Numerical Solution of Differential-Difference Equations

by

V.K. Barwell

CS-76-04

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

January, 1976

ABSTRACT

Many of the properties of methods for solving ordinary differential equations are similar to the properties of methods for differential difference equations. For example, Tavernini has shown that convergence of a consistent method for ordinary differential equations implies convergence of a consistent method for differential difference equations. Cryer has given a generalization of the definition of A-stability of methods for ordinary differential equations by considering the scalar equation $y'(t) = qy(t-\beta)$, $\beta > 0$, q real, and has illustrated methods satisfying his generalized definitions.

In this thesis a complete characterization is given for the asymptotic behaviour of the equation $y'(t) = qy(t-\beta)$, $\beta > 0$, q complex, and a partial characterization is given for the asymptotic behaviour of $y'(t) = py(t) + qy(t-\beta)$, $\beta > 0$, p and q complex. This enables the author to generalize the definitions and theorems due to Cryer. The backward differentiation methods are shown to have nice stability properties.

These backward differentiation methods and the Adams methods are incorporated into an automatic package (similar to Gear's package for solving ordinary differential equations) for solving the equation $\underset{\sim}{y}'(t) = \underset{\sim}{f}(t,\underset{\sim}{y}(t),\underset{\sim}{y}(t-\beta))$, $\beta > 0$.

Sample problems to test the effectiveness of the package are given, and one example illustrates the surprising result that stiffness can occur in a scalar differential difference equation. An appropriate definition for stiffness of differential difference equations is given.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1

REVIEW OF NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

Existence and Uniqueness

An initial value problem (I.V.P.) in ordinary differential equations (O.D.E.) consists of a differential equation of the form

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

together with an initial condition

(1.2) $$y(a) = y_a.$$

The numerical solution of (1.1) and (1.2) consists of calculating a sequence of values $\{y_n\}$ which approximate the solution on a set of nodes $\{t_n\}$. This entire process assumes that (1.1) and (1.2) have a solution. The following theorem, whose proof can be found in Henrici [12, p.112] gives conditions on the function $f(t,y(t))$ such that (1.1) and (1.2) have a unique solution.

Theorem 1.1

Let $f(t,y(t))$ be defined and continuous in a region

$$D = \{(t,y) | a \le t \le b, -\infty < y < +\infty\}$$

and suppose the function f satisfies the Lipschitz condition:

$$\exists L > 0 \ni \forall (t,y),(t,y^*) \in D$$
$$|f(t,y)-f(t,y^*)| \le L|y-y^*|.$$

Then for any given number y_a, there exists a unique solution $y(t)$ to (1.1), where $y(t)$ is continuous and differentiable for all $(t,y) \in D$ and $y(a) = y_a$. □

Although this theorem and most of the theorems and results given in this thesis refer, for simplication, to a scalar equation they are valid with the obvious changes for systems. Any changes for systems which are not obvious are clearly pointed out.

Linear Multistep Methods

Consider the sequence of points $t_n = a+nh$ where $n = 0,1,2,\ldots$. The parameter h which is regarded as constant (unless otherwise noted) is called the steplength. The numerical problem is to determine a sequence of numbers $\{y_n\}$ which is an approximation to the theoretical solution $\{y(t_n)\}$.

Let $f_n = f(t_n,y_n)$. Then if the numerical method for determining the sequence $\{y_n\}$ is a linear relationship between y_{n+j}, f_{n+j} for $j = 1,\ldots,k$, we call the method a k-step linear multistep method.

The linear multistep method (L.M.S.) may be written as

$$\sum_{j=0}^{k} \alpha_j \, y_{n+j} = h \sum_{j=0}^{k} \beta_j \, f_{n+j} \qquad (1.3)$$

where α_j, β_j are constants, $\alpha_k \neq 0$ and both α_0, β_0 are not zero. As (1.3) is arbitrary in the sense that all constants could be multiplied by the same factor, a normalization is usually done by requiring $\alpha_k = 1$.

Assuming that $y_n, y_{n+1}, \ldots, y_{n+k-1}$ are known, we note that (1.3) is a nonlinear (algebraic) equation for y_{n+k} which may be difficult to solve particularly for implicit methods ($\beta_k \neq 0$). If $\beta_k = 0$, we say the method is explicit and the solution is direct.

An implicit method requires at each stage of the computation the solution y_{n+k} of the equation

$$(1.4) \qquad y_{n+k} = h\beta_k f(t_{n+k}, y_{n+k}) + g$$

where g is a known function of the previously calculated values.

Review of the Solution of Nonlinear Equations

Consider the nonlinear equation

$$(1.5) \qquad y = f(y).$$

Define the sequence of iterates $\{y^m\}$ by the equation

$$(1.6) \qquad y^{m+1} = f(y^m).$$

Then the following theorem whose proof can be found in Henrici [12, p.216] gives conditions under which (1.5) has a unique solution and the sequence of iterates defined by (1.6) converges to that solution as $m \to +\infty$.

Theorem 1.2

Let f(y) satisfy the Lipschitz condition $|f(y)-f(y^*)| \le L|y-y^*|$ $\forall y, y^*$ and $0 \le L < 1$. Then (1.5) has a unique solution to which the iterates defined by (1.6) converge. □

If the Lipschitz constant in Theorem 1.2 is large, then an alternative method which may be used is the well-known Newton iteration. When applied to the equation $F(y) = 0$, this has the form:

$$y^{m+1} = y^m - F(y^m)/F'(y^m) \qquad m = 0,1,2,...$$

Sufficient conditions for the convergence of Newton's method may be found in [12,p.366]. We note, however, that convergence depends primarily upon the closeness of y^0 to the solution.

Basic Concepts of Linear Multistep Methods (L.M.S.)

The L.M.S. (1.3) is said to be convergent if for all I.V.P. (1.1), (1.2) subject to the hypothesis of Theorem 1.1

$$\lim_{\substack{h\to 0 \\ t-a=nh}} y_n = y(t_n)$$

$\forall t \in [a,b]$ and all solutions of (1.4) which have starting values which are a function of h and converge to $y(a)$ as $h \to 0$.

With the L.M.S. (1.3) we can associate the linear difference operator

$$(1.7) \qquad L[y(t),h] = \sum_{j=0}^{k} [\alpha_j y(t+jh) - h\beta_j y'(t+jh)]$$

where $y(t)$ is an arbitrary function possessing as many higher order derivatives as we wish. Formally, expanding $y(t+jh)$ and $y'(t+jh)$ in a Taylor series about t gives $L[y(t),h] = \sum_{j=0}^{+\infty} C_j h^j y^{(j)}(t)$, where the C_j are constants. The L.M.S. (1.3) is said to be of *order p* if $C_j = 0$ for $0 \leq j \leq p$ and $C_{p+1} \neq 0$. C_{p+1} is called the *error constant*. The L.M.S. is said to be *consistent* if the order $p \geq 1$.

The *local truncation error* at t_{n+k} of the L.M.S. (1.3) is defined to be $L[y(t_n),h]$ where $y(t)$ is the theoretical solution to the I.V.P. (1.1), (1.2). If the previous values were exact (no truncation error was made) and the theoretical solution $y(t)$ has continuous derivatives of sufficiently high order, then we could show [14, p28] $y(t_{n+k}) - y_{n+k} = C_{p+1}h^{p+1}y^{(p+1)}(t_n) + O(h^{p+2})$. The term $C_{p+1}h^{p+1}y^{(p+1)}(t_n)$ is called the *principal local truncation error*. In practice, of course, truncation error is made in the previous values. The actual error $y(t_{n+k}) = y_{n+k}$ is called the *global truncation error*. It can be shown that if the local truncation error is $O(h^{p+1})$ then under certain conditions the global truncation error is $O(h^p)$ [12, p.247]. Hence we try to choose our methods with as great an order as possible to reduce the global error.

As a L.M.S. method is specified by the coefficients α_j and β_j, $j = 0,\ldots,k$, then we may specify a L.M.S. method by the *first and second characteristic polynomials*

$$\rho(z) = \sum_{j=0}^{k} \alpha_j z^j, \quad \sigma(z) = \sum_{j=0}^{k} \beta_j z^j.$$

Consider the scalar equation

$$(1.8) \qquad y'(t) = \lambda y(t), \qquad \lambda \text{ a constant.}$$

circle. A region R of the complex plane is called a region of absolute stability if the method is absolutely stable $\forall \lambda h \in R$.

We can plot the boundary ∂R of the region of absolute stability for a method by using the boundary locus method [14,p.82]. Since the roots of (1.9) are a continuous function of λh the λh will lie on ∂R when one of the roots lies on the unit circle and hence the root has the form $\exp(i\theta)$. Substituting this into (1.9) and solving for λh gives $\lambda h = \rho(\exp(i\theta))/\sigma(\exp(i\theta))$. Letting θ vary over the interval $[0,2\pi]$ and plotting the corresponding values of λh gives us a plot of ∂R in the λh plane.

Examples

Consider the well known Euler method $y_{n+1} = y_n + hf_n$. Here $\rho(z) = z-1$, $\sigma(z) \equiv 1$. Clearly $\rho(1) = 0$, $\rho'(1) = 1 = \sigma(1)$, and the zero of $\rho(z)$ is a simple zero on the unit circle. Hence the method is consistent and zero stable. Applying the boundary locus method gives $\lambda h = \exp(i\theta)-1$. Clearly the region of absolute stability is the disc $|z+1| \leq 1$.

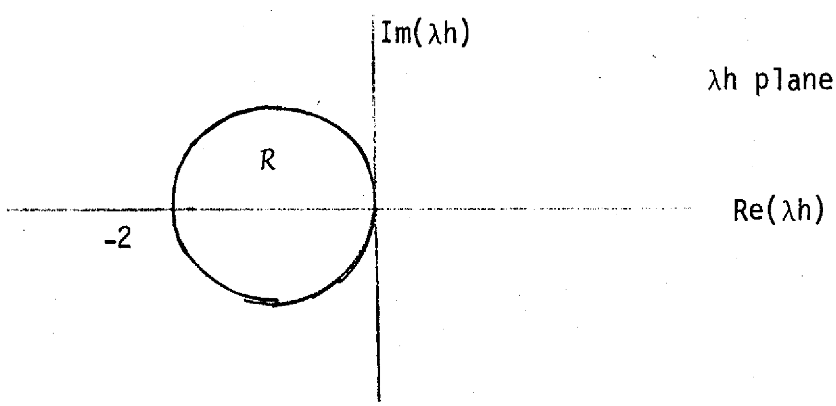


Fig.1.1 Region of absolute stability of Euler method

Another method to consider is the Backward Euler method

$$y_{n+1} = y_n + hf_{n+1}.$$

Here $\rho(z) = z-1$; $\sigma(z) = 1$. Again we see that the method is consistent and zero-stable. The boundary locus method gives $\lambda h = 1-\exp(-i\theta)$ so that the region of absolute stability is the entire λh plane except for the disc $|z-1| < 1$.

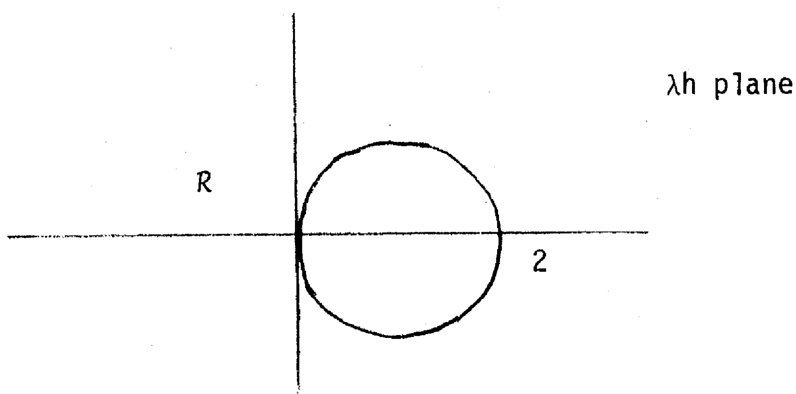


Fig.1.2 Region of absolute stability for backward Euler method

Note that solutions to equation (1.9) go to zero asymptotically for $Re(\lambda h) < 0$. We would like the numerical method to behave in a similar manner. Hence we say a method is A-stable if its region of absolute stability contains the region $Re(\lambda h) < 0$.

Clearly from Figures (1.1) and (1.2), we see that the Backward Euler method is A-stable while the Euler method is not.

Derivation of a Method

Two special classes of L.M.S. methods which we will use later are the explicit Adams-Bashforth and the implicit Adams-Moulton method. The Adams methods are characterized by the first characteristic polynomial $\rho(z)$ which has the form $z^k - z^{k-1}$. The coefficients of $\sigma(z)$ are then chosen to maximize the order of the method with $\beta_k = 0$ for the explicit method and $\beta_k \neq 0$ for the implicit method. In fact, this produces a k step method of order k.

Predictor-Corrector Methods

Recall that for implicit methods we must solve, at each step, the equation $y_{n+k} = h\beta_k f(t_{n+k}, y_{n+k}) + g$. By Theorem 1.2 this can be solved by the iteration

$$y_{n+k}^{m+1} = h\beta_k f(t_{n+k}, y_{n+k}^{m}) + g$$

where y_{n+k}^{0} is arbitrary provided

(1.10) $\quad h < 1/(L|\beta_k|)$.

Normally the acceptable limit on h is determined by other considerations (such as accuracy) except in those differential equations with a very large L which are considered separately.

It is obviously desirable to keep the number of iterations to a minimum so as to minimize the number of function evaluations. We would therefore like to make the initial guess y_{n+k}^{0} as close as possible to y_{n+k}. This is normally done by using an explicit method (called a predictor) to compute y_{n+k}^{0}. The implicit method is called the corrector.

Normally, the restriction (1.10) is not important, however there is a class of problems which exhibit a property called 'stiffness' in which the restriction (1.10) is important and a Newton iteration must be used to solve the nonlinear equation (1.4).

The linear system $\underset{\sim}{y}'(t) = A\underset{\sim}{y}(t)$ is said to be <u>stiff</u> if the eigenvalues of A are widely separated in magnitude and the time scale is large enough. This definition is somewhat ambiguous because it really depends on whether we are interested in transient or asymptotic solution behaviour. For the more general problem $\underset{\sim}{y}'(t) = f(t,\underset{\sim}{y})$ we can do a local linearization and thus we would modify our definition to apply to the eigenvalues of the Jacobian of $\underset{\sim}{f}$. Then it is necessary to use methods which are A-stable in order to take a large step relative to the time scale.

The following (depressing) theorem by Dahlquist [7] restricts the order of linear multi-step A-stable methods.

<u>Theorem 1.4</u>

An explicit linear multi-step method cannot be A-stable and the order of an A-stable method cannot exceed two. □

In order to overcome this problem Gear [11,p.213] suggests a slackening of the A-stability requirement with the following definition. A numerical method is said to be <u>stiffly stable</u> if its region of absolute stability contains R_1, R_2 and it is accurate for all $h \in R_2$ when applied to equation (1.9) with $Re(\lambda) < 0$, where

$$R_1 = \{h\lambda \mid Re(h\lambda) < -a\}$$

$$R_2 = \{h\lambda \mid -a \le Re(h\lambda) \le b,\ -c \le Im\,(h\lambda) \le c\}$$

and a, b, c are positive constants.

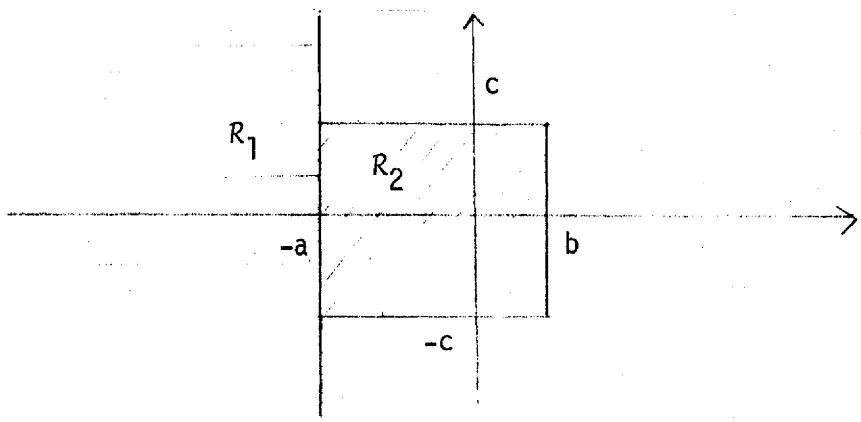


Fig.1.3 Stiff stability region

Gear then proposed a class of methods called backward differentiation methods. These methods have the form

$$\sum_{j=0}^{k} \alpha_j \, y_{n+j} = \beta_k \, f_{n+k}.$$

That is we, take $\sigma(z) = \beta_k z^k$ and choose β_k and $\rho(z)$ to maximize the order. This produces a k-step method of order k. The first and second order backward differentiation methods are A-stable [11,p.214]. In fact, in [11,p.214] Gear plots the region of absolute stability for these methods.

The first to sixth order methods are stiffly stable.

A method is called A_0 stable if its region of absolute stability includes the negative real axis. Cryer has shown [5] that the higher order backward differentiation methods are not even A_0 stable.

Alternately, one could consider using implicit Runge Kutta methods which are A-stable, but these methods are not considered in this thesis.

CHAPTER 2

REVIEW OF THE NUMERICAL SOLUTION OF DIFFERENTIAL DIFFERENCE EQUATIONS

Basic Existence and Uniqueness Theorem

Consider the equation

$$(2.1)\qquad y'(t) = f(t,y(t),y(t-\beta(t)))$$

where $\beta(t) \geq 0$ and $y(t) = g(t)$ on the initial set E_{t_0} defined by

$$E_{t_0} = \{t-\beta(t)\,|\,t-\beta(t) \leq t_0 \text{ for } t > t_0\}.$$

Thus the right-hand side of the differential equation depends on the solution at the given time and the solution at a previous time. D.D.E's are also different from O.D.E's in that the solution must be specified on the initial set E_{t_0} which is frequently an interval. If, for example, $\beta(t) \equiv \beta$, a constant, then $E_{t_0} = [t_0-\beta, t_0]$ and we would want the solution $y(t)$ for $t > t_0$.

If we apply the method of steps [9, p.6] to (2.1) we obtain the equation

$$(2.2)\qquad \begin{aligned} y'(t) &= f(t,y(t),g(t-\beta(t))) \\ y(t_0) &= g(t_0) \end{aligned}$$

to be solved on the interval $[t_0,t_1]$, where t_1 is chosen so that $t_1 - \beta(t_1) = t_0$. This equation has a solution if f and g are continuous and the solution is unique if $f(t,y,x)$ satisfies a Lipschitz condition in its second argument for t near t_0, for y near $y(t_0)$ and x near

$g(t_0-\beta(t_0))$. The following theorem, whose proof can be found in El'sgol'ts [9, p.20] gives a more general theorem.

Theorem 2.1

Consider the equation

$$(2.3)\qquad y'(t) = f(t,y(t),y(t-\beta(t)))$$

$$y(t) = g(t) \text{ on an initial set } E_{t_0}.$$

Suppose $\beta(t)$ is continuous and non-negative, $g(t)$ is continuous on E_{t_0} and f satisfies a Lipschitz condition in all arguments beginning with the second. Then there exists a unique continuous solution $y_g(t)$ to (2.3) for $t > t_0$. □

We note that in general for (2.3) (unless the initial function $g(t)$ satisfies some very special conditions [1, p.51]) that discontinuities can occur in the higher order derivatives at those points t_k such that $t_{k+1}-\beta(t_{k+1}) = t_k$, even for "smooth" f. The discontinuity at t_k can occur in the $k+1$ derivative, but the lower order derivatives will be continuous at t_k. Thus the solution smooths itself out for increasing t. The discontinuities in the lower order derivatives cause problems for numerical methods and must be accounted for as we shall see later.

Numerical Methods for D.D.E's

To simplify the analysis and eventually the coding of a method, we will consider differential difference equations of the form

$$y'(t) = f(t,y(t),y(t-\beta)) \qquad t > 0$$

(2.4)

$$y(t) = g(t) \qquad t \in [-\beta,0]$$

where y is a scalar, $\beta > 0$, and f and g satisfy the hypothesis of Theorem 2.1. As Wiederholt [20, p.3] notes, this type of equation describes physical systems in many different areas such as rocket propulsion and control theory and hence includes a reasonable class of problems.

Recall that L.M.S. methods are based on a linear relationship among $\{y_n\}$ and the values of the function $\{f(t_n)\}$ at the points $\{t_n\}$ using y_n as an approximation to $y(t_n)$ to evaluate f. Define $f_n = f(t_n,y_n,y_n^*)$ where y_n^* is an approximation to $y(t_n-\beta)$. Then the formula (1.3) for L.M.S. method can be applied directly to solving (2.4) provided we prescribe how to obtain y_n^*. Clearly as $y_n^* \sim y(t_n-\beta)$ we must save sufficient past values in order to obtain an accurate approximation y_n^* to $y(t_n-\beta)$. Note that if the step size is small compared to β then this can require saving many values.

If the step size is chosen so that $\beta = mh$ then $t_n-\beta = (n-m)h$ coincides with a previous node and we can use y_{n-m} as an approximation to y_n^*.

To obtain an arbitrary step size Cryer [6] suggests choosing $m \in I^+$ (set of positive integers) and $u \in [0,1)$ such that $\beta = (m-u)h$. Then $t_n-\beta = (n-m)h + uh$. Cryer suggests obtaining the value of y at t_{j+u} from the $\ell+1$ values of y at $t_{j+1},t_j,\ldots,t_{j-\ell+1}$. Let E denote the operator defined by $Ey_j = y_{j+1}$ and then take the approximation to $y(t_{j+u})$ to be

$E^{-\ell+1}\gamma(E;u)y_j$ where $\ell \in I^+$, γ is a polynomial in E of degree at most ℓ, and whose coefficients depend on u. We require that γ be exact if $u = 0$ or $y(t)$ is a constant. That is $\gamma(E,0) = E^{\ell-1}$ and $\gamma(1,u) \equiv 1$. Normally γ is taken to be an interpolating polynomial since such a polynomial has these properties.

Example

The well-known Euler method $y_{n+1} = y_n + hf_n$ could be used with linear interpolation, whence $\gamma(E,u) = uE + 1 - u$.

Convergence of Numerical Methods

The above description allows us to derive numerical methods for (2.4) by modifying numerical methods for O.D.E's. This is done in the hope that properties of methods for O.D.E's such as convergence and stability will carry over to solving (2.4).

Taverini [19] has shown for the L.M.S. method $\{\rho,\sigma\}$ [6] that

$$\lim_{\substack{t-t_0=nh \\ \beta/m=h\to 0}} y_n = y(t)$$

where $y(t)$ is the solution of (2.4) subject to the hypothesis of Theorem 2.1 if and only if the method $\{\rho,\sigma\}$ is convergent for O.D.E's. This is a nice property, since we need only consider convergent methods for O.D.E's, which have been well studied. For a discussion of the case $\beta = (m-u)h$ the reader is referred to Taverini [19].

Recall that the definition of local truncation error and order of a method was independent of the differential equation and thus the same definitions can be used for methods in D.D.E's. Neves [17] has shown that when using a method whose local truncation order is $O(h^{p+1})$ one must use an interpolation formula whose order is $O(h^p)$ in order to preserve the global order of convergence of the original O.D.E. method. That is, a method of order p must use an interpolation formula whose error is $O(h^{p+1})$ and hence must use at least p+1 points, if only function values are saved. In the case of equation (2.4) one simply needs to save at least p+1 function values which is always possible.

Also we must handle the problem of possible discontinuities in the higher order derivatives. One way of handling this problem is to modify the method using jumps in y(t) and lower order derivatives of y(t). Instead of Taylor series, one uses an extended Taylor series due to Zverkina [22]. For an English translation of this paper and a description of this technique the reader is referred to [13].

The other way of overcoming the problem is to use a variable order, variable step algorithm such as [11,p.158] and include the points of discontinuity in the set of mesh points. For equation (2.4) we know that smoothing of the solution occurs and that we need only include the p+1 points $t_0 + j\beta$, $j = 0,\dots,p$ in the set of nodes t_n where p is the maximum order of the method being used.

Stability of Numerical Methods for D.D.E's

The stability of numerical methods for D.D.E's has been studied previously by Brayton and Willoughby [3], Wiederholt [20], Brayton [2] and Cryer [6].

Brayton and Willoughby show by means of an example that when Euler's method is used to solve a neutral differential difference equation (a neutral D.D.E. has the form $y'(t) = f(t,y(t),y(t-\beta),y'(t-\beta))$ the range of values of the step size h for which the method is stable can differ from the corresponding range of values of h for O.D.E's.

Wiederholt considers the linear D.D.E.

$$(2.5) \qquad \begin{aligned} y'(t) &= qy(t-\beta) \qquad && \beta,t > 0 \\ y(t) &= g(t) && t \in [-\beta,0]. \end{aligned}$$

Applying a L.M.S. method $\{\rho,\sigma\}$ to (2.5) with $\beta = mh$ we obtain the linear difference equation

$$\sum_{j=0}^{k} \alpha_j \, y_{n+j} = hq \sum_{j=0}^{k} \beta_j \, y_{n+j-m}.$$

The associated characteristic polynomial is

$$(2.6) \qquad C(z,q,m) = z^m\rho(z) - hq\sigma(z).$$

Wiederholt determines numerically for m = 1,2,3 and for specific choices of ρ,σ the set of values of q for which the zeros of $C(z,q,m)$ lie inside the unit circle.

Cryer considers equation (2.5) with q real since it is known (Bellman and Cooke [1, p.444]) that $y(t) \to 0$ as $t \to +\infty$ for all initial functions g(t) if and only if $\beta q \in (-\pi/2,0)$. Cryer then defines a method $\{\rho,\sigma\}$ with $\beta = mh$ to be $\underline{DA_0 \text{ stable}}$ if the numerical solution converges to zero asymptotically for all $\beta q \in (-\pi/2,0)$, $m \in I^+$ and all initial functions

$g(t)$. Cryer remarks that he does not consider equation (2.5) with complex q or the more general equation $y'(t) = py(t) + qy(t-\beta)$ because of a lack of results on the asymptotic behaviour of D.D.E. with complex coefficients. As we shall see in Chapter 3 some of these more general equations can be considered.

Cryer then generalizes the above definition to an arbitrary step size by using the technique discussed before. The method $\{\rho,\sigma,\gamma\}$ [6] is called <u>GDA_0 stable</u> if the numerical solution converges to zero asymptotically for all $\beta q \in (-\pi/2,0)$, $m \in I^+$, $u \in [0,1)$ and all initial functions $g(t)$. The corresponding characteristic polynomial is

$$(2.7) \qquad C(z,q,m,u) = z^{m+\ell-1}\rho(z) - hq\sigma(z)\gamma(z;u)$$

so that the method $\{\rho,\sigma,\gamma\}$ is GDA_0 stable iff all the zeros of $C(z,q,m,u)$ lie in the unit circle for all $\beta q \in (-\pi/2,0)$, $m \in I^+$ and $u \in [0,1)$.

Cryer then proves the following interesting theorems.

<u>Theorem 2.2</u>

If the method $\{\rho,\sigma\}$ is DA_0 stable then it is zero-stable. Similarly, if the method $\{\rho,\sigma,\gamma\}$ is GDA_0 stable then $\{\rho,\sigma\}$ is zero stable. Furthermore, if the k-step method $\{\rho,\sigma\}$ is DA_0 stable and of order k it is implicit. Similarly, if the k-step method $\{\rho,\sigma,\gamma\}$ is GDA_0 stable and of order k it is implicit.

This is a good theorem because much is known [14] about the stability of methods for O.D.E's, and justifies the belief that we should modify methods for O.D.E's to D.D.E's.

Theorem 2.3

The Backward Euler method and the Trapezoidal rule [14, p.15] used with linear interpolation are GDA_0 stable.

Theorem 2.4

The modified trapezoidal rule $y_{n+1} - y_n = hf_{n+1/2}$ where $f_{n+1/2}$ is computed by linear interpolation is DA_0 stable but not GDA_0 stable.

The modified trapezoidal rule belongs to the family of modified Adams methods considered by Zverkina [22] which are particularly useful for delay differential equations since they preserve the order even when stepping over discontinuities and of course when the delay is small compared to the step size h, as is the case in equations with harmless delay [8], one will step over discontinuities. It is surprising that these methods are not as stable as the ordinary trapezoidal rule.

The following theorem enables us to prove results on the location of the zeros of (2.6) and (2.7) by only considering those zeros on the unit circle.

Theorem 2.5

Let the zeros of $\rho(z)$ other than $z = 1$ be inside the unit circle and let $\{\rho,\sigma\}$ be convergent. Then $\{\rho,\sigma\}$ is DA_0 stable iff $\forall \beta q \in (-\pi/2,0)$, $m \in I^+$, (2.6) has no zeros on the unit circle. Furthermore, $\{\rho,\sigma,\gamma\}$

is GDA_0 stable iff $\beta q \in (-\pi/2,0)$, $m \in I^+$, $u \in [0,1)$, (2.7) has no zeros on the unit circle.

