A STIFFLY STABLE INTEGRATION PROCESS USING CYCLIC COMPOSITE METHODS

by

JOEL MARVIN TENDLER

B.E., The Cooper Union, 1964

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of Syracuse University, May, 1973

Approved Theodore A Bickart

Date 26 February 1973

A STIFFLY STABLE INTEGRATION PROCESS USING CYCLIC COMPOSITE METHODS

by

JOEL MARVIN TENDLER

B.E., The Cooper Union, 1964

ABSTRACT OF DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of Syracuse University, May, 1973

Approved Theodore A Bickart

Date 26 February 1973

ABSTRACT

Let $\dot{\underset{\sim}{y}} = \underset{\sim}{f}(\underset{\sim}{y}, t)$ with $\underset{\sim}{y}(t_0) = \underset{\sim}{y}_0$ possess a solution $\underset{\sim}{y}(t)$ for $t \geq t_0$. Set $t_n = t_0 + nh$ and let $\underset{\sim}{y}_n$ denote the approximate solution of $\underset{\sim}{y}(t_n)$ defined by the composite linear multistep method

$$\sum_{j=-k+1}^{\ell} (\alpha_{i,j}\, \underset{\sim}{y}_{n\ell+k-1+j} - h\beta_{i,j}\, \dot{\underset{\sim}{y}}_{n\ell+k-1+j}) = \underset{\sim}{0}, \qquad i=1, \ldots, \ell,$$

with $\dot{\underset{\sim}{y}}_n = \underset{\sim}{f}(\underset{\sim}{y}_n, t_n)$ and $n=1, 2, \ldots$. Asymptotic stability properties of the method as reflected in the behavior of $\underset{\sim}{y}_n$ as $n \to \infty$ are examined; in particular, $A(\alpha)$-stability and stiff stability are treated. It is shown that, when the method is used to numerically solve $\dot{y} = qy$, q constant, with a fixed integration step-size h, the characteristic polynomial, a polynomial in two variables, ζ and $\lambda = qh$, may be examined to determine the asymptotic behavior of the given method. The Lambda and Zeta Loci determined by the characteristic equation are introduced and their properties examined. Specifically, the Lambda Locus is the λ-plane image of the ζ-plane unit circle under the mappings $\lambda_i = f_i(\zeta)$, $i=1, \ldots, \ell$, and determines the stability regions for the method. Similarly, the image of the λ-plane imaginary axis in the ζ-plane under the mappings $\zeta_j = g_j(\lambda)$, $j=1, \ldots, k$, determines the Zeta Locus, a useful tool in

checking if a method is A-stable. Note: $\lambda_i = f_i(\zeta)$ for each i is a zero of the characteristic polynomial viewed as a polynomial in λ; similarly, $\zeta_j = g_j(\lambda)$ for each j is a zero of the characteristic polynomial viewed as a polynomial in ζ. A modified Zeta Locus is also discussed and it is shown that it can be used to check if a method is stiffly stable. A computer program to plot both the Lambda and Zeta Loci for a given method is described. Using an interactive version of this program new and efficient cyclic composite linear multistep methods, that is, methods given as

$$\sum_{j=-k+i}^{i} \alpha_{i,j}\, \underset{\sim}{y}_{n\ell+k-1+j} - h \sum_{j=1}^{i} \beta_{i,j}\, \dot{\underset{\sim}{y}}_{n\ell+k-1+j} = \underset{\sim}{0},$$

$$i=1, \ldots, \ell,$$

of orders 3 through 7 are derived. These new methods are stiffly stable and exhibit better stability properties than the backward differentiation formulas of corresponding order. A new variable step, variable order integration algorithm incorporating these new methods is presented. This new algorithm is ideally suited for the numerical integration of stiff systems of first order differential equations wherein the Jacobian matrix $\underset{\sim}{f}_{\underset{\sim}{y}}$ has complex eigenvalues. Numerical results to support this claim are presented, including a comparison of the new algorithm with Gear's widely accepted computer program DIFSUB and with a third algorithm based on

the backward differentiation formulas. For one of the test problems solved an improvement of better than an order of magnitude in computer program execution time was noted using the new algorithm over results obtained using Gear's algorithm DIFSUB.

A STIFFLY STABLE INTEGRATION PROCESS USING CYCLIC COMPOSITE METHODS

by

JOEL MARVIN TENDLER

B.E., The Cooper Union, 1964

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of Syracuse University, May, 1973

Approved Theodore A Bickart

Date 26 February 1973

To the two Philips

in my life—

one of whom has affected me

as I pray I can the other

PREFACE

With the advent of larger and faster computers new problems have been opened to solution. More complex systems, described by systems of ordinary differential equations, are now being solved than ever before. This places a bigger burden on numerical integration subroutines as they must be capable of efficiently and accurately supplying answers. In the case of stiff systems, that is, systems with both fast and slow transients, the numerical integration process can be both time consuming and expensive. This work was undertaken to help alleviate this problem. A new integration algorithm is presented that does not suffer from many of the shortcomings of presently available algorithms. In the very least, the techniques developed to arrive at this new algorithm should provide others with the insight and tools for pursuing this field.

I would like to express my sincerest appreciation to my advisor, Professor Theodore A. Bickart, for his guidance and supervision during the course of this work. His insight and knowledge of this research area has been invaluable. I am also grateful to Mr. William B. Rubin, a fellow student, for the many enlightening conversations we had.

Support provided by the Syracuse University Research Corporation in the form of unlimited free computer time and

secretarial services has proven invaluable. I would like to especially thank Mr. James D. Rodems for providing me with the opportunity of pursuing this work while still remaining as an employee, and Mrs. Christine Peters and Miss Janet Sinclair for typing and overseeing this manuscript. The financial support provided by the National Science Foundation, to myself in the form of a traineeship and to my advisor under Grant GK-23010, and by Syracuse University is also gratefully acknowledged.

Finally, and perhaps foremost, I am indebted to my wife Debbie for her understanding and encouragement during my studies.

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER I

INTRODUCTION

We are concerned with the numerical integration of a stiff system of first order ordinary differential equations*

$$\dot{\underset{\sim}{y}}(t) = \underset{\sim}{f}(\underset{\sim}{y}(t), t), \tag{1.1a}$$

given the initial conditions

$$\underset{\sim}{y}(t_o) = \underset{\sim}{y}_o. \tag{1.1b}$$

By stiff we mean a system with widely separated eigenvalues of the Jacobian matrix $J = \underset{\sim}{f}_{\underset{\sim}{y}}$ of (1.1) evaluated along a particular solution $\underset{\sim}{y}(t) = \underset{\sim}{g}(t)$. Note: stiff systems are encountered in many disciplines, e.g., reactor calculations, circuit analysis, chemical kinetics, etc.

---

*It is assumed that (1.1) possesses a unique solution $\underset{\sim}{y}(t)$ for $t \geq t_o$. Such will be the case if $\underset{\sim}{f}(\underset{\sim}{y}(t), t)$ satisfies a Lipshitz condition, i.e., there exists a constant $L \geq 0$ such that

$$||\underset{\sim}{f}(\underset{\sim}{y}, t) - \underset{\sim}{f}(\underset{\sim}{z}, t)|| \leq L\, ||\underset{\sim}{y} - \underset{\sim}{z}||$$

for all $t \geq 0$ and $\underset{\sim}{y}, \underset{\sim}{z} \in R^n$.

Conventional numerical integration processes when applied to stiff systems limit the integration step-size, h, to a value determined by the largest eigenvalue of the Jacobian matrix, even after the transient effects due to the larger eigenvalues have decayed. This unnecessarily causes the numerical integration process to become both time consuming and expensive. Thus there is a need for numerical integration processes which perform well when used to solve stiff systems.

A detailed discussion of the history and state-of-the-art as of 1970 of the methods used for the numerical solution of stiff systems can be found in Bjurel et al [BJU70]. A more recent bibliography covering the field can be found in [ENR72].

Almost all of the previous work has been limited to classical multistep and Runge-Kutta type processes. Of all the schemes proposed the most widely accepted process is a variable order, variable step algorithm due to Gear [GEA71a]. Gear's algorithm is based on the backward differentiation multistep formulas described in [HEN62]. Recent modifications to Gear's algorithm by Brayton, Gustavson, and Hachtel [BRA72a] and by Krogh [KRO72] have attempted to improve on some of the shortcomings in Gear's method, but their processes are based on the same formulas used by Gear. The performance of all three algorithms deteriorate when they

are used to integrate a stiff system wherein the eigenvalues of the Jacobian matrix are complex.

Recently Sloate and Bickart [SLO71b] introduced the concept of a composite linear multistep method wherein a block of ℓ future points are simultaneously computed using ℓ different equations. Sloate [SLO71a] has reported an algorithm suitable for the solution of stiff systems that is unaffected by the presence of complex eigenvalues in the Jacobian matrix. However, the algorithm consisted of solving for two future points and retaining only the first of these points at each stage of the process.

Since then Bickart, Burgess, and Sloate [BIC71], Bickart and Picel [BIC72], and Watts [WAT71] have reported on composite linear multistep methods wherein all ℓ future points are retained with the additional desirable property of requiring only one backpoint making the processes self-starting and simplifying step-size changes. These processes are applicable to the numerical solution of stiff systems and, to varying degrees, are not as sensitive to complex eigenvalues in the Jacobian matrix as are the processes based on the backward differentiation formulas. However, they suffer the disadvantage of requiring the simultaneous solution of ℓ coupled equations.

One of the goals of the research effort reported herein was to obtain a new integration process applicable to

stiff systems that is efficient to use and that is not "too" sensitive to complex eigenvalues in the Jacobian matrix. As composite linear multistep methods satisfy two of the three requirements they were considered. In order to make the process efficient attention was further directed toward the class of cyclic composite linear multistep methods wherein each of the ℓ component formulas are solved cyclically for one of the ℓ future points. The concept of a cyclic composite linear multistep method is not new. It was first formally considered by Donelson and Hansen [DON71] and later by Chesler and Pierce [CHE71], and by Dyer, Pierce, Haney, and Chesler [DYE72]. However, their interest was in orbit computation and the processes they considered are not suitable to stiff systems.

A search of the available literature soon revealed that tools for characterizing the general class of composite linear multistep methods that would be required in obtaining the new integration processes were noticably missing. Hence a second goal of the research effort reported herein was to obtain new techniques to be used as tools in characterizing numerical integration processes, specifically composite linear multistep methods.

Chapter II begins with a discussion of classical multistep numerical integration processes and procedures used to characterize them and concludes with a description

of composite linear multistep methods. Some historical perspectives are also presented. Furthermore, since it is a key ingredient of subsequent results reported, the concept of the characteristic equation of a multistep process is introduced. Chapter III describes a program used to obtain stability boundaries for multistep processes from the characteristic equation. The program described in Chapter III was used to obtain a new composite linear multistep method; that result is reported in Chapter IV. A variable order, variable step-size algorithm using this new method is described in Chapter V. Results obtained with this new algorithm from some sample problems are also reported. Chapter VI contains conclusions about the new method and an outline for future research.

CHAPTER II

PRELIMINARIES

2.1 Problem Statement

Consider the system of first order differential equations

$$\dot{\underset{\sim}{y}}(t) = \underset{\sim}{f}(\underset{\sim}{y}(t), t), \quad \underset{\sim}{y}(t_o) = \underset{\sim}{y}_o. \tag{2.1}$$

It is desired to obtain a sequence $\{\underset{\sim}{y}_n\}$ such that $\underset{\sim}{y}_n$ approximates $\underset{\sim}{y}(t_n) = \underset{\sim}{y}(t_o + nh)$.

Historically predictor-corrector methods and explicit Runge-Kutta type processes have been used to solve (2.1) numerically. However, when (2.1) is stiff, classical methods often encounter serious problems. For stiff systems one wants to increase the step-size, once the transient effects due to smaller time constants have decayed, in order to decrease computer program execution time. However, this is in conflict with the small step-size required to keep the numerical processes stable. Hence one is forced to use a small step-size resulting in unnecessarily long and expensive program execution times.

The phenomenon of a numerical integration process becoming unstable even when used to integrate a stable differential equation can be exhibited by considering the forward Euler formula

$$y_{n+1} = y_n + h\dot{y}_n, \tag{2.2}$$

when used to integrate the differential equation

$$\dot{y} = qy, \quad y(0) = y_o, \tag{2.3}$$

q real and negative. The differential equation (2.3) is stable; however, when (2.2) is used to numerically integrate it, the solution becomes

$$y_n = (1 + qh)^n y_o, \tag{2.4}$$

which for $h > -2/q$ is unstable. This follows from the fact that, for $h > -2/q$, $|y_{n+1}| > |y_n|$, hence, $y_n \to \infty$ as $n \to \infty$. Thus, the forward Euler method is unstable under these circumstances, even though the differential equation itself is stable.

Our aim is to present a new integration method which avoids the pitfall exemplified by the forward Euler method.

2.2 Linear Multistep Methods (LMM)*

Linear multistep methods may be written in the form

$$\sum_{j=-k+1}^{1} (\alpha_j y_{n+j} - h\beta_j \dot{y}_{n+j}) = 0. \qquad (2.5)$$

The LMM as given by (2.5) may be used to numerically integrate (2.1) by setting $\dot{y}_i = f(y_i, t_i)$. As the differential equation requires an initial condition so does (2.5). In fact (2.5) requires $y_o, \ldots, y_{k-1}$ as starting values. As information at k different points in time is required it is called a k-step method.

Next consider

$$L[y(t_n), h] \equiv \sum_{j=-k+1}^{1} \{\alpha_j y(t_{n+j}) - h\beta_j \dot{y}(t_{n+j})\}. \qquad (2.6)$$

Under the assumption that y(t) in (2.6) possesses a convergent Taylor series at $t = t_n$, (2.6) can be written as

$$L[y(t_n), h] = \sum_{i=o}^{\infty} C_i h^i y^{(i)}(t_n), \qquad (2.7)$$

---

*For notational simplicity only the scalar form of (2.1) will be considered. The results extend, in a straightforward manner, to the vector case.

where

$$C_i = \sum_{j=-k+1}^{1} \left(\frac{1}{i!} j^i \alpha_j - \frac{1}{(i-1)!} j^{i-1} \beta_j\right). \qquad (2.8)$$

The basis now exists for

<u>Definition 2.1</u>: The linear multistep method (2.5) is said to be of <u>order</u> p if $C_i = 0$ for $i=0, \ldots, p$ but $C_{p+1} \neq 0$, where C_i is given by (2.8).

The order of a method is related to the local discretization error as follows: If

i) the multistep method is of order p;

ii) $y_{n+j} = y(t_{n+j}) + O(h^{p+1})$, $j=-k+1, \ldots, 0$; and

iii) $y(t)$ is $(p+1)$ times continuously differentiable

then $y_{n+1} = y(t_{n+1}) + O(h^{p+1})$.

In the light of the relationship between order and local discretization error one would like to maximize the order of a multistep formula.

2.3 A-Stability

In 1963 Dahlquist [DAH63] introduced the concept of A-stability; namely,

<u>Definition 2.2</u>: A linear multistep method is said to be <u>A-stable</u> if all solutions of (2.5) tend to zero as $n \to \infty$ when the method is used to integrate the differential equation

$$\dot{y}(t) = q\, y(t) \tag{2.9}$$

with any fixed positive h where $Re\{q\} < 0$.

Substituting (2.9) into (2.5) yields

$$\sum_{j=-k+1}^{1} (\alpha_j - qh\beta_j)\, y_{n+j} = 0. \tag{2.10}$$

The solution of (2.10) is given as

$$y_n = \sum_{i=1}^{k} a_i\, \zeta_i^n, \tag{2.11}$$

where, with $qh = \lambda$, the ζ_i's are the roots* of

$$\sum_{j=-k+1}^{1} (\alpha_j - \lambda\beta_j)\, \zeta^{k-1+j} = 0, \tag{2.12}$$

---

*The form of the solution as given by (2.11) assumes that the zeros of (2.12) are all distinct. In general the solution is given as [HEN64]

$$y_n = \sum_{i=1}^{s} \sum_{j=1}^{m_i} a_{ij} n^{j-1} \zeta_i^n, \tag{2.11'}$$

where the ζ_i's are the s distinct roots of (2.12) with multiplicity m_i, respectively, and the a_{ij}'s are functions of the initial conditions. As the primary concern of the present discussion is with A-stability it suffices to assume that there are k distinct zeros of (2.12).

and the a_i's are functions of the initial conditions. (Equation (2.12) is called the characteristic equation of the method.*)

As a result of (2.11), if an LMM is to be A-stable, then $|\zeta_i| < 1$, $i=1, \ldots, k$, for all values of λ such that $\mathrm{Re}\{\lambda\} < 0$.

The forward Euler method given by (2.2) is not A-stable. Its characteristic equation is

$$\zeta - (1 + \lambda) = 0, \tag{2.13a}$$

or

$$\zeta = 1 + qh. \tag{2.13b}$$

It is readily verified that for $\mathrm{Re}\{q\} < 0$, $|\zeta| > 1$ for

$$|1 + \lambda| > 1, \tag{2.14a}$$

or

$$h > -2\frac{\mathrm{Re}\{q\}}{|q|^2}. \tag{2.14b}$$

---

*The characteristic equation is discussed in detail in Section 2.7.

