SERIES, follows -

```
ATTACH,STAN.
ATTACH,SOL.
ATTACH,DIFFUN.
FTN,I=SOL,B=X.
FTN,I=DIFFUN,B=Y.
LIBRARY(IMSLLIB,PLOT10)
LOAD(STAN,X,Y)
NOGO,Z.
```

PLOT10 and IMSLLIB are the TEKTRONIX graphics library and the IMSL library, respectively. Z is the resulting load module, to be executed.

The available commands are of three types - those that describe the system of equations being investigated; those describing the solution method being used, including the analytic solution; and those describing the desired display. These will be described here. The first group includes INIT, CENTER, SYSTEM, NEQ, and NORM. A two-dimensional subsystem of N-differential equations will be investigated by holding N-2 variables constant at t=0, and varying the other 2 "active" variables about an approximate equilibrium

point CENTER where the derivatives of the two active variables are nearly zero. Such a point must be known in advance from analytic considerations, but the equilibrium can be improved by using Mueller's method [Conte, de Boor] for finding the root of 2-dimensional nonlinear function. The commands are -

NEQ - specify N, the dimension of the system. Default is two.

SYSTEM - specify i1, i2 the two dependent variables to be investigated. Default is i1 = 1, i2 = 2.

INIT - set the values the N-dependent variables take at t = 0.

Default is 0.

CENTER - set the equilibrium point CENTER of the two-dimensional subspace where f(y,t) = 0. When the program prompts IMPROVE CENTER,..., a reply of YES will use the initial value given and try to solve

$$f_{i1}(y,t)=0$$

and

$$f_{i2}(y,t) = 0.$$

NORM - find Q such that B^t Q+QB=I where B is the 2 by 2 Jacobian of the system with respect to y_{i1}, y_{i2}. Whenever SYSTEM is called after the initial GO command, this is done automatically. Calling NORM is usually not necessary unless the stepsize H has been changed, or the initial values are drastically changed. B is computed by differencing.

Commands describing the solution being used are NSTEPS, STEPSIZE, and TYPE. These commands tell whether the analytic solution or a k-step numerical method is being used, and what stepsize H is employed. Since the desired output is either D' or D'', initial values of y_{i1} and y_{i2} such that the function $V(Y)_{G,k,h}$ is decreasing, these commands are used to set k and h. Note that one should compare the analytic D' and D'' for the same k as the numerical method, since D' and D'' are likely to be smaller as KH grows larger KH < 1 is required if SOL is generated by a power series expansion. G is picked automatically for the various numerical methods, G = [g_{kk} = 1, g_{ij} = 0 otherwise] for the exact case, so the interior of D' satisfies

$$\langle y(kh),y(kh)\rangle < \; < \langle y((k-1)h),y((k-1)h)\rangle$$

for the exact solution y(t).

The commands are -

NSTEPS - set K, the number of steps in the numerical methods being compared. When K is changed, D' and D'' for the exact case should be recomputed. Default is 1, maximum is 10.
STEPSIZE - H, the stepsize in the independent variable, is set. This is normally set by GO the first time, and need not be changed unless D' is needed for various stepsizes.
TYPE - EXACT or NUMERICAL, sets indicators so that when the GO command is given a K step numerical method and corresponding matrix G are chosen.

The commands that describe the display are NPOINTS, SYMMETRIC, PARAM, and REPEAT.

NPOINTS - sets NP, the NP rays spaced at equal angular displacement about CENTER in R^2 are generated to find the boundaries of D' and D''. $3 \leq NP \leq 80$, default 13.

SYMMETRIC - if the system is known to be symmetric, then the NP plotted points are concentrated in only half of R^2. When the system prompts is answered by 1, this means the system is symmetric about $y_{i_2} = 0$ (the upper half-plane is graphed); 2 means symmetry about $y_{i1} = 0$ (the right half plane is graphed); 0 cancels the symmetric display.

PARAM - display most of the parameters that have been set.

REPEAT - repeat the last graph, allowing display limits to be set.

The command GO initiates calculation of D' and D''. On the first GO call, the user must supply H, then Q is computed and listed. If this is a numerical test, one of the numerical methods in Appendix A must be chosen, or else a k step predictor corrector (PC) is input by the user. When asked for the predictor, enter

$$a_0, a_1, \ldots, a_k, b_0=0, b_1, \ldots, b_k.$$

If the number of corrector iterations is given as 0, no corrector need be entered. Otherwise, give

$$a_0^*, \ldots, a_k^*, b_0^*, \ldots, b_k^*.$$

The program computes G, depending on whether the corrector

is known to be stable at infinity or not. If the approximate equilibrium is not inside D, the Liapunov stability region, then the message INVERSE REGION will be printed; try a new CENTER or a different subsystem. If the system is too irregular or too nonlinear, then the message UNABLE TO... and a PAUSE will be printed. A carriage return will cause the program to return to the beginning command sequence. Choice of a different subsystem or rewriting the system of equations may solve this problem.

After D', the domain of attraction, is computed, it can be displayed. On the first GO call, the left, right, bottom, and top values in D' are printed, and the user can choose his display coordinates. Thereafter, these coordinates can be changed or left alone. After choosing whether to display D', one can compute and display D'' in a similar fashion with the default coordinates being those chosen for D'. In many problems, D'' may be so close to D' that only D' need be displayed. This will save a great deal of computation. After this, the user is returned to the beginning command sequence, but now all parameters have been initialized and, after changing any of them, a new GO will proceed faster than the first.

EXAMPLE

The following set of equations form a subsystem that models the longitudinal stability of the F4 aircraft [Steinmetz et al] -

$$A = ATAN(W/U)$$

$$V^2 = U^2 + W^2$$

$$U' = -C1*SIN\ \phi - W*Q + (C1 + C2*A)V^2 + C3*Q*V$$

$$W' = C1*COS\ \phi + U*Q + (C4+C5*D+C6*A)\ V^2 + C7*Q*V$$

$$Q' = (C8+C9*(A+D))V^2 + C10*W'/U + C11*Q*V$$

$$\phi' = Q$$

$$D = 2*T*.174533 \text{ if } 0 \leq T \leq 0.5$$

$$0.1734533 \text{ if } 0.5 < T$$

where U is the horizontal velocity along the aircraft body, W is the vertical velocity perpendicular to U, A is angle of attack, ϕ is pitch, Q is rate of change of pitch, and D is the driving function, the stabilator deflection angle. These equations will be put in a usable form for input to SERIES and a sample run of STAN will illustrate the various features. The constants Ci appear in the program listing in Figure 2.2.

Figure 2.2a Input to Series

```
/* C=32.16, C1=-.5018521E-9, C2=.1192125E-5, C3=-.42110675
     E-3, C4=-.14598254E-3, C5=.46345531E-5,
     C6=-.45006065E-5, C7=-.6971899E-3, C8=-.587215E-8,
     C9=-.73872E-6,  C10=.3800645E-5,  C11=-.7422436  E-2*/

U.=-C*SIN(THETA)-W*Q+(C1+C2*A)*(U**2+W**2)
     +C3*Q*(U**2+W**2)**.5;

W.=C*COS(THETA)+U*Q+(C4+C5*D+C6*A)*(U**2+W**2)
     +C7*Q*(U**2+W**2)**.5;

Q.=(U**2+W**2)*(C8+C9*(A+D))+(C10*((C*COS(THETA)
     +U*Q+(C4+C5*D+C6*A)*(U**2+W**2)
     +C7*Q*(U**2+W**2)**.5)/U)+C11*Q)
     *(U**2+W**2)**.5;
     THETA.=Q;

A.=(U*(C*COS(THETA)+U*Q+(C4+C5*D+C6*A)*(U**2+W**2)
     +C7*Q*(U**2+W**2)**.5)-W*(-C*SIN(THETA)
     -W*Q(C1+C2*A)*(U**2+W**2)
     +C3*Q*(U**2+W**2)**.5))/(U**2+W**2);
     D.=.349066;;
```

Figure 2.2b.

DIFFUN, the output from SERIES used by STAN and CLMP.

```
SUBROUTINE DIFFUN(T,Y,DY)

DIMENSION DY(20), Y(20)

DATA C/32.16/, C1/-.5018521E-9/, C2/.1192125E-5/,
     C3/-.421106759E-3/,  +C8/-.587215E-8/,  C9/-.73872E-6/,
     C10/.3800645E-5/, C11/-.7422436E-2/

DY(1)=-C*SIN(Y(4))-Y(2)*Y(3)+(C1+C2*Y(5))*(Y(1)**2+Y(2)**2)
     +C3*Y(+3)*(Y(1)**2+Y(2)**2)**.5

DY(2)=C*COS(Y(4))+Y(1)*Y(3)+C4+C5*Y(6)+C6*Y(5))*(Y(1)**2
     +Y(2)**2+)+C7*Y(3)*(Y(1)**2+Y(2)**2)**.5

DY(3)=(Y(1)**2+Y(2)**2)*(C8+C9*(Y(5)+Y(6)))
     +(C10*((C*COS(Y(4))+Y(+1)*Y(3)
     +(C4+C5*)(6)+C6*Y(5))*(Y(1)**2+Y(2)**2)
     +C7*Y(3)*(Y(1)**2+Y(+2)**2)**.5)/Y(1)
     +C11*Y(3))*(Y(1)**2+Y(2)**2)**.5

DY(4)=Y(3)

DY(5)=(Y(1)*(C*COS(Y(4))+Y(1)*Y(3)+(C4+C5*)(6)
     +C6*6(5))*(Y(1)**2+Y(2)**2)+C&*Y(3)*(Y(2)**2+Y
     (+2)**2)**.5)/Y(2))+C11*Y(3))*(Y(2)**2+Y(2)**2)**.5

DY(4)=Y(3)

DY(5)+(Y(1)*(C*COS(Y(4))+Y(1)*Y(3)+(C4+C5*Y(6)+C6*Y(5))
     *(Y(1)**2+Y(2)**2(+C7*Y(3)*(Y(1)**2+Y(2)**2)
     **.5)Y(2)*(-C*SIN(Y(4))-Y)2*Y(+3)
     +(C1+C2*Y(5))*(Y(1)**2+Y(2)**2)
     +C3*Y(3)*(Y(1)**2+Y(2)**2)**.5))/
     +(Y(1)**2+Y(2)**2)

DY(6)=.349066

RETURN

END
```