This theorem greatly simplifies the work involved in proving stability results about methods. Even so (Cryer [6]) the proofs are long and tedious. However, no other technique seems to be known for proving stability results.

CHAPTER 3

NEW RESULTS ON THE ASYMPTOTIC BEHAVIOUR OF A LINEAR D.D.E

Introduction

As Cryer has noted in [6], he did not consider the more general equation

(3.1) $$y'(t) = py(t) + qy(t-\beta)$$

where y is a scalar, p and q are complex constants and $\beta > 0$, in generalizing the definition of A-stability, because of a lack of results on the asymptotic behaviour of the solutions to (3.1) where p and q are complex. It is possible in the case p = 0, q an arbitrary complex number to completely characterize the asymptotic behaviour of (3.1) in terms of a simple condition on q. This permits us to easily generalize the definition of A-stability to differential difference equations, for the case p = 0, as we shall see later. In the case where p and q are arbitrary complex numbers it is not yet possible to completely characterize the asymptotic behaviour of the solutions to (3.1) in terms of simple conditions on p and q. However, we can give a simple sufficient condition on p and q to ensure that all solutions to (3.1) converge to zero as $t \to +\infty$. We characterize the asymptotic behaviour in the following two theorems.

Theorem 3.1

Consider the equation

$$(3.2) \qquad y'(t) = qy(t-\beta)$$

$$(3.3) \qquad y(t) = g(t) \quad \text{on} \quad [0,\beta],$$

where $y(t)$ is a scalar, $q = \gamma\exp(i\phi)$, $\phi \in [0,2\pi)$ is a complex number, and $g(t) \in C^0[0,\beta]$.

Then all continuous solutions to (3.2), (3.3) satisfy $\lim_{t\to+\infty} x(t) = 0$ if

$$(3.4) \qquad \mathrm{Re}(q) < 0 \quad (\phi \in (\pi/2,\ 3\pi/2)) \text{ and}$$

$$0 < \beta\gamma < \min\{\phi-\pi/2,\ 3\pi/2-\phi\}.$$

Proof

It is first shown that a necessary and sufficient condition for all continuous solutions to (3.2) to approach zero as $t \to +\infty$ is that all roots of the corresponding characteristic equation have negative real parts. That is, we need only show that all exponential solutions of the form $\exp(\alpha t)$ approach zero as $t \to +\infty$.

We cannot apply the theorem in [1,p.115] on the asymptotic behaviour of linear D.D.E. since it is only valid for real coefficients. However, writing $y(t) = y_1(t) + iy_2(t)$, $q = q_1+i\, q_2$ where $y_1(t)$, $y_2(t)$, q_1, q_2 are real, and equating real and imaginary parts of equation (3.2) gives

$$\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = Q \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

where Q is the real matrix $\begin{bmatrix} q_1 & -q_2 \\ q_2 & q_1 \end{bmatrix}$.

The corresponding characteristic equation [1,p.166] for this system of D.D.E. is $\det|Is-Q\exp(-\beta s)| = 0$ which yields the two equations $s = q\exp(-\beta s)$ and $s = \bar{q}\exp(-\beta s)$. If these equations have roots with negative real parts then the solution to the vector equation decays to zero are $t \to +\infty$ [1,p.190].

As the characteristic equation for (3.2) is $s = q\exp(-\beta s)$ it follows that we need only consider exponential solutions to equation (3.2).

To simplify the algebra, consider exponential solutions of the form $\exp(\alpha qt)$, where $\alpha = r\exp(i(\theta+2k\pi))$, $\theta \in [-\phi,-\phi+2\pi)$, $k \in I$ (set of all integers), $r > 0$. Then $\exp(\alpha qt) = \exp(r\gamma\exp(i(\theta+\phi+2k\pi))t) = \exp(r\gamma t\cos(\theta+\phi))\exp(ir\gamma t\sin(\theta+\phi))$, so that the modulus of the exponential solution is $\exp(r\gamma t\cos(\theta+\phi))$. Clearly, the asymptotic behaviour of the exponential solution is determined by the sign of $\cos(\theta+\phi)$ and we need only show $\cos(\theta+\phi) < 0$.

The characteristic equation for 3.2 is given by $\alpha q \exp(\alpha q t) = q \exp(\alpha q(t-\beta))$. That is, $\alpha = \exp(-\alpha q\beta)$. Since there cannot be a non-zero constant solution, we may assume $\alpha \neq 0$. Solving for β we get

$$\begin{aligned}
\beta &= \ln \alpha/(-\alpha q) \\
&= (\bar{\alpha} \ln \alpha)/(-q|\alpha|^2) \\
&= -[\ln r+i(\theta+2k\pi)]r \exp(-i(\theta+2k\pi))/(\gamma r^2 \exp(i\phi)) \\
&= -[\ln r+i(\theta+2k\pi)][\cos(\theta+\phi)-i \sin(\theta+\phi)]/(\gamma r).
\end{aligned}$$

$$\therefore\ \mathcal{R}e(\beta) = -[\ln r \cos(\theta+\phi)+(\theta+2k\pi)\sin(\theta+\phi)]/(\gamma r).$$

$$\mathcal{I}m(\beta) = -[(\theta+2k\pi)\cos(\theta+\phi)-\ln r \sin(\theta+\phi)]/\gamma r.$$

As β is real, then $\mathcal{I}m(\beta) = 0$; hence $(\theta+2k\pi)\cos(\theta+\phi) = \ln r \sin(\theta+\phi)$. If $\sin(\theta+\phi) = 0$, then $(\theta+2k\pi) = 0$, so that $\cos(\theta+\phi) = \cos(\phi-2k\pi) = \cos \phi < 0$. We may then assume $\sin(\theta+\phi) \neq 0$.

Now $r(\theta) = \exp((\theta+2k\pi)\cot(\theta+\phi))$ and

$$\begin{aligned}
\beta &= -[(\theta+2k\pi)\cot(\theta+\phi)\cos(\theta+\phi)+(\theta+2k\pi)\sin(\theta+\phi)]/(\gamma r) \\
&= -(\theta+2k\pi)/(\gamma r \sin(\theta+\phi)) \\
&= A(\theta)/(\gamma r(\theta)) \text{ where } A(\theta) = -(\theta+2k\pi)/\sin(\theta+\phi).
\end{aligned}$$

Clearly as $\beta > 0$, $A(\theta) > 0$.

If $\sin(\theta+\phi) \to 0$ <u>and</u> $\theta+2k\pi \to 0$, then $\phi = \pi$. This is the case if q is real and we have seen this result in Chapter 2. We may assume $\phi \neq \pi$ and $A(\theta) \to \pm\infty$ as $\sin(\theta+\phi) \to 0$.

Case (i) ($\theta+2k\pi > 0$)

Now $\sin(\theta+\phi) < 0$ and $\theta + \phi \in (\pi,2\pi)$.

As $\theta \to \pi^+ - \phi$, $\beta \to 0$

$\theta \to 3\pi/2 - \phi$, $\beta \to 2k\pi + (3\pi/2-\phi)/\gamma$

$\theta \to 2\pi - \phi$, $\beta \to +\infty$.

Using the following lemma we can complete the proof.

Lemma 3.1

The function $\beta(\theta) = A(\theta)/(\gamma r(\theta))$ is a strictly increasing function of θ for $\theta + \phi \in (\pi,2\pi)$.

Proof We need only show that $\beta'(\theta) = d\beta/d\theta > 0$.

$$\gamma\beta'(\theta) = (A'(\theta)r(\theta)-r'(\theta)A(\theta))/r^2(\theta) \text{ and}$$

$$A'(\theta) = [(\theta+2k\pi)\cos(\theta+\phi)-\sin(\theta+\phi)]/\sin^2(\theta)$$

$$r'(\theta) = r(\theta)[\cos(\theta+\phi)\sin(\theta+\phi)-(\theta+2k\pi)]\sin^2(\theta+\phi).$$

$$\therefore \quad \gamma\beta'(\theta) = -[(\theta+2k\pi)^2+\sin^2(\theta+\phi)-2\sin(\theta+\phi)\cos(\theta+\phi)(\theta+2k\pi)]/\sin^3(\theta+\phi).$$

As $(\theta+2k\pi)^2+\sin^2(\theta+\phi)-2\sin(\theta+\phi)\cos(\theta+\phi)(\theta+2k\pi)$

$$> (\theta+2k\pi)^2+\sin^2(\theta+\phi)\cos^2(\theta+\phi)-2\sin(\theta+\phi)\cos(\theta+\phi)(\theta+2k\pi)$$

$$= [(\theta+2k\pi)-\sin(\theta+\phi)\cos(\theta+\phi)]^2 \geq 0,$$

then clearly $\beta'(\theta) > 0$. □

We can now easily complete the proof of the theorem. As β is strictly increasing for $\theta + \phi \in (\pi,2\pi)$, then $0 < \beta\gamma < (3\pi/2-\phi)$ implies $\theta + \phi \in (\pi,3\pi/2)$ and hence $\cos(\theta+\phi) < 0$.

Case (ii) ($\theta+2k\pi < 0$)

Now $\sin(\theta+\phi) > 0$ and $(\theta+\phi) \in [0,\pi)$.

As $\theta \to 0^+ - \phi$, $\beta \to +\infty$

$\theta \to \pi/2 - \phi$, $\beta \to -(\pi/2-\phi+2k\pi)/\gamma$

$\theta \to \pi^- - \phi$, $\beta \to 0$.

It is easy to see from the proof of the above lemma that β is a strictly decreasing function of θ for $\theta + \phi \in [0,\pi)$. Then $0 < \beta\gamma < \phi - \pi/2$ implies $\theta + \phi \in (\pi/2,\pi)$ and hence $\cos(\theta+\phi) < 0$.

Therefore, if $0 < \beta\gamma < \min\{3\pi/2-\phi, \phi-\pi/2\}$, $\cos(\theta+\phi) < 0$. □

Remarks

The criterion on q in Theorem 3.1 is sharp in the sense that if q does not satisfy (3.4) then we can find an exponential solution to (3.3) which does not converge to zero as $t \to +\infty$. That is, we simply choose $\theta+2k\pi$ so that $\cos(\theta+\phi) = 0$.

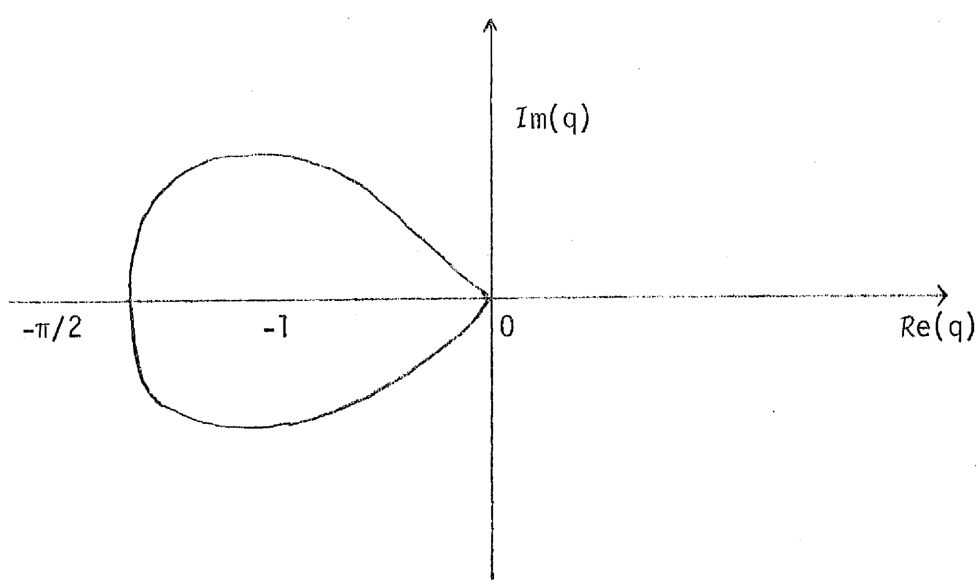


Fig.3.1 Region of q-stability for $\beta = 1$

The region of stability of equation (3.2), shown in Figure 3.1 for $\beta = 1$, is determined by plotting the boundary. For $Im(q) > 0$, that is $\phi \in (\pi/2, \pi)$ we have $\gamma = \phi - \pi/2$ and thus $Re(q) = (\phi-\pi/2)\cos\phi$ and $Im(q) = (\phi-\pi/2)\sin\phi$. Therefore, we can parametrize the boundary of the region in terms of ϕ.

<u>Theorem 3.2</u>

Consider the equation

(3.5) $$y'(t) = py(t) + qy(t-\beta)$$

where y is a scalar, p and q are complex constants and $\beta > 0$. All exponential solutions (hence all solutions) converge to zero as $t \to +\infty$ under the condition

(3.6) $$Re(p) < -|q|.$$

<u>Proof</u> As usual, we may write $p = \rho\exp(i\psi)$, $q = \gamma\exp(i\phi)$ where $\rho, \gamma > 0$. Condition (3.6) can be written as $-\rho\cos\psi > \gamma$.

To simplify the algebra consider exponential solutions of the form $\exp(\alpha q+p)t$ with $\alpha = (\rho/\gamma) r \exp(i(\theta+2k\pi))$ where $\theta \in [0,2\pi)$, $k \in I$, $r > 0$. Then

$$\alpha q+p = \rho r \exp(i(\theta+\phi+2k\pi)) + \rho\exp(i\psi)$$
$$= \rho[\cos\psi + r\cos(\theta+\phi) + i(\sin\psi + r\sin(\theta+\phi))].$$

If $z = \cos\psi + r\cos(\theta+\phi) + i(\sin\psi + r\sin(\theta+\phi))$ then $\alpha q + p = \rho z$ so that the modulus of the exponential solution is $\exp\{\rho\, Re(z)t\}$. The asymptotic behaviour of the solution is determined by the sign of $Re(z)$ and we need only show $Re(z) < 0$.

The characteristic equation of (3.5) is

$(\alpha q+p)\exp(\alpha q+p)t = p \exp(\alpha q+p)t + q \exp[(\alpha q+p)(t-\beta)]$.

That is $\alpha = \exp[-(\alpha q+p)\beta]$.

$\therefore$ $\ell n(\rho r/\gamma) + i(\theta+2k\pi) = -\rho z\beta$.

$$|z|^2 = \cos^2\psi + 2r\cos\psi\cos(\theta+\phi) + r^2\cos^2(\theta+\phi)$$
$$+ \sin^2\psi + 2r\sin\psi\sin(\theta+\phi) + r^2\sin^2(\theta+\phi)$$
$$= 1 + 2r\cos(\theta+\phi-\psi) + r^2 \geq (1-r)^2.$$

If $z = 0$ then $r = 1$ and $\ell n(\rho/\gamma) = 0$ which contradicts condition (3.6), so we may assume z is nonzero.

Solving for β, we get

$$\beta = -\bar{z}[\ell n(\rho r/\gamma) + i(\theta+2k\pi)]/(\rho|z|^2).$$

$$Re(\beta) = -\{\ell n(\rho r/\gamma)[\cos\psi+r\cos(\theta+\phi)]+(\theta+2k\pi)[\sin\psi+r\sin(\theta+\phi)]\}/(\rho|z|^2).$$

$$Im(\beta) = -\{-\ell n(\rho r/\gamma)[\sin\psi+r\sin(\theta+\phi)]+(\theta+2k\pi)[\cos\psi+r\cos(\theta+\phi)]\}/(\rho|z|^2).$$

As β is real then $Im(\beta) = 0$, hence $(\theta+2k\pi)(\cos\psi+r\cos(\theta+\phi)) = \ell n(\rho r/\gamma)[\sin\psi+r\sin(\theta+\phi)]$.

<u>Case (i) $(\sin\psi + r\sin(\theta+\phi) = 0)$</u>

Either $(\theta+2k\pi) = 0$ or $\cos\psi + r\cos(\theta+\phi) = 0$.

If $\cos\psi = -r\cos(\theta+\phi)$ then, since $\sin\psi = -r\sin(\theta+\phi)$, squaring, adding and solving for r gives $r = 1$, which as we saw before is impossible. Thus we cannot have $\cos\psi + r\cos(\theta+\phi) = 0$.

If $(\theta+2k\pi) = 0$ then $\theta = 0$ and $\beta = -\ell n(\rho r/\gamma)/[\rho(\cos\psi+r\cos\phi)] = -\ell n(\rho r/\gamma)/[\rho\cos\psi(1+r\cos\phi/\cos\psi)]$. Condition (3.6) implies $-Re(p) > |q| \geq Re(q)$, hence $-\rho\cos\psi > \gamma\cos\phi$. If $\cos\phi < 0$ then $Re(z) = \cos\psi + r\cos(\theta+\phi) = \cos\psi + r\cos\phi < 0$ and the exponential solution decays. If $\cos\phi > 0$ then

$\cos\psi/\cos\phi < -\gamma/\rho$. For $0 < r < \gamma\,\rho$, $\beta < 0$

$$\gamma/\rho < r < -\cos\psi/\cos\phi,\ \beta > 0$$

$$r > -\cos\psi/\cos\theta,\ \beta < 0$$

so that we must have $\gamma/\rho < r < -\cos\psi/\cos\phi$, which implies $\mathcal{R}e(z) < 0$. Therefore, the exponential solution decays.

Case (ii) $(\sin\psi + r\sin(\theta+\phi) \neq 0)$

$$\ell n(\rho r/\gamma) = (\theta+2k\pi)[\cos\psi+r\cos(\theta+\phi)]/[\sin\psi+r\sin(\theta+\phi)]$$

and $\beta = (\theta+2k\pi)/[-\rho(\sin\psi+r\sin(\theta+\phi)]$

$= -A(r,\theta)/\rho$ where $A(r,\theta) = (\theta+2k\pi)/[\sin\psi+r\sin(\theta+\phi)]$.

As $\beta > 0$ we require $A(r,\theta) < 0$. The equation derived from putting $\mathcal{I}m(\beta) = 0$ becomes

$$A(r,\theta)[\cos\psi+r\cos(\theta+\phi)] = \ell n(\rho r/\gamma).$$

Suppose, if possible, that $\cos\psi + r\cos(\theta+\phi) \geq 0$. Then $\ell n(\rho r/\gamma) < 0$; $r < \gamma/\rho$. $\cos(\theta+\phi) \geq -\cos\psi/r \geq -\rho\cos\psi/\gamma = -\mathcal{R}e(p)/|q| > 1$. This contradicts condition (3.6). Thus, we must have $\cos\psi + r\cos(\theta+\phi) < 0$. □

Remarks

Note that (3.6) is a sufficient condition for the solution to converge to zero as $t \to +\infty$. However, the solution can converge to zero when condition (3.6) is not satisfied. It is difficult to specify a good condition for solutions to (3.5) to converge to zero as $t \to +\infty$.

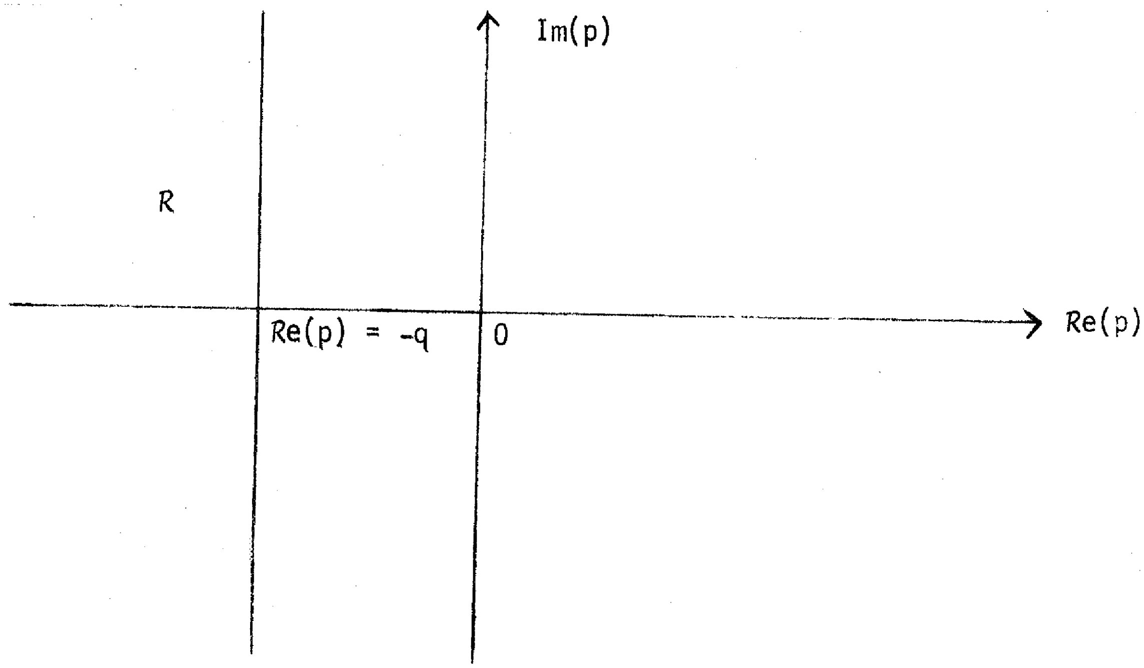


Fig.3.2 Region of p-stability

Theorem 3.3

Consider the equation

(3.7) $$y'(t) = qy(t-\beta) + f(t)$$

where y is a scalar, q a complex constant, $\beta > 0$, and f is a continuous function satisfying $|f(t)| \le c\exp(-\alpha t)$, α and c positive constants. Then all continuous solutions to (3.7) satisfy $|y(t)| < c^*\exp(-\alpha^* t)$ where c^*, α^* are positive constants and q satisfies condition (3.4).

Proof Clearly $\int_t^{+\infty} |f(t)|dt$ is bounded, so applying Theorem 3.1 along with the theorem from [1, p.361] yields the desired result. □

We can use this theorem to determine the behaviour of a system of D.D.E's which depend only on the past solution and such that each component has the same lag.

Theorem 3.4

Consider the vector equation

$$(3.8) \qquad y'(t) = Qy(t-\beta)$$

where $\beta > 0$ and Q is an $n \times n$ complex matrix. Suppose the eigenvalues of Q are $q_j = \gamma_j \exp(i\theta_j)$, $j = 1,\ldots,n$. Then all solutions to (3.8) converge to zero under the conditions

$$(3.9) \qquad 0 < \beta\gamma_j < \min\{3\pi/2-\phi_j,\ \phi_j-\pi/2\} \quad j = 1,\ldots,n.$$

Proof By [20, p.11], $\exists$ a matrix $R \ni RQR^{-1}$ has the form

$$\begin{pmatrix} J_0 & & & \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ & & & & J_r \end{pmatrix}$$

where $$J_0 = \begin{pmatrix} q_1 & & & 0 \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & q_s \end{pmatrix}$$

$$J_1 = \begin{pmatrix} q_{s+1} & 1 & & 0 \\ & q_{s+1} & 1 & \\ & & \ddots & \ddots \\ & & & 1 \\ 0 & & 0 & q_{s+1} \end{pmatrix}_{m \times m}$$

where $m > 1$ is the multiplicity of the eigenvalue q_{s+1} and $J_2,\ldots,J_r$ have the same form as J_1. Clearly, the variables $y_1(t),\ldots,y_s(t)$ become decoupled and we may apply Theorem 3.1 to each component to show that $y_i(t) \to 0$ as $t \to +\infty$ for $i = 1,\ldots,s$.

Let $\underset{\sim}{y}^*(t) = [y_{s+1}(t),\ldots,y_{s+m}(t)]^T$ and consider the equation

(3.10) $\quad \underset{\sim}{y}^*(t) = J_1 y^*(t-\beta)$.

Clearly, the last equation of (3.10) is $y_{s+m}(t) = q_{s+1}y_{s+m}(t-\beta)$ and so by Theorem 3.1 $y_{s+m}(t)$ is exponentially decaying. The second last equation of (3.10) is $y_{s+m-1}(t) = q_{s+1}y_{s+m-1}(t-\beta) + y_{s+m}(t-\beta)$ which has the form of equation (3.7) if we take $f(t) = y_{s+m}(t-\beta)$. Thus, $y_{s+m-1}(t)$ is an exponentially decaying function. Similarly, we can show that all the components of $\underset{\sim}{y}^*(t)$ are exponentially decaying and similarly all the components of $\underset{\sim}{y}(t)$ are exponentially decaying. □

We might hope to give a similar generalization to a system of D.D.E's for equation (3.5). However, this can be done only by imposing fairly restrictive conditions.

Theorem 3.5

Consider the vector equation

(3.11) $\underset{\sim}{y}'(t) = P\underset{\sim}{y}(t) + Q\underset{\sim}{y}(t-\beta)$

where P, Q are $n \times n$ complex matrices having eigenvalues p_i, q_i respectively, $i = 1,\ldots,n$. Suppose P, Q are simultaneously diagonalizable, that is, $\exists$ a matrix R $\ni$ $RPR^{-1} = \text{diag}[p_1,\ldots,p_n]$ and $RQR^{-1} = \text{diag}[q_1,\ldots,q_n]$, and further that each pair p_i, q_i satisfies condition (3.6) or $p_i = 0$ and q_i satisfies (3.4). Then all continuous solutions to (3.11) converge to zero as $t \to +\infty$.

Proof The fact that P, Q are simultaneously diagonalizable allows us to reduce (3.11) to a decoupled system and we may apply either Theorem 3.1 or 3.2 to each component to complete the proof. □

CHAPTER 4

NEW RESULTS ON THE STABILITY OF NUMERICAL METHODS

Theorems 3.1 and 3.2 in Chapter 3 give a more complete characterization of the asymptotic behaviour of a linear D.D.E. We can use these theorems to generalize the definitions of DA_0 stability and GDA_0 stability of Cryer [6]. As we shall see later, we can also generalize Theorems 2.2 and 2.5. Using Theorems 3.1 and 3.2 we get the following definitions:

Definition 4.1

A L.M.S. method $\{\rho,\sigma\}$ with $\beta = mh$ is called Q-stable if all the roots of the characteristic equation (2.6) $C(z,q,m) = 0$ are inside the unit circle whenever q satisfies the conditions (3.4) and $m \in I^+$.

Definition 4.2

A L.M.S. method $\{\rho,\sigma,\gamma\}$ with $\beta = (m-u)h$ is called GQ-stable if all the roots of the characteristic equation (2.7) $C(z,q,m,u) = 0$ are inside the unit circle whenever q satisfies the conditions (3.4), $m \in I^+$ and $u \in [0,1)$.

Applying the L.M.S. method (1.3) to equation (3.1) with $\beta = mh$ yields the difference equation

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = ph\sum_{j=0}^{k} \beta_j y_{n+j} + qh\sum_{j=0}^{k} \beta_j y_{n-m+j}.$$

The associated characteristic polynomial is given by

$$CP(z,p,q,m) = z^m(\rho(z) - ph\,\sigma(z)) - qh\,\sigma(z). \tag{4.1}$$

Definition 4.3

A L.M.S. method $\{\rho,\sigma\}$ with $\beta = mh$ is called P-stable if all the roots of the characteristic polynomial (4.1) are inside the unit circle whenever p,q satisfy condition (3.6) and $m \in I^+$.

Applying the L.M.S. method (1.3) to equation (3.1) with $\beta = (m-u)h$ yields the difference equation

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = ph \sum_{j=0}^{k} \beta_j y_{n+j} + qh \sum_{j=0}^{k} \beta_j E^{-\ell+1} \gamma(E;u) y_{n-m+j}.$$

The associated characteristic polynomial is given by

$$\text{(4.2)} \qquad CP(z,p,q,m,u) = z^m(\rho(z) - ph\,\sigma(z)) - qh\gamma(z,u)\sigma(z).$$

Definition 4.4

A L.M.S. method $\{\rho,\sigma,\gamma\}$ with $\beta = (m-u)h$ is called GP-stable if all roots of the characteristic polynomial (4.2) are inside the unit circle whenever p,q satisfy condition (3.6), $m \in I^+$ and $u \in [0,1)$.

Using these definitions we can state the following theorems which are just a generalization of those in [6].

Theorem 4.1

If a L.M.S. method $\{\rho,\sigma\}$ is Q-stable it is zero-stable. Furthermore, if the method $\{\rho,\sigma,\gamma\}$ is GQ-stable, it is zero-stable.

Proof Clearly any method which is Q-stable must be DA_0 stable and hence by Theorem 2.2 is zero-stable. Similarly, the result is proved for GQ-stability.

Theorem 4.2

If the k step method $\{\rho,\sigma\}$ is Q-stable and of order k it is implicit. Furthermore, if the k step method $\{\rho,\sigma,\gamma\}$ is GQ-stable and of order k it is implicit.

Proof Again any such methods are DA_0 and GDA_0 stable respectively and the result follows by Theorem 2.2.

Theorem 4.3

Any method which is P-stable is A-stable. Furthermore, any k step method which is order k and P-stable, is implicit.