On the other hand, the backward Euler method, given as

$$y_{n+1} = y_n + h\dot{y}_{n+1}, \tag{2.15}$$

is A-stable. Its characteristic equation is

$$\zeta(1 - \lambda) - 1 = 0, \tag{2.16a}$$

or

$$\zeta = \frac{1}{1-qh}. \tag{2.16b}$$

For $\mathrm{Re}\{q\} < 0$ it is seen that $|\zeta| < 1$.

The backward Euler method is implicit* and hence more difficult to solve as y_{n+1} appears on both sides of the equation. Dahlquist [DAH63] has proven that no explicit* LMM can be A-stable. He further showed that if (2.5) is to be A-stable the order of the integration formula cannot exceed 2.

As high order integration processes that do not restrict the step-size due to stability considerations are desirable, researchers have looked for ways to

---

*A linear multistep method is called implicit if in (2.5) $\beta_1 \neq 0$. If $\beta_1 = 0$ it is said to be explicit.

circumvent Dahlquist's result.

2.4 A(α)-Stability

Widlund [WID67] introduced the concept of A(α)-stability; namely,

Definition 2.3: A linear multistep method is said to be A(α)-stable, $\alpha \varepsilon (0,\pi/2)$, if all solutions of (2.5) tend to zero as $n \to \infty$ when the method is used to integrate the differential equation

$$\dot{y}(t) = q\, y(t) \tag{2.17}$$

with any fixed positive h where q is a complex constant which lies in the set

$$S_\alpha = \{z : |\arg(-z)| < \alpha,\ z \neq 0\}. \tag{2.18}$$

A method is $A(\pi/2)$-stable if it is A(α)-stable for all $\alpha \varepsilon (0,\pi/2)$, and A(0)-stable if it is A(α)-stable for some (sufficiently small) $\alpha \varepsilon (0,\pi/2)$.

As the set S_α defines a wedge in the complex domain, the angle α in Definition 2.3 is referred to as the Widlund wedge angle.

Dahlquist's concept of A-stability is seen to correspond to $A(\pi/2)$-stability.

Widlund proved that an explicit method cannot be $A(0)$-stable and that for all $\alpha \varepsilon [0,\pi/2)$ there exist $A(\alpha)$-stable methods of third and fourth order.

By allowing α to be less then $\pi/2$, methods as high as 12-th order have been reported [JAI70]. Indeed, Gear [GEA71a] and Brayton, Gustavson, and Hachtel [BRA72a] both have operational general purpose programs specifically designed for stiff systems. Both programs are similar and incorporate and make use of multistep methods up to sixth order, namely the backward differentiation formulas [HEN62], each of which is $A(\alpha)$-stable for some $\alpha \varepsilon (0,\pi/2]$.

The effect of decreasing α is to restrict the eigenvalues of the Jacobian matrix to a smaller region of the complex domain wherein the method remains asymptotically stable for arbitrary, fixed h.

As the order of the integration process may exceed 2 for $A(\alpha)$-stable methods, $\alpha<\pi/2$, researchers have focussed their attention on obtaining higher order methods that are $A(\alpha)$-stable, with α as large as possible, for the solution of stiff systems.

2.5 Runge-Kutta Methods (RKM)

In addition to linear multistep methods Runge-Kutta

type formulas have been used to numerically integrate (2.1). A q-stage RKM may be written as [BUT64]

$$y_{n+1} = y_n + h \sum_{j=1}^{q} b_j k_j, \tag{2.19a}$$

where $k_1, k_2, \ldots, k_q$ satisfy the equations

$$k_i = f\left(y_n + h \sum_{j=1}^{q} A_{ij} k_j,\ t_n + C_i h\right), \tag{2.19b}$$

with $i = 1, \ldots, q$,

and

$$C_i = \sum_{j=1}^{q} A_{ij} \quad \text{for } i = 1, \ldots, q. \tag{2.19c}$$

All RKM require only one initial condition and hence are self-starting. However, for every step taken, a q-stage Runge-Kutta process requires q function evaluations* in contrast to an LMM which only requires one.

If in (2.19b) $A_{ij} = 0$ for $j \geq i$, the process is said to be explicit. Otherwise it is said to be implicit. The commonly used fourth order Runge-Kutta method is a 4-stage explicit process. It is given as

---

*Evaluating f(y(t), t) is termed a function evaluation. If this is a complicated function, as it oftentimes is, it may be a costly operation.

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4), \qquad (2.20a)$$

where

$$\begin{aligned} k_1 &= f(y_n, t_n), \\ k_2 &= f(y_n + \tfrac{1}{2}hk_1, t_n + \tfrac{1}{2}h), \\ k_3 &= f(y_n + \tfrac{1}{2}hk_2, t_n + \tfrac{1}{2}h), \\ k_4 &= f(y_n + hk_3, t_n + h). \end{aligned} \qquad (2.20b)$$

In the notation of (2.19)

$$\begin{aligned} &2b_1 = b_2 = b_3 = 2b_4 = \tfrac{1}{3}, \\ &C_1 = A_{1,j} = 0,\ j = 1, 2, 3, 4, \\ &C_2 = A_{2,1} = \tfrac{1}{2},\ A_{2,j} = 0,\ j = 2, 3, 4, \\ &C_3 = A_{3,2} = \tfrac{1}{2},\ A_{3,j} = 0,\ j = 1, 3, 4, \\ &C_4 = A_{4,3} = 1,\ A_{4,j} = 0,\ j = 1, 2, 4. \end{aligned} \qquad (2.21)$$

Though it has the desirable property of being explicit it is not $A(\alpha)$-stable for any $\alpha \varepsilon (0,\pi/2]$ and is hence unsuited for the solution of stiff systems.

Butcher [BUT64] has explored the general class of q-stage RKM. He has shown that an implicit q-stage process can have order 2q. Ehle [EHL68] has shown that the root of the resulting characteristic equation is the P_{qq} Padé approximation to the exponential and hence is A-stable.

Implicit RKM, hence, can have arbitrarily high order

and still be A-stable at the cost of q function evaluations per time step for an order 2q method. This is in contrast to implicit LMM which require only one function evaluation per time point.*

Though implicit Runge-Kutta methods are applicable to the solution of stiff systems of ordinary differential equations they were not considered in this research effort due to the high number of function evaluations required per time point.

2.6 Composite Linear Multistep Methods (CLMM)

In 1971 the concept of a composite linear multistep method was introduced by Sloate and Bickart [SLO71b]. It consists of ℓ linear multistep formulas used to simultaneously solve for ℓ future time points.† They may be written as§

---

*Both implicit LMM and implicit RKM may have to be solved by an iterative method resulting in a higher number of function evaluations per time point—1 function evaluation for an implicit LMM and q function evaluations for an implicit RKM, per iteration. However, on a per iteration basis, an implicit LMM offers a distinct advantage over an implicit RKM.

†A variation of (2.22) has been used in which (2.22) is used to solve for ℓ future time points, keeping only the first m ($<\ell$) of these values for the next iteration. As an extra (ℓ-m) function evaluations are performed at each stage, this form of the CLMM was not considered in this research effort.

§Note: The index for n = 0 and j = -k + 1 is not zero. To make it zero results in an unnecessarily cumbersome equation. It is left to the reader to recognize that to make the equations precise k - 1 should be added to all the indices on t and y.

$$\sum_{j=-k+1}^{\ell} (\alpha_{i,j}\, y_{n\ell+j} - h\beta_{i,j}\, \dot{y}_{n\ell+j}) = 0,$$

$$\text{with } i = 1, \ldots, \ell. \qquad (2.22)$$

As with an LMM one may consider

$$L_i[y(t_{n\ell}),h] \equiv \sum_{j=-k+1}^{\ell} \{\alpha_{i,j}\, y(t_{n\ell+j}) - h\beta_{i,j}\dot{y}(t_{n\ell+j})\},$$

$$\text{with } i = 1, \ldots, \ell. \qquad (2.23)$$

Under the assumption that $y(t)$ in (2.23) possesses a convergent Taylor series at $t = t_{n\ell}$, (2.23) may be rewritten as

$$L_i[y(t_{n\ell}),h] = \sum_{j=0}^{\infty} C_{i,j} h^j y^{(j)}(t_{n\ell}),$$

$$\text{with } i = 1, \ldots, \ell, \qquad (2.24)$$

where

$$C_{i,j} = \sum_{n=-k+1}^{\ell} \left(\frac{1}{j!} n^j \alpha_{i,n} - \frac{1}{(j-1)!} n^{j-1} \beta_{i,n}\right),$$

$$\text{with } i = 1, \ldots, \ell. \qquad (2.25)$$

The order of a CLMM may now be defined, as in Definition 2.4, which follows.

<u>Definition 2.4</u>: The i-th equation of the composite

linear multistep method (2.22) is said to be of order p_i if $C_{i,j} = 0$ for $j = 0, \ldots, p$ but $C_{i,p+1} \neq 0$, where $C_{i,j}$ is given by (2.25). Let $p = \min_{1 \le i \le \ell} \{p_i\}$. Then the composite linear multistep method (2.22) is said to be of <u>order</u> p.

As with linear multistep methods the order of a CLMM is related to its local discretization error as follows: If

i) the composite linear multistep method is of order p;

ii) $y_{n\ell+j} = y(t_{n\ell+j}) + O(h^{p+1})$, $j = -k+1, \ldots, 0$; and

iii) $y(t)$ is $(p+1)$ times continuously differentiable

then $y_{n\ell+i} = y(t_{n\ell+i}) + O(h^{p+1})$ for $i=1, \ldots, \ell$.

The definitions of A-stability and A(α)-stability are also readily extended to CLMM with the words "composite linear multistep method" replacing the words "linear multistep method" in Definitions 2.2 and 2.3.

Sloate and Bickart [SLO71b] have offered the following conjecture:

<u>Conjecture</u>: The order of an A-stable composite linear multistep method cannot exceed 2ℓ.

For $\ell = 1$ Dahlquist [DAH63] has proven the conjecture. Sloate [SLO71a] has derived a fourth order A-stable method

with $\ell = 2$, the limit anticipated by the conjecture. For only one backpoint, $k = 1$, the conjecture has been proven [SLO71b], though the maximal order of 2ℓ cannot be achieved for $\ell > 1$.*

Though at first glance the introduction of a composite system seems to complicate the problem, as ℓ equations must be solved simultaneously, the real benefit is derived from the hope that A-stable processes of at least order 2ℓ, the limit hypothesized in the conjecture, can be found.

High order composite linear multistep methods have been reported. Reference has already been made to Sloate's algorithm [SLO71a] which is a 4-th order method and consists of two equations.† A-stable methods of order $\ell + 1$ which retain all future points and with only one backpoint§ have been independently reported by Bickart, Burgess, and Sloate [BIC71] and Watts [WAT71]. The former reference also describes results obtained from a program utilizing these formulas for $\ell = 1, \ldots, 6$. Watts [WAT71] has proven that

---

*For $\ell = 1$, the trapezoidal formula which has only one backpoint is A-stable, of order 2. For $k = 1$ and $\ell > 1$, only $(\ell + 1)$ degrees of freedom exist in choosing the parameters for each of the ℓ equations. Hence the maximal order achievable is $\ell + 1$.

†Sloate's algorithm only maintains one of the two future time points solved for in each stage of the process.

§The advantage of only one backpoint, $k = 1$, manifests itself in the ease with which the step-size may be changed.

these methods are not A-stable for $\ell > 8$ but has shown that it is possible to achieve A-stability with order equal to the number of equations, for all ℓ, again with only one back-point.

Composite linear multistep methods that achieve high order, with the desirable properties of A-stability and only one function evaluation per time point retained, reported so far have done so at the expense of increased computation since ℓ equations must be solved simultaneously at each stage of the process. As one of the primary objectives of this research effort was to derive processes that were efficient, cyclic composite linear multistep methods were considered. (A composite linear multistep method is said to be cyclic if in (2.22) $\alpha_{i,j} = \beta_{i,j} = 0$ for $j > i$.) At first it was hoped that high order A-stable cyclic methods could be found. This goal was not achieved. However, the new integration methods described in Chapters IV and V are $A(\alpha)$-stable and hence applicable to the solution of stiff systems. Moreover, they exhibit better stability properties than the backward differentiation formulas in Gear's widely accepted algorithm.

2.7 The Characteristic Equation

Consider the composite linear multistep method (2.22). The set of ℓ scalar equations can be expressed

in matrix notation as

$$\{A_o \underset{\sim}{Y}_n + A_{-1}\underset{\sim}{Y}_{n-1} + \dots + A_{-K}\underset{\sim}{Y}_{n-K}\}$$

$$-h\{B_o \dot{\underset{\sim}{Y}}_n + B_{-1}\dot{\underset{\sim}{Y}}_{n-1} + \dots + B_{-K}\dot{\underset{\sim}{Y}}_{n-K}\} = \underset{\sim}{0}, \tag{2.26}$$

where

$$\underset{\sim}{Y}_{n-j} \equiv \begin{pmatrix} y_{(n-j)\ell+\ell} \\ \vdots \\ y_{(n-j)\ell+1} \end{pmatrix}, \quad \dot{\underset{\sim}{Y}}_{n-j} \equiv \begin{pmatrix} \dot{y}_{(n-j)\ell+\ell} \\ \vdots \\ \dot{y}_{(n-j)\ell+1} \end{pmatrix},$$

$$A_{-j} \equiv \begin{pmatrix} \alpha_{1,-j\ell+\ell} & \dots & \alpha_{1,-j\ell+1} \\ \vdots & & \vdots \\ \alpha_{\ell,-j\ell+\ell} & \dots & \alpha_{\ell,-j\ell+1} \end{pmatrix}, \tag{2.27}$$

$$B_{-j} \equiv \begin{pmatrix} \beta_{1,-j\ell+\ell} & \dots & \beta_{1,-j\ell+1} \\ \vdots & & \vdots \\ \beta_{\ell,-j\ell+\ell} & \dots & \beta_{\ell,-j\ell+1} \end{pmatrix},$$

for $j = 0, \dots, K$, K is the integer part of $(k + \ell - 1)/\ell$, and $\alpha_{i,m} = \beta_{i,m} = 0$ whenever $m < -k + 1$.

Now consider using the CLMM as expressed by (2.26)

to integrate the linear differential equation $\dot{y}(t) = q\, y(t)$, or equivalently $\dot{\underset{\sim}{Y}}_{n-j} = q\, \underset{\sim}{Y}_{n-j}$, $j = 0, \ldots, K$. With $\lambda = qh$, (2.26) becomes

$$(A_o - \lambda B_o)\underset{\sim}{Y}_n + \cdots + (A_{-K} - \lambda B_{-K})\underset{\sim}{Y}_{n-K} = \underset{\sim}{0}, \quad (2.28)$$

which is a vector difference of equation of the form

$$\sum_{j=-K}^{0} R_j(\lambda)\underset{\sim}{Y}_{n-j} = \underset{\sim}{0}, \quad (2.29)$$

where $R_j(\lambda) \equiv A_j - \lambda B_j$. Define

$$p(\zeta,\lambda) \equiv \det \left\{ \sum_{j=-K}^{0} R_j(\lambda)\zeta^{K+j} \right\} . \quad (2.30)$$

Then for a given step-size, i.e., fixed λ, (2.29) has a solution, governed by the zeros $\zeta_i(\lambda)$ of $p(\zeta,\lambda)$, given as

$$\underset{\sim}{Y}_n = \sum_{i=1}^{K\ell} \underset{\sim}{C}_i \zeta_i^n(\lambda), \quad (2.31)$$

where the vectors $\underset{\sim}{C}_i$, $i = 1, \ldots, K\ell$ are functions of the

initial conditions.* (Note: Of the $K\ell$ zeros of $p(\zeta,\lambda)$,

---

*The formulation as given by (2.31) can be proven by an extension of the scalar case as given in Section 2.3. As in the scalar case a simplification has been made in (2.31). In general one must consider the invariant polynomials of the polynomial matrix [GAN60]

$$R(\zeta,\lambda) = \sum_{j=-K}^{0} R_j(\lambda)\zeta^{K+j}. \tag{2.32}$$

The invariant polynomials, $\iota_1(\zeta,\lambda), \ldots, \iota_\ell(\zeta,\lambda)$, are defined as

$$\iota_j(\zeta,\lambda) \equiv \frac{D_{\ell+1-j}(\zeta,\lambda)}{D_{\ell-j}(\zeta,\lambda)}, \text{ with } j = 1, \ldots, \ell, \tag{2.33}$$

where $D_j(\zeta,\lambda)$ is the greatest common divisor of all minors of order j of the determinant of $R(\zeta,\lambda)$ and $D_o(\zeta,\lambda) \equiv 1$. The general solution is then given as [DEJ67]

$$\underset{\sim}{Y}_n = \sum_{i=1}^{s} \sum_{j=1}^{\ell} \sum_{\sigma=1}^{m_{i,j}} \underset{\sim}{C}_{i,j,\sigma}\, n^{\sigma-1}\, \zeta_i^n(\lambda), \tag{2.31'}$$

where the ζ_i's are the s distinct zeros of (2.30) of multiplicity $m_{i,j}$ in the j-th invariant polynomial $\iota_j(\zeta,\lambda)$ and the $\underset{\sim}{C}_{i,j,\sigma}$'s are functions of the initial conditions. (Note: It is possible that $m_{i,j} \equiv 0$ for $j \geq j_o > 1$. Moreover, $m_{i,j+1} \leq m_{i,j}$.) For the scalar case, i.e., $\ell=1$, (2.31') reduces to (2.11').