Figure 2.2C - SOL, used by STAN.

```
      SUBROUTINE SOL(T,YO,YNEW,IND)

      DIMENSION  YO(20),  YNEW(20),  ZZZB(20),  TBC(20),  DW  (20),
           TB7(20),  TBD(20),  +D(20),  TB8(20),  TBE(20),  TC7(20),
           TDA(20),  TBF(20),  TCD(20),  TC8(20),TBG+(20),  TCE(20),
           TC9(20),  TBH(20),  TBI(20),  TBJ(20),  TDF(20),  TBK(20),
           TCI+(20),  TDG(20),  TBL(20),  TCJ(20),  DA(20),  TBM(20),
           TCK(20),  TBN(20),  TCL+20,  DTHETA(20),  DD(20),  Q(20),
           TBR(20),  TBS(20),  TBT(20),  TEN(20),  U+(20),  TBU(20),
           TEO(20),  TBV(20),  W(20),  TBW(20),  TBX(20),  THETA(20),
           TBY +20),  TES(20),  TBZ(20),  TBO(20),  TB1(20),  DQ(20),
           TB2(20),  DU(20),  TBA+(20),  A(20),  TBB(20),  TB6(20),
           TD2(20)

      DATA C/32.16/, C1/-.5018521E-9/, C2/.1192125E-5/,
           C3/-.42110675E-3/, C+4/-.14598254E-3/,
           C5/.46345531E-5/, C6/-.45006065E-5/,
           C7/-.6971899+E-3/,C8/-.587215E-8/, C9/-.73872E-6/,
           C10/.3800645E-5/, C11/-.74224+36E-2/

      EPS=1.0E-10
      U(1)=YO(1)
      W(1)=YO(2)
      Q(1)=YO(3)
      THETA(1)=YO(4)
      A(1)=YO(5)=ATAN2(YO(2), YO(1))
      D(1)=YO(6)
      IND=0
      DO 1 III=1,19
      NIII=III

      III1=III-1
      IF(III.EQ.1) GO TO 100
      TBB(III)=0. TBA(III)+0.
      DO 101  JJJ=1,III1
      TBA(III)=TBA(III)+TBB(JJJ)*(III-JJJ)*THETA(III-JJJ+1)

      101  TBB(III)+TBB(III)-TBB(III)-TBA(JJJ)*(III-JJJ)*THETA
           (III-JJJ+1)
           TBA(III)=TBA(III)/(III-1.)
           TBB(III)=TBB(III)/(III-1.)
           GO TO 102
      100  TBA(III)=SIN(THETA(III))
           TBB(III)+COS(THETA(III))
      102  CONTINUE
           TBC(III)=TBA(III)*C
           TBD(III)=-TBC)III)
           TBD(III)=0. Do 103 JJJ=1,III
```

```
      Do 103 JJJ=1,III
103   TBE(III)=TBE(III)+W(JJJ)*Q(III-JJJ+1)
      TBF(III)=TBD(III)-TBE(III)
      TBG(III)=A(III)*C2
      TBH(III)=TBF(III)
      IF (III.EQ.1)TBH(III)=C1+TBG(III)
      TBI(III)=0.
Do 104 JJJ=1,III
104   TBI(III)=TBI(III)+U(JJJ)*U(III-JJJ+1)
      TBJ(III)=0.
      Do 105 JJJ=1,III
105   TBJ(III)=TBJ(III)+W (JJJ)*W(III-JJJ+1)
      TBK(III)=TBI(III)+TBJ(III)TBL(III)=0.
      DO 106 JJJ=1,III
106   TBL(III)=TBL(III)+TBH(JJJ)*TBK(III-JJJ+1)
      TBM(III)=TBF(III)+TBL(III)
      TBN(III)=Q(III)*C3
      IF(III.EQ.1) Go to 107
      TBR(III)=0.
      Do 108 JJJ=1,III1
108   TBR(III)=TBR(III)+(((.5+1.)*(III-JJJ))
      /(III-1.)-1.)*TBR(JJJ)*TBK+(III-JJJ+1)
      TBR(III)=TBR(III)/TBK(1)
      Go to 110
107   TBR(III)=TBK(III)**.5
110   CONTINUE
      TBS(III)=0.
      Do 111 JJJ=1,III
111   TBS(III)=TBS(III)+TBN(JJJ)*TBR(III-JJJ+1)
      TBT(III)=TBM(III)+TBS(III)
      DU(III)=TBT(III)
      U(III+1)=DU(III)/FLOAT(III)
      IF (III.EQ.1)  Go to 112
      TBU (III)=0.
      TBV(III)=0.
      Do 113 JJJ=1,III1.
      TBV(III)=TBV(III)+TBU(JJJ)*(III-JJJ)*THETA(III-JJJ+1)
113   TBU(III)=TBU(III)-TBV(JJJ)*(III-JJJ)*THETA(III-JJJ+1)
      TBV(III)=TBV(III)/(III-1.)
      TBU(III)=TBU(III)/(III-1.)
      Go to 114
112   TBV(III)=SIN(THETA(III))
      TBU(III)=COS(THETA(III))
114   CONT.NUE
      TBW(III)=TBU(III)*C
      TBX(III)=0.
      Do 115 JJJ=1,III.
115   TBX(III)=TBX(III)+U(JJJ)*Q(III-JJJ + 1)
      TBY(III)=TBW(III)+TBX(III)
      TBZ(III)=D(III)*C5
      TBO(III)=TBZ(III)
```

```
      If (III.EQ.1)TBO(III)=C4+TBZ(III)
      TB1(III)=A(III)*C6
      TB2(III)=TBO(III)+TB1(III)
      TB6(III)=0.
      Do 116 JJJ=1,III.
  116 TB6(III)=TB6(III)+TB2(JJJ)*TBK(III-JJJ+1)
      TB7(III)=TBY(III)+TB6(III)
      TB8(III)=Q(III)*C7
      TCD(III)=0.
  117 TCD(III)=TCD(III)+TB8(JJJ)*TBR(III-JJJ+1)
      TCE(III)=TB7(III)+TCD(III)
      DW(III)=TCE(III)
      W(III+1)=DW(III)/FLOAT(III)
      TCI(III)=A(III)+D(III)
      TCJ(III)=TCI(III)*C9
      TCK(III)=TCJ(III)
      IF(III.EQ.1)TCK(III)=C8+TCJ(III)
      TCL(III)=0.
      Do 118 JJJ=1,III
  118 TCL(III)=TCL(III)+TBK(JJJ)*TCK(III-JJJ+1)
      IF (III.EQ.1) GO TO 120
      TC7(III)=TCE(III)-TC7(1)*U(III)
      IF(III.EQ.2) GO TO 121
      DO 122 JJJ=2, IIII
  122 TC7(III)=TC7(III)-TC7(JJJ)*U(III-JJJ+1)
  121 TC7(III)=TC7(III)/U(1)
      GO TO 123
  120 TC7(III)+TCE(III)/U(III)
  123 CONTINUE
      TC8(III)=TC7(III)*C10
      TC9(III)=Q(III)*C11
      TDA(III)=TC8(III)+TC9(III)
      TDF(III)=0.
      DO 124 JJJ=1,III
  124 TDF(III)=TDF(III)+TDA(JJJ)*TBR(III-JJJ+1)
      TDG(III)=TCL(III)+TDF(III)
      DQ(III)=TDG(III)
      Q(III+1)=DQ(III)/FLOAT(III)
      DTHETA(III)=Q(III)
      THETA(III+1)=DTHETA(III)/FLOAT(III)
      TD2(III)=0.
      DO 125 JJJ=1,III
  125 TD2(III)=TD2(III)+U(JJJ)*TCE(III-JJJ+1)
      TEN(III)=0.
      DO 126 JJJ=1,III
  126 TEN(III)=TEN(III)+W(JJJ)*TBT(III JJJ+1)
      TEO(III)=TD2(III)-TEN(III)
      IF (III.EQ.1) GO TO 127
      TES(III)=TEO(III)-TES(1)*TBK(III)
      IF(III.EQ.2) GO TO 128
      DO 130 JJJ=2,IIII
```

```
130   TES(III)=TES(III)-TES(JJJ)*TBK(III-JJJ+1)
128   TES(III)=TES(III)/TBK(1)
      GO TO 131
127   TES(III)=TEO(III)/TBK(III)
131   CONTINUE
      DA(III)=TES(III)
      A(III+1)=DA(III)/FLOAT(III)
      IF(III.GT.1)DD(III)=0.
      DD(1)=.349066
      D(III+1)=DD(III)/FLOAT(III)
      IF (III.LT.4)GO TO 1
      III1=III+1
      ZZZZ1=0.
      ZZZZ2=0.
      DO 132 JJJ=1,III1
      ZZZZ1=ZZZZ1+U(JJJ)
      IF(JJJ.LT.III-4) GO TO 132
      ZZZZ2=ZZZZ2+ABS(U(JJJ))
132   CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      ZZZZ1=0.
      ZZZZ2=0.
      DO 133 JJJ=1,III1
      ZZZZ1=ZZZZ1+W(JJJ)
      IF(JJJ.LT.III-4) GO TO 133
      ZZZZ2=ZZZZ2+ABS(W(JJJ))
133   CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      ZZZZ1=0.
      ZZZZ2=0.
      DO 134 JJJ=1,III1
      ZZZZ1=ZZZZ1+Q(JJJ)
      IF(JJJ.LT.III-4) GO TO 134
      ZZZZ2=ZZZZ2+ABS(Q(JJJ))
134   CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      ZZZZ1=0.
      ZZZZ2=0.
      DO 135 JJJ=1,III1
      ZZZZ1=ZZZZ1+THETA(JJJ)
      IF (JJJ.LT.III-4) GO TO 135
      ZZZZ2=ZZZZ2+ABS(THETA(JJJ))
135   CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      ZZZZ1=0.
      ZZZZ2=0.
```

```
      DO 136 JJJ=1,III1
      ZZZZ1=ZZZZ1+A(JJJ)
      IF (JJJ.LT.III-4) GO TO 136
      ZZZZ2=ZZZZ2+ABS(A(JJJ))
  136 CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      ZZZZ1=0.
      ZZZZ2=0.
      DO 137 JJJ+1,III1
      ZZZZ1=ZZZZ1+D(JJJ)
      IF(JJJ.LT.III-4) GO TO 137
      ZZZZ2=ZZZZ2+ABS(D(JJJ))
  137 CONTINUE
      ZZZZ1=EPS*(ABS(ZZZZ1)+1.)
      IF(ZZZZ2.GT.ZZZZ1) GO TO 1
      GO TO 2
      1 CONTINUE
      2 CONTINUE
      DO 138 JJJ=1,NIII
      IF(ABS(TBK(JJJ)).LT.EPS) GO TO 138
      KKK=JJJ
      GO TO 140
  138 CONTINUE
  140 ZZZZ1=0.
      KKK1=KKK+1
      DO 141 JJJ=KKK1,NIII
  141 ZZZZ1=ZZZZ1+ABS(TBK(JJJ))
      IF(ZZZZ1/ABS(TBK(KKK)).GE.1)IND=IND+1
      DO 142 JJJ=1,NIII
      IF (ABS(U(JJJ)).LT.EPS) GO TO 142
      KKK=JJJ
      GO TO 143
  142 CONTINUE
  143 ZZZZ1=0.
      KKK1=KKK+1
      DO 144 JJJ=KKK1,NIII
  144 ZZZZ1=ZZZZ1+ABS(U(JJJ))
      IF(ZZZZ1/ABS(U(KKK)).GE.1)IND=IND+1
      DO 145 JJJ=1,NIII
      IF(ABS(TBK(JJJ)).LT.EPS) GO TO 145
      KKK=JJJ
      GO TO 146
  145 CONTINUE 146
      ZZZZ1=0.
      KKK1=KKK+1
      DO 147 JJJ=KKK1,NIII
  147 ZZZZ1=ZZZZ1+ABS(TBK(JJJ))
      IF(ZZZZ1/ABS(TBK(KKK)).GE.1)IND=IND+1
      NIII+NIII+1
```

```
      ZZZB(1)=U(NIII)
      DO 148 JJJ=2,NIII
148   ZZZB(JJJ)=U(NIII-JJJ+1)+T*ZZZB(JJJ-1)
      YNEW(1)=ZZZB(NIII)
      ZZZB(1)=W(NIII)
      DO 150 JJJ=2,NIII
150   ZZZB(JJJ)=W(NIII-JJJ+1)+(T*ZZZB(JJJ-1)
      YNEW(2)=ZZZB(NIII)
      ZZZB91)=Q(NIII)
      DO 151 JJJ=2,NIII
151   ZZZB(JJJ)=Q(NIII-JJJ+1)+T*ZZZB(JJJ-1)
      YNEW(3)+ZZZB(NIII)
      ZZZB(1)=THETA(NIII)
      DO 152 JJJ=2, NIII
152   ZZZB(JJJ)=THETA(NIII-JJJ+1)+T*ZZZB(JJJ-1)
      YNEW(4)=ZZZB(NIII)
      ZZZB(1)=A(NIII)
      DO 153 JJJ=2,NIII
153   ZZZB(JJJ)=A(NIII-JJJ+1)+T*ZZZB(JJJ-1)
      YNEW(5)=ZZZB(NIII)
      ZZZB(1)=D(NIII)
      DO 154 JJJ=2,NIII
154   ZZZB(JJJ)=D(NIII-JJJ+1)+T*ZZZB(JJJ-1)
      YNEW(6)=ZZZB(NIII)
      RETURN
      END
```

Since SERIES only accepts input with derivatives on the left of the =, and a limited subset of FORTRAN functions of derivative free variables on the right, the equations for A, V^2, V, and D must be changed. For example, since A = ATAN(W/U), the differential equation A' = (UW' - WU')/(U^2) could be added, but then the entire expression for W' and U' would have to replace these values (the actual output SOL of SERIES will have these common subexpressions eliminated; the restriction was imposed to remove the problem of sorting the input to SERIES. Sorting often is the cause of errors in simulation languages such as ACSL and DARE.) Further, the original formulation includes the approximation

$$A' = W'/U$$

so this slightly simpler expression can be used to change from an algebraic to a differential equation. Since A(0) is a function of W(0) and U(0) and SERIES cannot handle algebraic constraints, the output of SERIES in Figure 2.2 has been modified by the lines Y(5) = ATAN2(Y(2)/Y(1)) in DIFFUN and A(1)=YO(5)=ATAN2(YO(2)/YO(1)) in SOL to provide this algebraic constraint.

Since the stepsize H usually used in a real time simulation package is h = 0.032 and we are limited to k=10 steps, stabilator deflections occurring after 0.32 sec will never occur, so the proper equation for D is

$$D' = 2.*.174533,$$

$$D(0) = 0$$

V^2 and V are replaced by (U*U+W*W) and SQRT(U*U+W*W) where they occur. The resulting system of equations becomes

$$U' = -C1*SIN\ \phi - W*Q+(C1+C2*A)(U*U+W*W) + C3*Q*SQRT(U*U+W*W)$$

$$W' = C1*COS\ \phi + U*Q+(C4+C5*D+C6*A)(U*U+W*W) + C7*Q*SQRT(U*U+W*W)$$

$$Q' = (U*U+W*W)\ (C8+C9*(A+D)) + C10*(C1*COS\ \phi + /U*Q+(C4+C5*D+C6*A)(U*U+W*W))/U+11*Q*SQRT(U*U+W*W)/U$$

$$\phi' = Q$$

$$D' = 0.349066$$

$$A' = (C1*COS\ \phi + (C4+C5*D+C6*A)(U*U+W*W) + C7*Q*SQRT(U*U+W*W))/U$$

Initial conditions at t=0 are S = 0, A = ATAN(W,U), ϕ = 5.3^0 = 0.093 radians, Q = 0. Typical initial conditions for U and W are U = 660.18167, W = 5.74626.

Fig 2.3 shows an extensive terminal session to determine initial conditions and best numerical method to be used with these equations for explicit multistep or Runge-Kutta methods for real-time simulation. Sequence A shows initialization, choice of approximate equilibrium y^* = (271,360) from an initial guess of U=660.18167 and W=5.74626. This corresponds to the plane going slow and climbing, and is inside the Liapunov stability region. Computation of the Liapunov norm matrix shows that only b_{11} is not almost zero, so this means tha W is not sensitive to changes in U or W, and that U is not sensitive to changes in W, either. Thus, $\langle y,y \rangle = U^2$. Both D' and D'' are displayed, but since D'' is

proportional to D' only D' will be computed in the following.