Proof The result that a P-stable method is A-stable follows immediately by letting $q \to 0$ in equation (4.1). As the method is of order k and A-stable it must be implicit [7]. □

It would be nice if Theorem 4.3 was also true for Q-stable and GQ-stable methods, but it is not known to date if this theorem is true or false and definite results are difficult to obtain.

One of the basic methods [6] for showing that the stability polynomials (2.6), (2.7) have all their roots inside the unit circle is to first show that for small q inside the region defined by (3.6) that (2.6) and (2.7) have roots inside the unit circle and then show that there are no roots on the unit circle. The following theorem allows us to consider only the zeros of a stability polynomial on the unit circle.

Theorem 4.4

Let the zeros of $\rho(z)$ other than $z = 1$ lie inside the unit disc, and let $\{\rho,\sigma\}$ be convergent. Then $\{\rho,\sigma\}$ is Q-stable, if and only if $\forall q$ satisfying (3.4) and $m \in I^+$ the characteristic polynomial $C(z,q,m)$ (2.6) has no zeros on the unit circle.

Proof Let $z_1(q),\ldots,z_{m+k}(q)$ be the zeros of $C(z,q,m)$ with $z_1(0) = 1$ and $|z_j(0)| < 1$ for $j > 1$. The $z_j(q)$ are continuous functions of the variable q since the roots of a polynomial are continuous functions of its coefficients [15,p.3]. Thus, $\exists$ a region about the origin so that $|z_j(q)| < 1$ for $j > 1$.

As $\{\rho,\sigma\}$ is convergent, we have $\rho(1) = 0$ and $\rho'(1) = \sigma(1) \neq 0$. As $z_1(q)$ is a simple zero we may differentiate the equation $C(z_1(q),q,m) = 0$ with respect to q. This yields

$$\rho m z_1^{m-1} \frac{dz_1}{dq} + z_1^m \frac{d\rho}{dz_1} \frac{dz_1}{dq} - h\sigma(z_1) - hq \frac{d\sigma}{dz_1} \frac{dz_1}{dq} = 0.$$

Setting $q = 0$ in the above equation gives

$$\sigma(1)\left[\frac{dz_1(0)}{dq} - h\right] = 0,$$

hence $$\frac{dz_1(0)}{dq} = h.$$

Now $$z_1(q) = 1 + \frac{dz_1}{dq}(0)q + 0(q^2)$$

$$= 1 + hq + 0(q^2),$$

so that for small q in the region defined by (3.4), that is for small q belonging to the region of stability of $y'(t) = q\, y^{(t-\beta)}$, $|z_1(q)| < 1$.

As the $z_j(q)$ are continuous functions of q, then, if there is no root on the unit circle for q satisfying (3.6), there can be no roots outside the unit circle.

Theorem 4.5

Let the zeros of $\rho(z)$ other than z = 1 be inside the unit disc and let $\{\rho,\sigma\}$ be convergent. Then $\{\rho,\sigma,\gamma\}$ is GQ-stable if and only if $\forall q$ satisfying (3.4), $m \in I^+$ and $u \in [0,1)$ the characteristic polynomial C(z,q,m,u) has no zeros on the unit circle.

Proof The details of the proof are the same as for Theorem 4.4.

Theorem 4.6

The first and second order backward differentiation methods are P-stable.

Proof The backward Euler method (first order backward differentiation method) applied to (3.1) with β = mh gives the difference equation $y_{n+1}(1-hp) - y_n - hq\, y_{n-m+1}$. The associated characteristic polynomial is $(1-hp)z^m - z^{m-1} - \beta q/m = 0$.

Let $P(z) = z^{m-1}[z(1-hp)-1]$ and $Q(z) = -\beta q/m$. Clearly, both P and Q are analytic inside and on the unit circle, with P(z) having m zeros inside the unit circle for $Re(p) < 0$. On the unit circle

$$|P(z)| = |z(1-hp)-1| > ||1-hp|-1|$$
$$\geq -h\, Re(p) > h|q| = |Q(z)|.$$

Applying the theorem of Rouché [15, p.2] to P,Q we have that $P(z) + Q(z)$, which is the characteristic polynomial, has the same number of zeros inside the unit circle as $P(z)$, namely m zeros. Hence, the method is P-stable.

The second order backward differentiation method applied to (3.1) with $\beta = mh$ gives the difference equation

$$y_{n+2}(1-2hp/3)-(4/3)y_{n+1}+(1/3)y_n-(2/3)hq\ y_{n-m+2} = 0.$$

The associated characteristic polynomial is

$$z^m[(1-2hp/3)z^2-(4/3)z+1/3] - (2/3)hq = 0.$$

Let
$$f(z) = (1-2hp/3)z^2 - (4/3)z + 1/3,$$

$$P(z) = z^m f(z) \text{ and } Q(z) = -(2/3)hq.$$

$$\begin{aligned}|f(z)|^2 &= f(z)\overline{f(z)} \\ &= (17/9) - (8/9)\mathrm{Re}(z) + (2/3)\mathrm{Re}[(1-2hp/3)z^2] \\ &\quad - (8/3)\mathrm{Re}(1-2hp/3) + |1-2hp/3|^2.\end{aligned}$$

For z on the unit circle $z = \exp(i\theta)$, $0 \le \theta < 2\pi$ and $p = p_1 + ip_2$ where p_1, p_2 are real. Then

$$\begin{aligned}|f(z)|^2 &= (4/9)h^2p_1^2 - (8/9)p_1h(1-\cos\theta)^2 \\ &\quad + (4/9)G(h,p_2,\theta)\end{aligned}$$

where
$$\begin{aligned}G(h,p_2,\theta) &= (4/9)[h^2p_2^2 + 2hp_2 \sin\theta(\cos\theta-2) \\ &\quad + 2(1-\cos\theta) + 3(1-\cos\theta)^2]\end{aligned}$$

is a quadratic in hp_2 with coefficients a function of θ. The discriminant of G is $-4(1-\cos\theta)^4$, hence $G(h,p_2,\theta) \geq 0$. Thus

$$|f(z)|^2 \geq (4/9)h^2p_1^2 > (4/9)h^2|q|^2 = |Q(z)|^2.$$

Therefore, $|P(z)| > |Q(z)|$. Again, applying the theorem of Rouché yields the desired result.

Theorem 4.7

The first and second order backward differentiation methods are GP-stable, when used with linear and quadratic interpolation respectively.

Proof The backward Euler method used with linear interpolation, when applied to (3.1) with $\beta = (m-u)h$, gives the difference equation

$$y_{n+1}(1-ph) - y_n - hq(uy_{n-m+2}+(1-u)y_{n-m+1}) = 0.$$

The characteristic polynomial associated with this equation is

$$z^m(1-hp) - z^{m-1} - hq(uz+1-u) = 0.$$

Let $Q(z) = -hq(uz+1-u)$. For $|z| = 1$ and $u \in [0,1)$ we have that

$$|Q(z)| \leq h|q| \; [u+(1-u)] = h|q|,$$

hence the proof of Theorem 4.5 generalizes to this case. Similarly, the proof of Theorem 4.5 will generalize to the second order backward differentiation method provided that, for $|z| = 1$, the polynomial $\frac{1}{2}u(u+1)z^2 + (1-u)(1+u)z + \frac{1}{2}u(u-1)$ is less than one in magnitude. Letting $z = \exp(i\theta)$ we can easily show that this is true, hence the result follows.

□

In order to analyze the stability of various multistep methods we will need the following lemma concerning the location of the zeros of various polynomials.

Lemma 4.1

The polynomials $(z-1) - \beta q = 0$ and $z(z-1) - \beta q(z+1)/2 = 0$ have no zeros outside the unit circle for βq satisfying (3.4).

Proof (By contradiction)

For the first polynomial, $z = 1 + \beta q$. Thus, for $Re(q) < 0$ and β small, the root lies inside the unit circle. If $\exists$ a root z on the unit circle then we may assume z has the form $\exp(i\theta)$, $0 \le \theta < 2\pi$, which gives $(\cos\theta-1) + i \sin\theta = \beta\gamma(\cos\phi+i \sin\phi)$, Equating the real parts and the squares of the absolute values gives

$$(\cos\theta-1) = \beta\gamma \cos\phi$$

$$(\beta\gamma)^2 = 2(1-\cos\theta).$$

Solving for $\beta\gamma$ gives $\beta\gamma = -2 \cos\phi$. Noting that $-\cos\phi = \begin{cases} \sin(\phi-\pi/2) \\ \sin(3\pi/2-\phi) \end{cases}$ and that $\sin(\alpha) \ge (2/\pi)\alpha$ for $0 \le \alpha < \pi/2$ we easily see that $\beta\gamma > \min\{\phi-\pi/2, 3\pi/2-\phi\}$, which gives us a contradiction.

For the other polynomial clearly $z = -1$ is not a root (i.e. $\theta \ne \pi$) so assuming that $z \in$ unit circle we have $z(z-1)/(z+1) = \beta\gamma\exp(i\phi)/2$; hence $\exp[i(\theta-\phi+\pi/2)]\tan(\theta/2) = \beta\gamma/2$. Equating absolute values gives $\beta\gamma = 2|\tan(\theta/2)|$. Equating real parts gives $\tan(\theta/2)\sin(\theta-\phi+\pi/2) = 0$. Clearly $\tan(\theta/2) \ne 0$ so that

$\theta-\phi+\pi/2 = k\pi$, $k \in I$. The only values of θ of interest are $\theta = \phi-\pi/2$ and $\theta = \phi+\pi/2$.

If $\theta = \phi-\pi/2$ then $0 < \theta/2 < \pi/2$ and $\tan(\theta/2) = \tan((\phi-\pi/2)/2) > (\phi-\pi/2)/2$.

If $\theta = \phi+\pi/2$ then $0 < \pi-\theta/2 < \pi/2$ and $-\tan(\theta/2) = \tan(\pi-\theta/2) = \tan((3\pi/2-\phi)/2) > (3\pi/2-\phi)/2$.

Therefore, $|\tan(\theta/2)| > \frac{1}{2}\min\{\phi-\pi/2, 3\pi/2-\phi\}$ and $\beta\gamma = 2|\tan(\theta/2)| > \min\{\phi-\pi/2, 3\pi/2-\phi\}$, which is clearly a contradiction of condition (3.4).

Lemma 4.2

Consider the polynomial

$$z^m(z-1) - qh[(1-v) + vz]\ , \ q = \gamma\exp(i\phi) \tag{4.3}$$

where $\beta = (m-u)h$, $m \in I^+$, $v \in [1/2,1]$. Then this polynomial has no zeros on the unit circle for $m > 1$ and q satisfying (3.4).

Proof (By contradiction)

Suppose a zero on the unit circle, so that we may assume $z = \exp(i\theta)$, $-\pi < \theta \leq \pi$. Clearly, as ± 1 are not zeros we have $-\pi < \theta < \pi$, $\theta \neq 0$. Equating (4.3) to zero we obtain $\exp(i\theta)z^m = h\gamma[(1-v)+vz]/(z-1) = (-ih\gamma/2)[\cos(\theta/2)+i(2v-1)\sin(\theta/2)]/\sin(\theta/2)$. Equating real parts and squares of the absolute values to zero gives $\cos(m\theta-\phi) = (h\gamma/2)(2v-1)$ and $\tan^2(\theta/2) = (h\gamma/2)^2[1-(h\gamma/2)^2(2v-1)^2]$, or $\sin^2(\theta/2) = (h\gamma/2)^2[1+4v(v-1)\sin^2(\theta/2)]$. This last equation implies $\sin^2(\theta/2) < (h\gamma/2)^2$.

Let $c = (h\gamma/2)(2v-1)$ and $T = \tan(\theta/2)$. We may choose ψ such that $0 \le \psi < \pi/2$ and $\sin\psi = c$ since $0 \le c < 1$. Then $\cos(m\theta-\phi) = \sin\psi = \cos(\psi-\pi/2)$ hence $m\theta-\phi = 2k\pi \pm (\psi-\pi/2)$. As $0 < (m\theta)^2 = (2m(\theta/2))^2 < (2m(\pi/2)\sin(\theta/2))^2 < (m\pi h\gamma/2)^2 = (\pi\beta\gamma/2)^2 < \pi^2$, $-\pi < m\theta < \pi$, thus the only values of θ we are interested in are given by

$$m\theta = \begin{cases} (\phi-\pi/2) + \psi & \text{for } 0 < \theta < \pi \\ (\phi-3\pi/2) - \psi & \text{for } -\pi < \theta < 0. \end{cases}$$

Also, as $-\pi/2 < \theta/2 < \pi/2$, $(m\theta)^2 = [2m(\theta/2)]^2 < (2mT)^2 = (\beta\gamma)^2/[1-c^2]$.

<u>Case (i)</u> $(m\theta = (\phi-\pi/2)+\psi)$

$$(m\theta)^2 = [(\phi-\pi/2)+\psi]^2 > [(\phi-\pi/2)+\sin\psi]^2,$$

which implies

$$[(\phi-\pi/2)+c]^2 < (\beta\gamma)^2/[1-c^2].$$

For $\pi/2 < \phi \le \pi$, $\beta\gamma < (\phi-\pi/2) < \pi/2$ so that $g(\phi) = [(\phi-\pi/2)+c]^2 - (\phi-\pi/2)^2/(1-c^2)$ is negative for $\phi \in (\pi/2,\pi]$. However, $g(\pi/2) = c^2 > 0$ and $g'(\phi) = 2c[1-c(\phi-\pi/2)-c^2]/(1-c^2)$, so that we can easily show for $v \in [1/2,1]$ and $m > 1$ that $g'(\phi)$ is positive which implies $g(\phi)$ is positive, which is clearly a contradiction.

Similarly, for $\phi \in (\pi,3\pi/2)$, we consider the function $g(\phi) = [(\phi-\pi/2)+c]^2 - (3\pi/2-\phi)^2/(1-c^2)$ and obtain a contradiction.

<u>Case (ii)</u> $(m\theta = (\phi-3\pi/2)-\psi)$

The proof here is similar to case (i).

Theorem 4.8

The Backward Euler formula and the modified trapezoidal rule are Q-stable.

Proof The characteristic polynomial associated with the Backward Euler method is $z^m(z-1)-hqz$.

For q satisfying (3.4) this polynomial has no zeros on the unit circle for m = 1 by Lemma (4.1). Taking v = 1 in Lemma (4.2) shows that this polynomial has no zeros on the unit circle for m > 1. Hence, the Backward Euler method is Q-stable.

The characteristic polynomial associated with the modified trapezoidal rule is $z^m(z-1)-hq(z+1)/2 = 0$.

For q satisfying (3.4) this polynomial has no zeros on the unit circle for m = 1 by Lemma (4.1). Taking v = 1/2 in Lemma (4.2) shows that this polynomial has no zeros on the unit circle for m > 1. Thus the modified trapezoidal rule is Q-stable.

Theorem 4.9

The modified trapezoidal rule used with linear interpolation is not GQ-stable.

Proof Any method which is GQ-stable is GDA_0 stable and Cryer [6] has shown that the modified trapezoidal rule is not GDA_0 stable.

Remarks

The author believes that the Backward Euler method used with linear interpolation is GQ-stable although he does not have a proof for it. Firstly, Cryer [6] has shown that this method is GDA_0 stable.

Next, consider the characteristic polynomial of the method which is $z^m(z-1)-hqz(uz+1-u)$. Actually, we need only consider the polynomial $Q(z) = z^{m-1}(z-1)-hq(uz+1-u)$. Note that in order to have a large step (i.e. $h > \beta$) we must have $m = 1$ and that the zero of $P(z)$ for $m = 1$ is $z = (1+\beta q)/[1-\beta qu/(1-u)]$, which is inside the unit circle for q satisfying (3.4) and $u \in [0,1)$. For a large step size the method is stable.

Also suppose $u \in [1/2,1]$ and $m \geq 2$. Let $M = m-1$, $Q = q(m-1)/(m-u)$ and $v = u$. Then $Q(z) = z^M(z-1)-(\beta Q/M)(vz+(1-v))$ and $|\beta Q| < |\beta q|$ so that the zeros of $Q(z)$ are inside unit circle for $m > 2$ by Lemma 4.2.

Thus there is good reason to believe that the method is GQ-stable.

<u>Plotting of the Regions of Q-stability</u>

Let $t = \beta\tau$ in the equation $y'(t) = qy(t-\beta)$. Then $\frac{dy}{dt} = \frac{dy}{d\tau}\frac{d\tau}{dt} = \frac{1}{\beta}\frac{dy}{d\tau}$ so that $\frac{dy}{d\tau} = \beta qy(\beta(\tau-1))$. Define $y^*(\tau) = y(\beta\tau)$ then $\frac{dy^*}{d\tau} = \beta qy^*(\tau-1)$ so that we can always put the equation in the form $y'(t) = qy(t-1)$ by a simple scaling of the time variable. Hence for plotting stability regions for numerical methods applied to this type of delay equation, which just depends on past function values, we can just consider the equation $y'(t) = qy(t-1)$ and the stability region of this equation having the boundary defined by

$$(4.4) \qquad \gamma = \min\{3\pi/2-\phi,\ \phi-\pi/2\}$$

with $q = \gamma\exp(i\phi)$, $\phi \in (\pi/2,3\pi/2)$. Note that this boundary curve is symmetric about the axis $\mathcal{I}m(q) = 0$.

Applying the boundary locus method to a multistep method $\{\rho,\sigma\}$ we get,

$$\exp(im\theta)\rho(\exp(i\theta)) - (q/m)\sigma(\exp(i\theta)) = 0.$$

Solving for q gives

$$q = m \exp(im\theta)\rho(\exp(i\theta))\sigma(\exp(i\theta)).$$

This equation defines a countable number of curves as m is a positive integer. We may plot some of these curves defining the regions of Q-stability for small values of m and hopefully determine if methods are not Q-stable and obtain an intuitive feeling as to where the methods are not Q-stable. Hopefully, the higher order backward differentiation methods will have stability properties similar to stiff stability for O.D.E's.

For the backward differentiation methods $\sigma(z) = \beta_k z^k$, so that $q = m \exp(i(m-k)\theta)\rho(\exp(i\theta))/\beta_k$. The regions of Q-stability for the first order methods with m = 1,2,3 are given in Figure 4.1. Other regions of Q-stability are given in Appendix B.

It is interesting to note [Appendix B] that, the second order backward differentiation method is stable for m = 1,2,3 as we might well expect. It is conjectured that this method is Q-stable. The stability regions with m = 1, for the higher order methods show a similarity to the stiff stability regions of Gear for O.D.E's [11,p.215], with a section missing near the boundary of (4.4), except for the order six method which appears to be stable for m = 1.

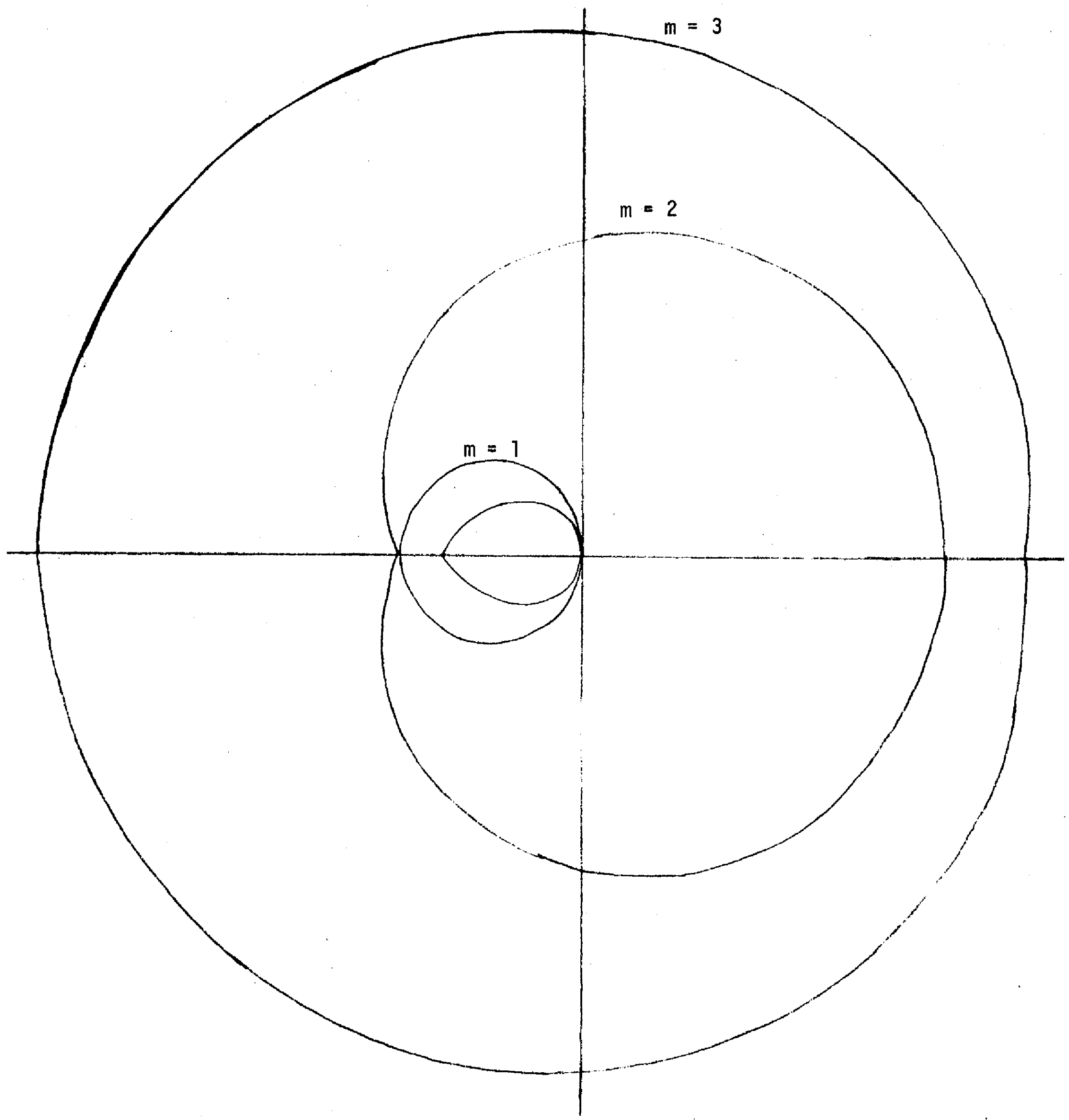


Fig. 4.1 Region of Q-Stability for B.D. method of Order 1 with $\beta = mh$

Plotting the Regions of P-Stability

Next, we wish to consider the equation $y'(t) = py(t) + qy(t-\beta)$. We can easily show by a simple scaling of the time variable, as above, that we need only consider the equation $y'(t) = py(t) + qy(t-1)$ and the stability region $Re(p) < -|q|$. Applying the boundary locus method for the multistep method $\{\rho,\sigma\}$ applied to this equation, we get

$$p = [\rho(z) - qh\sigma(z)z^{-m}]/(h\sigma/z))$$

so that for the backward differentiation methods we have

$$p = (\exp(i\theta))(\exp(-ik\theta)/\beta_k - q\exp(-im\theta).$$

This defines a curve in the p-plane which depends on m and q. To obtain a feeling for the stability behaviour of a method we can take $q = 1$ and plot the corresponding p curves for $m = 1,2,3,4$ along with the line $Re(p) = -|q|$. Figure 4.2 illustrates the boundary of this region for the Backward Euler method. Boundaries of the region for the higher order methods are given in Appendix B.

Clearly, the first and second order backward differentiation methods behave as expected since we know these methods are P-stable. The higher order methods have stability regions similar to those for O.D.E's. However, there is some surprising behaviour. The third order method is stable for $m = 1,2,3,4$ and $q = 1$, and the fourth order method is stable for $m = 1,2$ and $q = 1$. This results since, for small values of m, the contribution from the term $q\exp(-im\theta)$ is not small. Note that the third to sixth order methods cannot be P-stable.

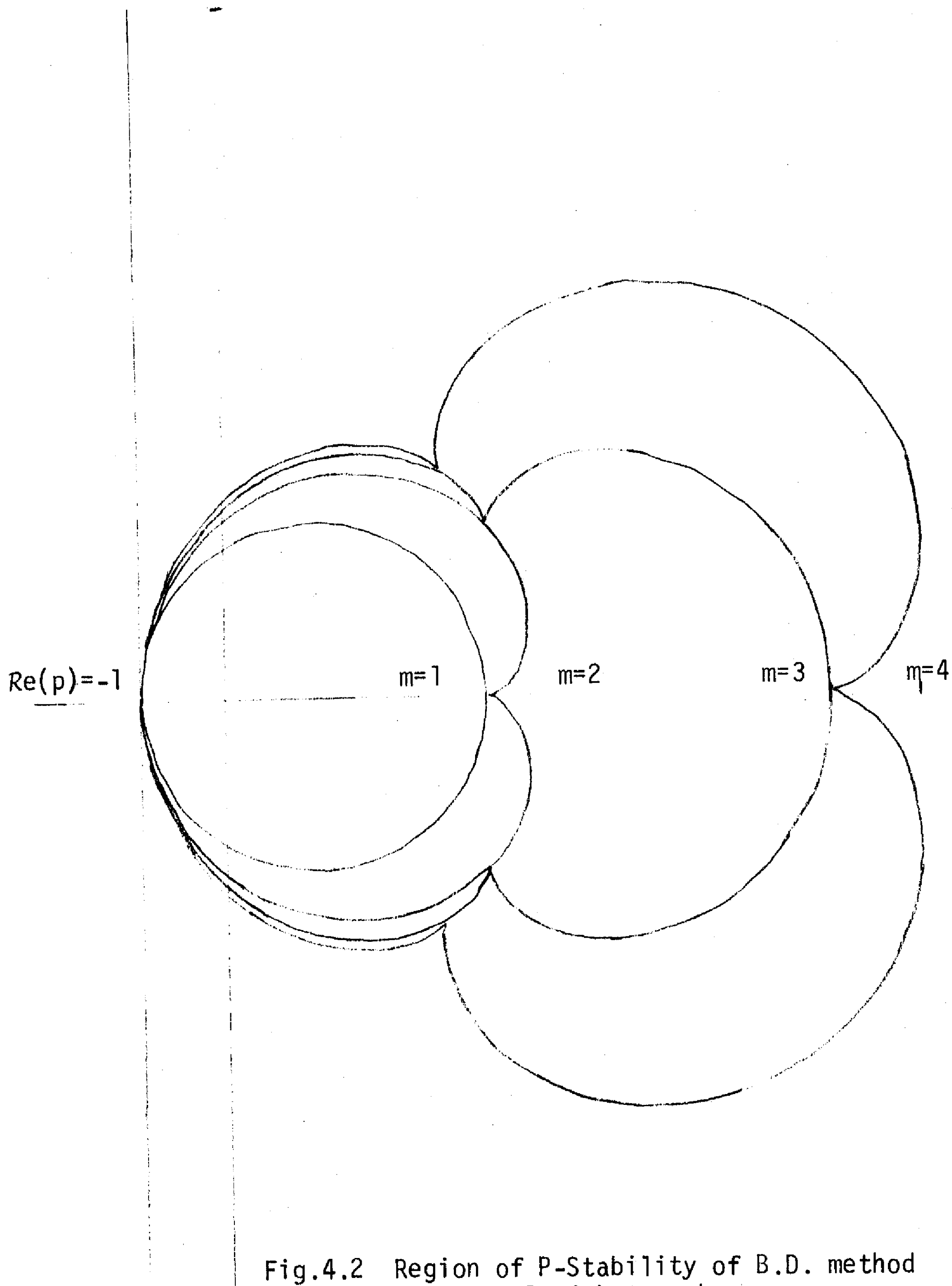


Fig.4.2 Region of P-Stability of B.D. method of Order 1 with β = mh

CHAPTER 5

DESCRIPTION OF AN AUTOMATIC PACKAGE FOR SOLVING $\underset{\sim}{y}'(t) = \underset{\sim}{f}(t,\underset{\sim}{y}(t),\underset{\sim}{y}(t-\beta))$ USING MULTI-VALUE ALGORITHMS

Consider the problem

$$(5.1) \qquad y'(t) = f(t,y(t),y(t-\beta))$$

$$y(t) = g(t) \text{ on } [t_0,t_0+\beta].$$

We will use the generalized Adam's methods (G.A.M.) [11,p.155] since little is known about a specific problem when writing a general package. Gear [11,p.158] has successfully incorporated these methods into an automatic package which changes step size and order to efficiently solve O.D.E. Many of his ideas can be used in designing a similar package for D.D.E. and we will discuss these ideas for O.D.E. and how they can be modified for D.D.E.

Basic Differences

The main differences between solving O.D.E. and D.D.E. are outlined below:

1) The initial function g(t) can be used to generate the starting values for the method; however, variable order methods are usually self starting in order to handle the problem outlined in 2).

2) Discontinuities can arise in the higher order derivatives. We can minimize this problem by ensuring that the set of points where discontinuities occur are included in the set of mesh points.

3) There is a need to save past function values to compute $y(t-\beta)$. This will be discussed more fully later.