$K\ell - k$ of them are identically zero for all values of λ.) Equation (2.30) is called the characteristic polynomial of the composite linear multistep method (2.26).

Now if a method is to be A-stable, then $\underset{\sim}{Y}_n \to \underset{\sim}{0}$ as $n \to \infty$. Therefore by (2.31) a method will be A-stable if and only if $|\zeta_i(\lambda)| < 1$ for $i = 1, \ldots, K\ell$, for all values of λ with $\mathrm{Re}\{\lambda\} < 0$. Similarly, if a method is to be $A(\alpha)$-stable for some $\alpha \varepsilon (0, \pi/2)$, then $|\zeta_i(\lambda)| < 1$, for $i = 1, \ldots, K\ell$, for all values of $\lambda \neq 0$ such that $|\arg(-\lambda)| < \alpha$.

In the next chapter the basis for necessary and sufficient conditions for graphically determining the stability boundaries of a composite linear multistep method is derived, i.e., determine the regions of the λ-plane for which all solutions of a CLMM, when used to integrate a linear differential equation, tend to zero as $n \to \infty$. The characteristic equation (2.30) is examined for those values of λ for which $|\zeta_i(\lambda)| < 1$.

CHAPTER III

STABILITY BY GRAPHICAL METHODS

3.1 Introduction

Consider the linear differential equation

$$\dot{y}(t) = q\, y(t), \tag{3.1}$$

and the composite linear multistep method

$$\{A_0 \underset{\sim}{Y}_n + \cdots + A_{-K} \underset{\sim}{Y}_{n-K}\} - h\{B_0 \dot{\underset{\sim}{Y}}_n + \cdots + B_{-K} \dot{\underset{\sim}{Y}}_{n-K}\} = \underset{\sim}{0}. \tag{3.2}$$

Asymptotic stability for a CLMM will be as defined in Definition 3.1 which follows.

<u>Definition 3.1</u>: The composite linear multistep method (3.2) is said to be <u>asymptotically stable</u> for $\lambda_0 = q\, h_0$ if all solutions of (3.2) tend to zero as $n \to \infty$ when it is used to integrate the linear differential equation (3.1).

If (3.1) is substituted into (3.2), it has been shown in Section 2.7 that the solution is governed by the zeros, $\zeta_i(\lambda)$, of the characteristic polynomial of (3.2), namely

$$p(\zeta,\lambda) = \det \{ \sum_{i=-K}^{0} (A_i - \lambda B_i)\zeta^{K+i} \}. \tag{3.3}$$

Definition (3.1) is equivalent to saying the CLMM (3.2) is asymptotically stable at the point λ_o if all zeros $\zeta_i(\lambda_o)$ of $p(\zeta,\lambda_o)$ are <u>well defined</u>, that is, assume a unique value, and less than unity in modulus.

The characteristic polynomial can be written as the product of three polynomials as

$$p(\zeta,\lambda) = Z(\zeta)\ \hat{p}(\zeta,\lambda)\ \Lambda(\lambda), \tag{3.4}$$

where $Z(\zeta)$ and $\Lambda(\lambda)$ are, respectively, polynomials in ζ and λ only, and $\hat{p}(\zeta,\lambda)$ is a polynomial in both ζ and λ that cannot be further factored into the form (3.4).

If $Z(\zeta)$ is a non-constant polynomial and its zeros, z_i, are such that for some i $|z_i| \geq 1$, then the method is not asymptotically stable for any λ. On the other hand, if for all i $|z_i| < 1$, then the stability of the method is not affected by $Z(\zeta)$. For those values of λ such that $\Lambda(\lambda) = 0$ the zeros $\zeta_i(\lambda)$ of $p(\zeta,\lambda)$ are <u>not well defined</u> (equivalently, <u>ill-defined</u>).* However, in a deleted neighborhood of each

---

*Note: The zeros of $\Lambda(\lambda)$ are also zeros of $\det\{A_o-\lambda B_o\}$; at the other zeros of $\det\{A_o-\lambda B_o\}$, the zeros $\zeta_i(\lambda)$ are <u>unbounded</u> (equivalently, <u>undefined</u>). Thus, the zeros $\zeta_i(\lambda)$ of $p(\zeta,\lambda)$ either are not well defined or are undefined at those values of λ for which $\det\{A_o-\lambda B_o\}$ vanishes. Now, it is for just these values of λ that (3.2) does not have a unique solution when used to solve (3.1).

of these points, $\zeta_i(\lambda)$ is well defined. (These zeros are discussed in greater detail in Section 3.2.)

Considering the complex λ-plane one may collect all points λ for which the CLMM is asymptotically stable into a set, $S(\lambda)$, i.e.,

$$S(\lambda) = \{\lambda: \text{ CLMM is asymptotically stable}\}, \tag{3.5a}$$

or, equivalently,

$$S(\lambda) = \{\lambda: \zeta_i(\lambda) \text{ is well defined and } |\zeta_i(\lambda)| < 1,\ i = 1, \ldots, K\ell\}. \tag{3.5b}$$

If $S(\lambda)$ contains the open left half plane then the method is A-stable; i.e., with $L_\lambda \equiv \{\lambda: \text{ Re}\{\lambda\} < 0\}$ if $L_\lambda \subset S(\lambda)$ then the CLMM (3.2) is A-stable. Note: In the complement of the set $S(\lambda)$, denoted as $S^c(\lambda)$, the composite linear multistep method is not asymptotically stable, by definition of $S(\lambda)$, and, hence, there must be either at least one zero of the characteristic polynomial such that $\zeta_j(\lambda)$ is well defined and $|\zeta_j(\lambda)| \geq 1$ or the $\zeta_i(\lambda)$'s are not well defined, corresponding to a zero of $\Lambda(\lambda)$.

Let $B(\lambda)$ be defined as

$$B(\lambda) \equiv \overline{S(\lambda)} \cap S^c(\lambda) - \{\lambda: \Lambda(\lambda) = 0\}, \tag{3.6a}$$

where $\overline{S(\lambda)}$ is the closure of $S(\lambda)$. $B(\lambda)$ may also be defined as

$$B(\lambda) = \{\lambda: \text{ for at least one index } j \ \zeta_j(\lambda) \text{ is well defined and } |\zeta_j(\lambda)| = 1\}. \quad (3.6b)$$

In the next section it is proven that $B(\lambda)$ can be described by continuous curves, $\gamma_i(\lambda)$, in the extended complex λ-plane. These curves, $\gamma_i(\lambda)$, together separate the λ-plane into regions* wherein the CLMM is either asymptotically stable or not asymptotically stable.†

3.2 The Lambda Plot

The characteristic polynomial (3.3)§ may be written as a polynomial in two variables as

$$p(\zeta,\lambda) = \sum_{j=0}^{k} \sum_{i=0}^{\ell} a_{ij}\lambda^i\zeta^j, \quad (3.7)$$

---

*The term components is used by Odeh and Liniger [ODE71].

†Except possibly at a finite set of points where the $\zeta_i(\lambda)$ are ill-defined, i.e., at those values of λ for which $\Lambda(\lambda) = 0$.

§The characteristic polynomial has a factor $\zeta^{K\ell-k}$; i.e., in (3.4) $Z(\zeta)$ has at least a factor $\zeta^{K\ell-k}$. Since this factor in no way affects stability considerations, as it only adds at most a multiple root at $\zeta = 0$ for all values of λ, it can be dropped from consideration. Therefore, in all that follows, $p(\zeta,\lambda)$ will be treated as though this factor has been removed. The resulting polynomial is hence of degree $k \leq K\ell$ in ζ.

or alternately as

$$p(\zeta,\lambda) = \sum_{i=0}^{\ell} p_i(\zeta)\lambda^i, \tag{3.8}$$

where

$$p_i(\zeta) \equiv \sum_{j=0}^{k} a_{ij}\zeta^j, \text{ with } i = 0, \ldots, \ell. \tag{3.9}$$

Similarly, one can express $p(\zeta,\lambda)$ as

$$p(\zeta,\lambda) = \sum_{j=0}^{k} q_j(\lambda)\zeta^j, \tag{3.10}$$

where

$$q_j(\lambda) \equiv \sum_{i=0}^{\ell} a_{ij}\lambda^i, \text{ with } j = 0, \ldots, k. \tag{3.11}$$

It shall be assumed that (3.7) is irreducible.* If it is not, then $p(\zeta,\lambda)$ can be written as the product of irreducible polynomials and the discussion to follow applies to each of the irreducible polynomials that is not a function of λ or ζ alone.

Since (3.8) is a polynomial in ζ and λ, it may be

---

*A polynomial in two variable is <u>irreducible</u> if it cannot be expressed as the product of two polynomials neither of which is constant. By assuming that $p(\zeta,\lambda)$ is irreducible $\{\lambda: \Lambda(\lambda) = 0\}$ will be empty, and need not be considered. These points will be reconsidered later.

written as*

$$p(\zeta,\lambda) = p_\ell(\zeta)[\lambda-f_1(\zeta)][\lambda-f_2(\zeta)]\cdots[\lambda-f_\ell(\zeta)], \quad (3.12)$$

where each $f_i(\zeta)$ is a well defined algebraic function of ζ, analytic except at, at most, a finite number of isolated singularities. Denoting these singular points as c_n they are either the zeros of the resultant of $p(\zeta,\lambda)$ and $\partial[p(\zeta,\lambda)]/\partial\zeta$ or the zeros of $p_\ell(\zeta)$.

At the singular points c_n such that $p_\ell(c_n) \neq 0$, Ahlfors [AHL66] has shown that each $f_i(\zeta)$ is bounded, that is, these singular points are <u>ordinary algebraic singularities</u> (equivalently, <u>branch points</u>). Furthermore, at those points c_n such that $p_\ell(c_n) = 0$ there exists a least rational number m,† non-negative and finite, such that $f_i(\zeta)(\zeta-c_n)^m$ remains bounded, that is, these singular points are <u>algebraic poles</u> or, in the special case corresponding to m=0, are ordinary algebraic singularities.

Now, consider

$$p(e^{j\theta},\lambda) = 0, \quad (3.13)$$

---

*The material in this and the following paragraph is excerpted from Chapter 8 of Ahlfors, <u>Complex Variables</u> [AHL66].

†The rational number m is bounded from above by the multiplicity of the zero c_n of $P_\ell(\zeta)$.

where $j = \sqrt{-1}$ and $\theta \in [0,2\pi)$. From the previous development, (3.13) defines ℓ loci

$$\lambda_i(\theta) = f_i(e^{j\theta}), \text{ for } \theta \in [0,2\pi) \text{ and } i=1, \ldots, \ell, \quad (3.14)$$

in the extended complex λ-plane. The basis now exists for

<u>Definition 3.2</u>: The set of points

$$\gamma(\lambda) = \{\lambda: \ \lambda = \lambda_i(\theta), \ \theta \in [0,2\pi), \ i=1, \ldots, \ell\}, \quad (3.15)$$

for a given composite linear multistep method, is termed the <u>Lambda Locus</u>, or the <u>Lambda Plot</u>, of the method.

The Lambda Locus, $\gamma(\lambda)$, is a set of closed curves which divide the extended complex plane into a finite collection of open connected regions. Let $N(\lambda)$ be one such open connected region and $\partial N(\lambda)$ be its boundary, i.e.,

$$\partial N(\lambda) = \overline{N(\lambda)} - N(\lambda). \quad (3.16)$$

Note: $\partial N(\lambda) \in \gamma(\lambda)$ but $N(\lambda) \cap \gamma(\lambda) = \phi$, ϕ being the empty set. The stage has now been set for

<u>Theorem 3.3</u>: For all points λ contained within the open region $N(\lambda)$, one, but not both, of the following statements is true:

i) The composite linear multistep method is asymptotically stable.

ii) The composite linear multistep method is not asymptotically stable.

The consequence of Theorem 3.3 is that $\gamma(\lambda)$ separates the complex λ-plane into regions wherein the method is either asymptotically stable or not asymptotically stable.

<u>Proof of Theorem 3.3</u>: Assume there exists a point $\lambda_1 \in N(\lambda)$ such that $\lambda_1 \in S(\lambda)$, i.e., λ_1 is a stable point of the CLMM. Next, assume that there exists another point $\lambda_2 \in N(\lambda)$ such that $\lambda_2 \notin S(\lambda)$. Now, consider the two equations $p(\zeta,\lambda_1) = 0$ and $p(\zeta,\lambda_2) = 0$. By a development similar to (3.12), we have,* with $i = 1, 2$,

$$\begin{aligned} p(\zeta,\lambda_i) &= q_k(\lambda_i)[\zeta-g_1(\lambda_i)][\zeta-g_2(\lambda_i)]\cdots[\zeta-g_k(\lambda_i)] \\ &= 0. \end{aligned} \tag{3.17}$$

Two cases need to be considered as follows:

Case 1. Neither λ_1 nor λ_2 is a singular point of $g_j(\lambda)$, $j=1, \ldots, k$. Note: Under these conditions $q_k(\lambda_i) \neq 0$, $i=1,2$.

Case 2. Either, or both, λ_1 or λ_2 is a singular point of $g_j(\lambda)$ for some j.

<u>Case 1</u>: By construction, for at least one index j, $|g_j(\lambda_1)| < 1$

---

*See (3.10) for a definition of $q_k(\lambda)$.

and $|g_j(\lambda_2)| \geq 1$. If $|g_j(\lambda_2)| = 1$ then $\lambda_2 \in \gamma(\lambda)$ but $N(\lambda) \cap \gamma(\lambda) = \phi$. Hence $|g_j(\lambda_2)| > 1$. As $g_j(\lambda)$ is analytic except at a finite number of points there must exist a path from λ_1 to λ_2, say L, completely contained within $N(\lambda)$ and along which $g_j(\lambda)$ is analytic. As $|g_j(\lambda_2)| > 1 > |g_j(\lambda_1)|$ there must exist a point $\lambda_3 \in L$ such that $|g_j(\lambda_3)| = 1$, i.e., $\lambda_3 \in \gamma(\lambda)$. But $N(\lambda) \cap \gamma(\lambda) = \phi$ which leads to a contradiction that $\lambda_3 \in \phi$. Hence, λ_1 and λ_2 cannot both be regular points of $N(\lambda)$ with $\lambda_1 \in S(\lambda)$ and $\lambda_2 \notin S(\lambda)$.

<u>Case 2</u>: As before, for at least one index j, $|g_j(\lambda_1)| < 1$ and $|g_j(\lambda_2)| \geq 1$. Again it can be assumed that $|g_j(\lambda_2)| \neq 1$ for otherwise $\lambda_2 \in \gamma(\lambda)$, which contradicts $\gamma(\lambda) \cap N(\lambda) = \phi$.

Assume λ_2 is a singular point of $g_j(\lambda)$ but $q_k(\lambda_2) \neq 0$, i.e., λ_2 is an ordinary algebraic singularity of $g_j(\lambda)$. Now, consider the function $G_j(\lambda)$ defined as

$$G_j(\lambda) \equiv |g_j(\lambda)|, \tag{3.17a}$$

where

$$G_j(\lambda_2) = \lim_{\lambda \to \lambda_2} G_j(\lambda). \tag{3.17b}$$

The function $G_j(\lambda)$ is well defined and continuous in a neighborhood of λ_2. Hence, for each $\varepsilon > 0$ there exists a

$\delta > 0$ such that

$$|G_j(\lambda_3) - G_j(\lambda_2)| < \varepsilon \tag{3.18}$$

for all λ_3 for which $|\lambda_3 - \lambda_2| < \delta$. Therefore, as $G_j(\lambda_2) > 1$, an ε can be chosen such that $G_j(\lambda_3) > 1$ for $\lambda_3 \in N(\lambda)$. At $\lambda = \lambda_3$ $g_j(\lambda)$ is analytic and $|g_j(\lambda_3)| > 1$. Therefore, by a similar argument to that in Case 1, a contradiction results.

Next, suppose $q_k(\lambda_2) = 0$, i.e., λ_2 is an algebraic pole of $g_j(\lambda)$,* thus $g_j(\lambda_2) \to \infty$ as $\lambda \to \lambda_2$. In some sufficiently small deleted neighborhood of λ_2 $g_j(\lambda_2)$ is analytic and $|g_j(\lambda)| > 1$. Again a contradiction results.