At B, numerical analysis is chosen, and D' for the AB1 (Euler) method cannot be plotted. Neither of the two Runge-Kutta methods work either, since their stability regions are too small, and incidentally identical.

At C, the number of steps is increased to 2, and the exact region D' is computed for k=2. As expected, this region is somewhat smaller, at least in the W coordinate. At D, the 2-step Adams predictor is analyzed, and at E, the shifted trapezoidal rule. Interestingly, this last has a much larger D' than either AB2 or the exact D', but this is not an advantage since we want a close match, not the largest region.

At F, a study of the analytic D' for H = 1/24, and 0.016, shows that, as expected, the larger H is, the smaller D' is.

We can now conclude that a two-step method is needed, and at G we look at other subsystems to pick which one. The subsystem 2,3 again is only sensitive to its second variable Q, but the 1,3 subsystem shows a definite dependence between U, the horizontal velocity, and Q, rate of change of pitch.

At H, we investigate AB2 and shifted trapezoid, and conclude that AB2 most closely follows the exact case, since shifted trapezoid slows down even when the nose is dropping fast ($Q<0$) to a greater extent than is the real case.

A

```
? HELP
AVAILABLE COMMANDS ARE:
HELP
GO
NPOINTS - SET NUMBER OF POINTS IN THE
     DISPLAY (DEFAULT 13)
NSTEPS - SET NUMBE ROF STEPS IN MULTISTEP
     METHOD AND FOR G (DEFAULT 1)
CENTER - SET CENTER FOR SEARCH OF DOMAIN
     (CAN BE AUTOMATED) (DEFAULT, VALUES
     SET BY INIT. OR ZERO)
NORM - COMPUTE THE LIAPUNOV NORM MATRIX BY
     DIFFERENCING
NEQ - (RE)SET NUMBER OF EQUATIONS
     (DEFAULT 2, MAX 20)
STEPSIZE - CHANGE STEPSIZE H
     (SET ON FIRST GO CALL)
SYSTEM - SPECIFY VARIABLES OF INTEREST
     (DEFAULT 1,2)
INIT - INITIALIZE DEPENDENT VARIABLES
     (DEFAULT 1,2)
INIT - INITIALIZE DEPENDENT VARIABLES
     (DEFAULT IS ZERO)
TYPE - EXACT (DEFAULT) OR NUMERICAL
     ANALYSIS
SYMMETRIC - SPECIFY THAT THE PROBLEM
     IS SYMMETRIC IN ONE VARIABLE
PARAM - LIST CURRENT VALUES OF PROGRAM
     VARIABLES
REPEAT - GRAPH LAST REGION AGAIN
     (CHANGE LIMITS END)
```



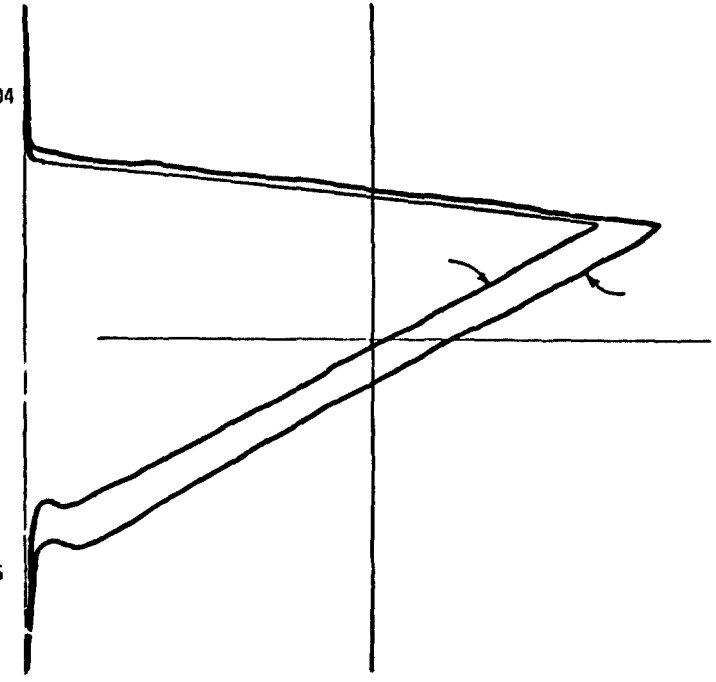
```
? NEQ
  HOW MANY DEPENDENT VARIABLES
? 6
? INIT
  GIVE INITIAL CONDITIONS FOR:
  Y(              1)
? 660.
  Y(              2)
? 5.7
  Y(              3)
? 0.
  Y(              4)
? .093
  Y(              5)
. .008
  Y(              6)
? 0.
? CENTER
  SET Y(          1), Y(        2)
? 660.18167,5.74626
  IMPROVE CENTER BY MUELLER'S METHOD
? Y
  MAXIMUM NUMBER OF ITERATIONS EXCEEDED.
  VALUE FOR ROOT(      1) MAY NOT BE ACCURATE
    NEW CENTER IS      (271.4240817345,358.8267985179)
```

Figure 2.3

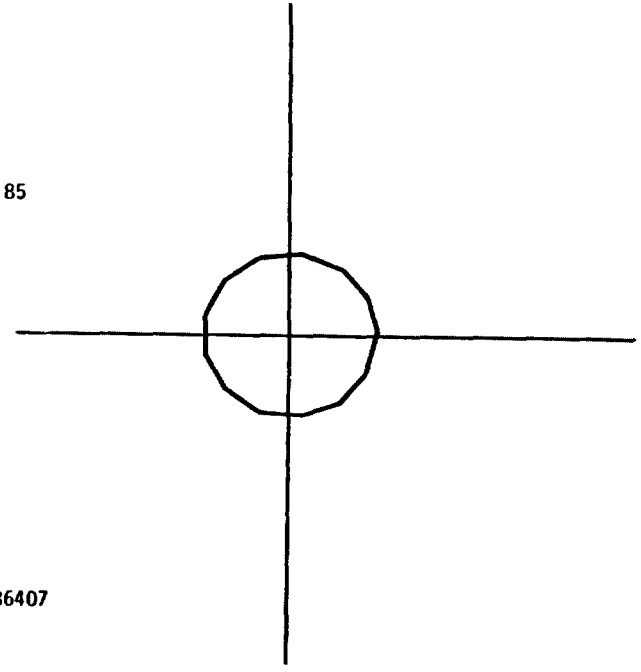
Terminal session for STAN.

```
?  GO
   INPUT H
?  .032
   2 BY 2 NORM MATRIX IS                1.  .00738800055952   .007388000055952
   .003913261862763
   at        (782., 360.)  1  VARIABLE HAVE RADIUS OF CONVERGENCE
   NOT EQUAL TO (-1, 1)
   DISPLAY DOMAIN OF ATTRACTION
?  YES
   COMPUTED LIMITS ARE:
   -20950.90436166  27706.  -22039.24910977   14415.5478640?
   SET DISPLAY LIMITS (L.R.BOT.TOP)
?  0., 30869.,  -1469., 1009.


DOMAIN OF ATTRACTION
   DISPLAY STABILITY REGION
?  YES
   COMPUTED LIMITS ARE:
   -18810.16274002  24938.28632812  -19779.63735876
   12997.6395688
   CHANGE DISPLAY LIMITS
?  NO
```

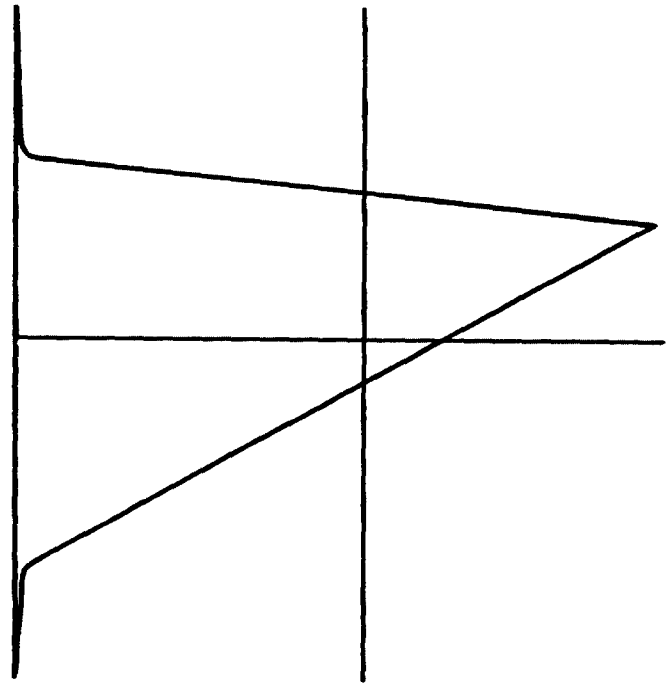
```
     B
STABILITY REGION
?  PARAM
EXACT ANALYSIS
NEQ = 6 STEPSIZE=  .320E-01NUMBER STEPS
NUMBER OF DISPLAY POINTS  13 SUBSYSTEM= 1  2
DISPLAY LIMITS=  0.     .300E+05  -.100E+04     .100E+04
CENTER = 270.   360.
INITIAL VALUES ARE:
270.  360.  0.  .093  -.263451896604  0.
?  TYPE
EXACT OR NUMERICAL ANALYSIS
?  NUMERICAL
?  GO
CHOOSE NUMERICAL METHOD
?  AB
   PAUSE
G  STABILITY MATRIX?
1.
UNABLE TO FIND BOUNDARY ALONG Y(1)-AXIS
?
GO
CHOOSE NUMERICAL METHOD
?  RK2
DISPLAY DOMAIN OF ATTRACTION
?  YES
COMPUTED LIMITS ARE:
269.7572645456  270.25 359.7518227815  360.2481772185
CHANGE DISPLAY LIMITS
?  YES
SET DISPLAY LIMITS (L.R.BOT.TOP)
?  269., 271., 359., 361.
```



```
        C

DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  GO
CHOOSE NUMERICAL METHOD
? RK4
DISPLAY DOMAIN OF ATTRACTION
? YES
COMPUTED LIMITS ARE:
269.7572645456  270.25  359.7512227815  360.2481772185
CHANGE DISPLAY LIMITS
?  NO

DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  TYPE
EXACT OR NUMERICAL ANALYSIS
?  EXACT
?  NSTEPS
INPUT NUMBER OF STEPS (1 TO 10)
?  2
?  GO

DISPLAY DOMAIN OF ATTRACTION
?  YES
COMPUTED LIMITS ARE:
-20950.90436166  27706.  -22039.24910977  14415.54786407
CHANGE DISPLAY LIMITS
?  YES
SET DISPLAY LIMITS (L.R.BOT.TOP)
?0.,30000.,-1000., 1000.
```



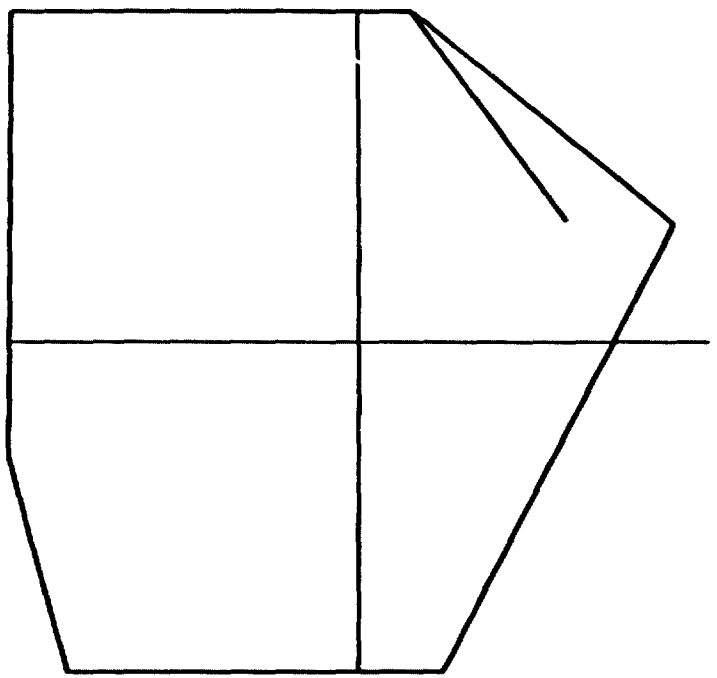
```
          D

DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  TYPE
EXACT OR NUMERICAL ANALYSIS
?  NUMERICAL
?  GO
CHOOSE NUMERICAL METHOD
?  AB
G  STABILITY MATRIX:
0.
0.  1.
DISPLAY DOMAIN OF ATTRACTION
?  YES
COMPUTED LIMITS ARE:
-19785.77418075  37306.  -20540.49825824 21639.70742517
CHANGE DISPLAY LIMITS
?  YES
SET DISPLAY LIMITS (L,R,BOT,TOP)
? 0.,40000., -1000.,1000.
```