Basic Algorithms

Let $\underset{\sim}{y}_n = [y_n, y_{n-1}, \ldots, y_{n-k+1}, hy'_{n-1}, \ldots, hy'_{n-k+1}]^T$. A multivalue algorithm [11,p.103] for O.D.E. has the form

(5.2) $$\underset{\sim}{y}_{n,(0)} = B\underset{\sim}{y}_{n-1}$$

(5.3) $$\underset{\sim}{y}_{n,(m+1)} = \underset{\sim}{y}_{n,(m)} + \underset{\sim}{c}\, G(\underset{\sim}{y}_{n,(m)}),$$

where B is a matrix reflecting the particular nature of the multivalue algorithm in use, and

$$G(\underset{\sim}{y}_n) = -hy'_n + hf(t_n, y_n).$$

(5.2) is called the predictor and (5.3) the corrector.

We may also consider equivalent methods [11,p.142] by using the following transformations:

$$\underset{\sim}{a}_n = T\underset{\sim}{y}_n$$

$$\underset{\sim}{a}_{n,(m)} = T\underset{\sim}{y}_{n,(m)}$$

$$\underset{\sim}{\ell} = T\underset{\sim}{c}$$

$$F(\underset{\sim}{x}) = G(T^{-1}\underset{\sim}{x})$$

$$A = TBT^{-1}.$$

The multivalue method is then written as

(5.4) $$\underset{\sim}{a}_{n,(0)} = A\underset{\sim}{a}_{n-1}$$

(5.5) $$\underset{\sim}{a}_{n,(m+1)} = \underset{\sim}{a}_{n,(m)} + \underset{\sim}{\ell}F(\underset{\sim}{a}_{n,(m)}).$$

In the case of generalized Adams methods [11,p.155] we consider the vector $(y_n, hy'_n, \ldots, hy'_{n-k+1})^T$ for a k-step method since the other components do not appear in the formulas. Using the equivalent representation which incorporates scaled derivatives, we have $\underset{\sim}{a}_n = (\underset{\sim}{y}_n, hy'_n, \ldots, h^{k-1}y_n^{k-1}/(k-1)!)$ for a k-step method. The transformation T is such that $F(T^{-1}\underset{\sim}{a}_{n,(m)}) = G(\underset{\sim}{y}_{n,(m)})$ so that F is easily evaluated as $-hy'_n + hf(t_n, y_n)$ and hy'_n is just the second component of $\underset{\sim}{a}_n$. The scaled derivative representation has the advantage of controlling round-off error better (although not as well as backward differences) and of changing step size easily. The matrix A in (5.4) is just the Pascal triangle matrix [11,p.149]. Hence, we can easily compute A $\underset{\sim}{a}_n$ using only additions [11,p.149] and thus easily perform the prediction step (5.4). The coefficients $\underset{\sim}{\ell}$ for the corrector algorithms in the scaled derivative representation are given in Table 5.1.

We may also use the backward differentiation methods [11,p.214] to overcome stiffness problems. Stiffness can even occur in the scalar D.D.E.problem with a single well behaved function. This will be discussed further in Chapter 6. The vector $\underset{\sim}{\ell}$ for the backward differentiation methods in scaled derivative form are given in Table 5.3 and can be found in [11,p.217].

<u>Error Control, Step Size and Order Change</u>

In an automatic package using a variable order method the error is normally controlled by controlling the local truncation error. A q-th

TABLE 5.1

Coefficients of $\underset{\sim}{\ell}$ for G.A.M.

Order q	1	2	3	4	5	6
ℓ_0	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$
ℓ_1	1	1	1	1	1	1
ℓ_2		1	$\frac{3}{4}$	$\frac{11}{12}$	$\frac{25}{24}$	$\frac{137}{120}$
ℓ_3			$\frac{1}{6}$	$\frac{1}{3}$	$\frac{35}{72}$	$\frac{5}{8}$
ℓ_4				$\frac{1}{24}$	$\frac{5}{48}$	$\frac{17}{96}$
ℓ_5					$\frac{1}{120}$	$\frac{1}{40}$
ℓ_6						$\frac{1}{720}$

TABLE 5.2

Error Constants for G.A.M.

Order q	1	2	3	4	5	6
C_{q+1}	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$

TABLE 5.3

Coefficients of $\underset{\sim}{\ell}$ for B.D.M.

Order q	1	2	3	4	5	6
ℓ_0	1	$\frac{2}{3}$	$\frac{6}{11}$	$\frac{12}{25}$	$\frac{60}{137}$	$\frac{60}{147}$
ℓ_1	1	1	1	1	1	1
ℓ_2		$\frac{1}{3}$	$\frac{6}{11}$	$\frac{7}{10}$	$\frac{225}{274}$	$\frac{406}{441}$
ℓ_3			$\frac{1}{11}$	$\frac{1}{5}$	$\frac{85}{274}$	$\frac{245}{588}$
ℓ_4				$\frac{1}{50}$	$\frac{15}{274}$	$\frac{175}{1764}$
ℓ_5					$\frac{1}{274}$	$\frac{7}{588}$
ℓ_6						$\frac{1}{1764}$

TABLE 5.4

Error Constants for B.D.M.

Order q	1	2	3	4	5	6
C_{q+1}	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$

order Adam's method has a local truncation error $C_{q+1}h^{q+1}y^{q+1}$ provided the order of the predictor plus the number of corrector iterations exceeds q [9, p.155]. We will use a q-1'th order predictor with a q'th order corrector, so that even after one iteration the method will be of order q [11,p.155]. The error constants for the G.A.M. are given in Table 5.2 and for the B.D.M. in Table 5.4.

Clearly, to estimate the local truncation error, we must estimate y^{q+1}. Let ∇a_q denote the change in the last component of $\underset{\sim}{a}_n$. Then $\nabla a_q = h^q(y^q_{n+1} - y^q_n)/q!$, so that $C_{q+1}q!\ \nabla a_q$ is an estimate of the local truncation error. The algorithm which is used in the program controls either the local relative error per unit step or the local error per step. That is, we accept a step provided the test,

(5.6) $$C_{q+1}q!\ \nabla a_q \le h\ \varepsilon\ \text{ymax}$$

or $$C_{q+1}q!\ \nabla a_q \le \varepsilon\ \text{ymax}.$$

where ε is the requested tolerance and ymax is the maximum absolute value of the previously computed solution, succeeds. YMAX is originally provided by the user and can be overridden by the user each time control returns to the user. The user must select error per unit step or error per step. In the case of a system of equations we replace ∇a_q with the L_2 norm of $\nabla \underset{\sim}{a}_q$ in the tests (5.6).

The step size is easily changed from h to αh by multiplying $\underset{\sim}{a}_n$ by the matrix $\text{diag}[1,\alpha,\ldots,\alpha^{q-1}]$, where we are using a q'th order method.

To decrease the order of a method, we simply omit the last component of $\underset{\sim}{a}_n$. To increase the order of a method, from q to q+1, we must add the component $h^{q+1}y_n^{q+1}/(q+1)!$ to $\underset{\sim}{a}_n = (y_n,\ldots,h^q y_n^q/q!)$! As before, $h^{q+1}y_n^{q+1}$ is estimated by using $\nabla a_q\, q!$ so that the last component becomes $\nabla a_q/(q+1)$.

Algorithm for Automatic Control

If we have solved the corrector equation (5.5), then (5.6) gives a criterion for accepting or rejecting the computed solution.

If a step fails, then we want to decrease the step size and/or the order. We consider using the order q or q-1 method which gives the maximum step size. Given the present step size h and order q then the new step size is αh, where the α for order q and error per unit step is given by

$$(5.7) \qquad \alpha = C_1[h\ \varepsilon\ \mathrm{ymax}/(|C_{q+1}\ a_q|q!)]^{1/q}$$

and the α for order q-1 is given by

$$(5.8) \qquad \alpha = C_2[h\ \varepsilon\ \mathrm{ymax}/(|C_q a_q|)]^{1/(q-1)}.$$

where a_q is the last component of $\underset{\sim}{a}_n$. For a system of equations we use the L_2 norm of $\underset{\sim}{a}_q$. If $\|\underset{\sim}{a}_q\|_2 = 0$, as frequently happens with D.D.E. (since with a constant initial function higher order derivatives are zero), the step size is decreased by 10. These values of α are chosen so that when $C_1 = C_2 = 1$ and there is no roundoff error, then the error test (5.6) would be satisfied exactly. C_1, C_2 are chosen slightly less than one in the hope that the error test (5.6) will be satisfied even in the presence of roundoff error, and inaccuracies in the asymptotic error formula. The program in Appendix A uses Gear's values [11,p.156] for C_1, C_2. The order corresponding to the largest step is chosen and the decrease in step size is performed. Of course, if the order is one to start, we can only decrease the step size.

If the step succeeds, we repeat with the same step size h and order q until at least q+1 steps after the last change in order or step size, and at least ten steps after the α were last estimated, if no increase in order was made at that time. In considering increasing the step to αh, the α for order q is given by (5.7) and the α for order q+1 with error per unit step is given by

$$\alpha = C_3[h\ \varepsilon\ y_{max}/(|C_{q+2}\nabla^2 a_q|q!)]^{1/q+1}. \tag{5.9}$$

In the case of a system of equations we use the $\|\nabla^2 \underset{\sim}{a}_q\|_2$ in (5.9). Again if $\|\nabla^2 \underset{\sim}{a}_q\|_2 = 0$, the step is increased by ten. If the order is less than six then the order corresponding to the largest step is selected provided that α is greater than 1.1. If neither α is at least 1.1, there is no change in step size or order. The value of $\nabla^2 \underset{\sim}{a}_q$ in (5.9) is obtained by saving the value of $\nabla \underset{\sim}{a}_q$ from the previous step, and computing $\nabla^2 \underset{\sim}{a}_q$ just before a step increase is imminent.

Solution of the Corrector Equation

The corrector iteration (5.5) can be written as $\underset{\sim}{a}_{n,(m+1)} = \underset{\sim}{a}_{n,(0)} + \underset{\sim}{\ell}(F(\underset{\sim}{a}_{n,(0)}) + \ldots + F(\underset{\sim}{a}_{n,(m)}))$ and only the first two components of $\underset{\sim}{a}_{n,(\cdot)}$ need be updated at once, since for the G.A.M. the function $F(\underset{\sim}{a})$ depends only on the first two components of $\underset{\sim}{a}$. The functional iteration (5.5) is performed a maximum of three times. After the m'th iteration, the test

(5.10) $F(\underset{\sim}{a}_{n,(m)}) < \varepsilon\ h\ ymax/(2q+1)$

is performed and if it succeeds the corrector iteration has converged. If this iteration does not converge in three steps then the step size h is decreased to maximum (h/4, hmin) where hmin is the smallest step size to be used.

Modifications for D.D.E.

The above formulas for the predictor step, the corrector step with the error estimation and step and order changing algorithm can be directly adapted to D.D.E. provided we replace $f(t,y_n)$ by $f(t,y_n,\bar{y}_n)$ where $\bar{y}_n$ is an approximation to $y(t_n-\beta)$.

At the point t_n we have previously computed the scaled derivative representation $\underset{\sim}{a}_n$ and we want to compute $\underset{\sim}{a}_{n+1}$. This can be done by using the formulas (5.4), (5.5) provided we replace $f(t_n,y_n)$ by $f(t_n,y_n,\bar{y}_n)$ where $\bar{y}_n = y(t_n-\beta)$. Thus we must provide a value for $f_{n+1} = f(t_{n+1},y_{n+1},\bar{y}_{n+1})$ and hence $\bar{y}_{n+1} = y(t_{n+1}-\beta)$. If $t_{n+1}-\beta \in [t_0,t_0+\beta]$, then we can generate $y(t_{n+1}-\beta)$ from the initial function g(t).

If $t_{n+1}-\beta \notin [t_0,t_0+\beta]$ we cannot compute $\bar{y}_{n+1}$ from g(t). However, $t_\beta = t_{n+1}-\beta$ must belong to the interval $(t_n-\beta,t_{n+1})$ and thus we require a representation of the solution on this interval. Clearly, we can save at least the points in this interval and possibly a fixed number outside the interval in order to compute an approximation to $y(t_\beta)$ by an interpolation formula. Suppose t_β is located between the nodes t_j and t_{j+1}, and that y_{n+1} was computed by using an order q formula. This implies that an interpolation formula of order q can be used to compute an approximation to $y(t_\beta)$ provided we use the points $t_{j+1}, t_j, \ldots, t_{j-q+1}$ since the Adams formulas are really interpolatory formulas. One problem arises when $h > \beta$ because then $t_{j+1} = t_{n+1}$ and we do not have the function value y_{n+1} needed for the interpolation process. This is overcome by including the interpolation in the corrector iteration and updating the approximation to $y(t_\beta)$ whenever y_{n+1} is updated.

Suppose we have the set of points $x_0, \ldots, x_N$ and the corresponding function values $y_0, \ldots, y_N$. To perform the interpolation to find $y(x)$ where $x_{N-1} < x < x_N$ we will use the Newton divided difference formula [4, p.195]. $P(x) = y_0 + (x-x_0)y_{01} + \ldots + (x-x_0)\ldots(x-x_{N-1})y_{0\ldots N}$.
The required divided differences are the diagonal entries in the divided difference table:

$$
\begin{array}{llll}
y_0 & & & \\
y_1 & y_{01} & & \\
y_2 & y_{02} & y_{012} & \\
y_3 & y_{03} & y_{013} & \ddots \\
\vdots & \vdots & \vdots & \ddots \\
y_N & y_{0N} & y_{01N} & y_{012\ldots N}
\end{array}
$$

The table is generated row by row, so that changing y_N has the effect of changing only the last row in the divided difference table. Also this changes only the last term in the divided difference formula, so the term $y_0 + (x-x_0)y_0 + \ldots + (x-x_0)\ldots(x-x_{N-2})y_0\ldots y_{N-1}$ is saved to efficiently evaluate polynomial $P(x)$ whenever only y_N changes. Thus, when $h > \beta$, including the interpolation in the corrector iteration can be made less expensive.

The adaptation of a Newton correction iteration for stiff problems in O.D.E. to D.D.E. requires some slight modifications. The Newton corrector iteration [11,p.217] for systems is

$$(5.11) \qquad \underset{\sim}{a}_{n,(m+1)} = \underset{\sim}{a}_{n,(m)} - \underset{\sim}{\ell}[\frac{\partial F}{\partial \underset{\sim}{a}} \circ \underset{\sim}{\ell}]^{-1} F(\underset{\sim}{a}_{n,(m)})$$

where $F(\underset{\sim}{a}) = hf(t,\underset{\sim}{a}_0,P(\underset{\sim}{a}_0)) - \underset{\sim}{a}_1$ and $\underset{\sim}{a}_0$, $\underset{\sim}{a}_1$ are the first and second row of $\underset{\sim}{a}_{n,(m)}$.

If $h \leq \beta$ then $P(a_0) = 0$, since the interpolation does not depend on $\underset{\sim}{y}_{n+1}$, and for $h > \beta$, $P(\underset{\sim}{a}_0)$ is the polynomial given by the Newton divided difference formula. Hence $P'(\underset{\sim}{y}_N) = 0$ for $h \leq \beta$ and

$$P'(\underset{\sim}{y}_N) = \frac{(x-x_0)\ldots(x-x_{N-1})}{(x_N-x_0)\ldots(x_N-x_{N-1})} .$$

Therefore $\qquad W = [\frac{\partial F}{\partial \underset{\sim}{a}} \circ \underset{\sim}{\ell}]$

$$= -\ell_1 I + \ell_0 h[\frac{\partial f}{\partial y} + P'(y_N)\frac{\partial f}{\partial \bar{y}}].$$

Normally these Jacobians would be revaluated at each step in the Newton iteration; however, in many stiff problems the Jacobians are slowly changing and thus can often be held constant over a number of steps.

Note that for $h > \beta$, the numerator of $P'(y_N)$ is computed during the interpolation stage so that it can simply be saved. Each term in the denominator is also computed during interpolation, so that if the re-evaluation of W is required, the product of these terms is calculated and saved.

The matrices $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial \bar{y}}$ are evaluated by numerical differencing. Thus, we approximate the (i,j) entry $\frac{\partial f_i}{\partial y_j}$ of the matrix $\frac{\partial f}{\partial y}$ by

$[f_i(t,y_j+r,\bar{y}) - f_i(t,y_j,\bar{y})]/r$, where $r = \max\{\varepsilon|y_j|,\varepsilon^2\}$.

Having evaluated W, we need not actually compute W^{-1}. To compute $W^{-1}F(a_{n,(m)})$ we need only find the LU decomposition of W and solve the equation W (correction term) = $F(\underset{\sim}{a}_{n,(m)})$ to find the correction term at the m'th iteration.

After each iteration the test

$$(5.12) \qquad \|W^{-1}F(a_{n,(m)})\|_2 < \varepsilon h \; ymax/(2q+1)$$

is performed. If (5.12) succeeds the corrector iteration has converged. If the corrector iteration fails to converge in three steps the Jacobian is re-evaluated. If the iteration still fails to converge the step size is decreased to max(h/4, hmin), and the process repeated.

Of course, with any change in step size the Jacobians are re-evaluated.

To handle the problem of discontinuities in the higher order derivatives, the user must call the subroutine with endpoints $t_0 + k\beta$ for k = 1 to 5 and then to the point at which the computed solution is desired.

Implementation of the Algorithm

We consider in this section the data structures used to store information and implement the algorithm. We also consider how to modularize the program.

The scaled derivatives are naturally represented as a vector, so an array A is used to store them. An array SAVE is used for temporary storage of the scaled derivatives so that if a step fails the values stored in SAVE may be used in restarting with a new step or order.

The coefficients of the vectors $\underset{\sim}{\ell}$ in equation (5.5) for both G.A.M. and B.D.M. are given in Tables 5.1, 5.3, and as we can see from the tables, the collection of coefficients can conveniently be stored in the upper Hessenberg part of a matrix. Thus the coefficients are stored in the upper Hessenberg part of the matrix CL(7,7) and are initialized during the first call to the subroutine. This makes the program more portable since the recompilation of the program on a different machine will cause the coefficients to be initialized to the accuracy of that machine. The error constants Cq given in Tables 5.2, 5.4 are naturally represented in the array CQ(7). Part of the computations, in 5.7-5.9 to determine the value of α to use in changing order and step size, and in 5.6 for controlling the error, involve constants such as $[1.0/|C_{q+1}q!|]^{1/q}$ in 5.7, which are independent of the step and thus need be computed only once. These constants are computed and stored in ERRCON. ERRCON(1,Q) contains the constant for order q-1, ERRCON(2,Q) the constant for order q, ERRCON(3,Q) the constant for order q+1 and ERRCON(4,Q) the constant used in the error test (5.6).

The past values are saved in a circular queue where each entry in the queue contains three pieces of information: the previous time TBACK, the computed solution at TBACK and the order used to compute the solution at TBACK. This is handled by using two real arrays PASTT(QMAX), PASTY(N,QMAX), and one integer array PASTQ(QMAX), where N is the dimension of the system being solved. All three arrays are of dimension QMAX with PASTT(I), PASTY(*,I), PASTQ(I) representing respectively the three pieces of information described above at the entry in the queue pointed to by the integer I. BEGIN and END are integers pointing to the beginning and end of the queue, with additions being made to the end of the queue at the end of a successful step by increasing END by one modulo QMAX. The circular queue is modified slightly so that when the step size H is greater than the lag BETA an entry can be added to the end of the queue, changed and deleted before the step is completed to enable the corrector iteration to use the predicted solution in the interpolation process. There is also a pointer INDEX into the queue which points to the last node used in the interpolation formula. Of course, when H > BETA we know that provided the predicted value is added to the end of the queue then INDEX will equal END and it is unnecessary to search the queue to find the nodes needed for interpolation. An array DELTAQ(N) is used to save $\nabla \underset{\sim}{a}_q$.

The program has been modularized by breaking it up into subroutines. Even though this increases cost especially on IBM machines the readability of the program improves and the flow of control is more evident. The following is a list of the subroutines used and a brief description of their function.

ADD - adds an entry to the end of the queue.

CHKERR - decides on the success of a step by controlling truncation error and determines and controls changes in order and step size.

CHSTEP - changes the step size.

CORECT - performs the functional corrector iteration for G.A.M. and changes step size if convergence does not occur.

DECOMP - routine found in [10] to find the LU decomposition of a matrix.

DDE - driver routine which determines when the endpoint of integration is reached and controls the addition and deletion routines for the circular queue.

DELETE - deletes unwanted entries from the circular queue.

DDIFF - computes the divided difference table needed in Newton's divided difference formula.

ERROR - a routine used for the printing of error messages.

EVAL - a routine to evaluate the Newton divided difference polynomial.

FUNCT - decides how to compute $y(t-\beta)$ and calls the appropriate routines.

JACOB - evaluates the partial derivatives $\frac{\partial f}{\partial y}, \frac{\partial f}{\partial \bar{y}}$.

PREDCT - performs the predictor step (5.4).

PUT - transfer one matrix to another.

SEARCH - performs a binary search of the queue to find the entries needed to do interpolation.

SETUP - initializes the method, the error control and the queue.

SOLVE - routine found in [10] for using the LU decomposition provided by DECOMP for solving a system of equations.

STIFFC - performs the Newton corrector iteration for B.D.M. and changes step size if convergence does not occur.

OUT - prints out a formatted vector or matrix.

Debugging Aids

The program in Appendix A contains a debugging facility, which of course, could be deleted from a production code since it involves some overhead. This debugging facility not only allows someone familiar with the program to determine the cause of bugs (no large program ever seems to be completely bug free), but also for a casual user of the program to write out intermediate results in a given time range. There are three debugging parameters contained in a blank common block, namely IDEBUG, KDEBUG and LDEBUG.

The last two debugging parameters are the simplest so we will discuss them first. If LDEBUG is nonzero then control is passed to the routine CHBUG (T, IDEBUG) after each step, even if it is not successful. Thus this routine may change the debugging parameter. Hence, if the routine is encountering trouble in a certain range of time values the user can gain control of the intermediate output by changing IDEBUG. If KDEBUG is nonzero then a call is made to the subroutine TRUE(T,Y) which calculates the known solution Y at the point T. This parameter is much more useful to the person correcting bugs in a program.

The other parameter IDEBUG permits the printing of more information when it is increased. For example, if IDEBUG equals four then all the information for IDEBUG = 0,1,2,3 is printed also. The following is a description of the information printed at each level:

IDEBUG ≤ 0 no information is printed.

= 1 prints out the values of ε, h, h_{min}, β and t on entry to the subroutine as well as the initial values of h and $\underset{\sim}{a}$.

= 2 prints out the value of t and y(t) after each successful step.

= 3 prints out when a change in step or order is being considered and in the stiff case when the Jacobian is re-evaluated.

= 4 prints out the scaled derivatives and other information before and after the corrector iteration and the partial derivatives when the Jacobian is evaluated.

= 5 prints out the matrix A during each corrector iteration.

= 6 prints out how $y(t-\beta)$ was evaluated.

= 7 prints out entries and deletions to the queue.

= 8 prints out the pointer to the queue values used for interpolation.

= 9 prints out the array A before and after the predictor step.

The user must provide two subroutines. The first routine DERIV(T,Y,YBETA,F) evaluates $f(t,y(t),y(t-\beta))$ given $t,y(t)$ and $y(t-\beta)$.

The second subroutine PHT(T,Y) evaluates the initial function $g(t)$ and stores it in Y. The subroutine for evaluating the partial derivatives called JACOB(F,FPLUSR,PDY,PDYBAR,Y,YBAR,YPLUSR,EPS,T,N) can be replaced by a user subroutine of the same name which stores $\frac{\partial f(t,y,\bar{y})}{\partial y}$ and $\frac{\partial f}{\partial \bar{y}}(t,y,\bar{y})$ in the matrices PDY(N,N) and PDYBAR(N,N) respectively.

CHAPTER 6

NUMERICAL RESULTS AND CONCLUSIONS

The debugging of an automatic package for solving delay differential equations involves finding a collection of problems which will exercise various parts and features in the package. The first problem is a system of D.D.E's which has an oscillatory solution where the initial function is a solution of the D.D.E. so that the true solution is known. Also there will be no discontinuities in the higher order derivatives.

<u>Problem 6.1</u>

$$\underset{\sim}{y}'(t) = -\underset{\sim}{y}(t-\pi/2) \quad \text{for } t > \pi/2$$

$$\underset{\sim}{g}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix} \qquad 0 \le t \le \pi/2$$

$$\underset{\sim}{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

This problem was integrated, using the generalized Adams methods with an error per unit step, from $\pi/2$ to 5.0. The tolerance was $\varepsilon = .002$ and an initial step size of $h = 0.1$ was attempted. KDEBUG was set to one since the true solution was known. The parameter IDEBUG was set to three which will print out information on step and order changing, to show how useful this parameter is. It allows a user to discover how the package is working on his problem. The output generated for this problem is given below.

```
DDE SOLVER ENTERED
EPS =  0.1999999E-02 H =  0.9999996E-01 HMIN =  0.9999999E-10
BETA =  0.1570794E 01 T =  0.1570794E 01
A SYSTEM OF DIMENSION  2 IS BEING SOLVED

INITIALIZATION DONE FOR ADAMS METHODS
ERROR PER UNIT STEP USED

STEP FAILED WITH ORDER = 1
STEP SIZE BEING CHANGED FROM
H =  0.9999996E-01 TO H =     0.3334740E-02


STEP SUCCEEDED WITH H=  0.3334740E-02
SOLUTION AT T =  0.1574128E 01 IS
    Y( 1) =   0.9999889E 00
    Y( 2) =  -0.3332500E-02
TRUE SOLUTION IS
    Y1 = SIN( 0.1574128E 01) =  0.9999945E 00
    Y2 = COS( 0.1574128E 01) = -0.3331816E-02


STEP SUCCEEDED WITH H=  0.3334740E-02
SOLUTION AT T =  0.1577462E 01 IS
    Y( 1) =   0.9999666E 00
    Y( 2) =  -0.6667163E-02
TRUE SOLUTION IS
    Y1 = SIN( 0.1577462E 01) =  0.9999778E 00
    Y2 = COS( 0.1577462E 01) = -0.6665818E-02

POSSIBLE INCREASE IN ORDER AND STEP SIZE
ORDER INCREASED TO 2
STEP SIZE BEING CHANGED FROM
H =  0.3334740E-02 TO H =     0.1111016E 00


STEP SUCCEEDED WITH H=  0.1111016E 00
SOLUTION AT T =  0.1688563E 01 IS
    Y( 1) =   0.9930691E 00
    Y( 2) =  -0.1173827E 00
TRUE SOLUTION IS
    Y1 = SIN( 0.1688563E 01) =  0.9930735E 00
    Y2 = COS( 0.1688563E 01) = -0.1174949E 00


STEP SUCCEEDED WITH H=  0.1111016E 00
SOLUTION AT T =  0.1799664E 01 IS
    Y( 1) =   0.9739388E 00
    Y( 2) =  -0.2266509E 00
TRUE SOLUTION IS
    Y1 = SIN( 0.1799664E 01) =  0.9739239E 00
    Y2 = COS( 0.1799664E 01) = -0.2268753E 00


STEP SUCCEEDED WITH H=  0.1111016E 00
SOLUTION AT T =  0.1910766E 01 IS
    Y( 1) =   0.9428115E 00
    Y( 2) =  -0.3331244E 00
TRUE SOLUTION IS
    Y1 = SIN( 0.1910766E 01) =  0.9427649E 00
    Y2 = COS( 0.1910766E 01) = -0.3334581E 00

POSSIBLE INCREASE IN ORDER AND STEP SIZE
ORDER INCREASED TO 3
STEP SIZE BEING CHANGED FROM
H =  0.1111016E 00 TO H =     0.2978103E 00
```

```
STEP SUCCEEDED WITH H=  0.2978103E 00
SOLUTION AT T =  0.2208575E 01 IS
     Y( 1 ) =   0.8037296E 00
     Y( 2 ) =  -0.5952091E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.2208575E 01 ) =  0.8034202E 00
     Y2 = COS( 0.2208575E 01 ) = -0.5954124E 00

STEP SUCCEEDED WITH H=  0.2978103E 00
SOLUTION AT T =  0.2506385E 01 IS
     Y( 1 ) =   0.5939073E 00
     Y( 2 ) =  -0.8049494E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.2506385E 01 ) =  0.5933447E 00
     Y2 = COS( 0.2506385E 01 ) = -0.8049484E 00

STEP SUCCEEDED WITH H=  0.2978103E 00
SOLUTION AT T =  0.2804194E 01 IS
     Y( 1 ) =   0.3317776E 00
     Y( 2 ) =  -0.9438897E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.2804194E 01 ) =  0.3310331E 00
     Y2 = COS( 0.2804194E 01 ) = -0.9436192E 00

STEP SUCCEEDED WITH H=  0.2978103E 00
SOLUTION AT T =  0.3102004E 01 IS
     Y( 1 ) =   0.4041755E-01
     Y( 2 ) =  -0.9997982E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.3102004E 01 ) =  0.3957826E-01
     Y2 = COS( 0.3102004E 01 ) = -0.9992165E 00

POSSIBLE INCREASE IN ORDER AND STEP SIZE
ORDER INCREASED TO 4
STEP SIZE BEING CHANGED FROM
H =  0.2978103E 00 TO H =    0.3762149E 00

STEP SUCCEEDED WITH H=  0.3762149E 00
SOLUTION AT T =  0.3478218E 01 IS
     Y( 1 ) =  -0.3297686E 00
     Y( 2 ) =  -0.9443701E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.3478218E 01 ) = -0.3303037E 00
     Y2 = COS( 0.3478218E 01 ) = -0.9438747E 00

STEP SUCCEEDED WITH H=  0.3762149E 00
SOLUTION AT T =  0.3854432E 01 IS
     Y( 1 ) =  -0.6537942E 00
     Y( 2 ) =  -0.7569280E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.3854432E 01 ) = -0.6539845E 00
     Y2 = COS( 0.3854432E 01 ) = -0.7565080E 00

STEP SUCCEEDED WITH H=  0.3762149E 00
SOLUTION AT T =  0.4230646E 01 IS
     Y( 1 ) =  -0.8864260E 00
     Y( 2 ) =  -0.4635737E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.4230646E 01 ) = -0.8861887E 00
     Y2 = COS( 0.4230646E 01 ) = -0.4633243E 00
```

```
STEP SUCCEEDED WITH H=  0.3762149E 00
SOLUTION AT T =  0.4606860E 01 IS
     Y( 1 ) =  -0.9951142E 00
     Y( 2 ) =  -0.1052335E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.4606860E 01 ) = -0.9944370E 00
     Y2 = COS( 0.4606860E 01 ) = -0.1053330E 00
STEP CHANGE TO REACH ENDPOINT EXACTLY
STEP SIZE BEING CHANGED FROM
H =  0.3762149E 00 TO H =    0.1965699E 00

STEP SUCCEEDED WITH H=  0.1965699E 00
SOLUTION AT T =  0.4803430E 01 IS
     Y( 1 ) =  -0.9966628E 00
     Y( 2 ) =   0.9111315E-01
TRUE SOLUTION IS
     Y1 = SIN( 0.4803430E 01 ) = -0.9958587E 00
     Y2 = COS( 0.4803430E 01 ) =  0.9091485E-01

DDE WILL TERMINATE IF STEP IS SUCESSFUL

STEP SUCCEEDED WITH H=  0.1965699E 00
SOLUTION AT T =  0.4999999E 01 IS
     Y( 1 ) =  -0.9598602E 00
     Y( 2 ) =   0.2840038E 00
TRUE SOLUTION IS
     Y1 = SIN( 0.4999999E 01 ) = -0.9589246E 00
     Y2 = COS( 0.4999999E 01 ) =  0.2836612E 00
```

The next problem was chosen to illustrate some of the difficulties with discontinuities in the higher order derivatives. Also as the initial function is a constant, then initially higher order derivatives are zero and so care must be taken in the step estimating algorithm for variable order methods since normally these derivatives appear in the denominators of the expression for estimating step size.