Now, let λ_1 be a singular point of $g_j(\lambda)$. Note: λ_1 cannot be an algebraic pole, by construction; hence $q_k(\lambda_1) \neq 0$ and λ_1 is an ordinary algebraic singularity. As before, consider the function $\hat{G}_j(\lambda)$ defined as

$$\hat{G}_j(\lambda) \equiv |g_j(\lambda)|, \tag{3.19a}$$

where

---

*The zeros of $q_k(\lambda)$ correspond to those values of λ for which the zeros of the characteristic polynomial, $\zeta_i(\lambda)$, are undefined, i.e., $q_K(\lambda) = \det\{A_o - \lambda B_o\}$. Further, by assumption, none of these points are ill-defined—$\Lambda(\lambda) =$ constant.

$$\hat{G}_j(\lambda_1) = \lim_{\lambda \to \lambda_1} \hat{G}_j(\lambda). \tag{3.19b}$$

$\hat{G}_j(\lambda)$ is well defined and continuous in a neighborhood of λ_1. Hence, for each $\varepsilon > 0$ there exists a $\delta > 0$ such that

$$|\hat{G}_j(\lambda_3) - \hat{G}_j(\lambda_1)| < \varepsilon \tag{3.20}$$

for all λ_3 for which $|\lambda_3 - \lambda_1| < \delta$. Therefore, as $\hat{G}_j(\lambda_1) < 1$, an ε can be chosen such that $\hat{G}_j(\lambda_3) < 1$ for $\lambda_3 \in N(\lambda)$. At $\lambda = \lambda_3$ $g_j(\lambda)$ is analytic and $|g_j(\lambda_3)| < 1$. Once again a contradiction results. Q.E.D.

From this theorem follows

Corollary 3.4: If $N(\lambda)$ is an open connected region in the extended complex λ-plane determined by the Lambda Locus and if there exists a point $\lambda \in N(\lambda)$ such that the composite linear multistep method is asymptotically stable at λ then the CLMM is stable for all $\lambda \in N(\lambda)$.

and

Corollary 3.5: If in any open connected region, $N(\lambda)$, of the extended complex λ-plane determined by the Lambda Locus, there exists a point λ_o such that $n \leq k$ of the values $p(\zeta, \lambda_o) = 0$ are greater than one in modulus then this condition is true for all $\lambda \in N(\lambda)$. Moreover, if $N_a(\lambda)$ and $N_b(\lambda)$ are two such regions

such that $\partial N_a(\lambda) \cap \partial N_b(\lambda) \neq \phi$ then as one crosses from $N_a(\lambda)$ to $N_b(\lambda)$ at least one root of the characteristic equation has gone from being less than unity in modulus to greater than unity, or vice versa.

As previously noted, $\gamma(\lambda)$ divides the extended complex λ-plane into a collection of open connected regions. At most $\ell+1$ such regions can be formed, each of which must be tested to determine if the composite linear multistep method is asymptotically stable or not in that region. Moreover, as the set of points in the extended λ-plane defined by (3.13) is a mapping of a closed contour, its image in the extended λ-plane must also be a set of closed contours [AHL66]. Hence, if one of the $f_i(e^{j\theta})$ of (3.14) does not form a closed contour in the extended λ-plane as θ goes from 0 to 2π, it must join at least one other segment to complete the contour.

It has been assumed that the characteristic polynomial is irreducible. If it is not then the results of this section apply to each of its irreducible factors, say $p_1(\zeta,\lambda), \ldots, p_n(\zeta,\lambda)$. Associated with each irreducible factor will be a stability region, $S_i(\lambda)$. The stability region of the CLMM, $S(\lambda)$, is then given as

$$S(\lambda) = S_1(\lambda) \cap \cdots \cap S_n(\lambda). \tag{3.21}$$

Excluded from the discussion have been polynomials which are functions of λ alone and of ζ alone; i.e., it has been assumed that in (3.4) both $\Lambda(\lambda)$ = constant and $Z(\zeta)$ = constant. If $\Lambda(\lambda)$ is a non-constant polynomial the zeros $\zeta_i(\lambda)$ of the characteristic polynomial $p(\zeta,\lambda)$ are ill-defined at the zeros of $\Lambda(\lambda)$ and $\{\lambda: \Lambda(\lambda) = 0\}$ cannot belong to the stability region $S(\lambda)$ of the method. If $Z(\zeta)$ is a non-constant polynomial and its zeros, z_i, are such that for some i, $|z_i| \geq 1$, then the method is not asymptotically stable for any λ. On the other hand, if for all i, $|z_i| < 1$, then the stability of the method is not effected by $Z(\zeta)$.

An example of a factorization of this type is given in (3.22) [WAT71].

$$y_{n+1} = y_n + h\{\dot{y}_n + \dot{y}_{n+2}\}$$
$$y_{n+2} = y_n + h\{\dot{y}_n + \dot{y}_{n+1}\} \tag{3.22}$$

The characteristic polynomial is given as

$$p(\zeta,\lambda) = \det\left\{\begin{pmatrix} \zeta & -1-\lambda(\zeta+1) \\ -\lambda\zeta & \zeta-1-\lambda \end{pmatrix}\right\}, \tag{3.23a}$$

or

$$p(\zeta,\lambda) = \zeta[(\zeta-1) - (\zeta+1)\lambda](1+\lambda). \tag{3.23b}$$

In the notation of (3.4),

$$Z(\zeta) = \zeta,$$
$$\hat{p}(\zeta,\lambda) = (\zeta-1) - (\zeta+1)\lambda, \tag{3.24}$$

and

$$\Lambda(\lambda) = 1+\lambda.$$

The zero of $Z(\zeta)$ at $\zeta = 0$ does not affect stability and can be ignored. The stability region for the irreducible polynomial $\hat{p}(\zeta,\lambda)$ consists of the open left-half plane as can be seen from

$$\hat{p}(\zeta,\lambda) = -(\zeta+1)(\lambda-\frac{\zeta-1}{\zeta+1}). \tag{3.25}$$

The Lambda Locus for $\hat{p}(\zeta,\lambda)$ consists of the imaginary axis which closes at infinity. Moreover, $\hat{p}(\zeta,\lambda)$ has an algebraic pole at $\lambda = +1$. The zero of $\Lambda(\lambda)$ at $\lambda = -1$ indicates the zero of $\hat{p}(\zeta,\lambda)$ is ill-defined there; hence, the point $\lambda = -1$ must be excluded from the stability region determined by a consideration of $\hat{p}(\zeta,\lambda)$ alone. The Lambda Locus for the irreducible polynomial $\Lambda(\lambda)$ consists of a single point,* namely $\lambda = -1$.

3.3 Stiff Stability

For a composite linear multistep method to be useful for the solution of stiff systems of ordinary differential

---

*Note: The Lambda Locus corresponding to $\Lambda(\lambda)$ consists of isolated points corresponding to the zeros of $\Lambda(\lambda)$.

equations one desires asymptotic stability in a "large" region of the left half of the complex λ-plane. The concept of stiff stability includes this attribute. Stiff stability may be defined as in Definition 3.6 [BIC72], which follows.

<u>Definition 3.6:</u> Let S_γ be an open connected region of the extended complex λ-plane such that

i) $\{\lambda: \ \mathrm{Re}\{\lambda\} < \gamma \leq 0\} \subset S_\gamma$.

ii) $\lambda = 0 \in \bar{S}_\gamma$.

Then the composite linear multistep method (3.2) is said to be <u>stiffly stable</u> if all solutions of (3.2) tend to zero as $n \to \infty$ when the method is used to integrate the differential equation

$$\dot{y}(t) = q\, y(t),$$

with any fixed positive h, such that

$$qh = \lambda \in S_\gamma.$$

If a method is stiffly stable and if $\{\lambda: \ \lambda$ is real and negative$\} \subset S_\gamma$ then it is $A(\alpha)$-stable for some $\alpha \in (0,\pi/2]$. Moreover, a method is A-stable if it is stiffly stable for all $\lambda \in S_0$, i.e., the method is stiffly stable with $\gamma = 0$ in Definition 3.6.

The development of Section 3.2 permits one to determine the stability boundaries of a CLMM and hence determine if it is stiffly stable or A-stable. The method may be summarized as

<u>Procedure 3.7</u>: Given a composite linear multistep method and its characteristic polynomial $p(\zeta,\lambda)$, the mapping of the unit circle in the ζ-plane into the λ-plane defined by (3.13) is applied, dividing the extended complex λ-plane into disjoint regions. (At most $\ell+1$ such regions exist.) If a region wherein the CLMM could be stiffly stable exists, an interior point is chosen, say λ_o. The k zeros $\zeta_i(\lambda_o)$ of $p(\zeta,\lambda_o)$ are determined and, if $|\zeta_i(\lambda_o)| < 1$ for $i = 1, \ldots, k$, then the method is stiffly stable. If, in addition, L_λ is contained within this region, the method is A-stable.

In applying the mapping (3.13), one must be careful to exclude the points in the λ-plane for which the zeros $\zeta_i(\lambda)$ of $p(\zeta,\lambda)$ are ill-defined from the region S_γ.*

Procedure 3.7 has been implemented in a computer program for determining the Lambda Locus of a method. Note: Definition 3.6 requires $\lambda = 0$ to be in $\bar{S}_\gamma$ and implies $\lambda = \infty$ is in $\bar{S}_\gamma$. Consequently as an aid in determining if a CLMM is stiffly stable the program solves for the zeros of $p_\ell(\zeta)$ and $p_o(\zeta)$, corresponding to $\{\zeta_i(\infty)\}$ and $\{\zeta_i(0)\}$, respectively.

---

*These points are the zeros of $\Lambda(\lambda)$. Each of these points, if any, must be determined and excluded from S_γ.

For $\lambda = 0$, $p(\zeta,\lambda)$ must have a zero at $\zeta = 1$;* thus the point $\lambda = 0 \notin S_\gamma$. As $|\zeta_i(\lambda)|$, $i=1, \ldots, k$, is a continuous function of λ, if the method is stiffly stable $\zeta_i(0)$ and $\zeta_i(\infty)$ cannot have modulus greater than unity.† If for some i this condition is not met then the method cannot be stiffly stable. Moreover, if $|\zeta_i(\infty)| < 1$ and a region does exist

---

*If the CLMM is to be at least of order zero, then each row of the matrix

$$A = \sum_{i=-K}^{0} A_i$$

must sum to zero; hence,

$$\det \left\{ \sum_{i=-K}^{0} A_i \zeta^{K+i} \right\} = p(\zeta,0) = p_o(\zeta)$$

must have a zero at $\zeta = 1$.

†An even stronger statement can be made as follows: The modulus of all roots of $p_\ell(\zeta) = 0$ and $p_o(\zeta) = 0$ must be bounded by 1 and the roots of modulus 1 must be "simple" [HEN62]. If $p_\ell(\zeta)$ or $p_o(\zeta)$, say $p_o(\zeta)$, has a zero of modulus 1 with multiplicity greater than 1 the minimal polynomial of $R(\zeta,\lambda)$, $\chi(\zeta,\lambda)$, must be examined at $\lambda = 0$. If $\chi(\zeta,0) = p_o(\zeta)$ the method cannot be stiffly stable. However, if $\chi(\zeta,0) \neq p_o(\zeta)$ the invariant polynomials $\iota_j(\zeta,\lambda)$, $j=1, \ldots, \ell$, as defined by (2.33), must be examined at $\lambda = 0$. If for all j the zeros of $\iota_j(\zeta,0)$ have multiplicity 1 then the method may be stiffly stable. Otherwise, the method cannot be stiffly stable. Note: $\chi(\zeta,\lambda) = \iota_1(\zeta,\lambda)$ and $\iota_{j+1}(\zeta,\lambda) | \iota_j(\zeta,\lambda)$ for $j=1, \ldots, \ell-1$.

such that the method might be stiffly stable (that is, a connected region exists, say S, such that $\lambda = 0 \in \bar{S}$ and $\lambda = \infty \in S$), then the method is stiffly stable and the zeros $\zeta_i(\lambda)$ of $p(\zeta,\lambda)$ for an additional interior point of the region need not be checked. This follows directly from Corollary 3.4 and the fact that in this instance $\lambda = \infty$ is an interior point.

3.4 The Zeta Locus

Since the characteristic polynomial of a composite linear multistep method is a polynomial* in ζ and λ, it may be written as

$$p(\zeta,\lambda) = q_k(\lambda)[\zeta-g_1(\lambda)][\zeta-g_2(\lambda)]\cdots[\zeta-g_k(\lambda)], \qquad (3.26)$$

where each $g_i(\lambda)$ is a well defined algebraic function of λ, analytic except at, at most, a finite number of isolated singularities. Denoting these singular points as d_m, they are either the zeros of the resultant of $p(\zeta,\lambda)$ and $\partial[p(\zeta,\lambda)]/\partial\lambda$ or the zeros of $q_k(\lambda)$. Let C_λ be the contour in the λ-plane shown in Figure 3.1. As the point R on the

---

*As was in the discussion with the Lambda Locus, it is assumed that the characteristic polynomial is irreducible. If it is not, then $p(\zeta,\lambda)$ can be written as the product of irreducible polynomials and the discussion applies to each of the irreducible polynomials that is not a function of λ or ζ alone. The special case wherein one of the irreducible factors may be a function of λ or ζ alone is considered later.

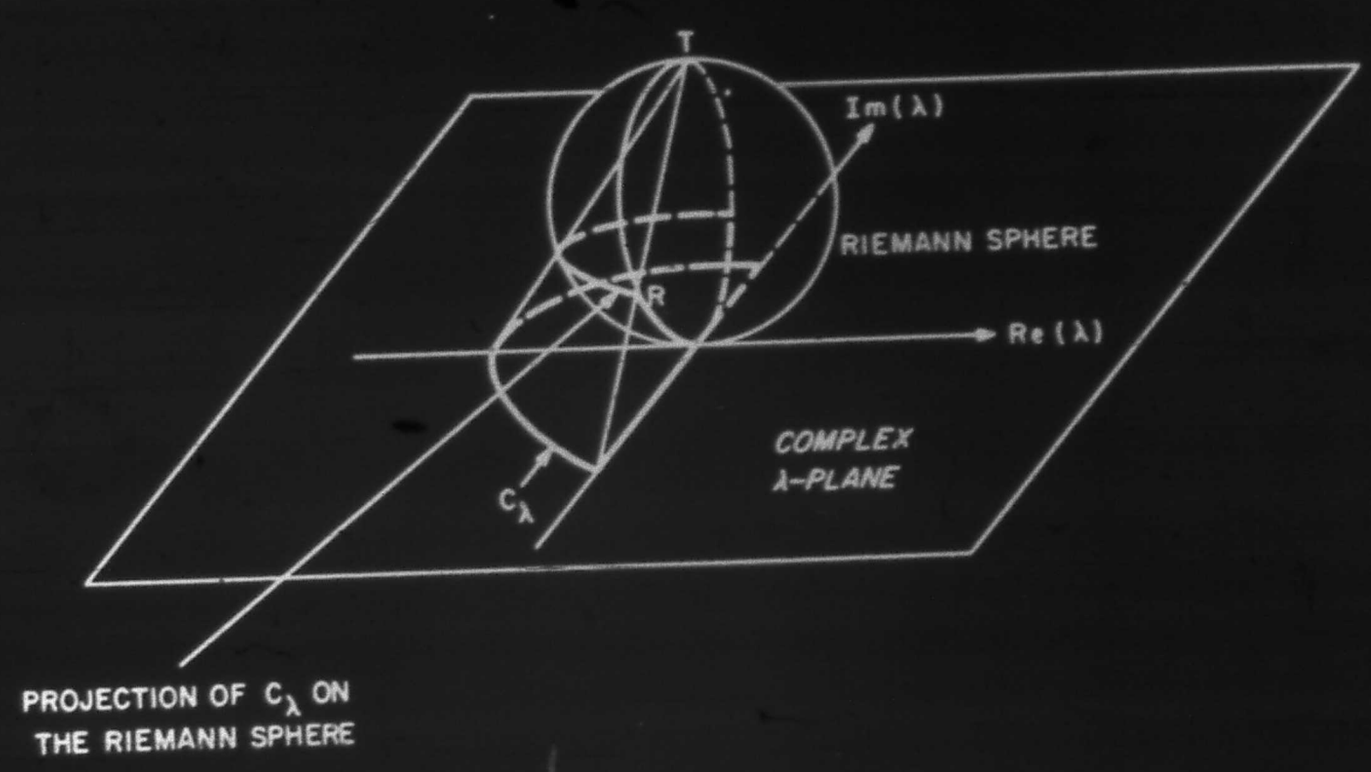


Figure 3.1: Definition of the Contour C_λ

Riemann sphere of Figure 3.1 moves towards the point T, C_λ completely encloses the open left half of the complex λ-plane.

Now, consider

$$p(\zeta,\lambda) = 0, \ \lambda \in C_\lambda. \tag{3.27}$$

As in Section 3.2, (3.27) defines k loci,

$$\zeta_i(\lambda) = g_i(\lambda) \text{ with } i = 1, \ldots, k \text{ and } \lambda \in C_\lambda, \tag{3.28}$$

in the complex λ-plane. The basis now exists for

<u>Definition 3.8</u>: The set of points

$$\Gamma(\zeta) = \{\zeta: \ \zeta = \zeta_i(\lambda), \ \lambda \in C_\lambda, \ i=1, \ldots, k\}, \tag{3.29}$$

for a given composite linear multistep method, is termed the <u>Zeta Locus</u>, or the <u>Zeta Plot</u>, of the method.