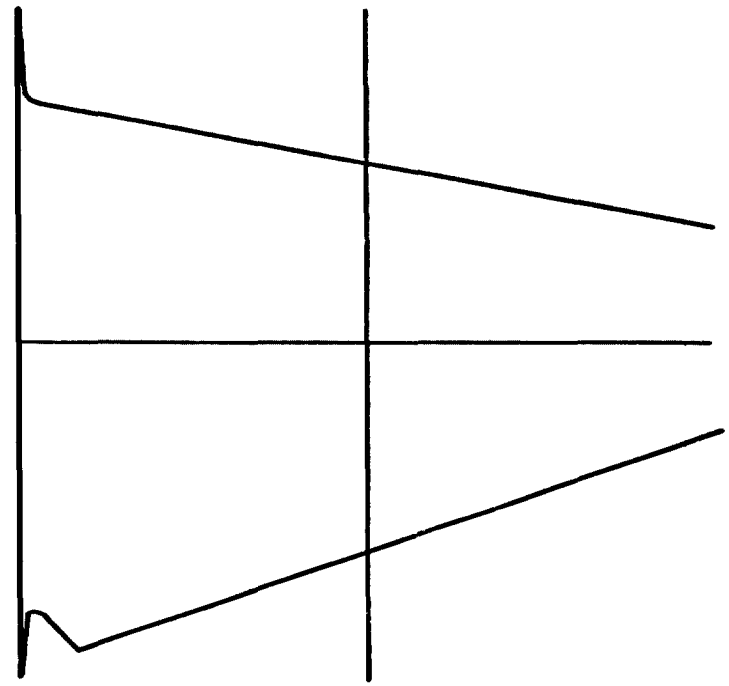
```
          E

DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  GO
CHOOSE NUMERICAL METHOD
?  PC
INPUT PREDICTOR FORMULA A(0)*Y(N)+A(1)*Y(N-1)
... A(K)*Y(N-K).B(0)=J.B(1)*Y'(N-1)...B(K)*Y'(N-K)
? -1.,1.,0.,0.,.5,.5
HOW MANY CORRECTOR ITERATIONS
? 0
THE LARGEST STABLE CIRCLE HAS DIAMETER 1.
G STABILITY MATRIX:
.3333333333333
-.3333333333333   1.

DISPLAY DOMAIN OF ATTRACTION
? YES
COMPUTED LIMITS ARE:
-23688.9602868  47506.  -29258.46196759  27000.33534529
CHANGE DISPLAY LIMITS
.  YES
SET DISPLAY LIMITS(L,R,BOT,TOP)
!  0.,50000., -1000.,1000.
```

F

```
DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  NSTEPS
INPUT NUMBER OF STEPS (1 TO 10)
?  1
?  TYPE
EXACT OR NUMERICAL ANALYSIS
?  EXACT
?  STEPSIZE
INPUT H
?  .041666666666
?  GO
DISPLAY
?  YES
COMPUTED LIMITS ARE:
-16115.61411088 21106.  -16934.06042471   4269.460691802
CHANGE DISPLAY LIMITS
?  YES
SET DISPLAY LIMITS (L,R,BOT,TOP)
?  0., 3000., -1000., 1000.
```

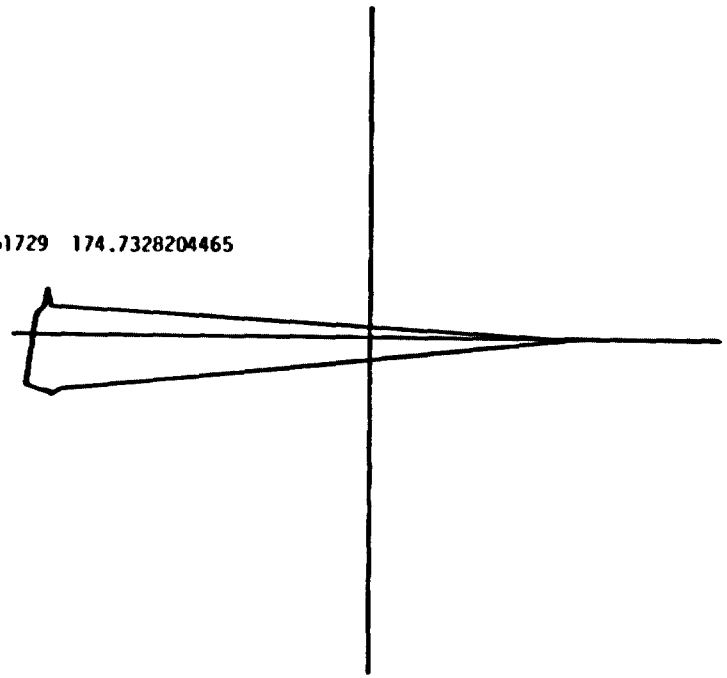


```
        G

DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
?  NO
?  NSTEPS
INPUT NUMBER OF STEPS (1 TO 10)
?  2
?  STEPSIZE
INPUT H
?  .032
?  SYSTEM
WHICH 2 DEP. VARIABLES WILL VARY
? 2.3
2 BY 2 NORM MATRIX IS      .0001608381708602   .01237868633224
.01237868633224   1.
?  INIT
GIVE INITIAL CONDITIONS FOR:
Y(        1)
?660.18167
Y(        2)
?5.74626
Y(        3)
? 0.
Y(        4)
? .093
Y(        5)
?.008
Y(        6)
? 0.
?  SYSTEM
WHICH 2 DEP. VARIABLES WILL VARY
?  1.3
2 BY 2 NORM MATRIX IS         .6609941144222     -.8130184570587
-.8130184570587   1.

DISPLAY DOMAIN OF ATTRACTION
?  YES
COMPUTED LIMITS ARE:
25.67119231207  15496.18167  - 184.4512589434   148.5030431346
CHANGE DISPLAY LIMITS
?  YES
SET DISPLAY LIMITS(L,R,BOT,TOP)
?  0.,20000.,-1000.,1000.
```



H

```
DOMAIN OF ATTRACTION
DISPLAY STABILITY REGION
? NO
? TYPE
EXACT OR NUMERICAL ANALYSIS
? NUMERICAL
? GO
CHOOSE NUMERICAL METHOD
? AB
G STABILITY MATRIX:
0.
0. 1.
DISPLAY DOMAIN OF ATTRACTION
? YES
COMPUTED LIMITS ARE:
335.1588966166  41896.18167 -173.0709351729  174.7328204465
CHANGE DISPLAY LIMITS
? NO
```



3. NUMERICAL SOLUTIONS

The analytic solution to the problem

$$y' = f(y,t) \qquad (1)$$
$$y(0) = y_0$$

is attacked numerically for a restricted set of input functions f using power series methods. That is we represent each of the y(i), i=1, ..., n for n <= 20 by a power series expanded about t=0, and given t and y_0 the value of y(i) can be calculated.

The general method of solution is to use recurrence relations to calculate successive terms of the power series. Algorithms for addition, subtraction, and multiplication of power series are well known. Recurrence relations for division and exponentiation can be derived which calculate the n^{th} term of the result using only the first n terms of the operand or operands and the first n-1 terms of the result [Knuth]. The same type of recurrence relations can also be derived for sin, cosine, natural logarithhm, and exponential of a power series [Gibbons] along with many other functions. The computation of the first n terms of any of the operations or functions mentioned above can be accomplished in $0(n^2)$ or less multiplications and divisions. In addition to this restricted set of inputs, any function which can be defined as an initial value problem with initial conditions at t=0 using the given set of functions and

operations can be added to the set of equations and consequently used as part of the input.

As an example of a recurrence relation, let

$$A = \sum_{i=0}^{\infty} a_i\, t^{i-1},\quad B = \sum_{i=0}^{\infty} b_i\, t^{i-1}$$

and consider

$$B = EXP\ (A) \tag{2}$$

differentiation of both sides with respect to t yields

$$B' = EXP\ (A) * A'$$

using (2)

$$B' = B * A'$$

$$\sum_{n=1}^{\infty} nb_n\, t^{n-1} = \sum_{n=1}^{\infty} b_n\, t^{n-1} * \sum_{n=1}^{\infty} i\, a_i\, t^{i-1}$$

$$\sum_{n=1}^{\infty} nb_n\, t^{n-1} = \sum_{n=1}^{\infty} \left(\sum_{i=0}^{n-1} b_i\, (n-i)\, a_{n-i}\right) t^{n-1}$$

equating the coefficients of t^{n-1}

$$nb_n = na_nb_0 + \ldots + a_1\, b_{n-1},\ n \geq 1 \tag{3}$$

$$b_0 = EXP\ (a_0)$$

giving us our recurrence relation.

The general form of the input is:

/* constant1 = value1, ..., constantn = valuen */

```
variable1.= equation1;
        .
        .                                    (4)
        .
variablem. = equationm;;
```

Constant1, ..., constantn are unique variable names and value1, ..., valuen are numbers. Derivatives with respect to time, represented by variable., can appear explicitly only on the left-hand side of the equation. If derivatives were allowed to appear on the right-hand side the equations would have to be ordered; however, by allowing derivatives only on the left-hand side no ordering is necessary by either the program or user.

The input is read as a character string from the data file EQIN and assembled into lexical items using a scanner. These lexical items are then sent to a recursive descent parser which enters them into a symbol table, and generates quadruples as the output of each operation, i.e. a*b would become *a b t1 where the result of a*b would be assigned to t1. Finally the codetable is examined to eliminate common subexpressions. It is from this optimized code table that the recurrence relations are generated, [Gries], [Barton et al].

The output from the program consists of two FORTRAN subroutines. The first is subroutine DIFFUN which is used in calculation of the numerical solution by other parts of this package and is described in more detail in Chapters 2

and 4. The second output is subroutine SOL, the subroutine which attempts to solve the system of equations to stated accuracy. The constant definition is transformed into a data statement and the quadruples generated by the parser are used to generate the recurrence relations. Enough terms (up to 20) are calculated by SOL so that all the input variables satisfy:

$$\sum_{i=n-4}^{n} \text{ABS}(Y(i)) \leq \text{EPS}*(\text{ABS}(\sum_{i=0}^{n} Y(i) + 1.))$$

as used in [Gibbons]. In general this condition states that the last five terms are negligible compared to the sum of the series. Obviously any alternating series will be convergent using this criteria and for an arbitrary power series this test provides some assurance that the rest of the terms of the series can be safely neglected since values of t only between -1 and 1 are being considered.

SERIES USER MANUAL

The purpose of this part of the software package is to generate the FORTRAN subroutines necessary to solve the input system of first order ordinary differential equations both analytically and numerically. As stated in (4) the general form of the input is:

```
/*constant1 = value1, ..., constantn = valuen */
        variable1. = equation1;
                .
                .
                .
        variablem. = equationm;;.
```

The first part of the input is the constant definition section. Any number of constants can be defined, and these will appear in data statements in both subroutines SOL and DIFFUN. The second part of the input is the specification of the differential equations themselves. Up to 20 first order equations can be input to the package at any one time. Note that at the end of each equation a semicolon must appear except for the last equation which must be followed by two semicolons. The file EQIN is used as the input data file for the program.

Variable names are restricted to 9 or less alphanumeric characters, the first of which must be alphabetic. Constant values are restricted to 20 or fewer characters. The following variables names are restricted; III, JJJ, KKK, EPS, IND, YO, YNEW, ZZZB, ZZZZ1, ZZZZ2, IIII, NIII, KKK1, and

3-letter variable name beginning with T whose second letter is not A, and if a variable is used as a time derivative, for example X., then both X and DX are also reserved. The name T is reserved for the independent variable.

The constant definition section consists of predefined user constants. The names constant1, ..., constantn are unique variable names, and value1, ..., valuen are numbers. These numbers can be integers, simple real numbers, or exponential real numbers. Their form is identical to FORTRAN constants, for example: 1, 10.3, -.3, -1.976E-5, and .43795E-15 are all valid. Recall that the constant definition becomes a data statement in the FORTRAN subroutines so that naming conventions for FORTRAN variables must be followed.

The differential equation specification section consists of a series of first order ordinary differential equations in the independent variable T, each equation of the form:

variableI. = equationI;. (5)

All variables used as derivatives must be real, i.e., begin with A..H or O..Z. Only derivatives may appear on the left-hand side of the equations; derivatives may not appear explicitly on the right-hand side. The right-hand side may contain constants from the constant definition, numbers, variables defined by differentiation on the left hand side, operators and predefined user functions listed below, the

independent variable t, parenthesis, and the terminating semicolon. Note that the syntax of the right-hand side will be identical to FORTRAN assignment statements.

The standard operators and 4 predefined user functions are available in this package. These operators are +, -, /, *, and **. Operands for these operators may be constants, expressions, the variable t, or series (variables defined by differentiation) with the following exceptions:

1) t, when appearing by itself may be raised only to integer powers

2) constants may not be explicitly divided by t or powers of t

3) series may not be explicitly divided by t raised to powers greater than 1.

The four predefined user functions are SIN, COS, LOG (natural logarithm), and EXP. The arguments for these functions must be completely enclosed in parenthesis and may be constants, t, expressions or series variables with one exception; LOG(t) or LOG(t**power) are not allowed.