Problem 6.2

$$y'(t) = y(t-1) \qquad t \geq 0$$

$$y(t) = 1.0 \qquad -1 \leq t \leq 0$$

The exact solution of this problem on the interval [0,4] is easily obtained by analytic integration and is given by:

$1 + t$	$0 \leq t \leq 1$
$(t^2+3)/2$	$1 \leq t \leq 2$
$7/2 + (t-2)(t^2-t+10)/6$	$2 \leq t \leq 3$
$t^4/24 - t^3/3 + 7t^2/4 - 5t/2 + 85/24$	$3 \leq t \leq 4$

Note that the solution has a discontinuity in the k-th derivative at the point $t = (k-1)$.

This problem was integrated from 0.0 to 3.2 with a tolerance $\varepsilon = .001$ and initial step size $h = 0.1$. The debugging parameter IDEBUG was set to one and KDEBUG was set to zero. The output generated is given below:

```
DDE SOLVER ENTERED
EPS =  0.9999999E-03 H =   0.9999996E-01 HMIN =   0.9999997E-07
BETA =   0.1000000E 01 T =   0.0
A SYSTEM OF DIMENSION  1 IS BEING SOLVED

INITIALIZATION DONE FOR ADAMS METHODS
ERROR PER UNIT STEP USED

DDE WILL TERMINATE IF STEP IS SUCESSFUL

DDE WILL TERMINATE IF STEP IS SUCESSFUL
    THE SOLUTION AT T =   0.3199999E 01 IS   0.6922497E 01
```

To illustrate the effect of the discontinuities in the higher order derivatives, the equation was integrated from 0.0 to 3.2 including the points 1,2,3 in the mesh and not including these points. The results are summarized in Table 6.1.

TABLE 6.1

	t	y(t)
True Solution	3.2	6.90806
Solution with points t=1,2,3 included	3.2	6.90964
Solution without	3.2	6.92250

The accuracy of the solution appears to be affected by the inclusion of the points, where discontinuities occur in higher order derivatives, in the mesh. It appears that including these points in the mesh improves the accuracy. It should be noted in both cases that the code would automatically decrease the order and step in the presence of the discontinuities; this behaviour is similar to that observed by Neves [16]. Of course, for the D.D.E. the possible points of discontinuity are known in advance and can easily be included in the set of mesh points. However, for a more general type of problem with variable time lags this can cause serious problems [17, 18].

The next examples deal with the problem of 'stiffness' for delay differential equations, and an appropriate definition of 'stiffness' for delay problems. The next example illustrates some stability problems with a scalar equation.

Problem 6.3

$$y'(t) = -10{,}000\ y(t) + y(t-\beta)$$

$$y(t) = \exp(-t) \quad \text{on } [-\beta, 0]$$

where $\beta = \ell n(10^4-1) \doteq 9.21024$. Note that β has been chosen so that exp(-t) is the solution to this problem. Clearly then, the solution is reasonably smooth on the interval [0,10]. However, the parameter 10,000 connected with y(t) causes problems for the Adams methods.

In fact, in attempting to integrate this problem to t = 10 with a tolerance of ε = .01 the Adams methods failed to solve the problem unless h was less than 10^{-4}.

The stiff option in the package easily overcame this problem since the backward differentiation methods are GP stable for these parameter values [Theorem 4.7]. The results of using it with an error per step algorithm, the parameter IDEBUG set to three and KDEBUG set to one are given below:

```
DDE SOLVER ENTERED
EPS =   0.9999996E-01  H =   0.9999999E-04 HMIN =   0.9999999E-15
BETA =  0.9210239E 01  T =  0.3088535E-83
A SYSTEM OF DIMENSION  1 IS BEING SOLVED

INITIALIZATION DONE FOR STIFF METHODS
ERROR PER STEP USED

STEP SUCCEEDED WITH H=  0.9999999E-04
SOLUTION AT T =  0.9210339E 01 IS
    Y( 1 ) =   0.1000001E-03
TRUE SOLUTION IS
    EXP( -T ) =  0.1000002E-03
```

The following example is a scalar problem which has two exponential components with very different arguments and yet causes no stability problems.

Problem 6.4

$$y'(t) = -y(t-\beta)$$
$$y(t) = \exp(-\alpha_1 t) + \exp(-\alpha_2 t) \text{ on } [-\beta,0]$$

where $\beta = 10^{-3}$ and α_1, α_2 are the two real positive roots of the equation $\alpha = \exp(\alpha\beta)$. Hence both $\exp(-\alpha_1 t)$ and $\exp(-\alpha_2 t)$ satisfy $y'(t) = -y(t-\beta)$ so that the solution to problem 6.4 is $y(t) = \exp(-\alpha_1 t) + \exp(-\alpha_2 t)$. $\alpha_1 \doteqdot 1.00100$ and $\alpha_2 \doteqdot 9118.01$. This is an interesting example since components like these in a system of O.D.E. would be associated with stiffness, but in problem 6.4 they cause no such problems, since $\frac{\partial f}{\partial \bar{y}}$ is not large.

These examples give rise to the following definition of stiffness for D.D.E.

Definition 6.1

The problem $y'(t) = f(t,y,\bar{y})$ is called stiff if $|\frac{\partial f}{\partial y}|$ or $|\frac{\partial f}{\partial \bar{y}}|$ is large relative to the time scale and the solution does not change drastically on the same time scale.

Of course in systems of D.D.E. one can encounter difficulties with stiffness similar to those for O.D.E. by having the eigenvalues of the Jacobian differ greatly on a suitable time scale.

Conclusions and Extensions

The exact relationship between A stable methods for O.D.E. and P,Q stable methods for D.D.E. is not known in general. Clearly, for the specific methods considered in Chapter 4, the properties of methods for D.D.E. are similar to those for O.D.E. More study is needed to determine the relationship between methods for O.D.E. and for D.D.E.

Although the computer program in Appendix A has no obvious bugs at this stage, it still needs exhaustive testing by people other than the author. Also the package encounters some difficulty with the discontinuities in the higher order derivatives, even when including the points of discontinuity in the mesh. For these points, one could possibly adapt the formulas of Zverkina [22] for incorporation into this package. This would not be suitable for stiff problems because of the stability properties of Zverkina's methods [6]. However, one might be able to modify the backward differentiation formulas to account for discontinuities.

For a more general package we would like a package similar to Neves [16], which solves the retarded differential equation

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

where α is a lag function and the initial function is defined on the appropriate interval, and incorporates the generalized Adams and backward differentiation methods.

APPENDIX A

Fortran Computer Programs to Solve a D.D.E.