The Zeta Locus is a useful tool for checking if a method is A-stable.

<u>Theorem 3.9</u>: Let $U_\zeta = \{\zeta: \ |\zeta| < 1\}$. If the singular points of the characteristic polynomial of a composite linear multistep method, denoted as d_m, are such that $d_m \notin \overline{L}_\lambda$, then the composite linear multistep method is A-stable if and only if its Zeta Locus is contained within $\overline{U}_\zeta$.

Proof: The proof is a direct consequence of the Maximum Modulus Theorem of complex variables [AHL66]. Under the assumptions of the theorem the $g_i(\lambda)$'s are analytic for all $\lambda \in \bar{L}_\lambda$; hence, each assumes its maximum on the boundary. If for $\lambda \in C_\lambda$ $|\zeta_i(\lambda)| \leq 1$ then $|\zeta_i(\lambda)| < 1$ for all $\lambda \in L_\lambda$ and by definition the method is A-stable. On the other hand, if a method is A-stable, then for $\lambda \in L_\lambda$, $|\zeta_i(\lambda)| < 1$. Therefore, $|\zeta_i(\lambda)| \leq 1$ for $\lambda \in C_\lambda$, and the Zeta Locus must be contained within $\bar{U}_\zeta$. Q.E.D.

The Zeta Locus, by itself, does not give necessary and sufficient conditions for checking if a method is A-stable, since the location of the singular points must be determined and checked independently. Now, let $d_m^\wedge$ denote those singular points that are in the left half of the complex λ-plane, a situation in contradiction with the primary hypothesis of the previous theorem, and let $L_\lambda^* = L_\lambda - \{d_m^\wedge\}$. As each $d_m^\wedge$ is an isolated point, the $g_i(\lambda)$ are analytic everywhere for $\lambda \in L_\lambda^*$ and, therefore, each assumes its maximum on the boundary. A proof similar to that for the last theorem easily establishes

Procedure 3.10: Given a composite linear multistep method and its characteristic polynomial $p(\zeta,\lambda)$, the mapping of C_λ in the λ-plane into the ζ-plane defined by (3.27) is applied, thereby specifying the Zeta Locus $\Gamma(\zeta)$ for the method. If $\Gamma(\zeta) \subset \bar{U}_\zeta$, then the

singular points of $p(\zeta,\lambda)$ in the λ-plane, d_m, are determined. The k zeros $\zeta_i(d_m)$ of $p(\zeta,d_m)$ for each $d_m \in L_\lambda$ are determined and if all are contained in U, the method is A-stable.

At those singular points d_m such that $q_k(d_m) = 0$ there exists at least one index i such that $g_i(\lambda)$ contains an algebraic pole at $\lambda = d_m$, i.e., $g_i(\lambda)$ becomes unbounded as $\lambda \to d_m$.* Hence

<u>Theorem 3.11</u>: A necessary condition for a composite linear multistep method to be A-stable is that the zeros of $q_k(\lambda)$, as defined by (3.11), be in the open right half of the complex λ-plane.

Note: Singular points may exist in L_λ and the method could still be A-stable. However, they can only be branch points, that is, they cannot be algebraic poles, namely, zeros of $q_k(\lambda)$. Moreover, if the singular points are not checked,† erroneous information may be inferred from the Zeta Locus.§

---

*Though $g_i(\lambda)$ becomes unbounded, there exists a positive integer n such that $(\lambda-d_m)^n g_i(\lambda)$ remains bounded as $\lambda \to d_m$.

†Determining the singular points that are not zeros of $q_k(\lambda)$ is a non-trivial task in practice, though possible.

§As an example, the Zeta Locus, alone, for the backward difference composite linear multistep method proposed by Bickart and Picel [BIC72] would indicate that the 6-th order method is A-stable. The Lambda Locus does, however, indicate that it is only stiffly stable.

The Zeta Plot, as defined by (3.28), cannot yield any information as to whether or not a method is stiffly stable if it is not A-stable. However, a slight modification of the above formulation establishes

<u>Corollary 3.12</u>: Let $U_\zeta = \{\zeta: \ |\zeta| < 1\}$ and let L'_λ be an arbitrary open connected region of the extended complex λ-plane. If, for a given composite linear multistep method:

i) All singular points of its characteristic equation, denoted as d_m, are such that for $d_m \in \overline{L}'_\lambda$, $\zeta_i(d_m) \in U_\zeta$, $i=1, \ldots, k$.

ii) The k loci, $\zeta_i(\lambda) = g_i(\lambda)$ with $i=1, \ldots, k$ and $\lambda \in \partial L'_\lambda$ are contained within $\overline{U}_\zeta$.

Then the composite linear multistep method is asymptotically stable for all $\lambda \in L'_\lambda$. In particular, if L'_λ is chosen such that S_γ, of Definition 3.6, is contained within L'_λ and conditions i) and ii) are met then the method is stiffly stable.

In practice the Lambda Locus has been found more useful in determining if a method is stiffly stable, since Corollary 3.12 requires an à priori knowledge of the set S_λ, which is unknown at the outset.

It has been assumed that the characteristic polynomial is irreducible. If it is not then the results of this section apply to each of its irreducible factors. Each irreducible

factor and its associated Zeta Locus must be investigated as discussed in this section.

Excluded from the discussion have been polynomials which are functions of λ alone and ζ alone; i.e., it has been assumed that in (3.4) both $\Lambda(\lambda)$ = constant and $Z(\zeta)$ = constant. If $\Lambda(\lambda)$ is a non-constant polynomial and if $\{\lambda: \Lambda(\lambda) = 0\} \subset L_\lambda$ then the Zeta Locus will yield erroneous results. Note: At those points $\{\lambda: \Lambda(\lambda) = 0\}$ the zeros $\zeta_i(\lambda)$ of the characteristic polynomial $p(\zeta,\lambda)$ are ill-defined. An example of a method exhibiting this type of factorization was given in (3.22). As the Zeta Locus for that method consists of the isolated point $\zeta = 0$ and the unit circle in the ζ-plane one is tempted to assume that the method (3.22) is A-stable. However, as $\Lambda(\lambda)$ has a zero at $\lambda = -1 \in L_\lambda$ the method cannot be A-stable.

If $Z(\zeta)$ is a non-constant polynomial then the Zeta Locus corresponding to the $Z(\zeta)$ consists of isolated points, namely, the zeros, z_i, of $Z(\zeta)$. As long as $|z_i| \neq 1$ for all i the Zeta Locus may be interpreted correctly. However, if for some i, $|z_i| = 1$, a casual examination of the Zeta Locus may lead to erroneous conclusions. An example of this type of factorization is given in (3.30).*

---

*This example was offered by a colleague, W. Rubin of the Department of Electrical and Computer Engineering at Syracuse University.

$$-12y_{n-1}+24y_n-12y_{n+1} = h\{5\dot{y}_{n-1}+3\dot{y}_n-9\dot{y}_{n+1}+\dot{y}_{n+2}\}$$

$$8y_{n-1}-12y_n+4y_{n+2} = h\{-3\dot{y}_{n-1}-5\dot{y}_n+7\dot{y}_{n+1}+\dot{y}_{n+2}\} \tag{3.30}$$

The characteristic polynomial is

$$p(\zeta,\lambda) = \det\left\{\begin{bmatrix} -12(\zeta+1)-\lambda(-9\zeta+5) & 24-\lambda(\zeta+3) \\ 8-\lambda(7\zeta-3) & 4(\zeta-3)-\lambda(\zeta-5) \end{bmatrix}\right\}, \tag{3.31a}$$

or

$$p(\zeta,\lambda) = -(\zeta-1)[(3-3\lambda+\lambda^2) - (3+3\lambda+\lambda^2)\zeta]. \tag{3.31b}$$

In the notation of (3.4),

$$Z(\zeta) = -(\zeta-1),$$
$$\hat{p}(\zeta,\lambda) = (3-3\lambda+\lambda^2) - (3+3\lambda+\lambda^2)\zeta, \tag{3.32}$$

and

$$\Lambda(\lambda) = 1.$$

The Zeta Locus associated with $\hat{p}(\zeta,\lambda)$ for $\lambda \in C_\lambda$ consists of the unit circle in the ζ-plane while the Zeta Locus corresponding to $Z(\zeta)$ consists of a single point, namely $\zeta = 1$. As the Zeta Locus for the method (3.30) is the unit circle and the isolated point $\zeta = 1$ one is tempted to assume that it is an A-stable method.* However, as (3.31) has a zero for all

*Furthermore, the singularities of the method are in the right half of the complex λ-plane.

λ for $\zeta = 1$ the method cannot be A-stable.

3.5 A Plotting Program

One of the primary objectives of the work reported herein was to obtain new and precise methods by which to characterize the stability properties of composite linear multistep methods. To that end the relationship of the Lambda and Zeta Loci to those stability properties has been established. To use those relationships in the search for "properly" stable CLMM's a computer program was written* for plotting both the Lambda Locus and the Zeta Locus of a CLMM. A block diagram showing the five segments of the program is given in Figure 3.2.

After program execution is begun control is passed immediately to the subroutine ACCEPT, which returns the characteristic polynomial in a doubly dimensioned array of real numbers, ICHAR.[†§] In particular, the characteristic

---

*The program was written for execution on an XDS SIGMA-5 computer. The program is in non-standard FORTRAN making full use of the power of XDS EXTENDED FORTRAN-IV. As an example, many arrays are indexed from zero rather than one as in standard FORTRAN-IV. Conversion to standard FORTRAN is readily made. Listings of all programs cited can be found in Appendix A.

[†]All computation is in double precision. On the XDS SIGMA-5 computer this corresponds to 56 binary bits of mantissa, or approximately 16.8 decimal digits of accuracy.

[§]All capitalized words correspond to actual program variable names.

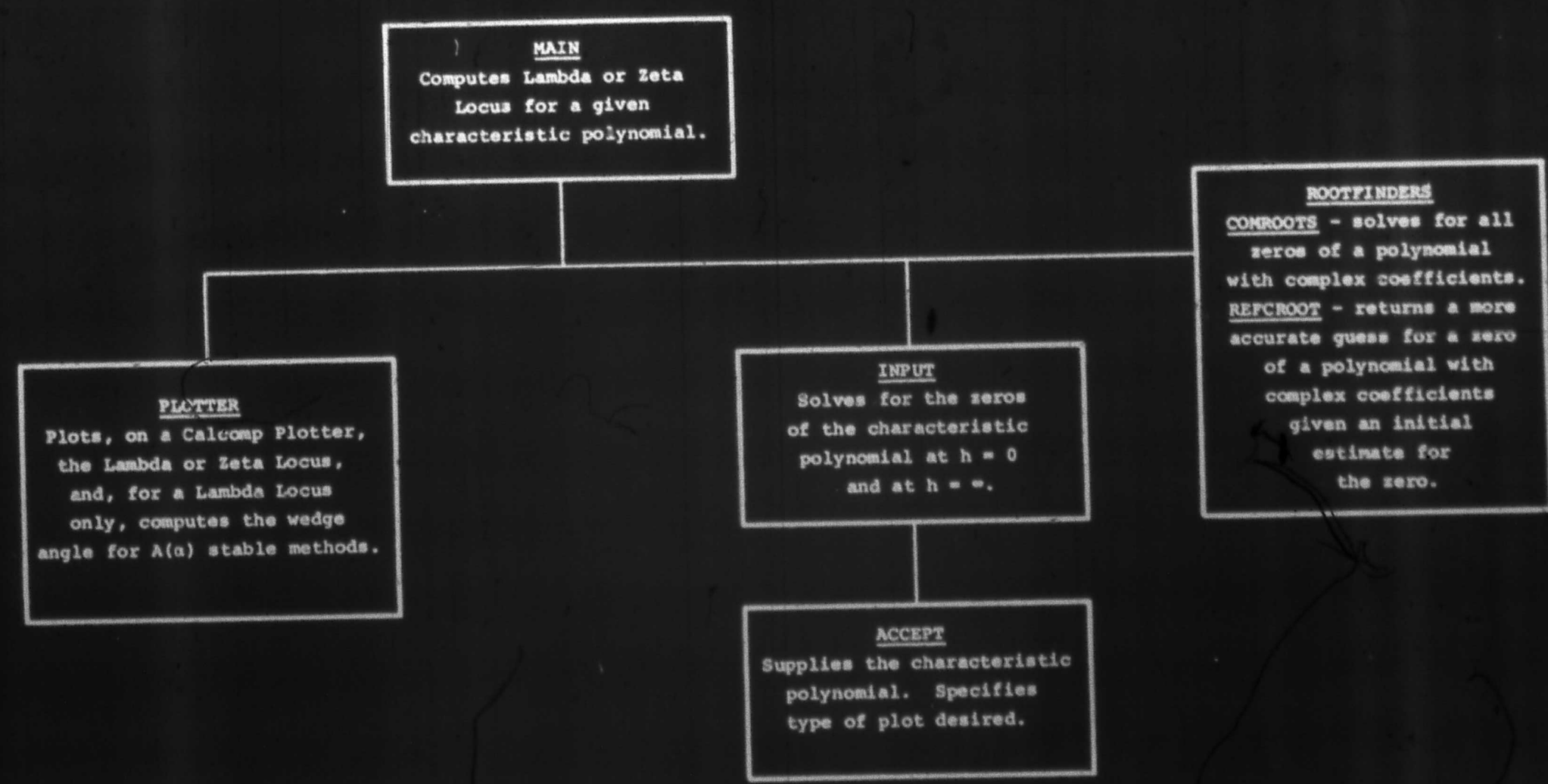


Figure 3.2: Plotting Program Organization

polynomial is

$$p(\zeta,\lambda) = \sum_{I=0}^{NL} \sum_{J=0}^{NZ} ICHAR(I,J)\,\zeta^{J}\lambda^{I}. \qquad (3.33)$$

For a CLMM consisting of ℓ equations and k back points,* NL = ℓ and NZ = k.

Though only one program is shown in Figure 3.2, there are in reality many programs. This is a consequence of the fact that numerous ACCEPT subroutines have been written. One version reads a CLMM from cards and computes the determinant as given by (3.3). Another version reads the characteristic polynomial directly from cards. Each returns the characteristic polynomial, for the moment assumed to be of maximum degree NL in λ and of maximum degree NZ in ζ. In addition an eight character alpha-numeric identification is supplied. This identification appears on all plots and printed output. The type of locus to be plotted, i.e., Lambda Locus, Zeta Locus or both, is also specified.

Control is then transferred to subroutine INPUT which prints the characteristic polynomial on a line printer and determines if the method is potentially stable by examining the zeros of the characteristic polynomial at $\lambda = 0$ and $\lambda = \infty$.

---

*This assumes that the factor $\zeta^{K\ell-k}$ has been removed from the characteristic polynomial and that all future points are retained. If all future points are not retained then NL $\leq \ell$ and NZ $\leq$ k. The program can accommodate this variation of a CLMM.

With the characteristic polynomial written as in (3.8), these zeros are, respectively the zeros of $p_o(\zeta)$ and $p_\ell(\zeta)$.* These zeros are computed and printed on the printer.

It is possible for a characteristic polynomial of a CLMM consisting of ℓ equations to be of degree $n < \ell$ in λ. An example of this phenomenon is found in the CLMM consisting of the forward Euler formula and the backward Euler formula applied cyclically:

$$y_{n+1} = y_n + h\dot{y}_n \qquad (3.34)$$

$$y_{n+2} = y_{n+1} + h\dot{y}_{n+2}$$

The characteristic polynomial is given by

$$p(\zeta,\lambda) = \det\left\{\begin{pmatrix} \zeta & -(1+\lambda) \\ -\zeta & \zeta(1-\lambda) \end{pmatrix}\right\}, \qquad (3.35a)$$

or

$$p(\zeta,\lambda) = \zeta[(-1+\zeta) - (1+\zeta)\lambda]. \qquad (3.35b)$$

Note: $p(\zeta,\lambda)$ is linear in λ even though the method consists

---

*As discussed in Section 3.3 these zeros are useful in determining if a method is stiffly stable.

of two equations.*

Subroutine INPUT checks for the possibility that $n < \ell$ in the process of determining the zeros at $h = \infty$. If the condition does exist, it decrements NL by one and uses $p_{\ell-1}(\zeta)$ to determine stability at $h = \infty$. The process is repeated as is necessary.

Control then returns to MAIN where the points comprising the loci, as defined by (3.13) for a Lambda Plot or by (3.27) for a Zeta Plot, are computed.

The MAIN program segment consists of three sections: initialization, start-up, and loci computation sections.

The initialization section, shown in flowchart form in Figure 3.3,† starts by reformatting the characteristic

---

*In fact the method (3.34) is A-stable as can be seen by setting $p(\zeta,\lambda) = 0$ and solving for ζ. Thus

$$\zeta = \frac{1+\lambda}{1-\lambda}$$

(The constant zero at $\zeta = 0$ in no way affects stability and may be ignored.) It is readily verified that $|\zeta| < 1$ for $Re\{q\} < 0$; in fact the characteristic equation is the same as that for the second order trapezoidal rule which is well known as being A-stable. However, this method is just of order 1. But of more interest is the fact that one of the equations is explicit. Hence, an implicit equation need be solved only every other time. The possibility of intermixing implicit and explicit equations to form a CLMM is an interesting concept worthy of further research.