The reason for these restrictions is inherent in the method used for solution of the problem itself. In the first case LOG(t) does not have a power series defined about t=o so it along with LOG(t**power) are not allowed; however, LOG(1+t) or LOG(constant + t**power), e.g., are allowed. The reason that t may be explicitly raised only to integer

powers is that the program considers each term to be a constant or a power series. If, however the term t**1/2*SIN(t)**3/2 appears one can simply rewrite it as(t*SIN(t))**1/2*SIN(t) which is a legal expression. In most cases this problem can be avoided by rewriting the input. Since division by power series with zero constant term is not defined series may not be divided explicitly by t raised to a power. However, division by t by itself is defined so that x/t**2 could be rewritten as (x/t)/t, or x**3/2/t**1/2 as (x/t)**1/2**. Note that COS(t)/(1 + t**2) or x/(8.7 + t) are no problem. Finally, the reason that constants may not be divided by t is that power series do not contain terms in inverse powers of t. To avoid this problem input can simply be rewritten, for example (const/t)*EXP(x) would become const*(EXP(x)/t) or (3/t**2)*COS(t) would become 3.*(COS(t)/t)/t.

Other than these restrictions on inputs stated above two other problems exist with respect to power series operations. The first is the problem encountered with division by powers of t; a power series used as a divisor must have a nonzero constant term. The second problem occurs when raising a power series to a power other than 2 in which case the series must also have a nonzero constant term. These problems will show up when subroutine SOL is executed and a division by zero error message will be generated.

The above problems occur in only a very limited number of equations; these are not in the author's opinion major stumbling blocks for use of this part of the software package. Although the user may be inconvenienced by having to rewrite some of his input due to these restrictions, the generation of SOL and DIFFUN by this program are still a great savings in terms of time and effort by the user.

EXAMPLES OF INPUTS

a) X' = -X-2Y+X**2*SIN(T)

Y' = 5X -(T+3)/(T+4)*Y

X(0) = Y(0) = 1

input would be

```
X. = -X-2.*y+X**2*SIN(T);
Y. = 5.*X-(T+3)/(T+4.)*Y;;
YO(1) 1., YO(2) = 1.
```

b) X' = Y

Y' = (5*EXP(T/TS) - 1 - 3/2*Y**2)/S + (A*Y+D)/X**2+C/X**(3*G+1)

G = 1.4, A = 4.0E-2, TS = 29, D = .1456,

C = 1+D

X(0) = 1, Y(0) = 0

input would be

```
/* G = 1.4, A=4.0E2, TS=29., D=.1456, C=1.1456*/
X.=Y;
Y.=(5.*EXP(-T/TS) - 1.  -(3./2.)*Y**2)/X -
(A*Y+D)/X**2 + C/X**(3.*G+1.);;
YO(1)=1., YO (2) = 0.
```

c) D=.349066T

A=ATAN(W/U)

THETA'=Q

U'=C*SIN(THETA) -W*Q + (C1+C2*A)*(U**2+W**2)

+ C3*Q*SQRT(U**2+W**2)

W' = C*COS(THETA) + U*Q + (C4+C5*D+C6*A)*(U**2

+W**2) + C7*Q*SQRT(U**2+W**2)

```
Q' = (U**2+W**2) * (C8+C9*(A+D)) + (C10*W'/U
+ C11*Q) *SQRT (U**2*W**2)
C = 32.16, C1 =  .5018521E-9, C1=0.1192125E-5,
C3=.42110675E-3, C4=.14598254E-3, C5=0.46345531E-5,
C6=.45006065E-5, C7=-.69718949E-3, C8=.587215E-8,
C9=.73872E-6, C10=0.3800645E-5, C11=.7422436E2,
U(0)=660.16187,  W(0)=5.74626,  Q(0)=0,  THETA(0)=5.3.
```

Notice that W' appears on the right-hand side, ATAN is not a user function, and A and D are not in the form of derivatives; however, we can substitute

```
D.=.349066, D(0)=0.
A.=(U*W.W*U.)/(U**2+W**2), A(0)=ATAN(W(0)/U(0))
```

the input becomes

```
/* C=32.16, C1=.5018521E-9, C2=0.1192125E-5,
C3=.42110675E-3, C4=.14598254E-3, C5=0.46345531E-5,
C6=-.45006065E-5, C7=.69718949E3, C8=.587215E8,
C9=.73872E=6, C100.3800645E-5, C11=.7422436E-2 */
U.=C*SIN(THETA) - W*Q + (C1+C2*A)*(U**2+W**2)
+ C3*Q*(U**2+W**2)**.5;
W.=C*COS(THETA) + U*Q + (C4+C5*D+C6*A)*(U**2+    W**2)
+ C7*Q*(U**2+W**2)**.5;
Q.=(U**2+W**2)*(C4+C5*D+C6*A) + (C10*(C*COS(THETA)
+ U*Q + (C4+C5*D+C6*A)*(U**2+W**2)
 + C7*Q*(U**2+W**2) **.5)/U+C11*Q)*(U**2+W**2)**.5;
THETA.=Q;
```

```
A.=(U*(C*COS(THETA)  +  U*Q  +  C4+C5*D+C6*A)  *  (U**2  +
W**2) + C7*Q*(U**2+W**2)**.5) - W*(C*SIN(THETA)
 - W*Q + (C1+C2*A)*(U**2+W**2) + C3*Q*(U**2+W**2)
**.5))/(U**2+W**2);
D.=.349066;;.
YO(1)=660.18167, YO(2)=5.74626, YO(3)=0.,
YO(4)=5.3, YO(5)=ATAN(W(0)/U(0)), YO(6)=0.
```

OUTPUTS

There are two fortran subroutines output. The first is subroutine DIFFUN(T,Y,DY) which appears in source form on file DIFFUN. The variables Y and DY are 20 element arrays. The second is subroutine SOL(T,YO,YNEW,IND) which appears on file SOL. The variables YO and YNEW are also 20 element arrays. One important fact to notice is that the ordering of the input equations determines the coefficients in Y, DY, YO, YNEW, beginning at 1 corresponding to each equation, for example if:

```
X.=Y;
Y.=-X;;
```

were entered X and DX would correspond to Y(1), DY(1), YO(1), and YNEW(1); Y and DY would correspond to the subscript 2 for both input and output values from the subroutines.

The purpose of subroutine SOL is to solve the system of equations analytically so that the first 10 significant digits are correct. The initial values at T=0 for the equations in order of occurrance are placed in YO and along with a value of T are input to SOL. The solution to the equations at this specific T are output in order of occurrence in YNEW. The value of IND at output is zero if the radius of convergence of all series is (-1,1) and greater than zero if the radius of convergence of any series is less than (-1,1). Subroutine DIFFUN is explained in chapters 2 and 4.

NOS CONTROL CARD EXAMPLES

```
BATCH JOB FROM CARDS
JOB.
USER.
CHARGE.
ATTACH,SERIES/UN=USERNAM.
COPYBR,INPUT,EQIN.
REWIND,EQIN.
SERIES,F=105000.
REPLACE,SOL.
REPLACE,DIFFUN.
*EOR
DATA ON CARDS
*EOF
```

INTERACTIVE SUBMISSION AFTER CREATION OF FILE EQIN

```
ATTACH,SERIES/UN=USERNAM
GET,EQIN
SERIES,F=105000
REPLACE,SOL
REPLACE,DIFFUN
*EOF
```

If running interactively using the SUBMIT command add the JOB, USER, and CHARGE cards to the above control card sequence.

ERRORS

When a compile time error occurs due to a mistake in the user input a dump is written by the program onto file OUTPUT. This dump contains the following information:

a) a general statement such as ILLEGAL OPERATOR to give the user some idea of what type of error has occurred

b) a listing of the input characters up to the point where the error was detected

c) a dump of the symbol table that shows which variable and constants have already been ·ered

d) a listing of the codetable which shows how far into the equations the parser has progressed.

The written messages in a) are neither completely descriptive nor always correct in the description given. If the parser becomes confused the detection of an error may occur well after the actual error has been made. If, for example, an opening parenthesis is missing in the expression the input may still remain legal until the extra closing parenthesis is encountered. The best way to find a compile time error is to study the listing b) near the end of the character listing. The symbol table and code table can also be helpful in error detection. When looking at the listing of the symbol table all the variable names and operators that have been encountered or generated will be listed. Next to each name is a number in parenthesis which tells the type of the variable:

0) user function

1) vector or series variable

2) constant

3) operator or delimeter

4) semicolon

5) differentiated variable

6) independent variable t.

Variables of type 1 and 5 should be declared as arrays in subroutine SOL. The code table is listed as a series of quadruples describing the operations which are to be performed in order of occurrance. By cross-referencing between the code and symbol tables many logical errors will be easier to find. Note that only the first 10 characters of numbers will be listed. Finally if the message RUNTIME STACK OVERFLOW appears in the user dayfile simply increase the F parameter on the SERIES control card.

LIMITED EXTENSIONS

The biggest thing to remember is that this part of the package is only a compiler to generate the FORTRAN code. All the variables need not be defined in the input at compile time when the subroutines are generated. If one has any function which does not depend on the differentiated variables (VAR.) one could add it to subroutine SOL or DIFFUN after creation. This could be done using a statement function in the subroutines themselves or by use of a function subprogram used in conjunction with SOL and/or DIFFUN. This extension is especially useful for functions of t. As a simple example let us redo the variable D in the previous example C. Recall that:

```
D = .349066T
```

one could simple define the statement function,

```
H(R) = .349066*R
```

and set D=H(T) at the beginning of the program. If any of the functions arguments are the dependent variables then one must define the functions as an initial value problem with initial conditions at t=0 such as A=ATAN(W/U) in example C.

4. USER'S INSTRUCTIONS FOR CLMP

The user of the closed Loop Modelling package (CLMP) is faced with the following system of equations:

$$y_i(t) = f_i(t,y(t),u(t)),i=1,2,\ldots,n \quad (1)$$

where u(t) is an input function of three possible types:

(i) Damped sine wave

(ii) Spike function

(iii) Step function

In order to use CLMP, the user must supply the following information:

(i) A system of equations

(ii) An interval $[t_0, t_n]$ on which y(t) is to be determined

(iii) Initial values for $y(t_0)$

(iv) Parameters of u(t)

(v) A fixed-step-size integration routine for solving differential equations.

A System of Equations

The system of equations (1) will be stored on file DIFFUN by the user, as outlined in section 3.

The system will be solved for y(t) for t in (t_0, t_n), where t_0 and t_n must be specified by the user. It is also essential that the values of y(t) at the point $t = t_0$ be specified.

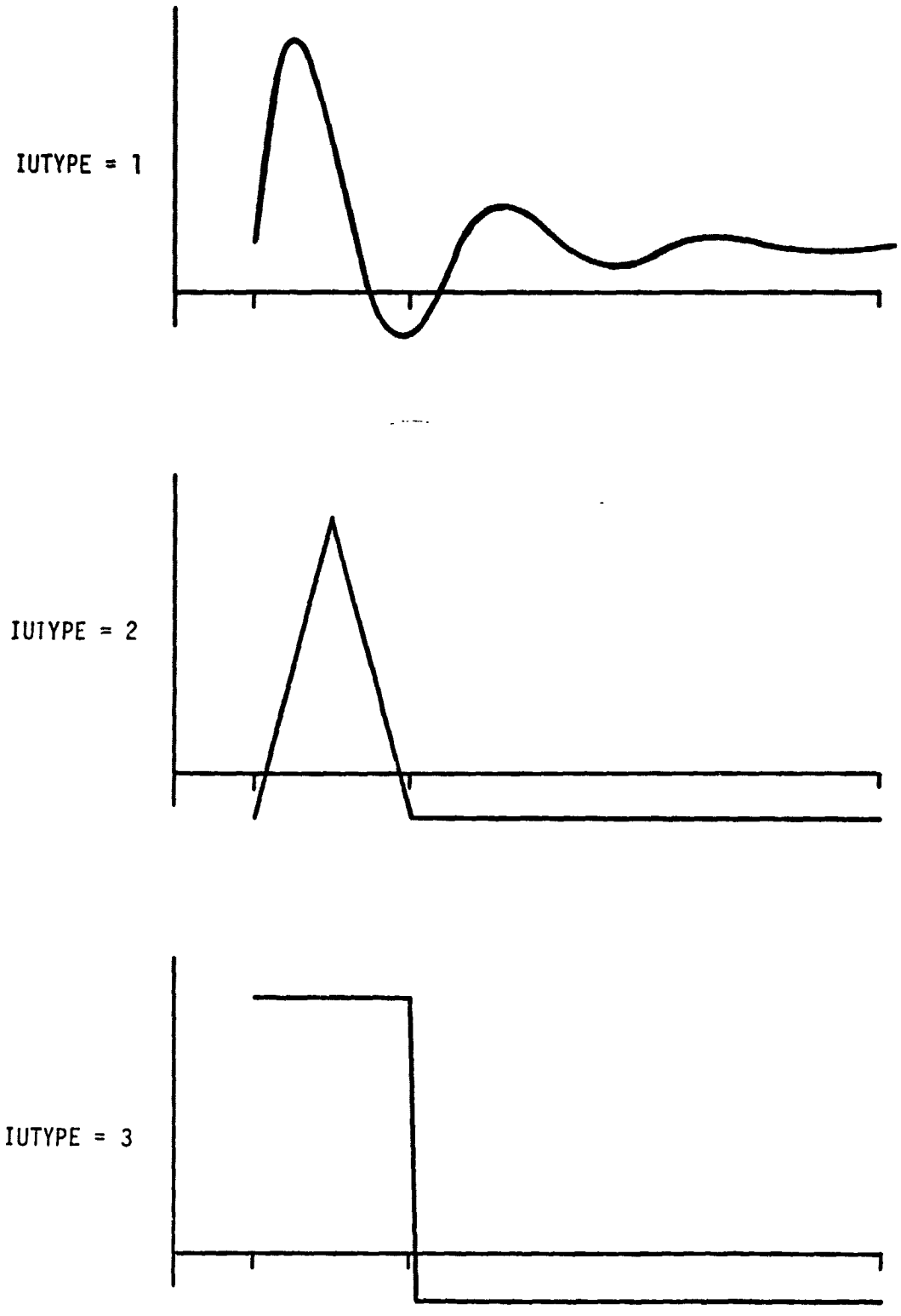


Figure 4.1 Graphs of various u(t) input functions.