List of Subroutines

```
C*********************************************************************1
C      MAINLINE ROUTINE
C      THIS ROUTINE TESTS THE DDE SOLVER ON THE PROBLEM
C            Y1'(T) = -Y1(T-BETA)
C            Y2'(T) = -Y2(T-BETA)
C
C       WHERE BETA = PI/2 AND
C
C            Y1(T) = SIN(T)     ON (0,PI/2)
C            Y2(T) = COS(T)
C
C*********************************************************************
      REAL A(7,2), DELTAQ(2), SAVE(7,2), W(38), WORK(38)
      REAL CL(7,7), ERRCON(4,6), PASTT(100), PASTY(2,100),
     +     BETA, EPS, H, HMIN, T, YMAX
      INTEGER PASTQ(100), BEGIN, END, INDEX, START, Q, QCOUNT, TYPE,
     +     QMAX
      LOGICAL REEVAL
      COMMON IDEBUG, KDEBUG, LDEBUG
C---------------------------------------------------------------------
      N = 2
      IDEBUG = 3
      KDEBUG = 1
      LDEBUG = 0
      START = 0
      YMAX = 1.0
      REEVAL = .FALSE.
      QMAX = 100
      HMIN = 1.0E-10
      TYPE = 0
      PI = 3.14159
      BETA = PI/2.0
      H = .1
      T = BETA
      TEND = 5.0
      T0 = 0.0
      EPS = .01/(TEND - T0)
      CALL DDE( A, CL, DELTAQ, ERRCON, PASTT, PASTY, SAVE, W,
     +    WORK, BETA, EPS, H, HMIN, T, T0, TEND, YMAX,
     +        PASTQ, BEGIN, END, INDEX, N, Q, QCOUNT, QMAX,
     +        START, TYPE, REEVAL )
      STOP
      END
C*********************************************************************
C
      SUBROUTINE DERIV(T,Y,YB,F)
C
C*********************************************************************
C     THIS ROUTINE COMPUTES THE DERIVATIVE F(T,Y(T),Y(T-BETA))
C*********************************************************************
      REAL Y(1), YB(1), F(1)
      F(1) = -YB(1)
      F(2) = -YB(2)
      RETURN
      END
C
C*********************************************************************
C
      SUBROUTINE PHI( T, Y )
C
C*********************************************************************
      REAL Y(1), T
      Y(1) = SIN(T)
      Y(2) = COS(T)
      RETURN
      END
C*********************************************************************
C
      SUBROUTINE TRUE( T )
C
C*********************************************************************
      REAL T, Y1, Y2
      Y1 = SIN(T)
      Y2 = COS(T)
      WRITE(6,1000) T, Y1, T, Y2
 1000 FORMAT('      Y1 = SIN(',E14.7,') = ',E14.7/'      Y2 = COS(',
     +       E14.7,') = ',E14.7)
      RETURN
      END
C*********************************************************************
C
      SUBROUTINE CHBUG( T, IDEBUG )
C
C*********************************************************************
      RETURN
      END
```

```
C*********************************************************************
C
      SUBROUTINE DDE( A, CL, DELTAQ, ERRCON, PASTT, PASTY, SAVE, W, WORK
     +               ,BETA, EPS, H, HMIN, T, T0, TEND, YMAX,
     +  PASTQ, BEGIN, END, INDEX, N, Q, QCOUNT, QMAX, START, TYPE,
     +  REEVAL )
C
C*********************************************************************
C
C     AUTHOR AND IMPLEMENTER -   VICTOR K. BARWELL
C
C     THIS IS AN AUTOMATIC ROUTINE FOR SOLVING THE SYSTEM OF
C     DIFFERENTIAL DIFFERENCE EQUATIONS
C
C         Y'(T) = F(T, Y(T), Y(T-BETA)) FOR T .GT. T0 + BETA
C         Y(T)  = PHI(T)     FOR T0 .LE. T .LE. BETA
C
C     THIS PROGRAM REQUIRES THE SUBROUTINE DERIV( T, Y, YBETA, F )
C     TO EVALUATE F AND THE SUBROUTINE PHI( T, Y ) TO EVALUATE THE
C     INITIAL FUNCTION. Y, YBETA, F ARE VECTORS OF DIMENSION N.
C         THE EQUATION IS INTEGRATED FROM   T0 + BETA    TO    TEND
C     USING VARIABLE STEP, VARIABLE ORDER METHODS.
C
C     ALL ARITHMETIC IS SINGLE PRECISION AND THE SAME VARIABLE
C     NAMES ARE USED IN ALL THE SUBROUTINES.  THE OUTPUT IS DONE USING
C     A FORMAT CODE OF E14.7 FOR THE REAL VARIABLES.  THIS WOULD HAVE
C     TO BE CHANGED DEPENDING ON THE PRECISION OF THE MACHINE.  THE
C     CODE SHOULD OTHERWISE BE PORTABLE.  THE AUTHOR HAS PUT THE
C     RESONABLE RESTRICTION THAT THE SYSTEM OF EQUATIONS BEGIN SOLVED
C     HAS DIMENSION LESS THAN 100.
C
C     THE UNFAMILIAR USER NEED ONLY CONCERN HIMSELF WITH THOSE
C     PARAMETERS CHECKED WITH A *, AND PROVIDE THE APPROPIATE
C     STORAGE OR VARIABLES FOR THE OTHER PARAMETERS.  THE ONLY
C     VARIABLES WHICH CAN BE CHANGED ON RETURN TO THE PROGRAM
C     ARE YMAX AND TEND.
C
C     THE SOPHISTICATED USER CAN USE THE SUPPORTING SUBROUTINES
C     TO PROVIDE ADDITIONAL INFORMATION.  FOR EXAMPLE THE SUBROUTINE
C     FUNCT CAN BE USED TO PROVIDE THE SOLUTION AT OFF MESH POINTS
C     BY INTERPOLATION.
C
C
C     -----------
C     REAL ARRAYS
C     -----------
C
C  *   A(7,N)          - A VECTOR CONTAINING THE SCALED DERIVATIVES
C      A(1,N)          - WILL CONTAIN THE SOLUTION AT ANY GIVEN TIME.
C
C      CL(7,7)         - A UPPER HESSENBURG MATRIX WHICH IS USED TO STORE
C                        THE VECTORS L WHICH DEFINE THE CORRECTOR.
C      DELTAQ(N)       - USED BY THE PROGRAM FOR ESTIMATING STEP SIZE
C
C      ERRCON(4,6)     - AN ARRAY USED TO STORE ERROR CONSTANTS FOR
C                        DETERMINING STEP SIZE AND ORDER
C
C      PASTT(QMAX)     - A VECTOR USED TO STORE THE PAST TIME VALUES.
C
C      PASTY(N,QMAX) - A MATRIX USED TO STORE THE PAST SOLUTION VALUES.
C
C      SAVE(7,N)       - A TEMPORARY STORAGE AREA TO SAVE SCALED
C                       DERIVATIVES FOR RESTARTS AFTER THE FAILURE OF A
C                       STEP
C      W(N**2)         - A MATRIX TO HOLD THE JACOBIAN MATRIX USED IN THE
C                        CORRECTOR ITERATION FOR STIFF METHODS
C
C      WORK(15*N+2*N**2) - WORKING STORAGE FOR THE SUBROUTINES
C
C      IN THE CASE OF ADAMS METHODS USE W(1) AND WORK(13N)
C      THE USE OF THE WORK AREAS IS OUTLINED BELOW
C
C      WORK(1,N)             -  POLY1(N)
C      WORK(N+1,N)            - YB(N)
C      WORK(2N+1,9N)            - DIVDIV(7,N)
C      WORK(9N+1,10N)            - F(N), F
C      WORK(10N+1,11N)            - FPLUSR(N), SIGMAF(N)
C      WORK(12N+1,13N)             - Y(N)
C      WORK(13N+1,14N)              - YBAR(N)
C      WORK(14N+1,15N)               - YPLUS(N)
C      WORK(15N+1,15N+N**2)          - PDY
C      WORK(15N+1+N**2,15N+1+2N**2) - PDYBAR
C
```

```
C
C      ---------------
C      REAL VARIABLES
C      ---------------
C
C  *   BETA - THE TIME LAG
C
C  *   EPS  - THE REQUESTED ERROR TOLERANCE
C
C  *   H    - STEP SIZE TO BE ATTEMPTED NEXT OR INITIALLY
C
C  *   HMIN - MINIMUM STEP SIZE TO BE USED
C
C  *   T    - THE INDEPENDENT VARIABLE
C
C  *   T0   - THE INITIAL FUNCTION IS GIVEN ON T0,T0+BETA.
C
C  *   TEND - THE ENDPOINT OF THE INTERVAL OF INTEGRATION.
C
C  *   YMAX - MAXIMUM L2-NORM OF THE SOLUTION TO DATE
C
C      ---------------
C      INTEGER ARRAYS
C      ---------------
C
C      PASTQ(QMAX) - A VECTOR USED TO STORE THE ORDER AT A PAST NODE.
C
C      -----------------
C      INTEGER VARIABLES
C      -----------------
C
C      BEGIN  - A POINTER TO THE BEGINING OF THE CIRCULAR QUEUE.
C
C      END    - A POINTER TO THE END OF THE CIRCULAR QUEUE.
C
C      INDEX  - A POINTER IN THE QUEUE SUCH THAT PASTT(INDEX) IS
C               THE FIRST NODE PAST T-BETA.
C
C  *   N      - DIMENSION OF THE SYSTEM
C
C  *   START  - 0 THE FIRST TIME THE SUBROUTINE IS CALLED.
C             - 1 TO CONTINUE COMPUTING THE SOLUTION FROM THE
C                 PRESENT TIME T WITH THE SAVED INFORMATION.
C
C  *   TYPE   - 0 ADAMS METHODS WITH ERROR PER UNIT STEP
C             - 1 ADAMS METHODS WITH ERROR PER STEP
C             - 2 BACKWARD DIFFERENTATION METHOD WITH
C             -   ERROR PER UNIT STEP ( NOT RECOMMENDED )
C             - 3 BACKWARD DIFFERENTATION METHOD WITH ERROR PER STEP
C
C      QCOUNT - INDICATER USED BY THE PROGRAM TO PREVENT FREQUENT
C             - TESTING FOR POSSIBLE STEP INCREASE
C
C      Q      - ORDER OF THE FORMULA PRESENTLY BEING USED.
C
C  *   QMAX   - THE MAXIMUM SIZE OF THE CIRCULAR QUEUE USED TO SAVE
C                 INFORMATION ABOUT THE PAST SOLUTION
C
C      -----------------
C      LOGICAL VARIABLES
C      -----------------
C
C      REEVAL - INDICATES WHEN TO REEVALUATE A JACOBIAN IN STIFF
C               PROBLEMS
C
C***********************************************************************
      INTEGER PASTQ(1), BEGIN, COL, END, INDEX, START, N, Q, QCOUNT,
     +        QMAX, QPLUS1, TYPE
      REAL A(7,1), CL(7,7), DELTAQ(1), ERRCON(4,6), PASTT(1),
     +     PASTY(N,1), SAVE(7,1), W(1), WORK(1),
     +     BETA, EPS, H, HMIN, T, TEMP, YMAX
      LOGICAL DONE, FINISH, REEVAL, SUCESS
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
      IF (  ( IDEBUG .GE. 1 ) .AND. ( START .EQ. 0 )  )
     +   WRITE(6,1000) EPS, H, HMIN, BETA, T, N
 1000 FORMAT('-DDE SOLVER ENTERED'/' EPS = ',E14.7,' H = ',E14.7,
     +       ' HMIN = ',E14.7/' BETA = ',E14.7,' T = ',E14.7/
     +       ' A SYSTEM OF DIMENSION ',I2,' IS BEING SOLVED')
C
C      ------------------
C      CHECK FOR RESTARTS
C      ------------------
C
      IF ( START .EQ. 0 ) CALL
     +   SETUP( A, CL, ERRCON, PASTT, PASTY, WORK,
     +          BETA, EPS, H, T, T0,
     +          PASTQ, BEGIN, END, INDEX, N, Q, QCOUNT, QMAX, TYPE,
     +          REEVAL )
```

```
C
      FINISH = .FALSE.
      DONE = .FALSE.
C
C     ---------------
C     HAVE WE FINISHED
C     ---------------
C
 10   IF ( FINISH ) RETURN
C
      IF ( LDEBUG .NE.  0 ) CALL CHBUG( T, IDEBUG )
C
          IF ( T + 1.5*H .LT. TEND ) GO TO 30
C
              IF ( DONE ) GO TO  20
C
                  ALPHA = 0.5*( TEND - T )/H
                  IF ( IDEBUG .GE. 3 ) WRITE(6,999)
999               FORMAT(' STEP CHANGE TO REACH ENDPOINT EXACTLY')
                  CALL CHSTEP( A, ALPHA, H, HMIN, N, Q, REEVAL )
                  QCOUNT = Q + 1
                  DONE = .TRUE.
                  GO TO  30
C
 20           FINISH = .TRUE.
              IF (  ( IDEBUG .GE. 1 ) .AND. ( START .EQ. 0 )   )
     +        WRITE(6,1001)
1001          FORMAT('0DDE WILL TERMINATE IF STEP IS SUCESSFUL')
C
C         ----------------------
C         SAVE SCALED DERIVATIVES
C         FOR RESTARTS
C         ----------------------
C
 30       CALL PUT( A, SAVE, N, Q + 1 )
C
              CALL PREDCT( A, N, Q )
C
              IF ( TYPE .LE. 1 ) CALL
     +        CORECT( A, CL, WORK( 2*N+1 ), WORK( 9*N+1 ), WORK( 10*N+1 ),
     +              PASTT,  PASTY, WORK( 1 ), SAVE, WORK( 12*N+1 ),
     +              WORK( N+1 ), BETA, EPS, H, HMIN, T, T0, YMAX,
     +              PASTQ, BEGIN, END, INDEX, N, Q, QMAX, REEVAL )
C
              IF ( TYPE .GE. 2 ) CALL
     +        STIFFC( A, CL, WORK( 2*N+1 ), WORK( 9*N+1 ), WORK( 10*N+1 ),
     +              PASTT, PASTY, WORK( 1 ), WORK( 15*N+1 ),
     +              WORK( 15*N+1+N**2 ), SAVE, WORK( 2*N+1 ), WORK( 12*N+1 ),
     +              WORK( N+1 ), WORK( 13*N+1 ), WORK( 14*N+1 ), W( 1 ),
     +              BETA, EPS, H, HMIN, T, T0, YMAX,
     +              PASTQ, BEGIN, END, INDEX, N, Q, QMAX, REEVAL )
C
              CALL CHKERR( A, DELTAQ, ERRCON, SAVE,
     +                     EPS, H, T, HMIN, YMAX,
     +                     INDEX, N, Q, QCOUNT, TYPE,
     +                     REEVAL, SUCESS, FINISH )
C
              IF ( SUCESS ) GO TO  40
C
              DONE = .FALSE.
          GO TO 30
C
C         --------------------
C         UPDATE QUEUE FOR
C         PAST FUNCTION VALUES
C         --------------------
C
 40       CALL ADD( A, PASTT, PASTY, T, PASTQ, BEGIN, END, N, Q, QMAX )
          CALL DELETE( BEGIN, END, INDEX, QMAX )
C
C         ----------------------
C         COMPUTE THE L2-NORM OF
C         MAXIMUM SOLUTION TO DATE
C         ----------------------
C
         TEMP = 0.0
         DO 50 COL = 1, N
             TEMP = TEMP + ABS( A( 1,COL) )**2
 50      CONTINUE
         YMAX = AMAX1( YMAX, SQRT( TEMP) )
      GO TO 10
      END
```

```
C***********************************************************************
C
      SUBROUTINE SETUP( A, CL, ERRCON, PASTT, PASTY, WORK,
     +                  BETA, EPS, H, T, T0,
     +                  PASTQ, BEGIN, END, INDEX, N, Q, QCOUNT, QMAX, TYPE,
     +                  REEVAL )
C
C***********************************************************************
C     THIS PROGRAM INITIALIZES THE ERROR CONSTANTS NEEDED TO ESTIMATE
C     THE ERROR IN THE STEP, THE VECTOR L USED IN THE CORRECTOR AND
C     THE INTIAL STARTING VALUES NEEDED FOR ORDER, STEP SIZE AND PAST
C     FUNCTION VALUES FOR THE ALL THE METHODS.
C***********************************************************************
      INTEGER PASTQ(1), BEGIN, END, I, J, K, N, Q, QCOUNT, QMAX, TYPE
      INTEGER COL
      REAL A(7,1), CL(7,7), CQ(7), ERRCON(4,6), PASTT(1), PASTY(N,1),
     +     W(1), WORK(1), BETA, H, EPS
      REAL QFACT(6) /1.0, 2.0, 6.0, 24.0, 120.0, 720.0/
      LOGICAL REEVAL
      COMMON IDEBUG, KDEBUG, LDEBUG
C     ------------------------------------------------------------------
C
C     ----------------------
C     TEST FOR ADAMS METHODS
C     ----------------------
C
      IF ( TYPE .GE. 2 ) GO TO 10
C
C         ------------------------------------------------
C         INITIALIZE ABS(CQ) IN THE ERROR ESTIMATE
C         CQ+1*H**(Q+1)*Y(Q+1)/Q-FACTORIAL FOR ADAMS METHODS
C         ------------------------------------------------
C
          CQ(1) = 1.0
          CQ(2) = 1.0/2.0
          CQ(3) = 1.0/12.0
          CQ(4) = 1.0/24.0
          CQ(5) = 19.0/720.0
          CQ(6) = 3.0/160.0
          CQ(7) = 863.0/60480.0
C
C         -----------------------------------
C         THE CORRECTOR STEP FOR ADAMS METHODS
C         -----------------------------------
C
          CL(1,1) = 1.0
          CL(2,1) = 1.0
C
          CL(1,2) = 1.0/2.0
          CL(2,2) = 1.0
          CL(3,2) = 1.0
C
          CL(1,3) = 5.0/12.0
          CL(2,3) = 1.0
          CL(3,3) = 3.0/4.0
          CL(4,3) = 1.0/6.0
C
          CL(1,4) = 3.0/8.0
          CL(2,4) = 1.0
          CL(3,4) = 11.0/12.0
          CL(4,4) = 1.0/3.0
          CL(5,4) = 1.0/24.0
C
          CL(1,5) = 251.0/720.0
          CL(2,5) = 1.0
          CL(3,5) = 25.0/24.0
          CL(4,5) = 35.0/72.0
          CL(5,5) = 5.0/48.0
          CL(6,5) = 1.0/120.0
C
          CL(1,6) = 95.0/288.0
          CL(2,6) = 1.0
          CL(3,6) = 137.0/120.0
          CL(4,6) = 5.0/8.0
          CL(5,6) = 17.0/96.0
          CL(6,6) = 1.0/40.0
          CL(7,6) = 1.0/720.0
C
          IF ( IDEBUG .GE. 1 ) WRITE(6,1000)
1000      FORMAT('0INITIALIZATION DONE FOR ADAMS METHODS')
          GO TO 30
```

```
C
C       -------------------------------------------------
C       INITIALIZE ABS(CQ) IN THE ERROR ESTIMATE
C       CQ+1*H**(Q+1)*Y(Q+1)/Q-FACTORIAL FOR STIFF METHODS
C       -------------------------------------------------
C
 10     DO 20 J = 1, 7
            CQ(J) = 1.0/FLOAT(J)
 20     CONTINUE
C
C
C       ----------------------------
C       INITIALIZE THE L-VECTORS IN
C       THE STIFF CORRECTOR
C       ----------------------------
C
        CL(1,1) = 1.0
        CL(2,1) = 1.0
C
        CL(1,2) = 2.0/3.0
        CL(2,2) = 1.0
        CL(3,2) = 1.0/3.0
C
        CL(1,3) = 6.0/11.0
        CL(2,3) = 1.0
        CL(3,3) = 6.0/11.0
        CL(4,3) = 1.0/11.0
C
        CL(1,4) = 12.0/25.0
        CL(2,4) = 1.0
        CL(3,4) = 7.0/10.00
        CL(4,4) = 1.0/5.0
        CL(5,4) = 1.0/50.0
C
        CL(1,5) = 60.0/137.0
        CL(2,5) = 1.0
        CL(3,5) = 225.0/274.0
        CL(4,5) = 85.0/274.0
        CL(5,5) = 15.0/274.0
        CL(6,5) = 1.0/274.0
C
        CL(1,6) = 60.0/147.0
        CL(2,6) = 1.0
        CL(3,6) = 406.0/441.0
        CL(4,6) = 245.0/588.0
        CL(5,6) = 175.0/1764.0
        CL(6,6) = 7.0/588.0
        CL(7,6) = 1.0/1764.0
C
        IF ( IDEBUG .GE. 1 ) WRITE(6,1010)
 1010   FORMAT('0INITIALIZATION DONE FOR STIFF METHODS')
C
C       --------------------------
C       TEST FOR ERROR PER UNIT STEP
C       --------------------------
C
 30     IF ( (TYPE .EQ. 1) .OR. (TYPE .EQ. 3) ) GO TO 60
C
C       ---------------------------------------------------------------
C       INITIALIZE THE ERROR CONSTANTS USED FOR ESTIMATING THE STEP SIZE
C       THE SECOND COMPONENT OF THE ARRAY IS ASSOCIATED WITH THE ORDER.
C       THIS INITIALIZES CONSTANTS FOR ERROR PER UNIT STEP
C       ---------------------------------------------------------------
C
        ERRCON(2,1) = 1.0/(CQ(2)*QFACT(1))/1.2
        ERRCON(3,1) = SQRT(1.0/(CQ(3)*QFACT(1)))/1.4
        DO 40 J = 2, 5
            ERRCON(1,J) = (1.0/(CQ(J)*QFACT(J)))**(1.0/FLOAT(J-1))/1.3
            ERRCON(2,J) = (1.0/(CQ(J+1)*QFACT(J)))**(1.0/FLOAT(J))/1.2
            ERRCON(3,J) = (1.0/(CQ(J+2)*QFACT(J)))**(1.0/FLOAT(J+1))/1.4
 40     CONTINUE
        ERRCON(1,6) = (1.0/(CQ(6)*QFACT(6)))**(1.0/5.0)/1.3
        ERRCON(2,6) = (1.0/(CQ(7)*QFACT(6)))**(1.0/6.0)/1.2
        DO 50 J = 1,6
            ERRCON(4,J) = QFACT(J)*CQ(J+1)
 50     CONTINUE
C
        IF ( IDEBUG. GE. 1 ) WRITE(6,1020)
 1020   FORMAT(' ERROR PER UNIT STEP USED')
```

```
C
      GO TO 90
C
C     ----------------------------------------------------------------
C     INITIALIZE THE ERROR CONSTANTS USED FOR ESTIMATING THE STEP SIZE
C     THE SECOND COMPONENT OF THE ARRAY IS ASSOCIATED WITH THE ORDER.
C     THIS INITIALIZES CONSTANTS FOR ERROR PER STEP
C     ----------------------------------------------------------------
C
 60   ERRCON(2,1) = 1.0/SQRT(CQ(2)*QFACT(1))/1.2
      ERRCON(3,1) = (1.0/(CQ(3)*QFACT(1)))**(1.0/3.0)/1.4
      DO 70 J = 2, 5
         ERRCON(1,J) = (1.0/(CQ(J)*QFACT(J)))**(1.0/FLOAT(J))/1.3
         ERRCON(2,J) = (1.0/(CQ(J+1)*QFACT(J)))**(1.0/FLOAT(J+1))/1.2
         ERRCON(3,J) = (1.0/(CQ(J+2)*QFACT(J)))**(1.0/FLOAT(J+2))/1.4
 70   CONTINUE
      ERRCON(1,6) = (1.0/(CQ(6)*QFACT(6)))**(1.0/6.0)/1.3
      ERRCON(2,6) = (1.0/(CQ(7)*QFACT(6)))**(1.0/7.0)/1.2
      DO 80 J = 1,6
         ERRCON(4,J) = QFACT(J)*CQ(J+1)
 80   CONTINUE
C
      IF ( IDEBUG .GE. 1 ) WRITE(6,1030)
 1030 FORMAT(' ERROR PER STEP USED')
C
C     ------------------------------------------
C     INITIALIZE ORDER, STEP SIZE
C     PAST FUNCTION VALUES,
C     QUEUE POINTERS, AND JACOBIANS
C
C     WORK(1-N) IS USED TO STORE Y(T0 + BETA)
C
C     WORK(N+1,2N) IS USED TO STORE Y(T0)
C
C     WORK(2N+1,3N) IS USED TO STORE Y'(T0+BETA)
C
C     ------------------------------------------
C
 90   Q = 1
      QCOUNT = 2
      T = T0 + BETA
      H = AMIN1( H, BETA )
C
C     -----------------------------------
C     INITIALIZE THE SOLUTION AT T0 + BETA
C     -----------------------------------
C
      CALL PHI( T, WORK(1) )
      CALL PHI( T0, WORK(N+1) )
      CALL DERIV( T, WORK(1), WORK(N+1), WORK(2*N+1) )
      DO 100 COL = 1, N
         A(1,COL) = WORK(COL)
         A(2,COL) = H*WORK(2*N+COL)
 100  CONTINUE
C
C     -------------------
C     INITIALIZE THE QUEUE
C     -------------------
C
      INDEX = 1
      BEGIN = 0
      END = 0
      CALL ADD( A, PASTT, PASTY, T,
     +          PASTQ, BEGIN, END, N, Q, QMAX )
      BEGIN = 1
C
      IF ( TYPE .GE. 2 ) REEVAL = .TRUE.
C
      IF ( IDEBUG  .GE. 5 ) WRITE(6,1040)
 1040 FORMAT(' INITIAL SCALED DERIVATIVES ARE')
      IF ( IDEBUG .GE. 5 ) CALL OUT( A, N, 2 )
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE CORECT( A, CL, DIVDIF, F, SIGMAF, PASTT, PASTY, POLY1,
     +                   SAVE, Y, YB,
     +                   BETA, EPS, H, HMIN, T, T0, YMAX,
     +                   PASTQ, BEGIN, END, INDEX, N, Q, QMAX, REEVAL )
C
C***********************************************************************
C     THIS PROGRAM PERFORMS THE CORRECTOR ITERATION
C     A(N,M) = A(N,0) + CL*( F(A(N,0)) + ... + F(A(N,M-1)))
C
C     CL(*,Q) - VECTOR FOR CORRECTOR OF ORDER Q
C***********************************************************************
      INTEGER PASTQ(1), BEGIN, COL, END, INDEX, J, Q, QMAX, QPLUS1, ROW
      REAL A(7,1), CL(7,7), DIVDIF(7,1), F(1), PASTT(1),
     +     PASTY(N,1), POLY1(1), SAVE(1), SIGMAF(1), TNODE(7),
     +     Y(1), YB(1),
     +     EPS, EPSC, H, HMIN, YMAX
      LOGICAL SMALLH, CORR, REEVAL
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
      REEVAL = .FALSE.
 10   EPSC = EPS*H*YMAX/(2.0*FLOAT(Q+1))
      IF ( IDEBUG .GE. 4 ) WRITE(6,1000) Q, EPSC, T, H
1000  FORMAT('-CORRECTOR STARTED WITH ORDER = ',I1
     +       /' EPSC= ',E14.7,' T= ',E14.7,' H= ',E14.7)
      DO 20 ROW = 1, N
         SIGMAF( ROW) = 0.0
 20   CONTINUE
      TPLUSH = T + H
      TBACK = TPLUSH - BETA
      REEVAL = .FALSE.
      SMALLH = .TRUE.
      CORR = .FALSE.
C
      IF ( H .GT. BETA ) SMALLH = .FALSE.
      IF ( SMALLH ) GO TO 30
C
         CALL ADD( A, PASTT, PASTY, TPLUSH, PASTQ, BEGIN, END,
     +             N ,Q, QMAX )
         INDEX = END
 30   CALL FUNCT( DIVDIF, TNODE, PASTT, PASTY, YB, POLY1,
     +            BETA, DIFF, GPRIME, TBACK, T0,
     +            PASTQ, BEGIN, END, INDEX, N, NPTS, QMAX,
     +            CORR, REEVAL, SMALLH )
C
C     ---------------------------------------------
C     FUNCT COMPUTES THE PAST FUNCTION VALUE Y(T-BETA)
C     DERIV COMPUTES F(T, Y(T), Y(T-BETA))
C     ---------------------------------------------
C
      DO 90 I = 1, 3
         CORR = .TRUE.
         DO 40 COL = 1, N
            Y(COL) = A(1,COL)
 40      CONTINUE
C
         CALL DERIV( TPLUSH, Y, YB, F )
C
         DO 50 COL = 1, N
            F(COL) = H*F(COL) - A(2,COL)
            SIGMAF(COL) = SIGMAF(COL) + F(COL)
 50      CONTINUE
C
         IF ( IDEBUG .GE. 5 ) WRITE(6,1001) I
1001     FORMAT('0BEFORE CORRECTION', I1, ' WE HAVE')
         IF ( IDEBUG .GE. 5 ) CALL OUT ( A, N, 2 )
C
C        ---------------------------
C        CORRECT FIRST TWO COMPONENTS
C        ---------------------------
C
         DO 60 COL = 1, N
            A(1,COL) = A(1,COL) + CL(1,Q)*F(COL)
            A(2,COL) = A(2,COL) + F(COL)
 60      CONTINUE
C
         IF ( IDEBUG .GE. 5 ) WRITE(6,1002) I
1002     FORMAT(' AFTER CORRECTION ',I1,' WE HAVE' )
         IF ( IDEBUG .GE. 5 ) CALL OUT ( A, N, 2 )
C
C        ------------------
C        TEST FOR CONVERGENCE
C        OF THE CORRECTOR
C        ------------------
C
```

```
          ABSF = 0.0
          DO 70 COL = 1, N
              ABSF = ABSF + ABS( F(COL) )**2
 70       CONTINUE
          ABSF = SQRT(ABSF)
          IF ( IDEBUG .GE. 5 ) WRITE(6,2000) ABSF
 2000     FORMAT(' ', 'L2-NORM OF CORRECTION TERM IS ',E14.7)
          IF ( ABSF .LE. EPSC ) GO TO 110
              IF ( SMALLH .OR. ( I .EQ. 3 ) ) GO TO 90
C
C                 -----------------------------------------------
C                 CHANGE LAST ENTRY IN THE DIVIDED DIFFERENCE TABLE
C                 -----------------------------------------------
C
                  DO 80 COL = 1, N
                      DIVDIF(NPTS,COL) = A(1,COL)
 80               CONTINUE
C
                  CALL FUNCT( DIVDIF, TNODE, PASTT, PASTY, YB, POLY1,
     +                        BETA, DIFF, GPRIME, TBACK, T0,
     +                        PASTQ, BEGIN, END, INDEX, N, NPTS, QMAX,
     +                        CORR, REEVAL, SMALLH )
C
 90       CONTINUE
C
C         --------------------------------
C         CORRECTOR FAILED TO CONVERGE SO
C         CHANGE THE STEP SIZE
C         --------------------------------
C
C
          IF ( IDEBUG .GE. 3 ) WRITE (6,1003)
 1003     FORMAT('0FUNCTIONAL CORRECTOR FAILED TO CONVERGE')
          ALPHA = 0.25
          IF ( H .LT. HMIN ) CALL ERROR( 2 )
C
          IF ( SMALLH ) GO TO 100
             END = END - 1
             IF ( END .EQ. 0 ) END = QMAX
C
 100      CALL PUT( SAVE, A, N, Q + 1 )
          CALL CHSTEP( A, ALPHA, H, HMIN, N, Q, REEVAL )
          CALL PUT( A, SAVE, N, Q + 1 )
          CALL PREDCT( A, N, Q )
          GO TO 10
C
C         -------------------
C         CORRECTOR CONVERGED
C         COMPLETE ITERATION
C         -------------------
C
 110  QPLUS1 = Q + 1
      IF ( SMALLH ) GO TO 120
         END = END - 1
         IF ( END .EQ. 0 ) END = QMAX
 120  IF ( QPLUS1 .LT. 3 ) GO TO 150
C
          DO 140 COL = 1, N
              DO 130 ROW = 3, QPLUS1
                  A(ROW,COL) = A(ROW,COL) + CL(ROW,Q)*SIGMAF(COL)
 130          CONTINUE
 140      CONTINUE
C
 150  IF ( IDEBUG .GE. 4 ) WRITE(6,1004) H, T, Q
 1004 FORMAT('0CORRECTOR DONE WITH '/' H= ',E14.7,' T= ',E14.7,
     +     ' ORDER= ',I1)
      IF ( IDEBUG .GE. 5 ) CALL OUT( A, N, Q + 1 )
      RETURN
      END
C***********************************************************************
```

```
C
      SUBROUTINE STIFFC( A, CL, DIVDIF, F, FPLUSR, PASTT, PASTY, POLY1,
     +                   PDY, PDYBAR, SAVE, SIGMAF, Y, YB, YBAR,
     +                   YPLUS, W,
     +                   BETA, EPS, H, HMIN, T, T0, YMAX,
     +                   PASTQ, BEGIN, END, INDEX, N, Q, QMAX, REEVAL )
C
C*********************************************************************
C     THIS PROGRAM PERFORMS THE CORRECTOR ITERATION
C     A(N,M+1) = A(N,M) - W-1*F(A(N,M))
C
C     W - THE JACOBIAN MATRIX
C
C     CL(*,Q) - VECTOR FOR CORRECTOR OF ORDER Q
C*********************************************************************
       INTEGER PIVOT(99)
      INTEGER PASTQ(1), BEGIN, COL, END, INDEX, J, Q, QMAX, QPLUS1, ROW
      REAL A(7,1), CL(7,7), DIVDIF(7,1), F(1),
     +     PASTT(1), PASTY(N,1), POLY1(1), PDY(N,1), PDYBAR(N,1),
     +     SAVE(7,1), SIGMAF(1), TNODE(7), W(1), Y(1), YB(1),
     +             YBAR(1), YPLUS(1),
     +     EPS, EPSC, H, HMIN, YMAX
      LOGICAL SMALLH, CORR, REEVAL
      COMMON IDEBUG, KDEBUG, LDEBUG
C---------------------------------------------------------------------
 10   EPSC = EPS*H*YMAX/(2.0*FLOAT(Q+1))
      IF ( IDEBUG .GE. 5 ) WRITE(6,996) Q, EPSC, T, H
 996  FORMAT('-NEWTON CORRECTOR STARTED WITH ORDER = ',I1
     +       /' EPSC= ',E14.7,' T= ',E14.7,' H= ',E14.7)
      TPLUSH = T + H
      TBACK = TPLUSH - BETA
      SMALLH = .TRUE.
      CORR = .FALSE.
C
      IF ( H .GT. BETA ) SMALLH = .FALSE.
      IF ( SMALLH ) GO TO 20
C
         CALL ADD( A, PASTT, PASTY, TPLUSH, PASTQ, BEGIN, END,
     +             N, Q, QMAX )
         INDEX = END
 20   CALL FUNCT( DIVDIF, TNODE, PASTT, PASTY, YB, POLY1,
     +            BETA, DIFF, GPRIME, TBACK, T0,
     +            PASTQ, BEGIN, END, INDEX, NPTS, QMAX,
     +            CORR, REEVAL, SMALLH )
C
      IF ( .NOT. REEVAL ) GO TO 90
C
C        -----------------------------------------------------
C        PARTIAL DERIVATIVES ARE EVALUATED, THE JACOBIAN MATRIX
C        IS EVALUATED AND THE LU-DECOMPOSITION IS FOUND
C        -----------------------------------------------------
C
         DO 30 COL = 1, N
            Y(COL) = A(1,COL)
            YBAR(COL) = YB(COL)
 30      CONTINUE
C
         CALL JACOB( F, FPLUSR, PDY, PDYBAR, Y, YBAR, YPLUSR,
     +               EPS, T, N )
C
         DO 50 COL = 1, N
            DO 40 ROW = 1, N
               W(N*(COL-1) + ROW) = CL(1,Q)*H*(   PDY(ROW,COL) +
     +                              GPRIME*PDYBAR(ROW,COL)   )
 40         CONTINUE
         W(N*(COL-1) + COL) = -1.0 + W(N*(COL-1) + COL)
 50      CONTINUE
C
         IF ( IDEBUG .LT. 4 ) GO TO 80
C
            WRITE(6,997)
 997        FORMAT(' NEW PARTIAL DERIVATIVES AND JACOBIAN ARE')
C
            DO 70 COL = 1, N
               WRITE(6,998) COL
 998           FORMAT('0',I2,'      COLUMNS OF THE MATRICES ARE'
     +                /' ',6X,'PDY',11X,'PDYBAR',6X,'W')
               DO 60 ROW = 1, N
                  WRITE(6,999) PDY(ROW,COL), PDYBAR(ROW,COL),
     +                         W(N*(COL-1) + ROW)
 999              FORMAT(' ',3(E14.7,1X))
 60            CONTINUE
 70         CONTINUE
C
 80      CALL DECOMP( N, N, W, PIVOT )
```

```
C
 90     CONTINUE
C
        DO 100 ROW = 1, N
            SIGMAF( ROW ) = 0.0
 100    CONTINUE
C
C       ------------------------------------------------
C       FUNCT COMPUTES THE PAST FUNCTION VALUE Y( T-BETA)
C       DERIV COMPUTES F(T, Y(T), Y( T-BETA))
C       ------------------------------------------------
C
        DO 170 I = 1, 3
            CORR = .TRUE.
C
            DO 110 COL = 1, N
                Y(COL) = A( 1,COL)
 110        CONTINUE
C
            CALL DERIV( TPLUSH, Y, YB, F )
C
            DO 120 COL = 1, N
                F(COL) = H*F( COL) - A( 2,COL)
 120        CONTINUE
C
C
            IF ( IDEBUG .GE. 5 ) WRITE( 6,1001) I
1001        FORMAT( '0BEFORE CORRECTION ',I1,' WE HAVE' )
            IF ( IDEBUG .GE. 5 ) CALL OUT ( A, N, 2 )
C
C           -------------------------
C           COMPUTE W-INVERSE*F
C           AND SAVE IN F
C           CORRECT FIRST TWO COMPONENTS
C           -------------------------
C
            CALL SOLVE( N, N, W, F, PIVOT )
C
            DO 130 ROW = 1, N
                SIGMAF( ROW) = SIGMAF( ROW) + F( ROW)
 130        CONTINUE
C
            DO 140 COL = 1, N
                A( 1,COL) = A( 1,COL) - CL( 1,Q)*F( COL)
                A( 2,COL) = A( 2,COL) - F( COL)
 140        CONTINUE
C
            IF ( IDEBUG .GE. 5 ) WRITE( 6,1002) I
1002        FORMAT( ' AFTER CORRECTION ',I1,' WE HAVE ' )
            IF ( IDEBUG .GE. 5 ) CALL OUT ( A, N, 2 )
C
C           -------------------
C           TEST FOR CONVERGENCE
C           OF THE CORRECTOR
C           -------------------
C
            ABSWF = 0.0
            DO 150 COL = 1, N
                ABSWF = ABSWF + F( COL)**2
 150        CONTINUE
            ABSWF = SQRT( ABSWF )
            IF ( IDEBUG .GE. 5 ) WRITE( 6,907) ABSWF
 907        FORMAT( ' L2-NORM OF THE CORRECTION TERM W-1*F IS')
C
            IF ( ABSWF .LE. EPSC ) GO TO 210
            IF ( SMALLH ) GO TO 170
C
C               ------------------------------------------------
C               CHANGE LAST ENTRY IN THE DIVIDED DIFFERENCE TABLE
C               ------------------------------------------------
C
                DO 160 COL = 1, N
                    DIVDIF(NPTS,COL) = A( 1,COL)
 160            CONTINUE
C
                CALL FUNCT( DIVDIF, TNODE, PASTT, PASTY, BETA, DIFF,
     +              GPRIME, YB, POLY1, TBACK, TO,
     +              PASTQ, BEGIN, END, INDEX, N, NPTS, QMAX,
     +              CORR, REEVAL, SMALLH )
C
 170    CONTINUE
C
```

```
C
C     ------------------------------------
C     CORRECTOR FAILED TO CONVERGE SO CHECK
C     FOR REEVALUATION OF THE JACOBIAN
C     ------------------------------------
C
      IF ( IDEBUG .GE. 3 ) WRITE(6,1003)
1003  FORMAT('ONEWTON ITERATION FAILED TO CONVERGE')
C
      IF ( REEVAL ) GO TO 190
C
          IF ( IDEBUG .GE. 3 ) WRITE(6,1004)
1004      FORMAT('+',38X,'JACOBIAN IS REEVALUATED')
          REEVAL = .TRUE.
C
          IF ( SMALLH ) GO TO 180
              END = END - 1
              IF ( END .EQ. 0 ) END = QMAX
 180      CALL PUT( SAVE, A, N, Q+1 )
          CALL PREDCT( A, N, Q )
          GO TO 10
C
C     -------------------------
C     JACOBIAN ALREADY REEVALUATED
C     SO CHANGE THE STEP SIZE
C     -------------------------
C
 190  ALPHA = 0.25
      IF ( IDEBUG .GE. 3 ) WRITE (6,1005)
1005  FORMAT('0JACOBIAN ALREADY REEVALUATED SO STEP SIZE IS CHANGED')
      IF ( H .LT. HMIN ) CALL ERROR( 2 )
C
      IF ( SMALLH) GO TO 200
         END = END - 1
         IF ( END .EQ. 0 ) END = QMAX
C
 200  CALL PUT( SAVE, A, N, Q + 1 )
      CALL CHSTEP( A, ALPHA, H, HMIN, N, Q, REEVAL )
      CALL PUT( A, SAVE, N, Q + 1 )
      CALL PREDCT( A, N, Q )
      GO TO 10
C
C     ------------------------------
C     CORRECTOR CONVERGED, DELETE FROM
C     QUEUE AND COMPLETE ITERATION
C     ------------------------------
C
 210  IF ( SMALLH ) GO TO 220
         END = END - 1
         IF ( END .EQ. 0 ) END = QMAX
 220  QPLUS1 = Q + 1
      IF ( QPLUS1 .LT. 3 ) GO TO 250
C
          DO 240 COL = 1, N
              DO 230 ROW = 3, QPLUS1
                  A( ROW,COL ) = A( ROW,COL ) - CL( ROW,Q )*SIGMAF( COL )
 230          CONTINUE
 240      CONTINUE
C
C
 250  IF ( IDEBUG .GE. 5) WRITE(6,1006) H, T, Q
1006  FORMAT('0CORRECTOR DONE WITH '/' H= ',E14.7,' T= ',E14.7,
     +       ' ORDER= ',I1)
      IF ( IDEBUG .GE. 5 ) CALL OUT( A, N, Q + 1 )
      REEVAL = .FALSE.
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE CHKERR( A, DELTAQ, ERRCON, SAVE, EPS, H, T,
     +         HMIN, YMAX,
     +      INDEX, N, Q, QCOUNT, TYPE, REEVAL, SUCESS, FINISH )
C
C***********************************************************************
C     THIS PROGRAM CONTAINS THE LOGIC FOR DETERMINING
C     THE SUCESS OF A STEP AND FOR CHANGING STEP SIZE
C     AND ORDER.
C***********************************************************************
      INTEGER COL, K, Q, QCOUNT, QPLUS1, TYPE
      REAL A(7,1), ALPH(2), DELTAQ(1), ERRCON(4,6), SAVE(7,1),
     +     DELSQ, EPS, H, NORMAQ, NRMDAQ
      LOGICAL FINISH, REEVAL, SUCESS
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
      SUCESS = .TRUE.
      QPLUS1 = Q + 1
C
      NRMDAQ = 0.0
      DO 10 COL = 1, N
         NRMDAQ = NRMDAQ + ( A(QPLUS1,COL) - SAVE(QPLUS1,COL) )**2
 10   CONTINUE
      NRMDAQ = SQRT( NRMDAQ )
C
      ERR = ABS( ERRCON(4,Q)*NRMDAQ )
      IF ( (TYPE .EQ. 0) .OR. (TYPE .EQ. 2) ) ERR = ERR/H
C
      IF ( ERR .GT. EPS*YMAX ) GO TO 120
C
C          -------------
C          STEP SUCEEDED
C          -------------
C
           IF ( IDEBUG .GE. 3 ) WRITE(6,1000) H
1000       FORMAT('-STEP SUCCEEDED WITH H= ',E14.7)
           T = T + H
           IF ( IDEBUG .GE. 2 ) WRITE(6,1001) T
1001       FORMAT(' SOLUTION AT T = ',E14.7,' IS ')
           IF ( IDEBUG .GE. 2 ) CALL OUT( A, N, 1 )
           IF ( KDEBUG .NE. 1 )  GO TO 20
                WRITE(6,1002)
1002            FORMAT(' TRUE SOLUTION IS')
                CALL TRUE(T)
C
20         QCOUNT = QCOUNT - 1
           IF ( QCOUNT .GT. 1 ) RETURN
           IF ( QCOUNT .EQ. 0 ) GO TO 40
C
C               ---------------------------------------------------
C               POSSIBLE INCREASE AT THIS OR THE NEXT STEP SO DELTAQ
C               IS SAVED
C               ---------------------------------------------------
C
                DO 30 COL = 1, N
                     DELTAQ(COL) = A(QPLUS1,COL) - SAVE(QPLUS1,COL)
 30             CONTINUE
                RETURN
C
C          --------------------
C          COMPUTE DELTA SQUARED
C          --------------------
C
 40        DELSQ = 0.0
           DO 50 COL = 1, N
                TEMP = A(QPLUS1,COL) - SAVE(QPLUS1,COL)
                DELSQ = DELSQ + ( TEMP - DELTAQ(COL) )**2
                DELTAQ(COL) = TEMP
 50        CONTINUE
           DELSQ = SQRT(DELSQ)
C
C          --------------------------------
C          PICK ORDER Q, Q+1 UP TO A MAXIMUM
C          OF SIX TO GIVE MAX STEP SIZE
C          --------------------------------
C
           IF ( IDEBUG .GE. 3 ) WRITE(6,1003)
1003       FORMAT('0POSSIBLE INCREASE IN ORDER AND STEP SIZE')
           QCOUNT = 10
           K = 1
C
           IF ( NRMDAQ .NE. 0.0 ) GO TO 60
                IF ( Q .NE. 6 ) K = 2
                ALPH(K) = 10.0
                GO TO 90
C
```

```
 60          IF ( (TYPE .EQ. 0) .OR. (TYPE .EQ. 2) )
     +       ALPH(1) = ERRCON(2,Q)*ABS( EPS*H*YMAX/NRMDAQ)**(1.0/FLOAT(Q))
             IF ( (TYPE .EQ. 1) .OR. (TYPE .EQ. 3) )
     +       ALPH(1) = ERRCON(2,Q)*ABS( EPS*YMAX/NRMDAQ)**(1.0/FLOAT(Q+1))
C
             IF ( Q .EQ. 6 ) GO TO 80
C
             IF ( DELSQ .NE. 0.0 ) GO TO 70
                K = 2
                ALPH(2) = 10.0
                GO TO 90
C
 70          IF ( (TYPE .EQ. 0) .OR. (TYPE .EQ. 2) )
     +       ALPH(2) = ERRCON(3,Q)*ABS( EPS*H*YMAX/DELSQ )**( 1.0/FLOAT(Q+1 ))
             IF ( (TYPE .EQ. 1) .OR. (TYPE .EQ. 3) )
     +       ALPH(2) = ERRCON(3,Q)*ABS( EPS*YMAX/DELSQ )**(1.0/FLOAT(Q+2))
C
C            ---------------------------
C            DETERMINE THE MAXIMUM ALPHA
C            ---------------------------
C
             IF ( ALPH(2) .GT. ALPH(1) ) K = 2
C
C            ---------------------------------
C            IF ALPHA IS TOO SMALL NO CHANGE
C            ---------------------------------
C
 80          IF ( ALPH(K) .LE. 1.1 ) RETURN
C
C            ----------------------------------
C            IF TRUE THEN NO INCREASE IN ORDER
C            ----------------------------------
C
 90          IF ( K .EQ. 1 ) GO TO 110
C
C               --------------
C               INCREASE ORDER
C               --------------
C
                DO 100 COL = 1, N
                   A(Q+2,COL) = DELTAQ( COL)/FLOAT( Q+1 )
 100            CONTINUE
                   Q = Q + 1
                   IF ( IDEBUG .GE. 3 ) WRITE( 6,1004) Q
 1004              FORMAT(' ORDER INCREASED TO ', I1 )
C
 110         CALL CHSTEP( A, ALPH(K), H, HMIN, N, Q, REEVAL )
             QCOUNT = Q + 1
             RETURN
C
C       -----------
C       STEP FAILED
C       -----------
C
 120    QCOUNT = QPLUS1
        FINISH = .FALSE.
        SUCESS = .FALSE.
        IF ( (TYPE .EQ. 0) .OR. (TYPE .EQ. 2) )
     +  ALPH(2) = ERRCON(2,Q)*ABS( EPS*H*YMAX/NRMDAQ)**(1.0/FLOAT(Q))
        IF ( (TYPE .EQ. 1) .OR. (TYPE .EQ. 3) )
     +  ALPH(2) = ERRCON(2,Q)*ABS( EPS*YMAX/NRMDAQ)**(1.0/FLOAT(Q+1))
        CALL PUT( SAVE, A, N, Q + 1 )
        IF ( IDEBUG .GE. 3 ) WRITE( 6,1005) Q
 1005   FORMAT( '0STEP FAILED WITH ORDER = ',I1)
C
        IF ( Q .GT. 1 ) GO TO 130
C
C           --------------------
C           CONNOT DECREASE ORDER
C           --------------------
C
            CALL CHSTEP( A, ALPH(2), H, HMIN, N, Q, REEVAL )
            RETURN
C
C       ---------------------------------
C       POSSIBLE DECREASE IN ORDER BY ONE
C       ---------------------------------
C
 130    NORMAQ = 0.0
        DO 140 COL = 1, N
            NORMAQ = NORMAQ + A( Q+1,COL)**2
 140    CONTINUE
        NORMAQ = SQRT( NORMAQ)
```

```
C
      IF ( NORMAQ .NE. 0.0 ) GO TO 150
           ALPH(1) = 0.1
           GO TO 160
 150  IF (  (TYPE .EQ. 0) .OR. (TYPE .EQ. 2)   )
     + ALPH(1) = ERRCON(1,Q)*ABS( EPS*H*YMAX/NORMAQ)**(1.0/FLOAT(Q-1))
      IF (  (TYPE .EQ. 1) .OR. (TYPE .EQ. 3)   )
     + ALPH(1) = ERRCON(1,Q)*ABS( EPS*YMAX/NORMAQ)**(1.0/FLOAT(Q))
C
C     ---------------------------
C     DETERMINE THE MAX STEP SIZE
C     ---------------------------
C
 160  K = 2
      IF ( ALPH(2) .LT. ALPH(1) ) K = 1
C
C     -------------------------
C     IF K=1 THEN DECREASE ORDER
C     -------------------------
C
      IF ( K .EQ. 1 ) Q = Q - 1
      IF ( IDEBUG .GE. 3 ) WRITE(6,1006) Q
 1006 FORMAT(' ALGORITHM WILL USE ORDER ',I1)
      CALL CHSTEP( A, ALPH(K), H, HMIN, N, Q, REEVAL )
      RETURN
      END

C**********************************************************************
C
      SUBROUTINE JACOB( F, FPLUSR, PDY, PDYBAR, Y, YBAR, YPLUS,
     +                  EPS, T, N )
C
C**********************************************************************
C     THIS ROUTINE EVALUATES THE PARTIAL DERIVATIVES OF F(T,Y,YBAR)
C     WITH RESPECT TO Y, YBAR BY USING NUMERICAL DIFFERENCING.  THE
C     VECTORS F, FPLUSR, Y, YBAR, YPLUSR, ARE USED IN GENERATING
C     THE PARTIAL DERIVATIVE MATRICES PDY, PDYBAR WHICH HOLD
C     THE PARTIAL OF F WITH RESPECT TO Y AND THE PARTIAL OF F
C     WITH RESPECT TO YBAR. T IS THE INDEPENDENT VARIABLE AND
C     N IS THE DIMENSION OF THE SYSTEM.  IN A USER WRITTEN
C     SUBROUTINE THE USER NEED ONLY CALCULATE PDY, PDYBAR.
C**********************************************************************
      INTEGER COL, N, ROW
      REAL F(1), FPLUSR(1), PDY(N,1), PDYBAR(N,1), Y(1), YBAR(1),
     +      YPLUS(1), EPS, RY, RYBAR
C----------------------------------------------------------------------
      DO 50 COL = 1, N
         RY = EPS*AMAX1( EPS, ABS( Y(COL)) )
         RYBAR = EPS*AMAX1( EPS, ABS( YBAR(COL)) )
C
         DO 10 ROW = 1, N
            YPLUS( ROW) = Y( ROW)
 10      CONTINUE
         YPLUS(COL) = Y(COL) + RY
C
         CALL DERIV( T, YPLUS, YBAR, FPLUSR )
         CALL DERIV( T, Y, YBAR, F )
C
C        ----------------------------------------------
C        EVALUATE VECTOR PARTIAL DF(T,Y,YBAR) BY DY(COL)
C        ----------------------------------------------
C
         DO 20 ROW = 1, N
            PDY(ROW,COL) = ( FPLUSR(ROW) - F(ROW) )/RY
 20      CONTINUE
C
         DO 30 ROW = 1, N
            YPLUS( ROW) = YBAR( ROW)
 30      CONTINUE
         YPLUS(COL) = YPLUS(COL) + RYBAR
C
         CALL DERIV( T, Y, YPLUS, FPLUSR )
C
C        -------------------------------------------------
C        EVALUATE VECTOR PARTIAL DF(T,Y,YBAR) BY DYBAR(COL)
C        -------------------------------------------------
C
         DO 40 ROW = 1, N
            PDYBAR( ROW,COL) = ( FPLUSR( ROW) - F( ROW) )/RYBAR
 40      CONTINUE
C
 50   CONTINUE
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE FUNCT( DIVDIF, TNODES, PASTT, PASTY, POLY, POLY1,
     +                  BETA, DIFF, GPRIME, T, T0,
     +                  PASTQ, BEGIN, END, INDEX, N, NPTS, QMAX,
     +                  CORR, REEVAL, SMALLH )
C
C***********************************************************************
C     THIS PROGRAM EVALUATES THE FUNCTION AT Y(T) BY USING
C     THE INITIAL FUNCTION IF T BELONGS TO T0, T0+BETA AND USES
C     INTERPOLATION IF NOT.  IT SAVES THE DIVIDED DIFFERENCE TABLE IN
C     CASE ONLY THE LAST NODE CHANGES FORM ONE CALL TO THE NEXT.
C***********************************************************************
      INTEGER PASTQ(1)
      INTEGER BEGIN, COL, END, INDEX, I, ISTART, N, NPTS, QMAX, ROW
      REAL DIVDIF(7,1), PASTT(1), PASTY(N,1), POLY(1), POLY1(1),
     +     TNODES(1), BETA, DIFF, T, T0
      LOGICAL CORR, REEVAL, SMALLH
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
C
      IF ( T .GT. T0 + BETA ) GO TO 10
C
         IF ( IDEBUG .GE. 8 ) WRITE(6,1000) T
1000     FORMAT(' F( T+H,Y(T+H),Y(T+H-BETA) )= PHI(',E14.7,')')
         GPRIME = 0.0
         CALL PHI( T, POLY )
         RETURN
C
 10   IF ( CORR ) GO TO 120
C
         IF ( .NOT. SMALLH ) GO TO 20
C
            CALL SEARCH( PASTT, T, BEGIN, END, INDEX, QMAX )
            IF ( IDEBUG .GE. 8 ) WRITE(6,1002) INDEX
1002        FORMAT('0SEARCH ROUTINE FINDS INDEX = ',I5)
C
 20   IF ( PASTT(INDEX) .NE. T ) GO TO 40
         DO 30 ROW = 1, N
            POLY(ROW) = PASTY(ROW,INDEX)
 30      CONTINUE
         GPRIME = 0.0
         IF ( IDEBUG .GE. 8 ) WRITE(6,1003) INDEX
1003     FORMAT(' F( T+H, Y(T+H), Y(T+H-BETA) ) = PASTY( *,',I5,')')
         RETURN
C
C     --------------------------------
C     STORE THE DIVIDED DIFFERENCE TABLE
C     ----------------------------------
C
 40   NPTS = PASTQ(INDEX)
      ISTART = INDEX - NPTS
C
      IF ( CORR ) GO TO 120
C
      IF (ISTART .LT. 0) GO TO 70
C
         DO 60 I = 1, NPTS
            DO 50 COL = 1, N
               ROW = COL
               DIVDIF(I,COL) = PASTY(ROW,ISTART+I)
 50         CONTINUE
            TNODES(I)  = PASTT(ISTART + I)
 60      CONTINUE
         GO TO 120
C
 70   ILIMIT = QMAX + ISTART + 1
      ITEMP = NPTS
      DO 90 I = ILIMIT, QMAX
         DO 80 COL = 1, N
            ROW = COL
            DIVDIF(ITEMP,COL) = PASTY(ROW,ITEMP)
 80      CONTINUE
         TNODES(ITEMP)  = PASTT(I)
         ITEMP = ITEMP - 1
 90   CONTINUE
C
      ITEMP = INDEX
      DO 110 I = 1, INDEX
         DO 100 COL = 1, N
            ROW = COL
            DIVDIF(ITEMP,COL) = PASTY(ROW,ITEMP)
 100     CONTINUE
         TNODES(I)  = PASTT(ITEMP)
         ITEMP = ITEMP - 1
 110  CONTINUE
```

```
C
 120    CALL DDIFF( DIVDIF, TNODES, N, NPTS, CORR )
C
C       ----------------------
C       DO FUNCTION EVALUATION
C       ----------------------
C
        CALL EVAL( DIVDIF, TNODES, POLY, POLY1, DIFF, GPRIME, T, N, NPTS,
     +            CORR, REEVAL )
        IF ( IDEBUG .GE. 8 ) WRITE(6,1004) T, POLY
 1004   FORMAT('FUNCTION AT T = ',E14.7,' COMPUTED BY INTERPOLATION',
     +         ' IS ',E14.7)
C
C
        RETURN
        END

C***********************************************************************
C
        SUBROUTINE PREDCT( A, N, Q )
C
C***********************************************************************
C       THIS PROGRAM PERFORMS THE PREDICTOR STEP BY EFFECTIVELY
C       MULTIPLYING THE SCALED DERIVATIVES BY THE PASCAL TRIANGLE
C       MATRIX.  THAT IS A = PASCAL MATRIX * A.
C***********************************************************************
        INTEGER COL, J, J1, J2, N, Q, QPLUS1
        REAL A(7,1)
        COMMON IDEBUG, KDEBUG, LDEBUG
C       ---------------------------------------------------------------
        QPLUS1 = Q + 1
        IF ( IDEBUG .GE. 9 ) WRITE(6,1000) Q
 1000   FORMAT('-PREDICTOR ENTERED WITH ORDER = ',I1)
        IF ( IDEBUG .GE. 9 ) CALL OUT( A, N, QPLUS1 )
        DO 30 J = 2, QPLUS1
             DO 20 J1 = J, QPLUS1
                  J2 = QPLUS1 - J1 + J - 1
C
C                 -----------------
C                 DO EACH COMPONENT
C                 OF THE SYSTEM
C                 ----------------
C
                  DO 10  COL = 1, N
                       A(J2,COL) = A(J2,COL) + A(J2+1,COL)
  10              CONTINUE
C
  20         CONTINUE
  30    CONTINUE
        IF ( IDEBUG .GE. 9 ) WRITE(6,1001)
 1001   FORMAT(' PREDICTED VALUES ARE')
        IF ( IDEBUG .GE. 9 ) CALL OUT( A, N, QPLUS1 )
C
        RETURN
        END
```

```
C*************************************************************************
C
      SUBROUTINE SEARCH( PASTT, T, BEGIN, END, E, QMAX )
C
C*************************************************************************
C     THIS PROGRAM SEARCHES THE CIRCULAR QUEUE OF SAVED FUNCTION
C     VALUES TO FIND THE INDEX SUCH THE PASTT( INDEX-1 ) .LE. T .LE.
C     PASTT( INDEX ) USING A BINARY SEARCH
C*************************************************************************
      INTEGER B, BEGIN, E, END, MID, QMAX
      REAL PASTT( 1 ), T
C-------------------------------------------------------------------------
C
      B = BEGIN
      E = END
      MID = E - B
C
      IF ( MID .GE. 0 ) GO TO 30
C
          IF ( PASTT( 1 ) .GE. T ) GO TO 10
              B = 1
              GO TO 30
C
 10       IF ( PASTT( 1 ) .GT. T ) GO TO 20
              E = 1
              RETURN
C
 20       E = QMAX
C
 30   MID = (E - B)/2
      IF ( MID .EQ. 0 ) RETURN
          MID = B + MID
          IF ( PASTT(MID) .LE. T ) B = MID
          IF ( PASTT(MID) .GT. T ) E = MID
          GO TO 30
      END

C*************************************************************************
C
      SUBROUTINE EVAL( DIVDIF, TNODES, POLY, POLY1, DIFF, GPRIME, T,
     +                  N, NPTS, CORR, REEVAL )
C
C*************************************************************************
C     THIS PROGRAM COMPUTES A FUNCTION VALUE BY INTERPOLATION
C     USING THE DIVIDED DIFFERENCE TABLE.  IF IN THE CORRECTOR
C     LOOP IT UPDATES THE FUNCTION VALUE BY USING THE LAST
C     DIVIDED DIFFERENCE IN THE TABLE.
C*************************************************************************
      INTEGER COL, I, N, NPTS
      REAL DIVDIF(7,1), POLY(1), POLY1( 1 ), TNODES( 1 ),
     +     DIFF, GPRIME, T
      LOGICAL CORR, REEVAL
C-------------------------------------------------------------------------
C
      IF ( CORR ) GO TO 60
          DO 10 COL = 1, N
              POLY1( COL ) = DIVDIF( 1,COL )
 10       CONTINUE
          DIFF = T - TNODES( 1 )
C
          IF ( NPTS .EQ. 2 ) GO TO 40
              NM1 = NPTS - 1
              DO 30 I = 2, NM1
                  DO 20 COL = 1, N
                      POLY1( COL ) = POLY1( COL ) +DIFF*DIVDIF( I,COL )
 20               CONTINUE
              DIFF = (T - TNODES( I ))*DIFF
 30           CONTINUE
C
 40       IF ( .NOT. REEVAL ) GO TO 60
              GPRIME = DIFF
              DO 50 I = 2, NPTS
                  GPRIME = GPRIME/( TNODES( NPTS ) -  TNODES( I-1 ))
 50           CONTINUE
C
 60   DO 70 COL = 1, N
          POLY( COL ) = POLY1( COL ) + DIFF*DIVDIF( NPTS,COL )
 70   CONTINUE
C
      RETURN
      END
```