†All statement numbers refer to actual statement numbers in the program, a listing of which can be found in Appendix A.

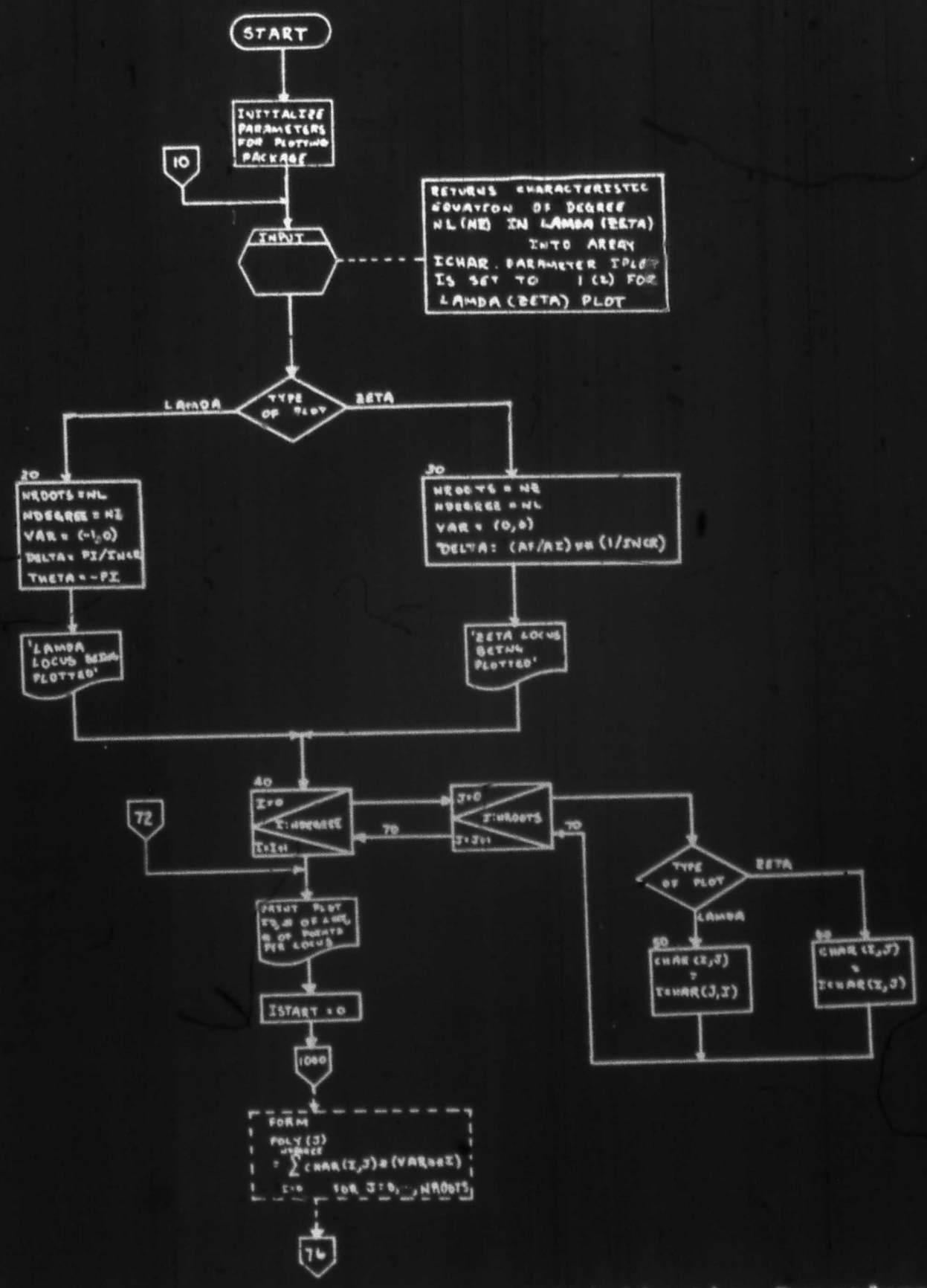


Figure 3.3: Main Program—Initialization Procedure

polynomial from the array ICHAR to a second doubly dimensioned array CHAR. The array CHAR defines a polynomial of degree NROOTS whose j-th row* is the coefficient of ζ^j [λ^j] for a Zeta [Lambda] Plot. Thus the entries of each row determine a polynomial of degree NDEGREE in λ [ζ] for a Zeta [Lambda] Plot. NROOTS and NDEGREE are set to NL and NZ, respectively, for a Lambda Plot and to NZ and NL, respectively, for a Zeta Plot. The parameter VAR, a complex variable, is initialized. VAR assumes the values $e^{j\theta}$ for a Lambda Plot and $j\omega$ ($\lambda = \sigma + j\omega$ with $\sigma = 0$) for a Zeta Plot.

During the start-up procedure, shown in flow chart form in Figure 3.4, the starting values for each of the NROOTS loci are computed.

For a Lambda Plot θ is initially set to $-\pi$. As only half the locus need be computed,† θ is varied from $-\pi$ to 0.

---

*Indexing starts at zero.

†The locus for θ varying from $0 \to \pi$ is the complex conjugate of the locus corresponding to θ varying from $0 \to -\pi$ and hence need not be computed. This can be seen as follows: The characteristic polynomial can be written as

$$p(\zeta,\lambda) = \sum_{j=0}^{k} \sum_{i=0}^{\ell} a_{ij}\lambda^i\zeta^j,$$

where the a_{ij}'s are real numbers. Now, with ~ denoting complex conjugate, consider

$$p(\tilde{\zeta},\tilde{\lambda}) = \sum_{j=0}^{k} \sum_{i=0}^{\ell} a_{ij}\tilde{\lambda}^i\tilde{\zeta}^j.$$

Figure 3.3: Main Program—Initialization Procedure

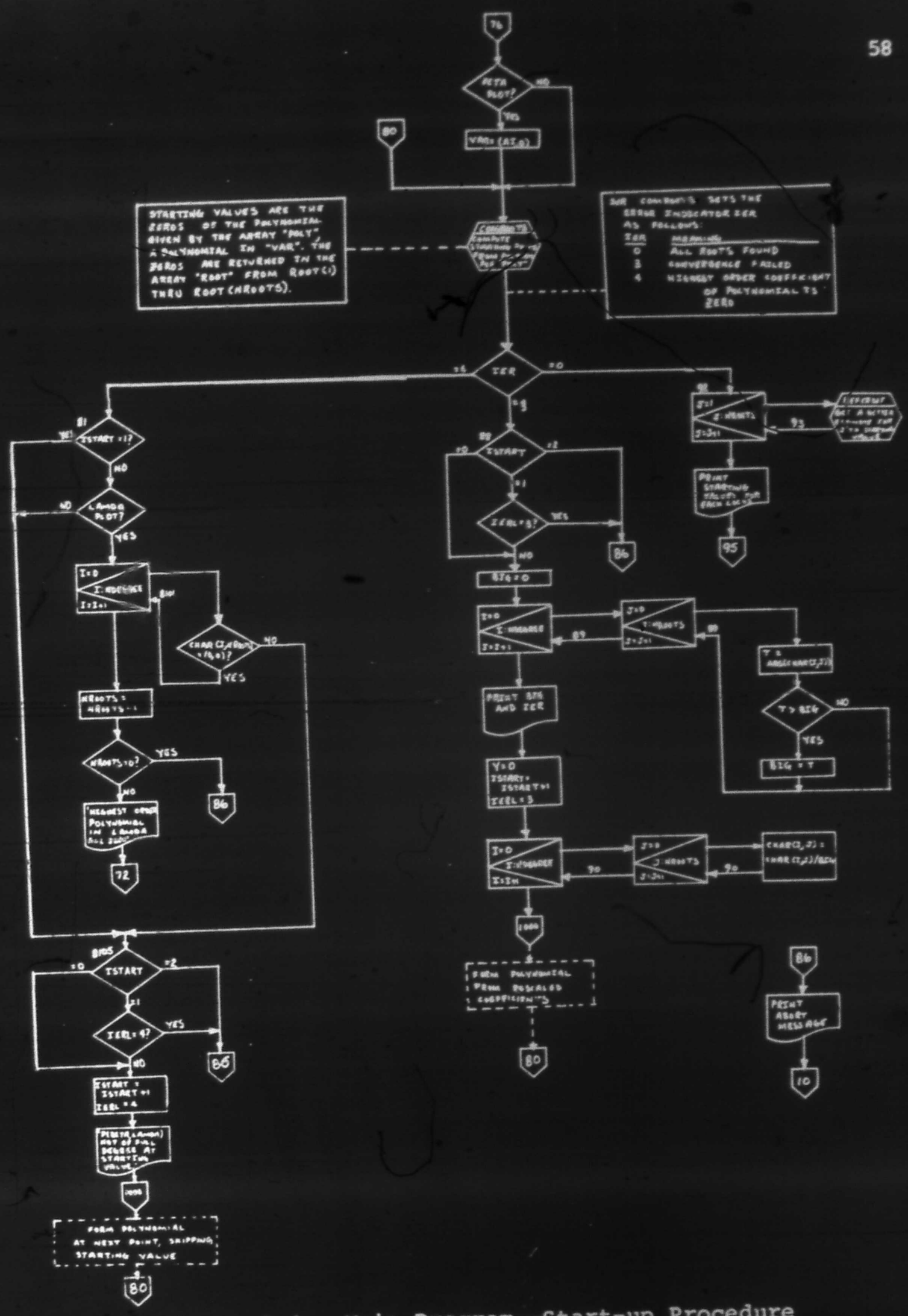


Figure 3.4: Main Program—Start-up Procedure

At each new plot point θ is incremented by π/INCR.* For a Zeta Plot ω is varied from 0 to AF† (chosen as 100§) in a geometric manner as follows: The start-up value is always chosen as $\omega = 0$. After the initial point for each locus is computed ω is set to AI (chosen as 0.1). At each new plot point, ω_{n+1} is chosen as

---

As $\tilde{a}_{ij} = a_{ij}$

$$p(\tilde{\zeta},\tilde{\lambda}) = \overline{\left(\sum_{j=0}^{k}\sum_{i=0}^{\ell} a_{ij}\lambda^{i}\zeta^{j}\right)} = \overline{p(\zeta,\lambda)}.$$

Hence, if for a given value of ζ, say ζ_o, λ_o is a root of the characteristic equation, i.e., $p(\zeta_o,\lambda_o) = 0$, then $p(\tilde{\zeta}_o,\tilde{\lambda}_o) = \overline{0}$. This implies that λ_o is a root of $p(\tilde{\zeta}_o,\lambda) = 0$. As $e^{-j\theta} = \overline{(e^{j\theta})}$ the claim is proven.

*INCR is chosen as follows: A singly dimensioned complex array ROOTS (dimensioned at 1500) is used to store all points to be plotted. Because it is desirable to compute a maximum number of plot points, INCR is chosen as the largest integer satisfying (INCR+1) NROOTS $\leq$ 1500. Each locus, i.e., each $f_i(\zeta)$ or $g_j(\lambda)$ of (3.12) or (3.26) for a Lambda or Zeta Plot, respectively, is evaluated at INCR+1 values resulting in a plot comprising (2×INCR+1) NROOTS points.

†In an argument similar to that for a Lambda Plot only half the locus need be computed as the locus for ω varying from $0 \to -\infty$ is the complex conjugate of the locus corresponding to ω varying from $0 \to \infty$.

§It has been found that, in general, a value of $\omega = 100$ is close enough to ∞ so as not to adversely effect the appearance of the plot.

$$\omega_{n+1} = \text{DELTA} \times \omega_n, \tag{3.36a}$$

where

$$\text{DELTA} = (\text{AF/AI})^{(1/\text{INCR})}. \tag{3.36b}$$

Varying ω as indicated by (3.36), rather than linearly, has been found to yield better results in the smoothness of the plots.

For the complex value VAR, a polynomial is computed which is a function of only one variable—λ if it is a Lambda Plot; ζ if it is a Zeta Plot. This polynomial, of degree NROOTS, is formed in the complex array POLY, given as

$$\text{POLY}(J) = \sum_{I=0}^{\text{NDEGREE}} [\text{CHAR}(I,J)][\text{VAR}^I],$$

$$\text{with } J=0, \ldots, \text{NROOTS}. \tag{3.37}$$

This procedure is repeated at each plot point—value of VAR.

The zeros of this polynomial, whose coefficients are complex numbers, are initially computed by the rootfinder COMROOTS using a Newton-Rhapson iteration to extract one root at a time.* If the rootfinder indicates success each root

---

*All zeros are computed with equal accuracy as follows: Assume that a root of the polynomial p(x) has been found, say x_1. A Newton-Rhapson iteration is then performed on the reduced polynomial $p(x)/(x-x_1)$ to locate a second zero.

is further refined by a second rootfinder, REFCROOT. (This rootfinder also uses a Newton-Rhapson iteration and improves on the guess of the root locations* as given by COMROOTS.)

This value is then used as an initial guess for a Newton-Rhapson iteration on the initial polynomial, p(x), resulting in a more accurate value, say x_2. The process is repeated for the third zero, this time starting with the reduced polynomial $[p(x)/(x-x_1)]/(x-x_2)$ to obtain an initial guess for a Newton-Rhapson iteration on the original polynomial. This process is repeated until all zeros have been extracted. By returning to the original polynomial at each stage, the error incurred in determining one zero (due to round-off, etc.) does not effect the accuracy to which a second zero is determined. Logic to detect multiple zeros is also built into COMROOTS. Note: If two zeros of the polynomial, say x_1 and x_2, are close to each other, but not equal, the procedure outlined may assume that a multiple zero exists. Such a condition can occur if the intial guess for the Newton-Rhapson iteration on the original polynomial for the zero x_2 is "closer" to x_1 than to x_2. In this case, the procedure will erroneously yield x_1 as the zero. This, however, does not present any major problem. Multiple zeros are checked for in the locus computation segment and handled as a special case. This procedure is described later.

*Convergence is determined in the following manner: The (n+1)-st guess at the root is given as

$$x_{n+1} = x_n + \delta x_n,$$

where x_n is the best guess at the end of the n-th stage and $\delta x_n = -p(x_n)/p'(x_n)$, p(x) being the polynomial whose zeros are desired. The process is terminated when $||\delta x_n||/g(x_{n+1}) < EPS$ (set at 10^{-10}), where

$$g(x) \equiv \begin{cases} ||x||, & ||x|| > 1 \\ 1, & ||x|| < 1 \end{cases},$$

and

$$||x|| \equiv |Re\{x\}| + |Im\{x\}|.$$

Success at this stage—namely, all roots found—results in being transferred to the last section, locus computation, of the program.

If COMROOTS failed to converge at the starting value of VAR and the highest order coefficient is non-zero (corresponding to IER = 3 in the flow chart) an attempt is made to rescale the characteristic equation. All coefficients of the array CHAR are divided by the modulus of the largest coefficient in CHAR. The polynomial (3.37) is then recomputed and the process repeated. If it fails a second time, the program aborts.

If COMROOTS failed to converge and the highest order coefficient, POLY(NROOTS), was zero (corresponding to IER = 4 in the flow chart), it is assumed that the point is an algebraic pole—a zero of $p_\ell(\zeta)$ or $q_k(\lambda)$ for a Lambda Plot or a Zeta Plot, respectively. In this case, a message is printed and the point is omitted from the plot. The next plot point is chosen as the starting point and the process repeated. If the same condition occurs a second time* the program aborts.

---

*As the poles are isolated singularities the condition will re-occur only due to numerical noise in the evaluation of POLY(NROOTS) or the unlikely possibility that a second pole has been found at the particular spacing of the plot point spacing.

It has been found beneficial to know the extent of each branch of the locus.* Hence, the starting values, as well as the mid-point values† are printed.

In the last part of the MAIN segment, shown in flow chart form in Figure 3.5, the variable VAR is set to its next value, as previously discussed, and the polynomial whose coefficients are given by (3.37) is re-computed. For each branch of the locus, the zeros of the polynomial at the previous setting of VAR are used as an initial guess to the zeros of the polynomial at the current setting of VAR. However, since it is possible that VAR may assume the value of a branch point,§ a check is made to see if two (or more) of the previous zeros coincided.‡ If they did, the zeros of the polynomial at the new setting of VAR are all recomputed using COMROOTS.** In addition, in an array ILOCUS, the

---

*This also aids in determining if $\Lambda(\lambda)$ or $Z(\zeta)$ in (3.4) are non-constant polynomials.

†As only half of the locus is computed, the last point computed is the mid-point of each branch.

§If an algebraic pole is encountered in this phase of the plotting, the program aborts.

‡This condition can also occur if two (or more) irreducible polynomials assume the same value for a given setting of VAR. The program handles this situation in a like manner.