Should the user wish to write his own DIFFUN routine, rather than use the SERIES package for this, he may do so. The format to use is

```
SUBROUTINE DIFFUN (T,Y,DY)
DIMENSION Y(20), DY(20)
```

The routine simply computes DY(I) = f_i (t,y(t),u(t)), as in (1). This routine should be compiled and stored on permanent file DIFFUN, as follows:

```
FTN.
SAVE,LGO=DIFFUN.
7/8/9
      SUBROUTINE DIFFUN
6/7/8/9
```

Parameters of u(t)

The first parameter is IUTYPE. If IUTYPE = 1, then u(t) is a damped sine wave with equation

$$u(t) = e^{-\alpha t}\sin(\beta t+\phi)$$

If IUTYPE = 2, then u(t) is a spike function, and if IUTYPE = 3, u(t) is a step function (see figure 4.1).

Other parameters which must be specified are a, b, UMIN, and UMAX, which simply indicate the domain and range of the function u(t). It should be noted that for IUTYPE = 2 and IUTYPE = 3, the function u(t) is different from UMIN

only on the first one-fourth of the interval [a,b]. (i.e., [a,a+(b-a)/4]).

Integration Routine

CLMP provides the user with 7 standard fixed-step-size integration routines to choose from. They are:

(1) Euler's method

(2) Improved Euler (2-pt. Runge-Kutta)

(3) 2-pt. Adams-Bashforth

(4) 3-pt. Runge-Kutta

(5) 4-pt. Runge-Kutta

(6) Runge-Kutta-Merson

(7) 4-pt. Adams-Moulton.

It should be noted that all of these methods are self-starting, except for (3) and (7), which use Improved Euler and 4-pt. Runge-Kutta to start, respectively.

Should the user desire to test his own integration routine, instead of one of the above, he may do so. Such a routine, however, must be called NEXTPT and have the calling sequence:

```
SUBROUTINE NEXTPT(T,Y,N,STEP,KTR)
```

NEXTPT, when given the values for y(t), should compute y(t+h), and return this value in y. Here, h is a fixed-step-size specified by the variable STEP. CLMP requires that STEP $\leq (t_n - t_0)/32$. The array Y is dimensioned as Y(20), and $y_i(t)$ is contained in Y(I). The variable N

denotes the number of equations in the system, and cannot exceed 20. The variable KTR may be used for whatever purpose is necessary. When the first call to NEXTPT is made from CLMP, the value KTR=0 is transmitted and remains at KTR=0 until changed in NEXTPT. No other information is passed through KTR.

A sample problem and terminal session follow:

Sample Problem

The following equations govern the longitudinal motion of a jet fighter [Steinmetz et al.]:

$$\dot{u} = -g \sin\theta - wq + \frac{\rho V^2 S}{2m}\left[(C_X)_{\alpha_T,\delta_T} + C_{X_\alpha}(\alpha-\alpha_T) + C_{X_q}\frac{q\bar{c}}{2V}\right]$$

$$\dot{w} = g\cos\theta + uq + \frac{\rho V^2 S\bar{c}}{2m}\left[(C_Z)_{\alpha_T,\delta_T} + C_{Z_\alpha}(\alpha-\alpha_T) + C_{Z_q}\frac{q\bar{c}}{2V} + (C_{Z_\delta}(\delta-\delta_T)\right]$$

$$\dot{q} = \frac{\rho V^2 S\bar{c}}{2I_Y}\left[(C_m)_{\alpha_T,\delta_T} + C_{m_\alpha}(\alpha-\alpha_T) + C_{m_{\dot{\alpha}}}\frac{\dot{\alpha}\bar{c}}{2V} + C_{m_q}\frac{q\bar{c}}{2V} + C_{m_\delta}(\delta-\delta_T)\right]$$

$$\dot{\theta} = q$$

$$a_z = w - g\cos\theta - qu$$

$$\dot{\alpha} \approx \frac{w}{u}$$

$$\alpha = \tan\text{-}1\left(\frac{w}{u}\right)$$

$$V = \sqrt{u^2 + w^2}$$

where all C values (C_{m_α}, etc.) are treated as constants.

These equations were set up in DIFFUN as shown in Figure 4.2. The input for δ was taken from the function u(t) with parameters IUTYPE, UMIN, UMAX, A, and B read as data. The data for the test run was as follows:

N	4
TO,TN	0., 10.
(YO(I), I=1,N)	660.18167, 5.74626, .5, .09251
IUTYPE	2
UMIN,UMAX,A,B	0., 0.174533, 0., 4.
STEP	.1

Here, Y(1) is y, Y(2) is w, Y(3) is Q, and Y(4) is θ. The values for C_{X_α}, C_{Z_α}, etc., were extracted from data given in [Steinmetz, et al.], and are shown in DATA statements in Figure 4.2.

Figure 4.2 Input differential system for CLMP.

```
      SUBROUTINE DIFFUN(T,Y,DY) REAL Y(13,20), CY(20)

      C*************************************************************
      C*   THIS REPRESENTS THE LONGITUDINAL STABILITY EQUATIONS
      C*   FOR A JET FIGHTER.  THE INPUT FOR STABILATOR DEFLECTION
      C*   IS FROM THE FUNCTION U(T), WITH PARAMETERS:  IUTYPE=1,
      C*   UMIN=-.05,  UMAX=.065,  A=0., B=10.
      C*************************************************************

      REAL M, IY.
      DATA S, G, M, RHO, CBAR, AT, DT, IY/530., 32.158,
           1285.5, 2.125E-3, +16.04, 0., 0., 129609/
      DATA CXATDT, OXA, OXQ / . 0158, 0890, -3.92 /
      DATA CZATDT, CZA, CZQ, CZD / -.121, -3.36, -6.49, .346 /
      DATA CMATDT, OMA, CMADOT, CMQ, CMD / -.00117, -.106,
           -1.66, -1.66, -1.66, .576/

      UU=Y(1,1)
       W=Y(1,2)
       Q=Y(1,3)
      THETA=Y(1,4)

      V=SQRT(UU*UU+W*W)
      ALFA=ATAN(W/UU)
      DELTA=U(T)
      C1=(.5*Q*OBAR)/V
      C2=.5*RHO*V*V*S
      A=ALFA-AT
      D=DELTA-DT

      DY(1)=-G*SIN(THETA)-W*Q+(O2/M)*(OXATDAT+OXA*A+OXQ*O1)DY(2)
           =G*COS(THETA)+UU*Q+(C2/M)*(CZATDT+CZA*A+CZQ*C1+CZD*D)

      ALFDOT=DY(2)/UU
      AZ=DY(2)-G*COS(THETA)-Q*UU

      DY(3)=(C2*CEAR/1Y)*(OMATDT=CMA*A
     +     =(.5*CMADOT*ALFDOT*CBAR)/V+CMQ*C1+CMD*D)
      DY(4) = Q

      RETURN END
```

Running a Job on CLMP

As mentioned earlier, the user may either supply the system of equations (1) directly (sect.3) or as SUBROUTINE DIFFUN. See Figure 4.3. Procedures for running CLMP on a CYBER 172 computer with NOS 1 operating system are given as follows:

Equations entered directly:

(Jobcards)

COPYBF(INPUT, TAPE6)

(CLMP commands)

6/7/8/9

Equations entered in DIFFUN:

(Jobcarbs)

FTN,B=DIFOBJ.

7/8/9

FIGURE 4.3 SAMPLE TERMINAL INPUT

```
WHAT IS THE RATE OF YOUR TERMINAL IN CHARACTERS PER SECOND?
>>30
HOW MANY EQUATIONS ARE IN THE SYSTEM?
?4
GIVE INITIAL AND FINAL VALUES OF T
? 0.,10.
GIVE INITIAL VALUES FOR
    Y(1)
? 660.18167
    Y(2)
? 5.74626
    Y(3)
? .5
    Y(4)
? 5.3 WHAT TYPE OF NOISE WILL BE INPUT (U(T))?
ENTER 1.  FOR DAMPED SINE WAVE
      2.  FOR SPIKE FUNCTION
      3.  FOR STEP FUNCTION

? 1
 GIVE MINIMUM AND MAXIMUM VALUES FOR U(T)
? -.05,.065
WHAT INTERVAL WILL U(T) BE DEFINED FOR?
? 0., 10. WHAT STEP SIZE WILL BE USED FOR T?
?.1
WHICH INTEGRATION ROUTINE DO YOU WISH TO COMPARE WITH DIFSUB?
ENTER  1  FOR EULER
       2      IMPROVED EULER
       3      2-PT ADAMS-BASHFORTH
       4      3-PT RUNGE-KUTTA
       5      4-PT RUNGE-KUTTA
       6      RUNGE-KUTTA-MERSON
       7      4-PT ADAMS-MOULTON
?6
```

```
      SUBROUTINE DIFFUN
              .
              .
              .
      END
```

7/8/9

(CLMP commands)

6/7/8/9

The section described simply as (CLMP commands) now follows:

```
GET,A=CLMP
UVAL1B,PLOT10.
ENTER./LOAD(A,DIFOBJ)/NOGO,ABS.
LABS,ABS.
```

(options)

Since values are output on TAPE6 and TAPE7, the user may desire to:

(i) Save TAPE6 and TAPE7 for further use

(ii) List TAPE6 and/or TAPE7 at the line printer.

(iii) Have the values on TAPE7 punched at the card punch

These may be accomplished by entering the following commands in the section referred to as (options):

(i) SAVE,TAPE6,TAPE7.

(ii) REWIND,TAPE6,TAPE7.

```
     COPYBF,TAPE6,OUTPUT.
     COPYSBY,TAPE7,OUTPUT.
```

(iii) REWIND,TAPE7.

COPYBF,TAPE7,PUNCH.

Notes

The Fourier coefficients given are of the form:

$$f(t) = \frac{Co}{2} + \sum_{k=1}^{\infty} C_k \cos\left(d_k + \left(\frac{2k\pi t}{t_n - t_o}\right)\right) \text{ for } t \in [t_o, t_n] \quad (2)$$

Only the first N coefficients are computed however, where N is a suitable value between 32 and 64, chosen by CLMP. They are stored in an array in the following fashion:

ARRAY(1) = $c_0/2$

ARRAY(2) = 0.

ARRAY(3) = c_1

ARRAY(4) = d_1

.

.

.

ARRAY (2N-1) = c_{n-1}

ARRAY (2N) = d_{n-1}

Using the construction in (2) gives not f(t), however, but instead, $f(t+t_0)$. Hence, in order to compute f(t) for t in $[t_0, t_n]$, one must first compute $t^* = t - t_0$, and then compute

$$f(t) = \text{ARRAY}(1) + \sum_{i=2}^{N} \text{ARRAY}(2k-1)*\text{COS}\left(\text{ARRAY}(2k) + \left(\frac{2k\pi t^*}{t_n - t_o}\right)\right)$$

Smoothing

The FORTRAN subroutine RFFT is used by CLMP to compute Fourier coefficients for f(t). RFFT requires 64 points to

be entered as tables values, whereas CLMP will supply only N values, with $32 \leq N \leq 64$. The remaining 64- N points are assigned value zero. The Fourier coefficients computed by RFFT yield a function $f^*(t)$ with the following property:

Given f(t) and the interval $[t_0, t_n]$, $f(t_i) - f^*(t_i) =$ constant for $1 \leq i \leq$ and $f(t_i) - f^*(t_i)$ has alternating signs for consecutive values of i. The value of the constant is unknown, but, because of the alternating signs, can be virtually eliminated graphically as follows:

Given t_i and $f^*(t_i)$, $1 \leq i \leq n$,

compute $T_i = (t_i + t_{i+1})/2$ and

$(F(t_i) = (f^*(t_i) + f^*(t_{i+1}))/2$.

Notice that $f(t_i) = (f(t_i) + f(t_{i+1}))/2$ as well. Then by plotting $F(t_i)$ at t_i, for $1 \leq i \leq N - 1$, a smooth graph can be drawn that closely approximates f(t).

By way of an example, examine figures 4.4, 4.5 and 4.6. Figure 4.5 shows the graphs of Y(1) computed from two separate sets of Fourier coefficients, before smoothing. Notice the undesirable effects of oscillation. Figure 4.6 shows the same graphs, after smoothing. Notice how closely the DIFSUB and NEXTPT graphs coincide on the smooth portion of the graph (i.e., away from discontinuities in the first derivative). This gives a very good view of what Y(1) looks like on that interval.

Which variable Y do you wish to see fourier coefficients for?