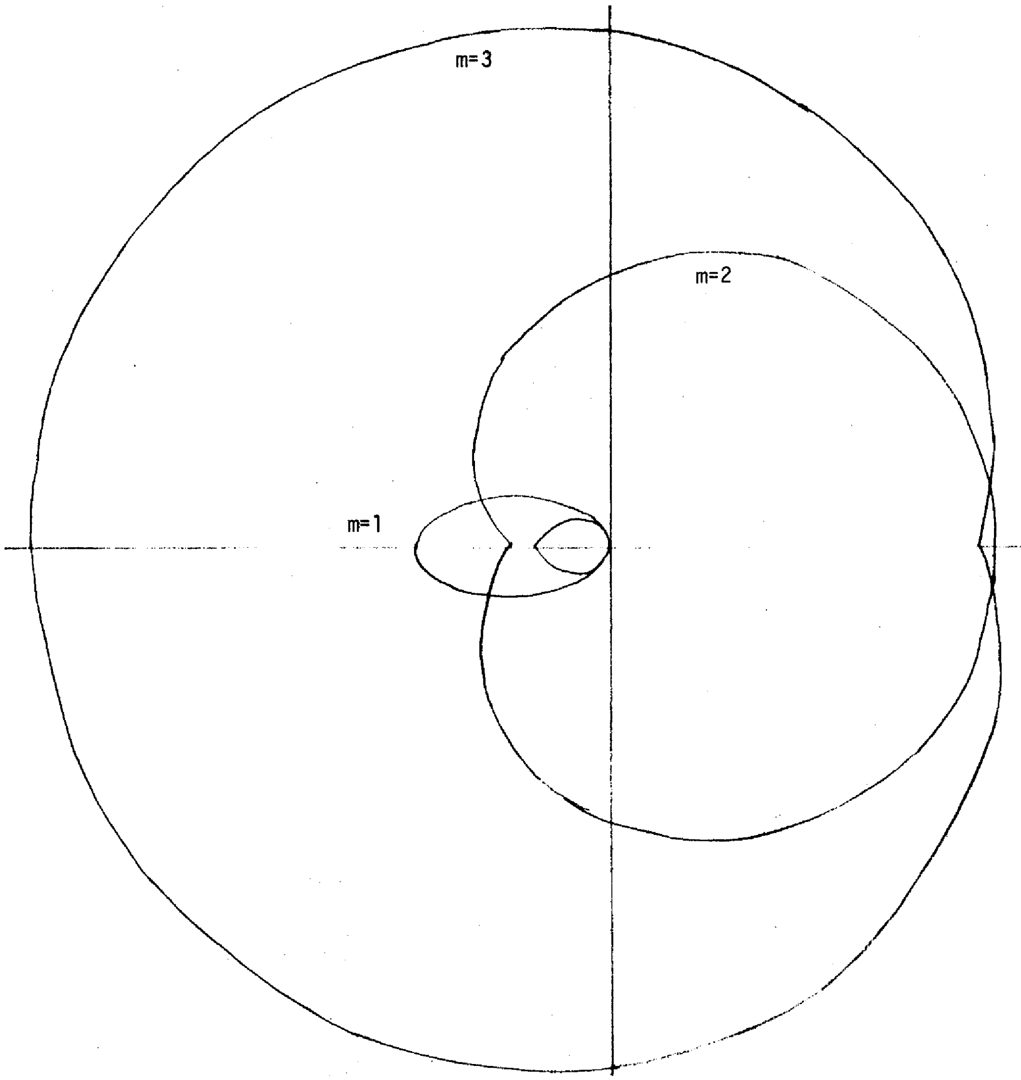
```
C***********************************************************************
C
      SUBROUTINE CHSTEP( A, ALPHA, H, HMIN, N, Q, REEVAL )
C
C***********************************************************************
C     THIS PROGRAM MULTIPLIES THE SCALED DERIVATIVES BY THE DIAGONAL
C     MATRIX C(ALPHA) WHOSE ENTRY IS C(I,I) = ALPHA**(I-1), THIS
C     CHANGES THE STEP SIZE H TO ALPHA*H
C***********************************************************************
      INTEGER COL, N, Q, QPLUS1, ROW
      REAL A(7,1), ALPHA, TEMP
      LOGICAL REEVAL
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
     IF (  ( H .LE. HMIN ) .AND. ( ALPHA .LE. 1.0 )  )  CALL ERROR(2)
     IF ( ALPHA*H .LT. HMIN ) ALPHA = HMIN/H
      IF ( IDEBUG .GE. 3 ) WRITE(6,1000) H
1000  FORMAT(' STEP SIZE BEING CHANGED FROM'/' H = ',E14.7,' TO H = ')
      TEMP = 1.0
      QPLUS1 = Q + 1
      DO  20 ROW = 1, QPLUS1
          DO 10 COL = 1, N
              A( ROW,COL ) = TEMP*A( ROW,COL)
 10       CONTINUE
          TEMP = TEMP*ALPHA
 20   CONTINUE
      H = ALPHA*H
      REEVAL = .TRUE.
      IF ( IDEBUG .GE. 3 ) WRITE(6,1001) H
1001  FORMAT('+',28X,E14.7)
C
      RETURN
      END

C***********************************************************************
C
      SUBROUTINE DELETE( BEGIN, END, INDEX, QMAX )
C
C***********************************************************************
C     THIS PROGRAM DELETES THOSE ENTRIES IN THE QUEUE WHICH
C     WILL NO LONGER BE NEEDED FOR INTERPOLATION. ALL THE NODES
C     PASTT IN T0, T0+BETA AND THE PREVIOUS SIX NODES LESS THAN T0
C     ARE RETAINED
C***********************************************************************
      INTEGER B, BEGIN, E, END, INDEX, QMAX
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
      IF ( IDEBUG .GE. 7 ) WRITE(6,1000) BEGIN, END
1000  FORMAT(' QUEUE POINTERS BEFORE DELETION ARE ',I3,' AND ',I3)
      B = BEGIN
      E = END
      IF ( B - E .LT. 0 ) GO TO 10
C
      IF ( B .GT. INDEX ) GO TO 20
C
 10       IF ( INDEX - 5 .GT. BEGIN ) BEGIN = INDEX - 5
          IF ( IDBBUG .GE. 7 ) WRITE(6,1001) BEGIN, END
1001      FORMAT(' QUEUE POINTERS AFTER DELETION ARE ',I3,' AND ',I3)
          RETURN
C
 20   IF ( INDEX .LT. 6 ) GO TO 30
          B = 1
          IF ( IDEBUG .GE. 7 ) WRITE(6,1001) BEGIN, END
          RETURN
C
 30   BEGIN = BEGIN + 1
      IF ( B .EQ. QMAX ) BEGIN = 1
      IF ( IDEBUG .GE. 7 ) WRITE(6,1001) BEGIN, END
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE PUT( A, B, N, R )
C
C***********************************************************************
C     THIS PROGRAM PUTS THE FIRST R ROWS OF THE MATRIX A
C     INTO THE MATRIX B WHERE A AND B HAVE 7 ROWS AND N COLUMNS.
C***********************************************************************
      INTEGER COL, N, R, ROW
      REAL A(7,1), B(7,1)
C
      DO 20 COL = 1, N
         DO 10 ROW = 1, R
            B( ROW,COL) = A( ROW,COL)
   10    CONTINUE
   20 CONTINUE
C
      RETURN
      END

C***********************************************************************
C
      SUBROUTINE ERROR( NUMBER)
C
C***********************************************************************
C     THIS PROGRAM PRINTS OUT THE ERROR MESSAGES
C***********************************************************************
      INTEGER NUMBER
      IF ( NUMBER .EQ. 1 ) WRITE(6,100)
      IF ( NUMBER .EQ. 2 ) WRITE(6,110)
  100 FORMAT('0QUEUE OF PAST VALUES OVERFLOWED')
  110 FORMAT('0CORRECTOR FAILED TO CONVERGE, STEP CANNOT BE DECREASED')
      STOP
      END
C

C***********************************************************************
C
      SUBROUTINE OUT( A, N, QPLUS1 )
C
C***********************************************************************
C     THIS PROGRAM OUTPUTS THE SCALED DERIVATIVES OF A SYSTEM
C     OF N EQUATIONS, WHEN A METHOD OF ORDER Q HAS BEEN USED.
C***********************************************************************
      INTEGER COL, N, QPLUS1, ROW
      REAL A(7,1)
C
      DO 30 COL = 1, N
         IF ( QPLUS1 .GT. 1 ) GO TO 10
            WRITE(6,1000) COL, A(1,COL)
 1000       FORMAT('       Y(',I2,') =  ',E14.7)
            GO TO 30
C
   10    WRITE(6,1010) COL
 1010    FORMAT('    THE SCALED DERIVATIVES FOR THE ',I2,' COMPONENT',
     +          ' OF THE SYSTEM' )
         DO 20 ROW = 1, QPLUS1
            WRITE(6,1020) ROW, COL, A( ROW,COL)
 1020       FORMAT('       A(',I2,',',I2,') =  ',E14.7)
   20    CONTINUE
   30 CONTINUE
C
      RETURN
      END
```

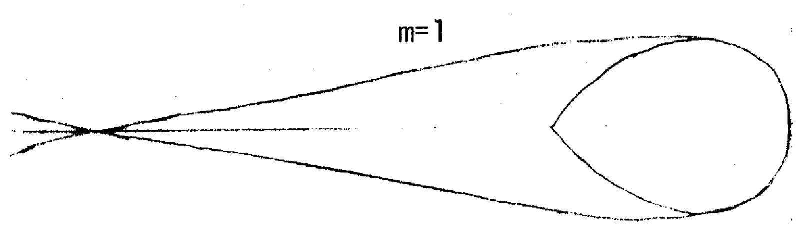
```
C***********************************************************************
C
      SUBROUTINE DDIFF( DIVDIF, TNODES, N, NPTS, CORR )
C
C***********************************************************************
C       THIS PROGRAM COMPUTES THE DIVIDED DIFFERENCE TABLE.  IT ONLY
C       UPDATES THE LAST ROW DURING THE CORRECTOR STEP
C
C       DIVDIF - CONTAINS THE FUNCTION VALUES ON ENTRY AND CONTAINS
C          - THE DIAGONAL OF THE DIVIDED DIFFERENCE TABLE ON EXIT.
C
C       TNODES - THE VALUES OF T IN THE DIVIDED DIFFERENCE TABLE
C
C       NPTS - NUMBER OF ENTRIES IN THE TABLE
C
C       N - DIMENSION OF THE SYSTEM
C
C       CORR - INDICATES TO ONLY UPDATE THE LAST ROW
C***********************************************************************
      INTEGER COL, I, J, N, NPTS
      REAL DIVDIF(7,1), TNODES(1), DENOM
      LOGICAL CORR
C-----------------------------------------------------------------------
C
      IF ( CORR ) GO TO 40
C
          DO 30 I = 2, NPTS
              DO 20 J = I, NPTS
              DENOM = TNODES(J) - TNODES(I-1)
              DO 10 COL = 1, N
                  DIVDIF(J,COL) = ( DIVDIF(J,COL) - DIVDIF(I-1,COL) )
                  DIVDIF(J,COL) = DIVDIF(J,COL)/DENOM
 10           CONTINUE
 20           CONTINUE
 30       CONTINUE
          RETURN
C
C       --------------------
C       UPDATE ONLY LAST ROW
C       --------------------
C
 40   DO 60 I = 2, NPTS
          DENOM = TNODES(NPTS) - TNODES(I-1)
          DO 50 COL = 1, N
              DIVDIF(NPTS,COL) = ( DIVDIF(NPTS,COL) - DIVDIF(I-1,COL) )
              DIVDIF(NPTS,COL) = DIVDIF(NPTS,COL)/DENOM
 50       CONTINUE
60    CONTINUE
C
      RETURN
      END

C***********************************************************************
C
      SUBROUTINE  ADD( A, PASTT, PASTY, T, PASTQ, BEGIN, END, N,
     +                   Q, QMAX )
C
C***********************************************************************
C     THIS PROGRAM ADDS AN ENTRY TO THE QUEUE
C***********************************************************************
      INTEGER PASTQ(1), BEGIN, COL, END, N, Q, QMAX, ROW
      REAL A(7,1), PASTT(1), PASTY(N,1), T
      COMMON IDEBUG, KDEBUG, LDEBUG
C-----------------------------------------------------------------------
      END = END + 1
C
      IF ( END .GT. QMAX ) END = 1
C
      IF ( BEGIN - END .EQ. 0 ) CALL ERROR( 1 )
C
      PASTT(END) = T
      DO 10 ROW = 1, N
          COL = ROW
          PASTY(ROW,END) = A(1,COL)
 10   CONTINUE
      PASTQ(END) = Q + 1
          IF ( IDEBUG .GE. 7 ) WRITE(6,1000) END, T
1000      FORMAT('0ENTRY ADDED TO THE QUEUE PASTT(',I5,') = ',E14.7)
          IF ( IDEBUG .GE. 7 ) CALL OUT( A, N, 1 )
C
      RETURN
      END
```

APPENDIX B

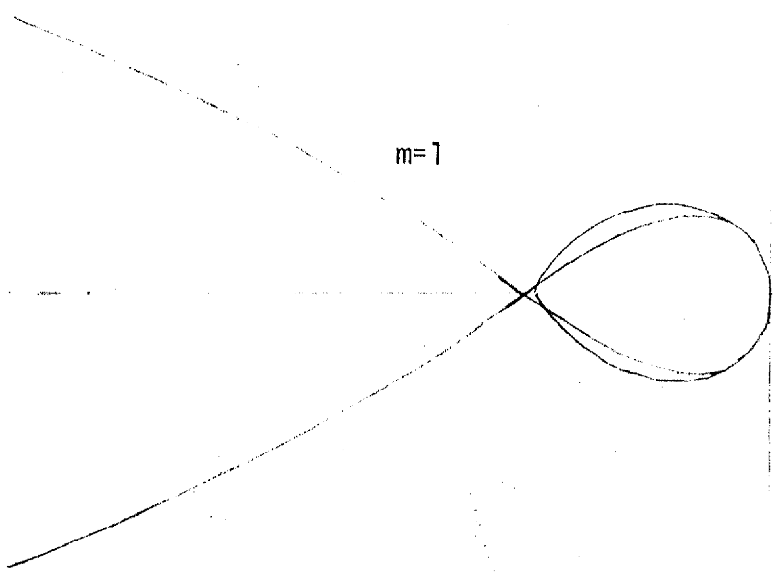
Plots of the Boundaries of Stability Regions for the Backward Differentiation Methods