**If this procedure were not followed then REFCROOT would fail to distinguish one branch from another and one locus, at least, would disappear from the plot. Hence all zeros are recomputed anew.

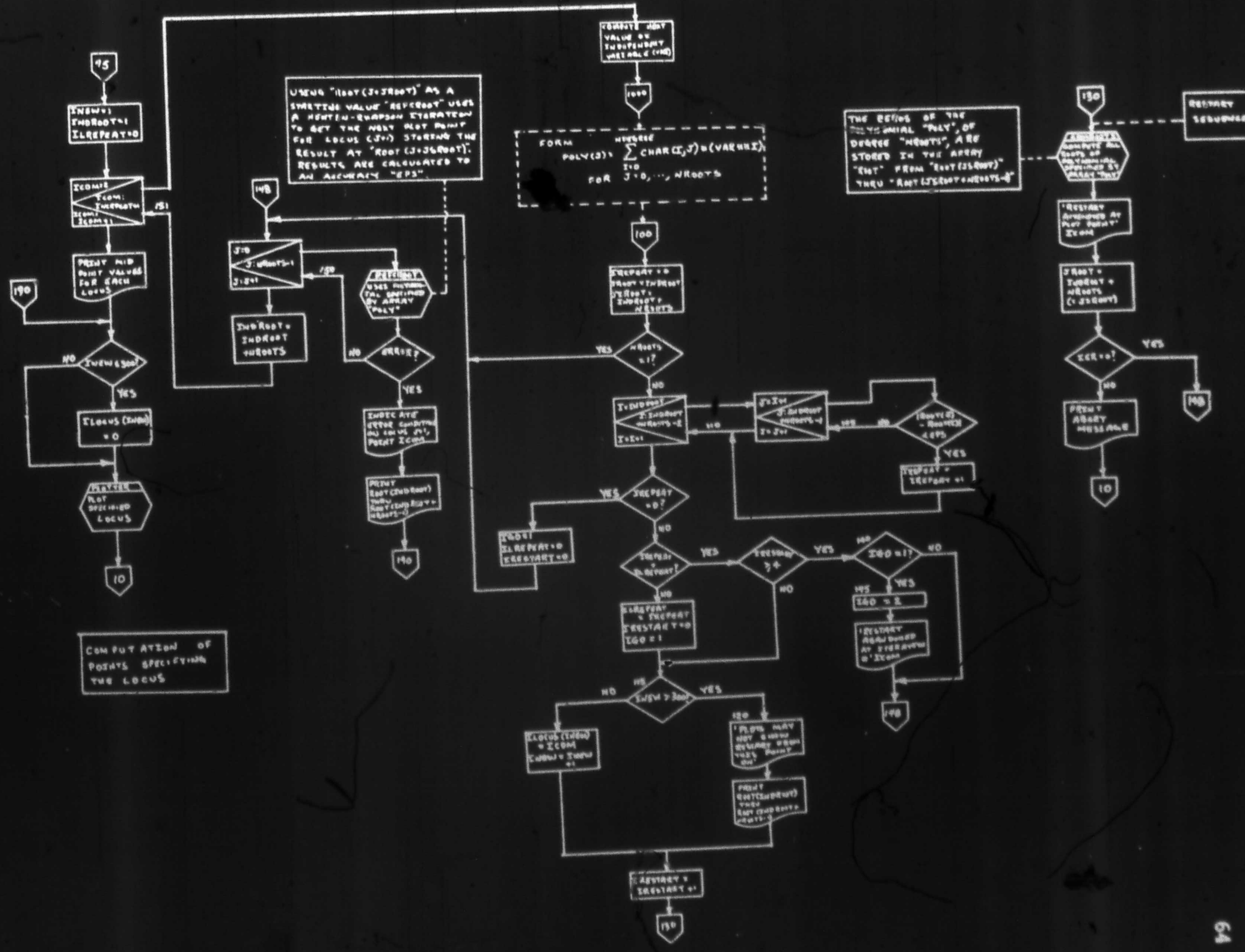


Figure 3.5: Main Program—Locus Computation

current plot point is stored, so that the plotting routine will have the information needed to lift the pen at this point.

The points comprising each locus are stored in the complex array ROOTS. The J-th locus (J=1, ..., NROOTS) is determined by the points stored at ROOTS(J+(ICOM-1)NROOTS), for ICOM=1, ..., INCR+1. Once computed the subroutine PLOTTER plots these points. For each locus the pen is dropped at ROOTS(J) and a continuous line is drawn through the points ROOTS(J+(ICOM-1)NROOTS), for ICOM=1, ..., INCR+1. The pen is then lifted, moved to ROOTS(J+(ICOM-1)NROOTS), dropped, and a continuous line drawn through the points ROOTS(J+(ICOM-1)NROOTS), for ICOM=INCR+1, ..., 1, completing the locus for the J-th root. This process is repeated for each locus—J=1, ..., NROOTS—except at those plot points as specified by the array ILOCUS. At these points, for each J the pen is lifted, moved to the next point, dropped, and the process is continued.

In addition, subroutine PLOTTER automatically chooses axes scalings that best suit the data. For a Zeta Locus, the unit circle is plotted as a reference. If the Zeta Plot is completely contained within the unit circle, axes limits of ± 1 are chosen. If it is not, then axes limits of ± 1.5 are chosen. If the Lambda Plot indicates the method is A-stable the minimum negative real axis limit is set to zero.

For Lambda Plots subroutine PLOTTER also computes the angle α for $A(\alpha)$-stability in accordance with Definition 2.3. The value γ for the region S_γ of Definition 3.6 is also computed and printed. The automatic scaling can be overridden and overplotting—one Lambda Plot on top of a previous one—accomplished.

These features have been used extensively in conjunction with an interactive version of subroutine ACCEPT used to vary certain "free parameters" in order to "optimize" the Lambda Plot. The composite linear multistep methods presented in Chapter IV were derived by using these properties.

The plotting program listed in Appendix A is limited to a characteristic polynomial of maximum degree 10 in both λ and ζ. This limitation is arbitrary.

CHAPTER IV

A NEW INTEGRATION PROCESS

4.1 Free Parameters

A composite linear multistep method may be written as

$$\sum_{j=-k+1}^{\ell} (\alpha_{i,j} y_{n\ell+j} - h\beta_{i,j} \dot{y}_{n\ell+j}) = 0, \quad \text{with } i=1, \ldots, \ell, \qquad (4.1)$$

or, in matrix notation, as

$$\{A_o \underset{\sim}{Y}_n + A_{-1} \underset{\sim}{Y}_{n-1} + \cdots + A_{-K} \underset{\sim}{Y}_{n-K}\}$$
$$-h\{B_o \dot{\underset{\sim}{Y}}_n + B_{-1} \dot{\underset{\sim}{Y}}_{n-1} + \cdots B_{-K} \dot{\underset{\sim}{Y}}_{n-K}\} = \underset{\sim}{0}, \qquad (4.2)$$

where A_{-j}, B_{-j}, $\underset{\sim}{Y}_{n-j}$, $\dot{\underset{\sim}{Y}}_{n-j}$ and K are as defined in (2.27). Each of the ℓ equations (4.1) has $2k + 2\ell$ coefficients. However, as they must constitute ℓ independent equations, only $2k + \ell$ coefficients are arbitrary. This may be seen as follows:

If (4.2) is to be at least of order zero, a solution for $\underset{\sim}{Y}_n$ must exist when it is used to numerically integrate the differential equation $\dot{y} = 0$, i.e.,

$$A_o \underset{\sim}{Y}_n + A_{-1} \underset{\sim}{Y}_{n-1} + \cdots + A_{-K} \underset{\sim}{Y}_{n-K} = \underset{\sim}{0} \qquad (4.3)$$

must have a solution for $\underset{\sim}{Y}_n$, or, equivalently, A_0^{-1} must exist. Pre-multiplying (4.2) by A_0^{-1} results in

$$\{\underset{\sim}{Y}_n + \hat{A}_{-1}\underset{\sim}{Y}_{n-1} + \cdots + \hat{A}_{-K}\underset{\sim}{Y}_{n-K}\}$$
$$-h\{\hat{B}_0\dot{\underset{\sim}{Y}}_n + \hat{B}_{-1}\dot{\underset{\sim}{Y}}_{n-1} + \cdots + \hat{B}_{-K}\dot{\underset{\sim}{Y}}_{n-K}\} = \underset{\sim}{0}, \quad (4.4)$$

where $\hat{A}_{-j} \equiv A_0^{-1}A_{-j}$, $\hat{B}_{-j} \equiv A_0^{-1}B_{-j}$, and $\hat{A}_0 \equiv I$, the identity matrix.* Hence each of the equations (4.1) only has $2k + \ell$ arbritrary coefficients.

---

*A CLMM when represented as in (4.4) is said to be in canonical form, a term suggested by W. Rubin (Electrical and Computer Engineering Department, Syracuse University). Any given CLMM can be put into its canonical representation by pre-multiplying the former by A_0^{-1}. The characteristic equation of (4.4), $\chi(\zeta, \lambda)$, is given as,

$$\chi(\zeta, \lambda) = \det\{\sum_{j=0}^{K} (\hat{A}_{-j} - \lambda\hat{B}_{-j})\zeta^{K-j}\},$$

whereas the characteristic equation of (4.2), $p(\zeta, \lambda)$, is given as

$$p(\zeta, \lambda) = \det\{\sum_{j=0}^{K} (A_{-j} - \lambda B_{-j})\zeta^{K-j}\}.$$

As A_0^{-1} exists, the latter may be written as

$$p(\zeta, \lambda) = \det\{A_0[\sum_{j=0}^{K} (\hat{A}_{-j} - \lambda\hat{B}_{-j})\zeta^{K-j}]\}$$
$$= \det\{A_0\}\det\{\sum_{j=0}^{K} (\hat{A}_{-j} - \lambda\hat{B}_{-j})\zeta^{K-j}\},$$

or

$$p(\zeta, \lambda) = \det\{A_0\}\,\chi(\zeta, \lambda).$$

Hence, within a constant factor, the characteristic equation, and therefore, the stability properties of a method is the same whether or not it is represented in canonical form.

If one requires a composite linear multistep method to be of a given order p, then p+1 coefficients, per equation, are predetermined. Any remaining coefficients may then be used to improve "other properties" of the method. These free parameters, $2k + \ell - p - 1$ per equation, can be used to

i) improve the stability characteristics of the method;

ii) make it efficient as implemented in an integration algorithm; and

iii) lower the discretization error introduced at each stage of the process.

In the next section a numerical integration process consisting of 7 composite linear multistep methods, of order 1 through 7, is presented. Free parameters were adjusted to make each method stiffly stable and efficient.

4.2 Some New Cyclic Composite Linear Multistep Methods

As one of the primary objectives of this research effort was to derive processes that were efficient to use, the class of composite linear multistep methods investigated was cyclic, i.e., in (4.1) $\alpha_{i,j} = \beta_{i,j} = 0$ for $j > i$, or, equivalently, in (4.2) A_0 and B_0 were limited to being lower triangular relative to the skew diagonal.* Moreover, as

*As pointed out in Section 2.6, this avoids the necessity of simultaneously solving ℓ equations.

these methods were to be applicable to the solution of stiff systems the conditions $B_{-j} = 0$ for $j > 0$ and B_0 non-singular were imposed. These conditions, though not a necessity, were set in order to guarantee asymptotic stability at $h = \infty$. That this property is achieved can be seen as follows: As the step-size increases, the zeros of the characteristic polynomial tend to those of $p_\ell(\zeta)$, as can be seen from (3.8). Now, $p_\ell(\zeta)$ is given as

$$p_\ell(\zeta) = \det\{\sum_{j=0}^{K} B_{-j}\zeta^{K-j}\}. \tag{4.4}$$

Requiring $B_{-j} = 0$ for $j > 0$ yields

$$p_\ell(\zeta) = \det\{B_0\}\zeta^{K\ell}, \tag{4.5}$$

which, as B_0 is non-singular, is equivalent to setting all zeros of $p_\ell(\zeta)$ to zero. This guarantees stability at $h = \infty$, independent of the coefficient settings.

Under these conditions the first of the ℓ equations (4.1) becomes

$$\sum_{j=-k+1}^{1} \alpha_{1,j}\, y_{n\ell+j} - h\beta_{1,1}\, \dot{y}_{n\ell+1} = 0. \tag{4.6}$$

If (4.6) is to be of maximal order k the backward differentiation formula (BDF) results [HEN62]. By imposing the condition that all ℓ equations have at least order k and

further restricting each equation to span only (k+1) time points, for efficiency considerations, it follows that i-1, i=1, ..., ℓ, free parameters exist in the i-th equation of (4.1), each of which can be adjusted to improve the stability characteristics of the method.

Summarizing, the class of composite linear multistep methods considered in this research was for i=1, ..., ℓ, restricted to

[R1]: $\alpha_{i,j} = \beta_{i,j} = 0,\ j > i;$

[R2]: $\beta_{i,j} = 0,\ j \leq 0$ and $\beta_{i,i} \neq 0;$*

[R3]: $\alpha_{i,j} = 0,\ j < -k+i;$ and

[R4]: $p_i \geq k$, where p_i is the order of the i-th equation.

Conditions [R1] and [R3] lead to the cyclic methods proposed by Donelson and Hansen [DON71] and used by Chesler and Pierce [CHE71], and Dyer, Pierce, Haney, and Chesler [DYE72] to obtain numerical integration processes suitable for use in orbit computation. In their methods free parameters were adjusted to improve stability properties and obtain higher accuracy as $h \to 0$. None of these methods are stiffly stable.

In this research effort the criterion used for choosing

---

*As B_0 is restricted to a triangular matrix this is equivalent to requiring B_0 to be non-singular.

the free parameters of a method was: Maximize the Widlund wedge angle α of Definition 2.3. The backward differentiation formulas for orders 1 and 2 are, respectively,

$$y_{n+1} = y_n + h\dot{y}_{n+1}, \tag{4.7}$$

and

$$3y_{n+1} = 4y_n - y_{n-1} + 2h\dot{y}_{n+1}. \tag{4.8}$$

They are both A-stable processes and, therefore, no further improvement in α can result. The backward differentiation formulas for orders 3 through 6 are stiffly stable, though for $\alpha < \pi/2$. For orders 7 through 15 they are not stiffly stable [GEA71b].

If conditions [R1], [R3], and [R4] are first imposed, then it can be shown that there exist k+1 linearly independent multistep formulas for each i, from linear combinations of which can be composed ℓ linearly independent formulas of a CLMM satisfying condition [R2]. Sets of such linearly independent multistep formulas for k=3, ..., 9 are given in Table 4.1.

Using an interactive version of subroutine ACCEPT* in conjunction with the plotting program described in Section 3.5, various linear combinations of these formulas were

---

*A brief description and a listing can be found in Appendix B.