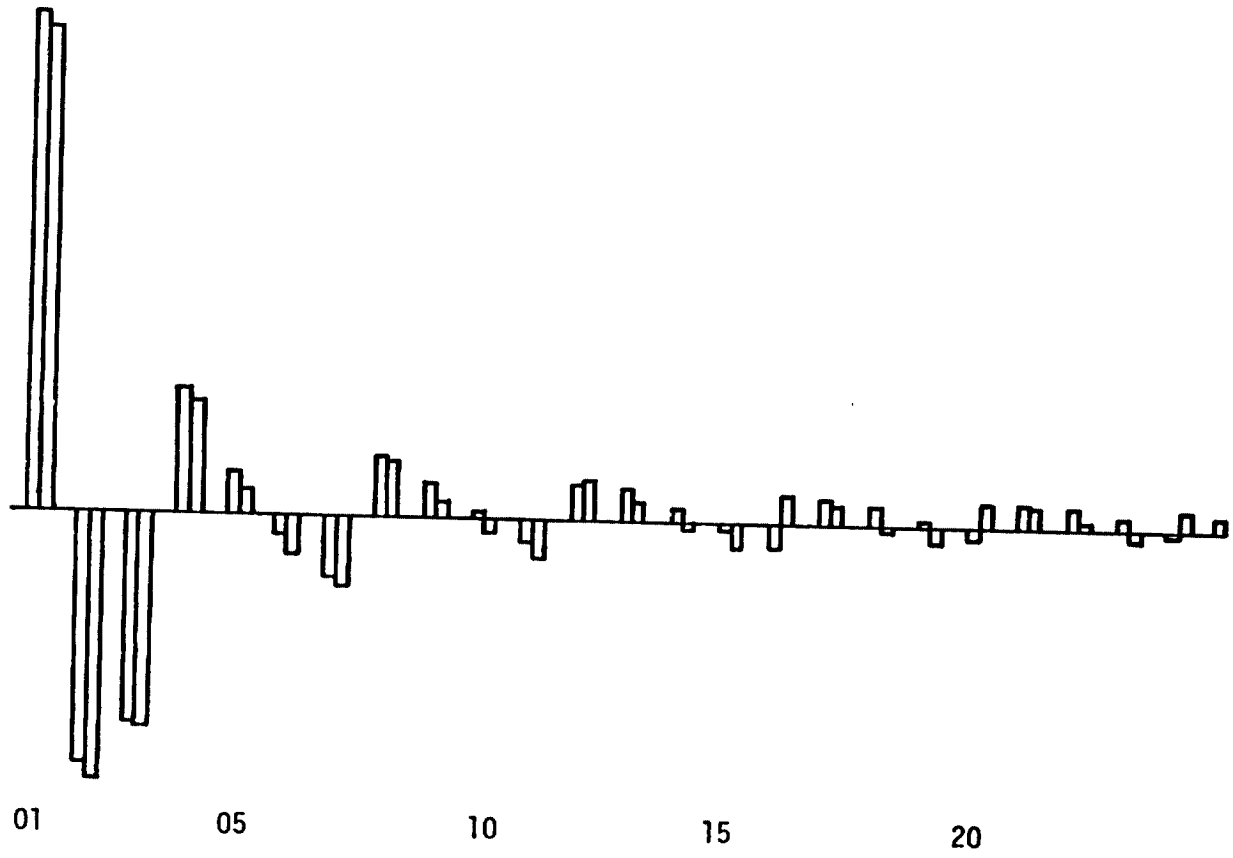


Figure 4.4 Bar Graph Depicting Fourier Coefficients for Y(1). For each pair of bars, bar on left denotes DIFSUB, bar on right denotes NEXTPT values.

ENTER 1 FOR SMOOTHING
-1 OTHERWISE

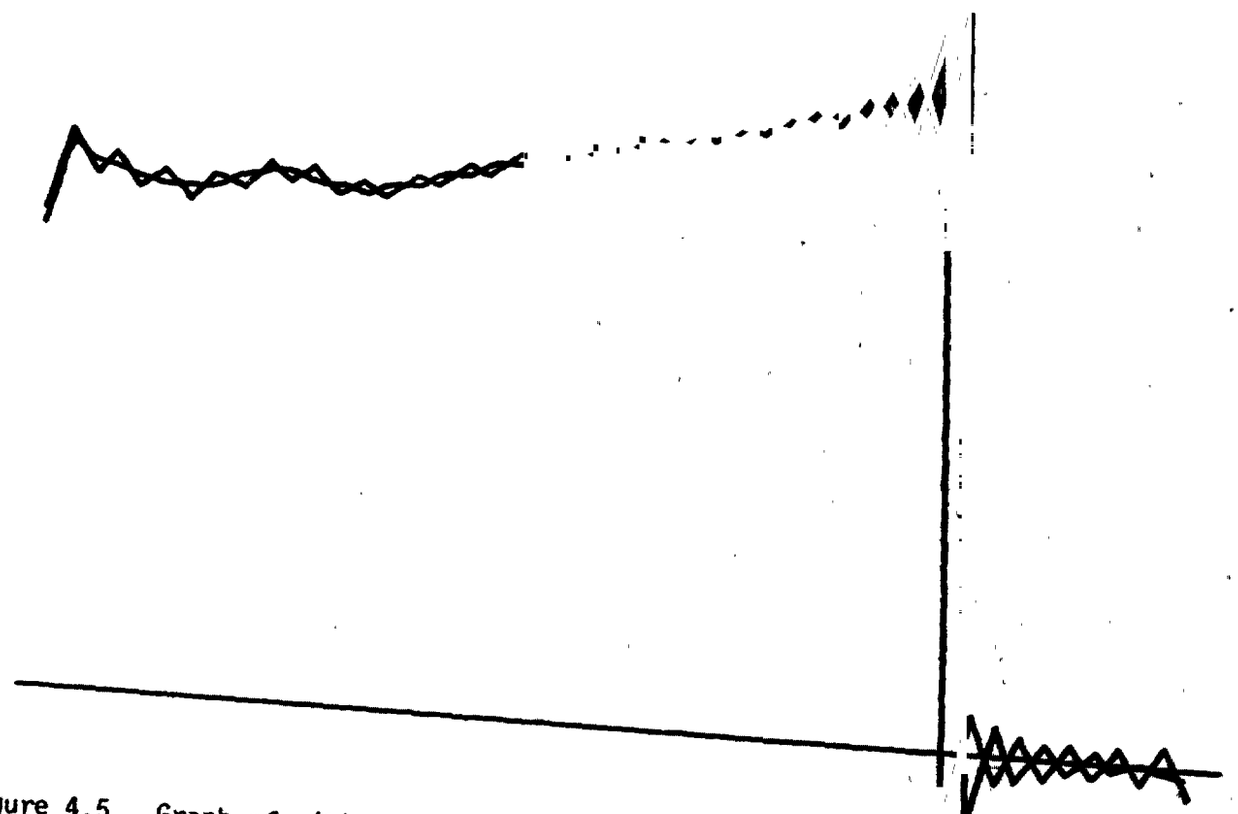


Figure 4.5 Graph of y(1), constructed from FOURIER coefficients given by DIFSUB and by NEXTPT, before smoothing.

?

WHICH VARIABLE Y DO YOU WISH TO SEE CURVES FOR?

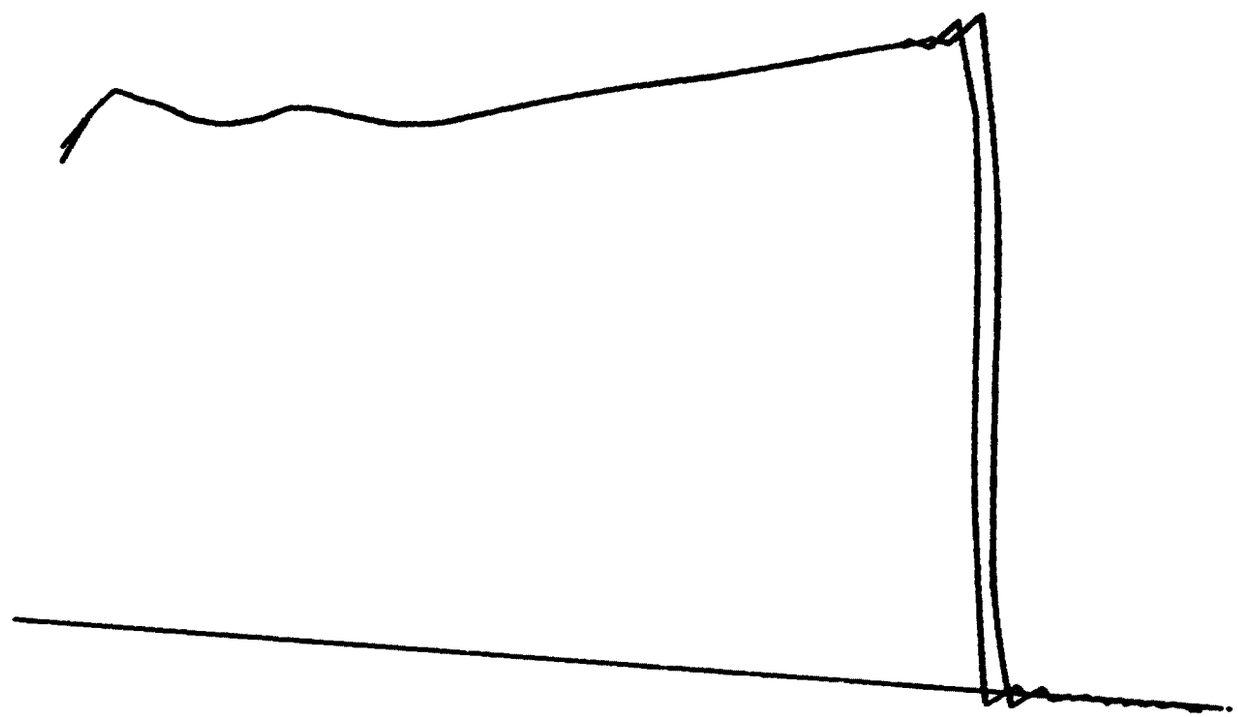


Figure 4.6 Graph of $y(1)$, given in Figure 5, after smoothing.

APPENDIX A

NUMERICAL METHODS

ABK Predictors

K=1 $y_n = y_{n-1} + hy'_{n-1}$

K=2 $y_n = y_{n-1} + h/2(3y'_{n-1} - y'_{n-2})$

K=3 $y_n = y_{n-1} + h/12(23y'_{n-1} - 16y'_{n-2} + 5y'_{n-3})$

K=4 $y_n = y_{n-1} + h/24(55y'_{n-1} - 59y'_{n-2} + 37y'_{n-3} - 9y'_{n-4})$

K=5 $y_n = y_{n-1} + h/720(1901y'_{n-1} - 2774y'_{n-2} + 2616y'_{n-3} - 1274y'_{n-4} + 251y'_{n-5})$

K=6 $y_n = y_{n-1} + h/1440\ (4277y'_{n-1} - 7923y'_{n-2} + 9982y'_{n-3} - 7298y'_{n-4} + 2877y'_{n-5} - 457y'_{n-6})$

AMK - Corrector (used with ABK Predictor)

K=1 $y_n = y_{n-1} + h/2(y'_n + y'_{n-1})$

K=2 $y_n = y_{n-1} + h/12(5y'_n + 8y'_{n-1} - y'_{n-2})$

K=3 $y_n = y_{n-1} + h/24(9y'_n + 19y'_{n-1} - 5y'_{n-2} + y'_{n-3})$

K=4 $y_n = y_{n-1} + h/720(251y'_n + 646y'_{n-1} - 264y'_{n-2} + 106y'_{n-3} - 19y'_{n-4})$

K=5 $y_n = y_{n-1} + h/1440(475y'_n + 1427y'_{n-1} - 798y'_{n-2} + 482y'_{n-3} - 173y'_{n-4} + 27y'_{n-5})$

BDF - Corrector (used with ABK Predictor)

K=1 $y_n = y_{n-1} + hy'_n$

K=2 $3y_n = 4y_{n-1} - y_{n-2} + 2hy'_n$

K=3 $11y_n = 18y_{n-1} - 9y_{n-2} + 2y_{n-3} + 6y'_n$

K=4 $25y_n = 48y_{n-1} - 36y_{n-2} + 16y_{n-3} - 3y_{n-4} + 12hy'_n$

$$K=5\quad 137y_n = 300y_{n-1} - 300y_{n-2} + 200y_{n-3} - 75y_{n-4} + 12y_{n-5} + 60hy'_n$$

$$K=6\quad 147y_n = 360y_{n-1} - 450y_{n-2} + 400y_{n-3} - 225y_{n-4} + 72y_{n-5} - 10y_{n-6} + 60hy'_n$$

RK2 - one step

$$K_0 = hf(y_{n-1})$$

$$K_1 = hf(y_{n-1} + K_0/2)$$

$$y_n = y_{n-1} + K_1$$

RK4 - one step

$$K_0 = hf(y_{n-1})$$

$$K_1 = hf(y_{n-1} + K_0/2)$$

$$K_2 = hf(y_{n-1} + K_1/2)$$

$$K_3 = hf(y_{n-1} + K_2)$$

$$y_n = y_{n-1} + h/6\ (K_0 + 2K_1 + 2K_2 + K_3)$$

APPENDIX B

THEOREM PROOFS

Definition. A *linear k-step formula* satisfies

$$0 = \sum_{j=0}^{k} [a_j \hat{y}_{n-j} + h b_j f(\hat{y}_{n-j}, t_{n-j})]. \tag{9}$$

Definition. A *one-leg k-step formula* corresponding to (9) satisfies

$$0 = \sum_{j=0}^{k} a_j y_{n-j} + hsf\left(\frac{1}{s}\sum_{j=0}^{k} b_j y_{n-j}, \frac{1}{s}\sum_{j=0}^{k} b_j t_{n-j}\right), \tag{10}$$

where $s = \sigma(1)$. Without loss of generality, set $s = 1$.