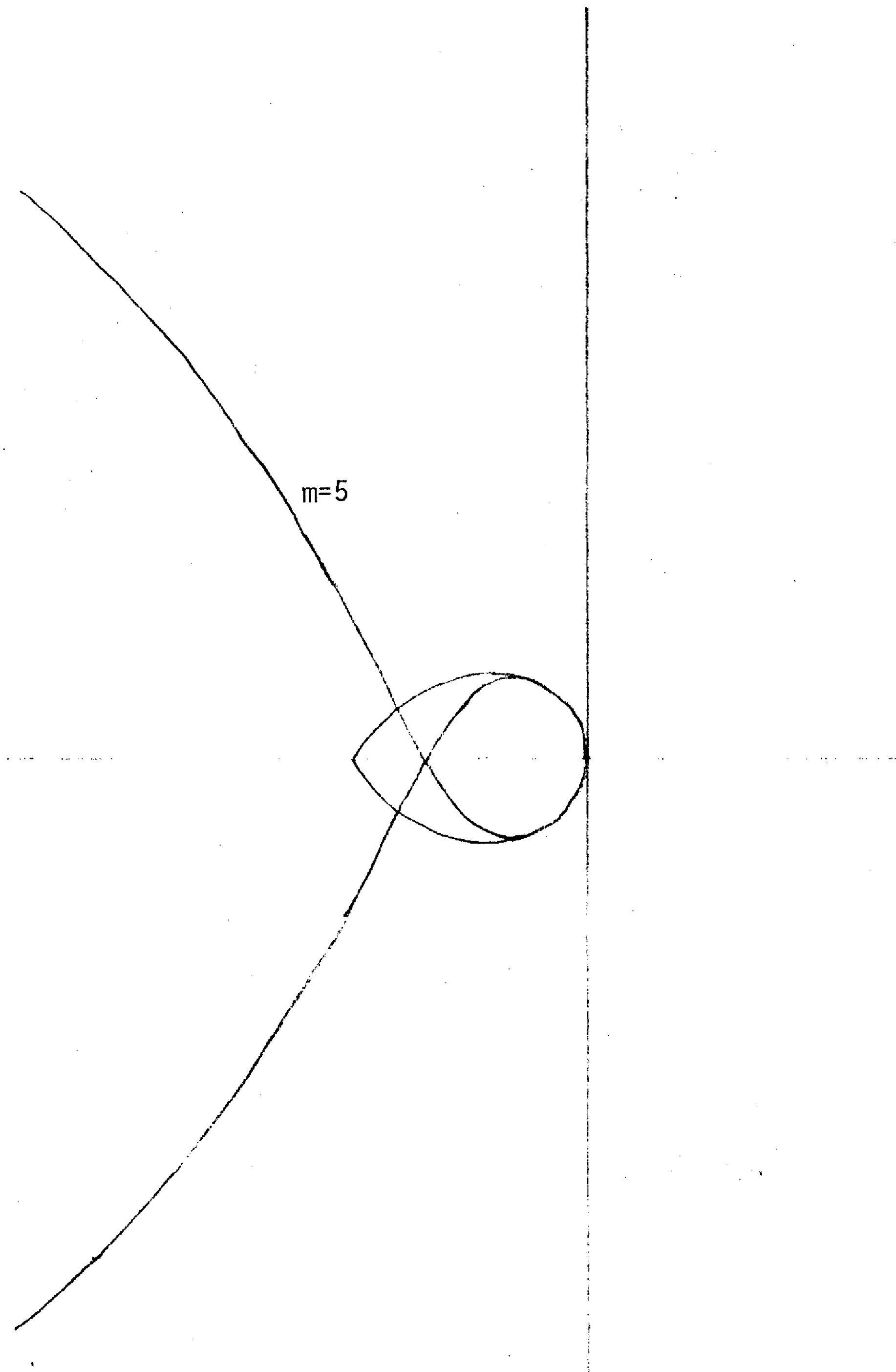
Region of Q-Stability for the B.D. method of Order 2
with β = mh



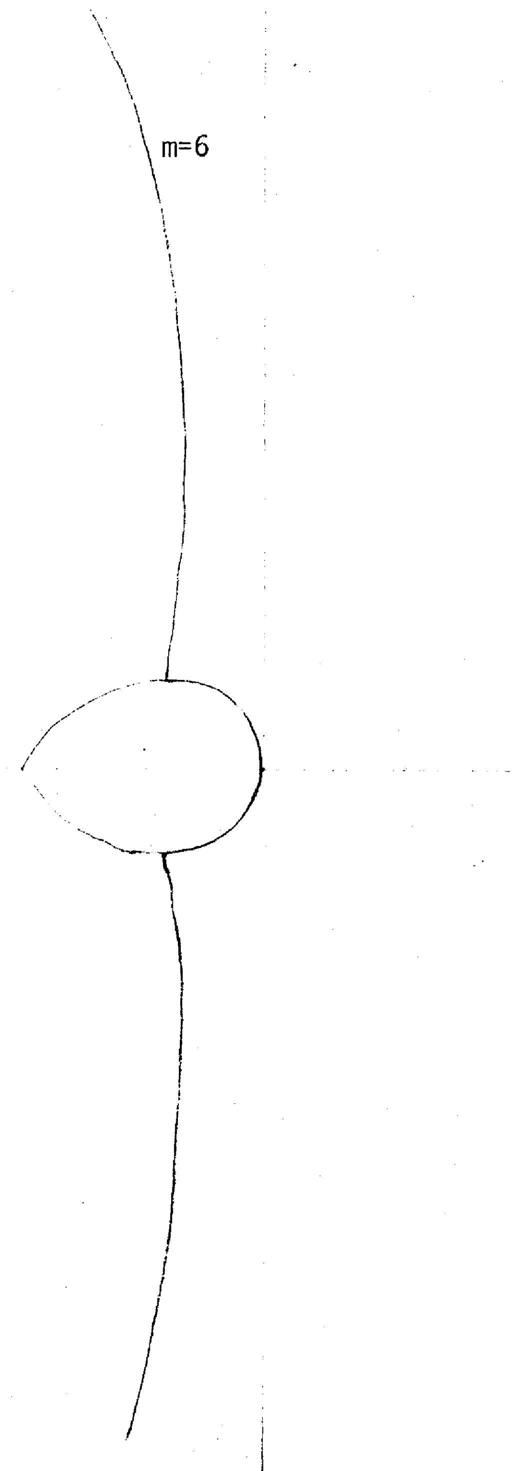
Region of Q-Stability for the B.D. method of Order 3 with $\beta = mh$



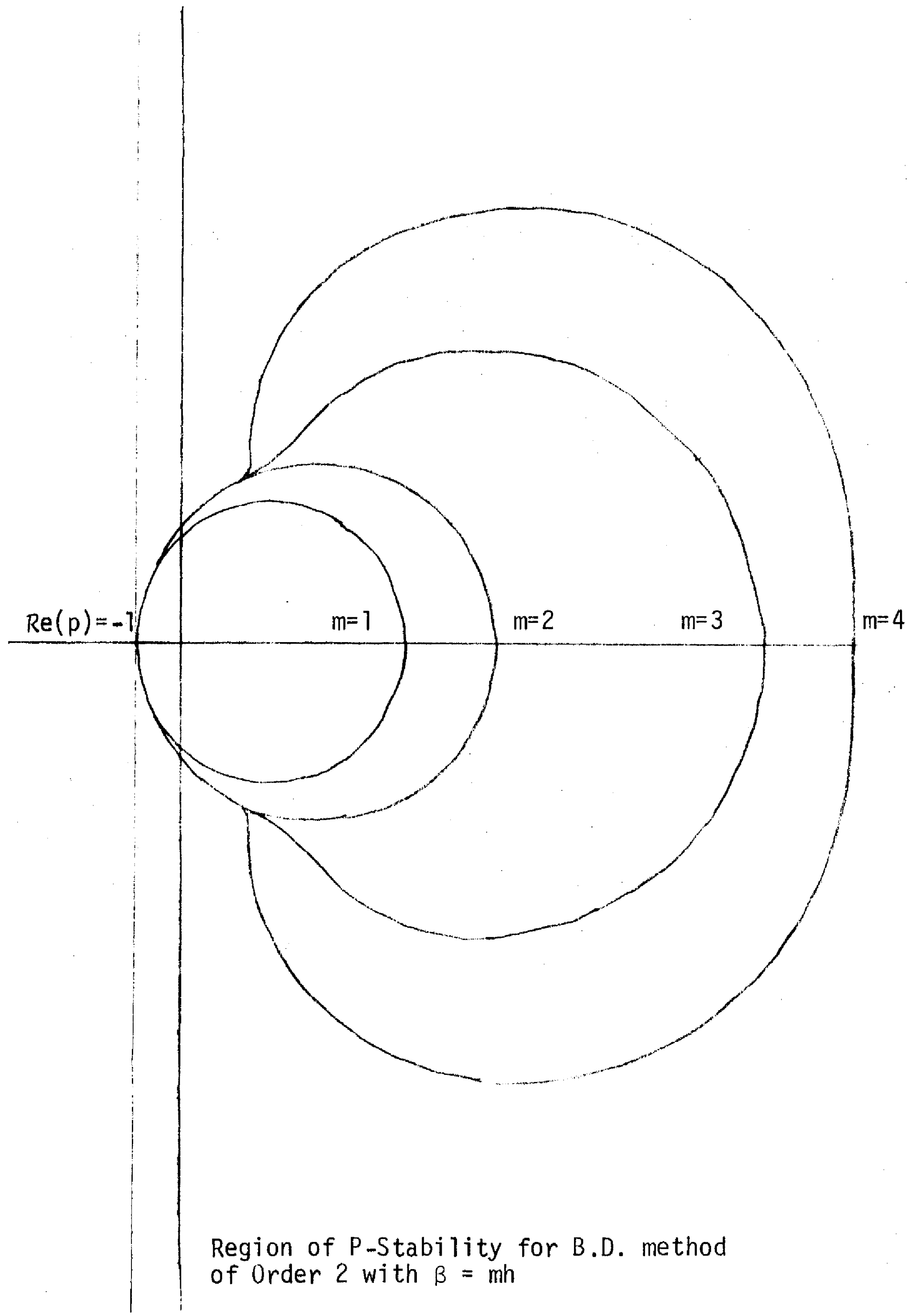
Region of Q-Stability for B.D. method of Order 4 with β = mh



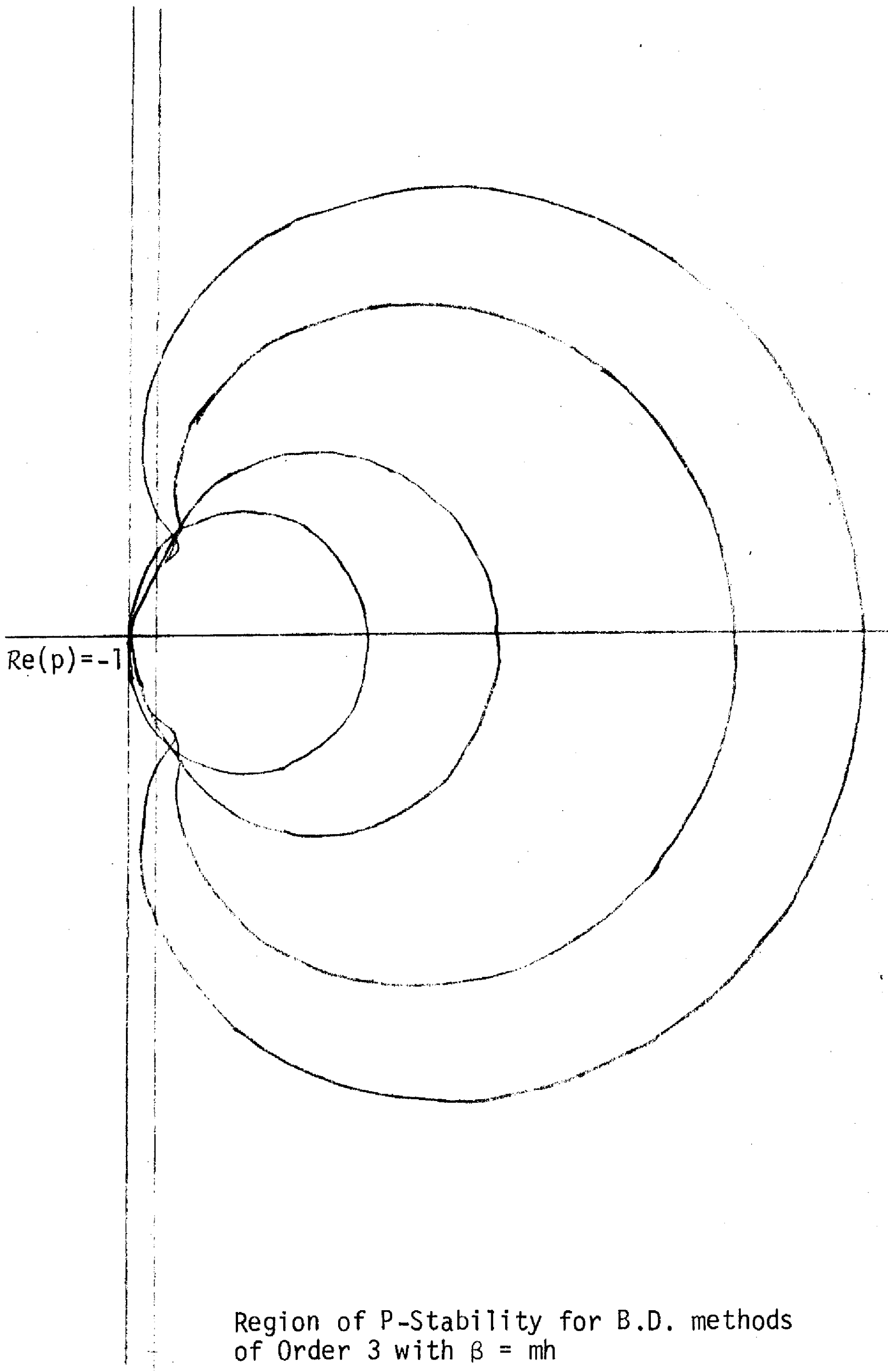
Region of Q-Stability for B.D. method of Order 5 with β = mh



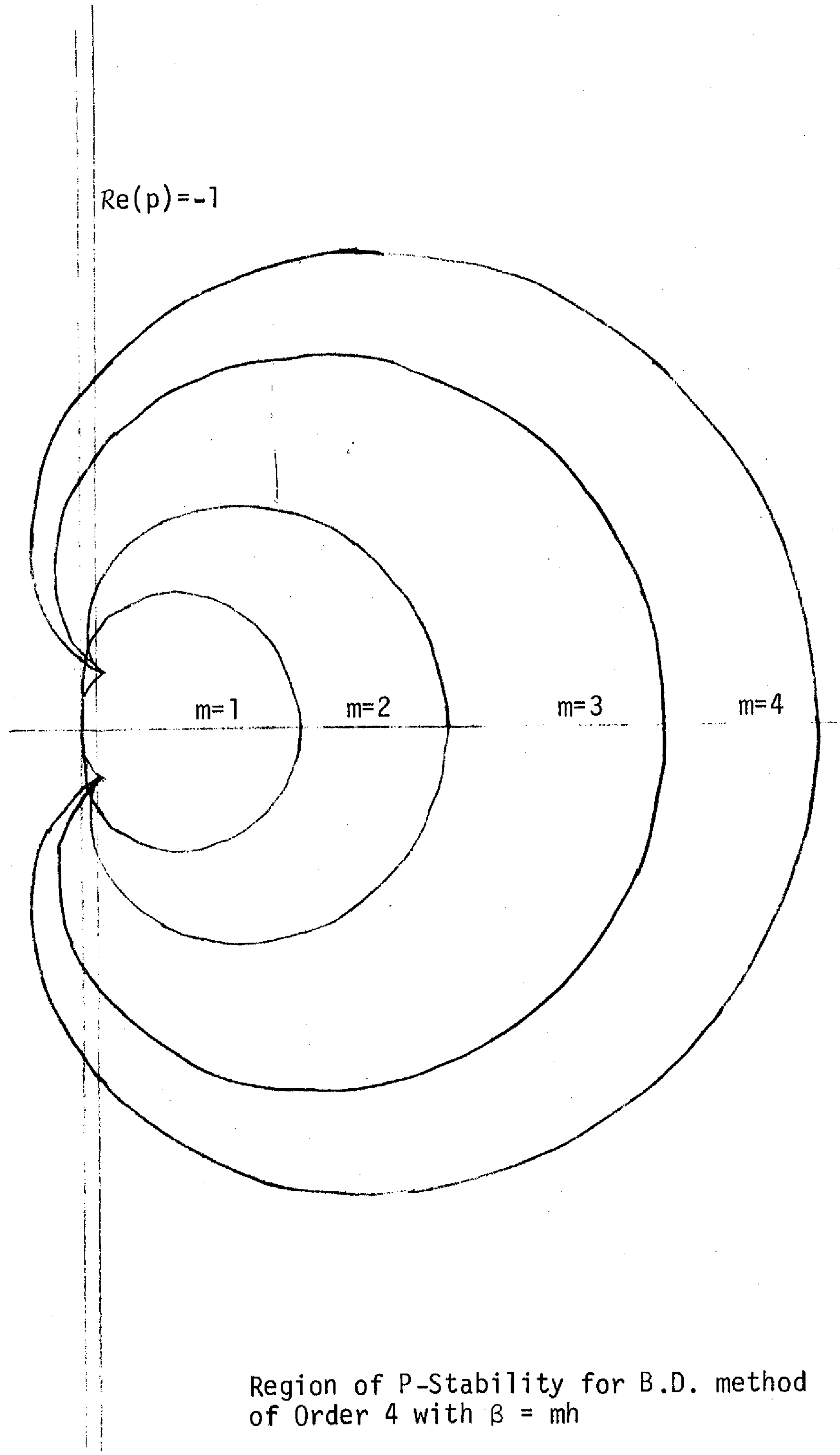
Region of Q-Stability of B.D. methods of Order 6 with β = mh



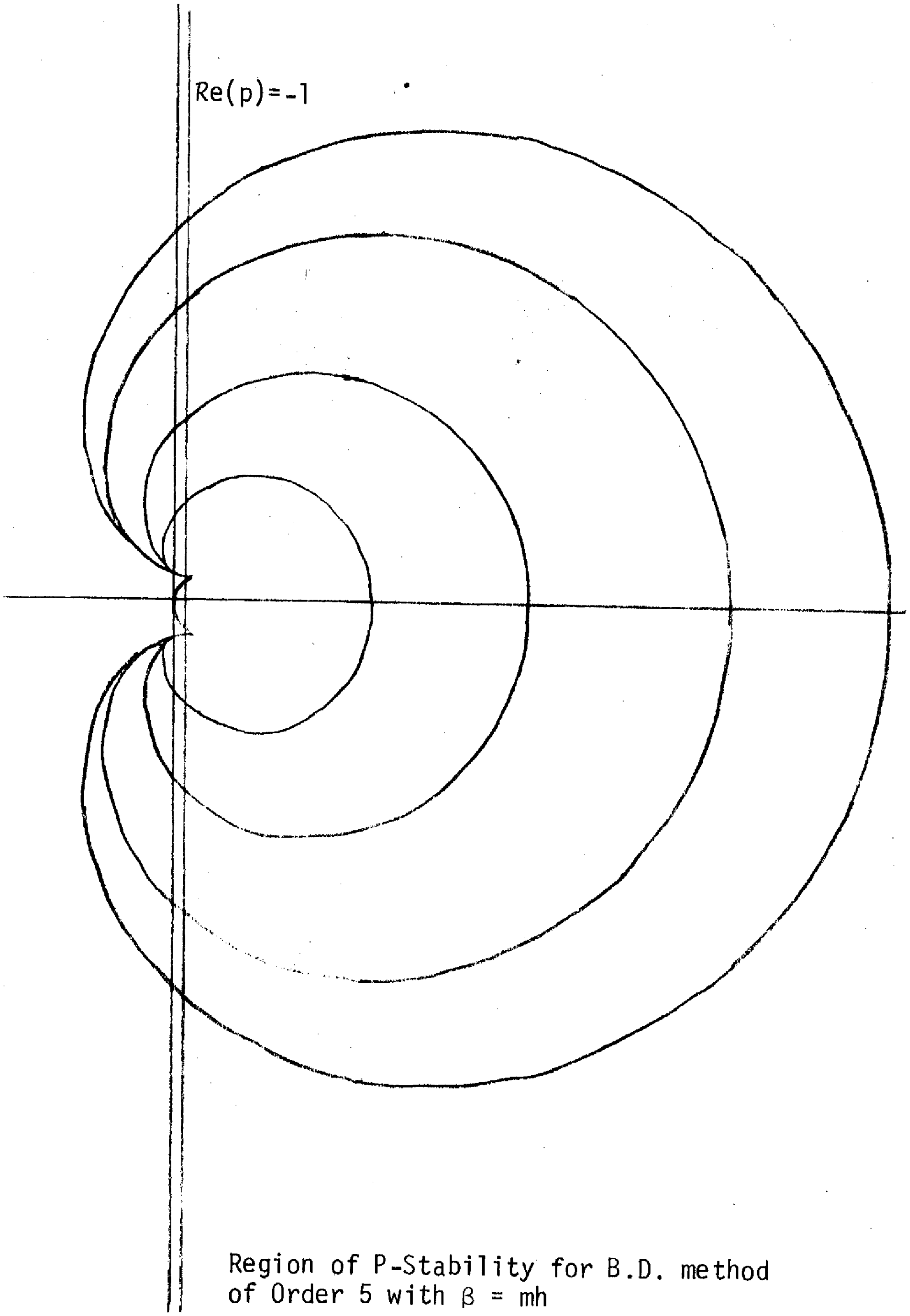
Region of P-Stability for B.D. method
of Order 2 with β = mh



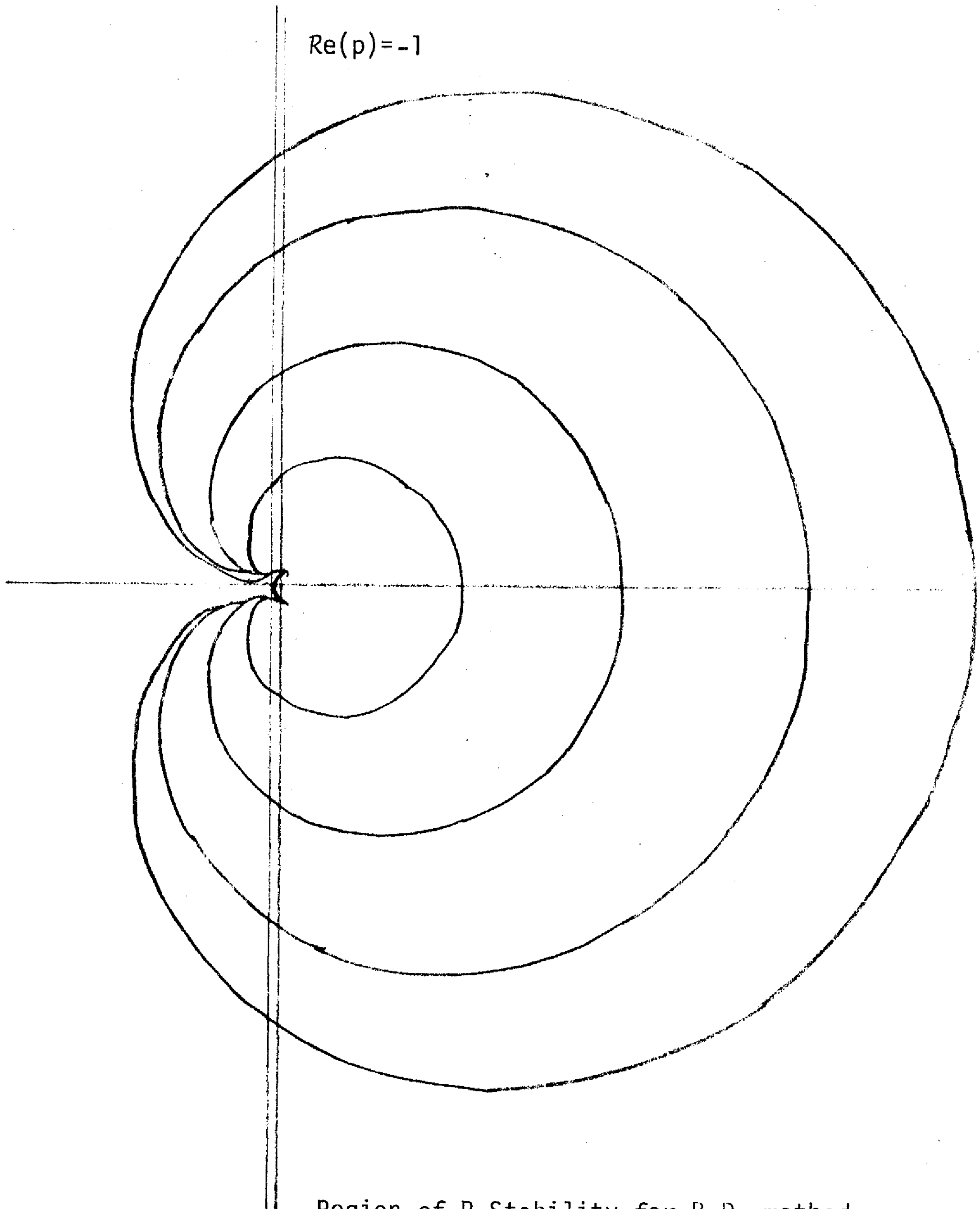
Region of P-Stability for B.D. methods
of Order 3 with β = mh



Region of P-Stability for B.D. method
of Order 4 with β = mh



Region of P-Stability for B.D. method
of Order 5 with β = mh



Region of P-Stability for B.D. method
of Order 6 with β = mh

BIBLIOGRAPHY

1. Bellman, R. and Cooke, K., Differential Difference Equations, Academic Press, New York, 1963.

2. Brayton, R.K., Numerical A-stability of Difference-Differential Systems, IBM Research Report, RC 4647, 1973.

3. Brayton, R.K. and Willoughby, R.A., On the numerical integration of a symmetric system of difference differential equations of neutral type, Journal of Mathematical Analysis Applied 18, (1967), pp.182-189.

4. Conte and de Boor, Elementary Numerical Analysis An Algorithmic Approach, McGraw Hill Book Company, 1972.

5. Cryer, C.W., A New Class of Highly Stable Methods: A-stable Methods, BIT, vol.13,(1973), pp.153-159.

6. __________, Highly Stable Multistep Methods for Retarded Differential Equations, SIAM Journal of Numerical Analysis, vol.11, No.4, 1974.

7. Dahlquist, G., A Special Stability Problem for Linear Multistep Methods, BIT, vol.3, (1963), pp.27-43.

8. Driver, R.D., Some Harmless Delays, Delay and Functional Differential Equations and Their Applications, Klaus Schmitt (editor), Academic Press, New York, 1972, pp.103-120.

9. El'sgol'ts L.E. and Norkin, S.B., Introduction to the Theory and Application of Differential Equation with Deviating Argument,

10. Forsythe, G.E., Malcolm, M.A. and Moler, C.B., Computer Methods for Solving Mathematical Problems, to be published by Prentice Hall.

11. Gear, W.C., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice Hall, New Jersey, 1971.

12. Henrici, P., Discrete Variable Methods in Ordinary Differential Equations, John Wiley & Sons, New York, 1962.

13. Hutchison, J., Finite Difference Solutions to Delay Differential Equations, Ph.D. Thesis, Department of Mathematics, Rensselaer Polytechnic Institute, 1971.

14. Lambert, J.D., Computational Methods in Ordinary Differential Equations, John Wiley & Sons, London, 1973.

15. Marden, Morris, Geometry of Polynomials, American Math Society, Providence, Rhode Island, 1966.

16. Neves, W.K., Automatic Integration of Functional Differential Equations: An Approach and an Algorithm, to be published, Babcock and Wilcox Co., Lynchburg, Va., 24505.

17. __________, Numerical Solution of Functional Differential Equations with State Dependent Lags, Ph.D. Thesis, Arizona State University, 1974.

18. __________, Feldstein, A., Characterization of Jump Discontinuities for State Dependent Delay Differential Equations, presented SIAM National Meeting, Hampton, Va., June 1973.

19. Taverini, L., Linear Multistep Methods for the Numerical Solution of Volterra Functional Differential Equations, to appear Journal Applicable Analysis.

20. Wiederholt, L., Numerical Integration of Delay Differential Equations, Ph.D. Thesis, University of Wisconsin, 1970.

21. Wilkinson, J.H., The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.

22. Zverkina, T.S., Modified Adams Formulas for Integrating Equations with a Deviating Argument, Trudy Sem. Teor. Differencial Uravensii S. Otklon. Argumentom, Univ. Druzdy Naradov Patrisa Lumumby, 1, MR 33 #5131, 1965, pp.221-232.