THEOREM 1 [4]. *Let Y_n be a sequence which satisfies* (10), *and let $\hat{Y}_n = \{\hat{y}_n\}$ be such that*

$$\hat{y}_n = \sum_{j=0}^{k} b_j y_{n-j} = \sigma(E) y_n, \tag{11}$$

where E denotes the back shifting operator. Then $\hat{Y}_n$ satisfies (9). *Conversely, if $\hat{Y}_n$ satisfies* (9), *then there exists a sequence Y_n such that $\hat{y}_n = \sigma(E)y_n$, and y_n satisfies* (10).

Proof. Without loss of generality assume the system of equations is autonomous. Write (10) as $\rho(E)y_n = -hf(\sigma(E)y_n)$, $n = 0, 1, 2, \ldots$. This together with (11) implies $\rho(E)\hat{y}_n = \rho(E)\sigma(E)y_n = -h\sigma(E)f(\hat{y}_n)$, which implies that $\hat{y}_n$ satisfies (9).

For the converse, Euclid's theorem on polynomials with no common divisors implies the existence of polynomials P, Q which satisfy $P(x)\sigma(x) + Q(x)\rho(x) = x^m$, $0 \leq m \leq k$. Writing (9) as $\rho(E)\hat{y}_n = -h\sigma(E)f(\hat{y}_n)$ and setting $y_n = E^{-m}(P(E)\hat{y}_n - hQ(E)f(\hat{y}_n))$ gives

$$\sigma(E)y_n = E^{-m}(P(E)\sigma(E)\hat{y}_n + Q(E)\rho(E)f(\hat{y}_n)) = E^{-m}(P(E)\sigma(E) + Q(E)\rho(E))\hat{y}_n,$$

which gives $\sigma(E)y_n = \hat{y}_n$. Next, set

$$\rho(E)y_n = -E^{-m}h(P(E)\sigma(E) + Q(E)\rho(E))f(\hat{y}_n) = -hf(\hat{y}_n) = -hf(\sigma(E)y_n), \quad n \geq m,$$

and Y_n satisfies (10), proving the converse.

This shows that Y_n given by the one-leg k-step formula will have similar stability properties to its corresponding linear k-step sequence $\hat{y}$. Dahlquist [4] has described a discrete Liapunov function $V_{G,l,h}$ which, applied to a sequence Y_n, characterizes the stability of that sequence generated by a nonlinear system (1). Let $V_{G,l,h}(Y_n) = \Sigma_{i=1}^{l}\Sigma_{j=1}^{l} g_{ij}\langle y_{n-i+1}, y_{n-j+1}\rangle$, where G is a positive definite, $l \times l$ matri... The structure of G assures that $V_{G,l,h}$ is positive for $Y_n \neq \{0\}$.

THEOREM 2. *If $V_{G,l,h}(Y_n) = c$ for the symmetric positive definite matrix G, then there exists a symmetric, positive definite matrix $\hat{G}$, dependent only on G and $\sigma(x)$, such that $V_{\hat{G},l,h}(\hat{Y}_n) = c$, where $\hat{y}_n = \sigma(E)y_n$ are the elements of $\hat{Y}_n$ and Y_n.*

Proof. Without loss of generality, consider a system of only one equation $y' = f(y, t)$ generating the sequence $Y_n = \{y_{n-k+1}, \cdots, y_n\}$. Since $\hat{y}_n$ and y_n are related only by the $k + 1$ coefficients of $\sigma(x)$, replace the sequences by the vectors $w_n = (y_n, y_{n-1}, \cdots, y_{n-k+1})^*$ and $w'_n = (y_n, y_{n-1}, \cdots, y_{n-l-k+1})^*$, where $*$ is the transpose operator. Let G' be a $(k + l)$ by $(k + l)$ matrix consisting of G in the upper l by l partition, and 0 elsewhere. Then $V(w_n) = w_n^* G w_n = c > 0$, and $V(w'_n) = w'^*_n G' w'_n = c$.

Define S such that $\hat{w}_n = Sw'_n$, thus

$$S = \begin{pmatrix} b_0 & b_1 & \cdots & b_k & 0 & \cdots & 0 \\ 0 & b_0 & & b_{k-1} & b_k & \cdots & 0 \\ & & & \cdot & & & \\ \vdots & & & & \cdot & & \\ & & & & & & \\ 0 & & & b_0 & b_1 & \cdots & b_k \end{pmatrix}$$

is an l by $l + k$ matrix. Then since G' is of rank l because G is positive definite, there exists a singular value decomposition of $G' = UFV^*$ for U, V $l + k$ by $l + k$ unitary matrices, and $F = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$ for D an l by l diagonal matrix of singular values of G. Thus, there exists an l by l matrix $\hat{G}$ such that $G' = S^* \hat{G} S$. This is seen by letting S have the singular value decomposition $U_s(\Sigma|0)V_s$, where Σ is an l by l diagonal matrix. Then

$$\hat{G} = U_s(\Sigma^{-1}|0)V_s^* UFV^* V_s^*(\Sigma^{-1}|0)^* U_s^*.$$

If $b_0, b_1, \ldots, b_{l-1}$ are all zero, a similar argument can be made using an l by $l + k - i$ matrix S where b_l is the coefficient of lowest index l such that $b_l \neq 0$, and

$$S = \begin{pmatrix} b_l & b_{l+1} & \cdots & b_k & \cdots & 0 \\ 0 & b_l & \cdots & b_{k-1} & \cdots & 0 \\ 0 & \cdots & \cdots & & & 0 \end{pmatrix}.$$

Thus, there exists a $\hat{G}$ dependent only on $\sigma(x)$ such that $V_{\hat{G},l,h}(Y_n) = c$ for all c, which was to be shown.

THEOREM 4. If the equilibrium $\bar{y}(t)$ of $f(y, t)$ has a Liapunov function of the form $v(y, t) = y^*Qy$ for a positive definite matrix Q, y^* the transpose of y, and $f(y, t)$ is continuously differentiable on a convex domain D whose boundary ∂D is defined by $v'(y_0, t_0) = 0$, and if $f(y, t)$ has the property on D that $(y_1 - y_2)^*Q(f(y_1, t) - f(y_2, t)) \le \mu\|y_1 - y_2\|_Q^2$ where $\mu \le 0$ and $x^*Qx = \|x\|_Q^2$, then for any point $\bar{y}_l = \bar{y}(t_l)$ in the interior of D, an (I, l, h)-stability region can be constructed using rays from $\bar{y}_l$, provided $h \le h_0(l, f)$.

Proof. Note that the solution from t_0 to $t_0 + h$ is spiraling in from ∂D toward the equilibrium, which is inside a circle $\|y(t) - y_0\|_2 \le ((hl)^2)M$ where $M = \max_{z \in D} f'(z)$, since the hypotheses and (8) gives $\|y(t_l) - y(\hat{t}_l)\| \le \exp(\mu l h)\|y(t_0) - \bar{y}(t_0)\|$. By definition, $\Delta V_{I,l,h}(Y_l) = \|y(t_l)\|_Q^2 - \|y(t_0)\|_Q^2$, which occurs when $0 = v(y_l, t_l) - v(y_0, t_0) = (y_l - y_0)v'(z)$ by the mean value theorem. Since $v'(z) = 0$ on the boundary ∂D, along any ray from $\bar{y}_l$ one can find $y(t_l)$ generated by a y_0 on ∂D, provided $hl < h_0 l$ is small enough that the trajectory from $\bar{y}_0$ to $\bar{y}_l$ is entirely in D.

The boundary of (I, l, h)-domain of attraction D' can be constructed of all such rays. The (I, l, h)-stability region D'' has a boundary of all points such that $V_{I,l,h} = v^* = \min_{\partial D'} V_{I,l,h}$, which can also be constructed.

THEOREM 5. If the hypotheses of theorem 4 are met and $(\partial/\partial y)(d^p f(z)/dt^p)$ is continuous in D, then for any $\bar{y}_l$ in the interior of D, an (I, l, h) stability region can be constructed for a pth order one-leg k-step method, for $h \le h_0(l, f)$.

Proof. $\Delta V = \|y_l\|_Q^2 - \|y(t_l - lh)\|_Q^2$ where

$$y_l = \sum_{j=1}^{k} a_j y(t_{l-j}) + hf\left(\sum_{j=0}^{k} b_j y(t_{l-j}), \sum_{j=0}^{k} b_j t_{l-j}\right)$$

and $y(t_l) = y_l + K_p h^{p+1} f^{(p)}(z)$, the truncation error formula. If the truncation error varies continuously as $y(t_0)$ varies along ∂D, Then two solutions y_l and z_l generated by y_0, z_0 on ∂D have the property that $y_l \to z_l$ uniformly as $y_0 \to z_0$, and a smooth curve $\partial D'$ exists on the boundary of the (I, l, h)-domain of attraction. Similar arguments show the existence of the (I, l, h)-stability region.

REFERENCES

1. Ames, W. F., Numerical Methods for Partial Differential Equations, 2nd Edition, Academic Press, NY (1977).

2. Barton, D., I. M. Willers, R. V. M. Zahar, The Automatic Solution of Systems of Ordinary Differential Equations by the Method of Taylor Series, Comp., J. 14, pp. 243-248 (1972).

3. Brown, R. L., Stability of Sequences Generated by Nonlinear Differential Systems, Math. Comp. 33, pp. 637-645 (1979).

4. Brown, R. L., Stability Analysis of Nonlinear Differential Sequences Generated Numerically, Int. J. Comp. Math with Appl. 5, pp. 187-192 (1979).

5. Conte, S. D. and C. de Boor, Elementary Numerical Analysis, An Algorithmic Approach, McGraw-Hill, N.Y. (1972).

6. Dahlquist, G., On Stability and Error Analysis for Stiff Non-Linear Problems, Report NA-7508, Dept. of Information Processing, Royal Inst. of Technology, Stockholm (1975).

7. Dahlquist, G., G-Stability is Equivalent to A-Stability, BIT 18, pp. 384-401 (1978).

8. Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, N.J. (1971).

9. Gibbons, A. A., A Program for the automatic Integration of Differential Equations Using the Method of Taylor Series, Comp. J. 3, pp. 108-111 (19--).

10. Gries, D., Compiler Construction for Digital Computers, Wiley, N.Y. (1971). 11. Jeltsch, R. and O. Nevanlinna, Largest Disk of Stability of Explicit Runge-Kunta Methods, BIT 18, pp. 500-502 (1978).

12. Knopp, K. Infinite Sequences and Series, Dover, N.Y., (1956).

13. Knuth, D. E., The Art of Computer Programming, Vol.2, Addison-Wesley, Reading, Massachusetts (1969).

14. Lehnigk, S. H., Stability Theorems for Linear Motions, Prentice-Hall, Englewood Cliffs, N.J. (1966).

15. Liniger, W. and F. Odeh, On Liapunov Stability of Stiff Non-Linear Multi-Step Diffference Equations, AFOSR-TR-76-1023, IBM Thomas J. Watson Research Center (1976).

16. Lucas, J. J. and J. V. Wait., Dare-P User's Manual, CSRL Report #255, University of Arizona.

17. Smail, L. L., Elements of the Theory of Infinite Processes. McGraw-Hill, N.Y. (1923).

18. Steinmetz, G. G., R. V. Parrish, R. L. Bowles, Longitudinal Stability and Control Derivatives of a Jet Fighter Airplane Extracted from Flight Test Data by Utilizing Maximum Likelihood Estimators, Stetter, H. J., Analysis of Discretization Methods for Ordinary Differential Equations, Springer-Verlag, N.Y. (1973).

BIBLIOGRAPHY

These were published as part of the research effort on NASA grant NSG-1335. They appear in approximate order of publication.

1. Brown, R. L., Suitability of Integrators for Non-linear Ordinary Differential Equations, presented at ACM Computer Science Conference, Atlanta, February 1, 1977.

2. Brown, R. L., Investigation of ODE Integrators using Interactive Graphics, Proceedings of IMACS Symposium on Simulation Software and Numerical Methods for Differential Equations, North-Holland, 1978.

3. R. L. Brown, Evaluation of Ordinary Differential Equation Software, BIT 18, pp. 103-105 (1978).

4. R. L. Brown, Stability of Sequences Generated by Non-Linear Differential Sequences, Math. Comp. 33, pp. 637-645 (1979).

5. R. L. Brown, Software Development for Stability Analysis of Non-Linear Differential Systems, Working Papers of 1979 SIGNUM Meeting on Numerical ODE's, Urbana, Il. (1979).

6. R. L. Brown, Stability Analysis of Non-linear Differential Sequences Generated Numerically; Comp. and Math. with Appls. 5, pp. 187-192 (1979).