P=k=3

-2	-1	0	1	|	-2	-1	0	1	$j-i,\ j=k+i,\dots,i,\ i=1,\dots,\ell$
-11	18	-9	2	|	6	0	0	0	
-2	-3	6	-1	|	0	6	0	0	
1	-6	3	2	|	0	0	6	0	
-2	9	-18	11	|	0	0	0	6	

$\alpha_{i,j-i}$ $\beta_{i,j-i}$

P=k=4

-3	-2	-1	0	1	|	-3	-2	-1	0	1
-25	48	-36	16	-3	|	12	0	0	0	0
-3	-10	18	-6	1	|	0	12	0	0	0
1	-8	0	8	-1	|	0	0	12	0	0
-1	6	-18	10	3	|	0	0	0	12	0
3	-16	36	-48	25	|	0	0	0	0	12

P=k=5

-4	-3	-2	-1	0	1	|	-4	-3	-2	-1	0	1
-137	300	-300	200	-75	12	|	60	0	0	0	0	0
-12	-65	120	-60	20	-3	|	0	60	0	0	0	0
3	-30	-20	60	-15	2	|	0	0	60	0	0	0
-2	15	-60	20	30	-3	|	0	0	0	60	0	0
3	-20	60	-120	65	12	|	0	0	0	0	60	0
-12	75	-200	300	-300	137	|	0	0	0	0	0	60

P=k=6

-5	-4	-3	-2	-1	0	1	|	-5	-4	-3	-2	-1	0	1
-147	360	-450	400	-225	72	-10	|	60	0	0	0	0	0	0
-10	-77	150	-100	50	-15	2	|	0	60	0	0	0	0	0
2	-24	-35	80	-30	8	-1	|	0	0	60	0	0	0	0
-1	9	-45	0	45	-9	1	|	0	0	0	60	0	0	0
1	-8	30	-80	35	24	-2	|	0	0	0	0	60	0	0
-2	15	-50	100	-150	77	10	|	0	0	0	0	0	60	0
10	-72	225	-400	450	-360	147	|	0	0	0	0	0	0	60

Table 4.1: Linearly Independent Multistep Formulas for p=3, ..., 9

P=k=7

-6	-5	-4	-3	-2	-1	0	1	-6	-5	-4	-3	-2	-1	0	1
-1089	2940	-4410	4900	-3675	1764	-490	60	420	0	0	0	0	0	0	0
-60	-609	1260	-1050	700	-315	84	-10	0	420	0	0	0	0	0	0
10	-140	329	700	-350	140	-35	4	0	0	420	0	0	0	0	0
-4	42	-252	-105	420	-126	28	-3	0	0	0	420	0	0	0	0
3	-28	126	-420	105	252	-42	4	0	0	0	0	420	0	0	0
-4	35	-140	350	-700	329	140	-10	0	0	0	0	0	420	0	0
10	-84	315	-700	1050	-1260	609	-60	0	0	0	0	0	0	420	0
-60	490	-1764	3675	-4900	4410	-2940	1089	0	0	0	0	0	0	0	420

P=k=8

-7	-6	-5	-4	-3	-2	-1	0	1	-7	-6	-5	-4	-3	-2	-1	0	1
-2283	6720	-11760	15680	-14700	9408	-3920	960	-105	840	0	0	0	0	0	0	0	0
-105	-1338	2940	-2940	2450	-1470	588	-140	15	0	840	0	0	0	0	0	0	0
15	-240	-798	1680	-1050	560	-210	48	-5	0	0	840	0	0	0	0	0	0
-5	60	-420	-378	1050	-420	140	-30	3	0	0	0	840	0	0	0	0	0
3	-32	168	-672	0	672	-168	32	-3	0	0	0	0	840	0	0	0	0
-3	30	-140	420	-1050	378	420	-60	5	0	0	0	0	0	840	0	0	0
5	-48	210	-560	1050	-1680	798	240	-15	0	0	0	0	0	0	840	0	0
-15	140	-588	1470	-2450	2940	-2940	1338	105	0	0	0	0	0	0	0	840	0
105	-960	3920	-9408	14700	-15680	11760	-6720	2283	0	0	0	0	0	0	0	0	840

P=k=9

-8	-7	-6	-5	-4	-3	-2	-1	0	1	-8	-7	-6	-5	-4	-3	-2	-1	0	1
-7129	22680	-45360	70560	-79380	63504	-35280	12960	-2835	280	2520	0	0	0	0	0	0	0	0	0
-280	-4329	10080	-11760	11760	-8820	4704	-1680	360	-35	0	2520	0	0	0	0	0	0	0	0
35	-630	-2754	5880	-4410	2940	-1470	504	-105	10	0	0	2520	0	0	0	0	0	0	0
-10	135	-1080	-1554	3780	-1890	840	-270	54	-5	0	0	0	2520	0	0	0	0	0	0
5	-60	360	-1680	-504	2520	-840	240	-45	4	0	0	0	0	2520	0	0	0	0	0
-4	45	-240	840	-2520	504	1680	-360	60	-5	0	0	0	0	0	2520	0	0	0	0
5	-54	270	-840	1890	-3780	1554	1080	-135	10	0	0	0	0	0	0	2520	0	0	0
-10	105	-504	1470	-2940	4410	-5880	2754	630	-35	0	0	0	0	0	0	0	2520	0	0
35	-360	1680	-4704	8820	-11760	11760	-10080	4329	-280	0	0	0	0	0	0	0	0	2520	0
-280	2835	-12960	35280	-63504	79380	-70560	45360	-22680	7129	0	0	0	0	0	0	0	0	0	2520

Table 4.1: Linearly Independent Multistep Formulas for p=3, ..., 9 (continued)

tried for each of the equations (4.1) for different ℓ and orders 3 through 7,* subject to restriction [R2]. Table 4.2 gives the resulting composite linear multistep methods derived in this manner.† For p = 3 and 4 the new methods consist of three equations (ℓ=3); for p = 5, 6, and 7 the new methods consist of four equations (ℓ=4). Table 4.3 compares the Widlund wedge angle of these methods with that of the corresponding backward differentiation formula of the same order. Table 4.4 provides a corresponding comparison for the value γ for the region S_γ of Definition 3.6.

As can be seen from Table 4.3 the Widlund wedge angles for the new methods have been significantly improved over the corresponding ones for the backward differentiation formulas. Table 4.4 indicates that for all of the new methods presented the half plane, defined by the parameter γ of Definition 3.6, wherein a composite linear multistep method is asymptotically stable has been enlarged when compared to the corresponding backward differentiation formulas.

---

*An unsuccessful attempt was made to obtain a stiffly stable order 8 composite linear multistep method. Failure can be reasonably attributed to the inadequacy of the "manual" search of the free parameter settings and not to the non-existence of such a method.

†In Table 4.2—printed by subroutine ACCEPT—A(I,j) and B(I,j), j = -k+1, ..., ℓ, correspond to $\alpha_{i,j}$ and $\beta_{i,j}$, respectively, of (4.1).

P=k=3

I	1	2	3
A(I,-2)	-2	0	0
A(I,-1)	9	-2	0
A(I, 0)	-18	9	0
A(I, 1)	11	-18	9
A(I, 2)	0	11	-12
A(I, 3)	0	0	3
B(I,-2)	0	0	0
B(I,-1)	0	0	0
B(I, 0)	0	0	0
B(I, 1)	6	0	-4
B(I, 2)	0	6	-4
B(I, 3)	0	0	2

P=k=4

I	1	2	3
A(I,-3)	3	0	0
A(I,-2)	-16	3	0
A(I,-1)	36	-16	11
A(I, 0)	-48	36	-48
A(I, 1)	25	-48	216
A(I, 2)	0	25	-272
A(I, 3)	0	0	93
B(I,-3)	0	0	0
B(I,-2)	0	0	0
B(I,-1)	0	0	0
B(I, 0)	0	0	0
B(I, 1)	12	0	-60
B(I, 2)	0	12	-48
B(I, 3)	0	0	48

P=k=5

I	1	2	3	4
A(I,-4)	-12	0	0	0
A(I,-3)	75	-12	0	0
A(I,-2)	-200	75	-118	0
A(I,-1)	300	-200	735	-133
A(I, 0)	-300	300	-1940	780
A(I, 1)	137	-300	2980	-1680
A(I, 2)	0	137	-3030	5470
A(I, 3)	0	0	1373	-5595
A(I, 4)	0	0	0	1158
B(I,-4)	0	0	0	0
B(I,-3)	0	0	0	0
B(I,-2)	0	0	0	0
B(I,-1)	0	0	0	0
B(I, 0)	0	0	0	0
B(I, 1)	60	0	-60	30
B(I, 2)	0	60	0	-1860
B(I, 3)	0	0	600	-1530
B(I, 4)	0	0	0	600

Table 4.2: New Cyclic Composite Linear Multistep Methods

P=k=6

I	1	1	2	3	4
A(I,-5)	10	0	0	0	0
A(I,-4)	-72	202	0	0	0
A(I,-3)	225	-1455	195	0	
A(I,-2)	-400	4550	-1399	285	
A(I,-1)	450	-8100	4340	-2039	
A(I, 0)	-360	9150	-7540	6225	
A(I, 1)	147	-7277	8905	-10360	
A(I, 2)	0	2930	-7445	18455	
A(I, 3)	0	0	2944	-14865	
A(I, 4)	0	0	0	2299	
B(I,-5)	0	0	0	0	
B(I,-4)	0	0	0	0	
B(I,-3)	0	0	0	0	
B(I,-2)	0	0	0	0	
B(I,-1)	0	0	0	0	
B(I, 0)	0	0	0	0	
B(I, 1)	60	-60	-420	180	
B(I, 2)	0	1200	-60	-4080	
B(I, 3)	0	0	1200	-4680	
B(I, 4)	0	0	0	1200	

P=k=7

I	1	2	3	4
A(I,-6)	-60	0	0	0
A(I,-5)	490	-60	0	0
A(I,-4)	-1764	490	-210	0
A(I,-3)	3675	-1764	1722	-774
A(I,-2)	-4900	3675	-6235	6349
A(I,-1)	4410	-4900	13100	-22988
A(I, 0)	-2940	4410	-17650	48160
A(I, 1)	1089	-2940	17710	-66290
A(I, 2)	0	1089	-11297	68159
A(I, 3)	0	0	2860	-42364
A(I, 4)	0	0	0	9748
B(I,-6)	0	0	0	0
B(I,-5)	0	0	0	0
B(I,-4)	0	0	0	0
B(I,-3)	0	0	0	0
B(I,-2)	0	0	0	0
B(I,-1)	0	0	0	0
B(I, 0)	0	0	0	0
B(I, 1)	420	0	-600	840
B(I, 2)	0	420	-1860	-2100
B(I, 3)	0	0	1200	-8400
B(I, 4)	0	0	0	4200

Table 4.2: New Cyclic Composite Linear Multistep Methods (continued)

ORDER	NEW METHOD	BDF
3	89.43°	86.03°
4	80.88°	73.35°
5	77.48°	51.84°
6	63.25°	17.84°
7	33.53°	*

*The order 7 BDF is not stiffly stable.

Table 4.3: Comparison of Widlund Wedge Angles

ORDER	NEW METHOD	BDF
3	-0.0048	-0.083
4	-0.24	-0.67
5	-1.4	-2.3
6	-2.9	-6.1
7	-10.2	*

*The order 7 BDF is not stiffly stable.

Table 4.4: Comparison of γ-values.

The effect of these improvements and the introduction of a new order seven method is to permit a numerical integration process based on these new methods to increase the integration step-size at a higher order much more rapidly than processes based on the backward differentiation formulas, for a larger class of problems. This effect will be especially evident in stiff problems wherein the eigenvalues of the Jacobian matrix are complex.

The Lambda Loci for the backward differentiation formulas of orders 1 and 2 are shown in Figure 4.1.* In Figure 4.2 the Lambda Loci for the new methods, as well as those for the corresponding backward differentiation formulas of the same order, are shown. The Lambda Locus for the new order 7 method is shown in Figure 4.3. The effect of the optimization criterion used in selecting the free parameters—that is, maximize the Widlund wedge angle—is especially apparent in the Lambda Loci for the orders 5, 6, and 7 methods. The "humps" in the Lambda Loci for the new methods are almost all tangential to the rays forming the boundary of the set S_α of Definition 2.3.

These new methods, together with the backward

---

*An identifier consisting of 2 letters and 3 digits, is used to label each plot. The three digits are, respectively, the values for p, ℓ, and k corresponding to the methods for which the Lambda Loci were plotted.

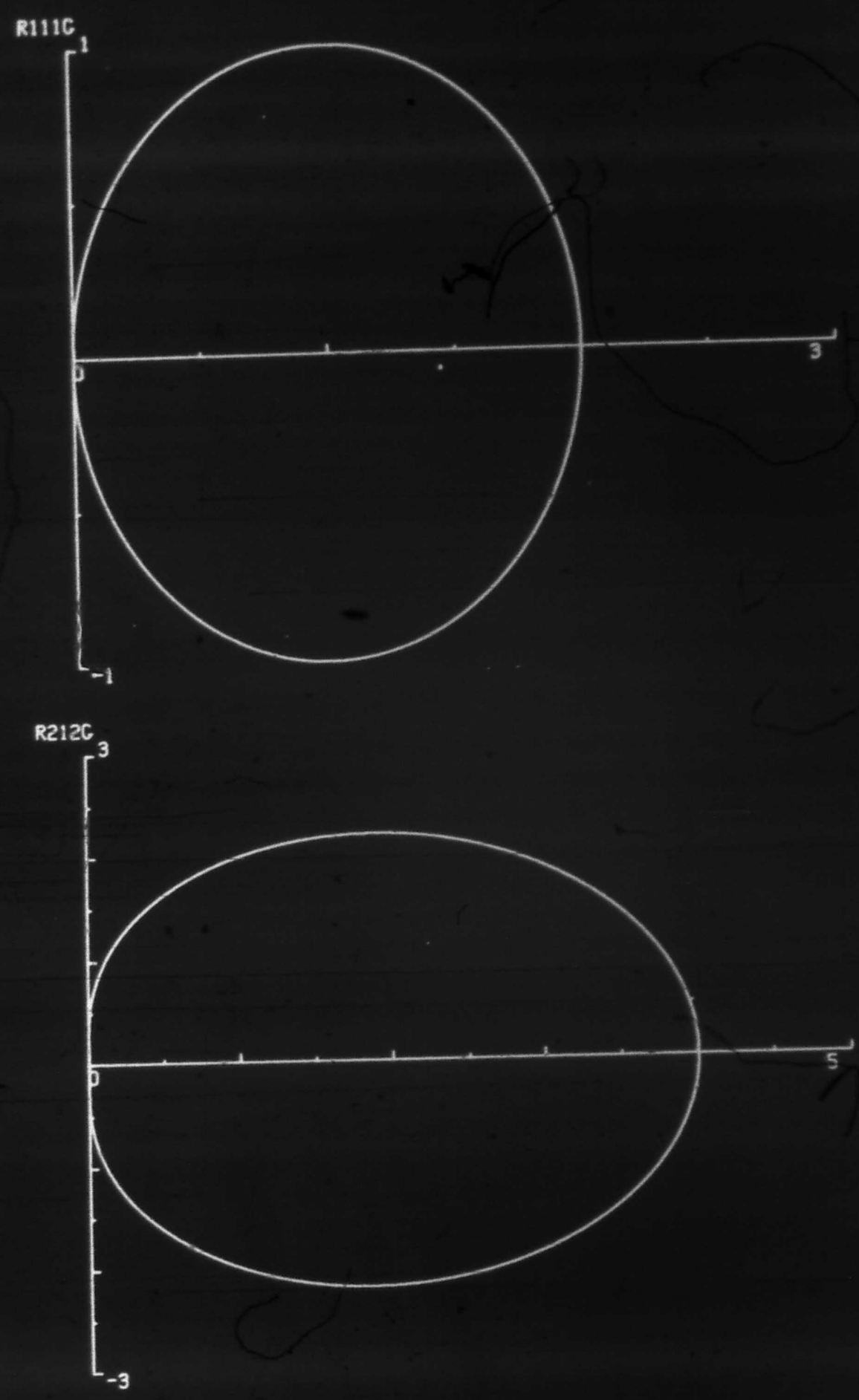
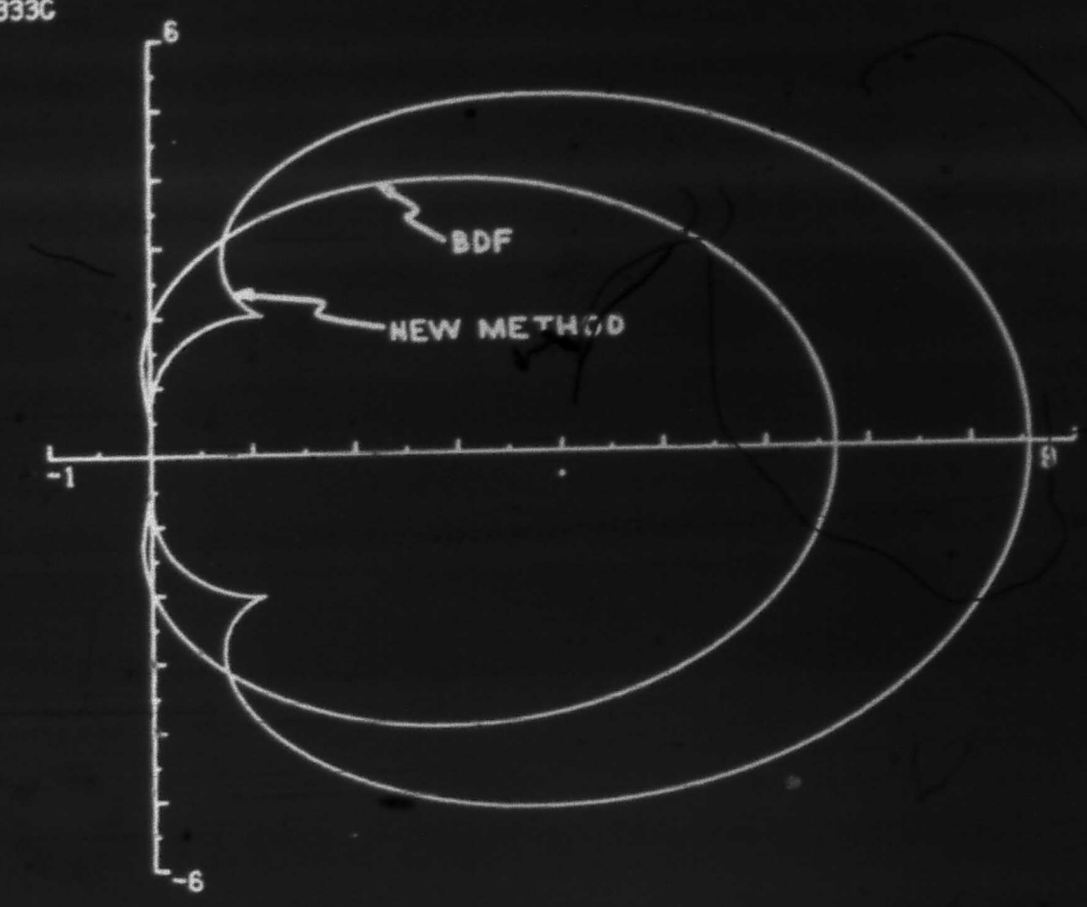


Figure 4.1: Lambda Loci for Order 1 and 2 Methods



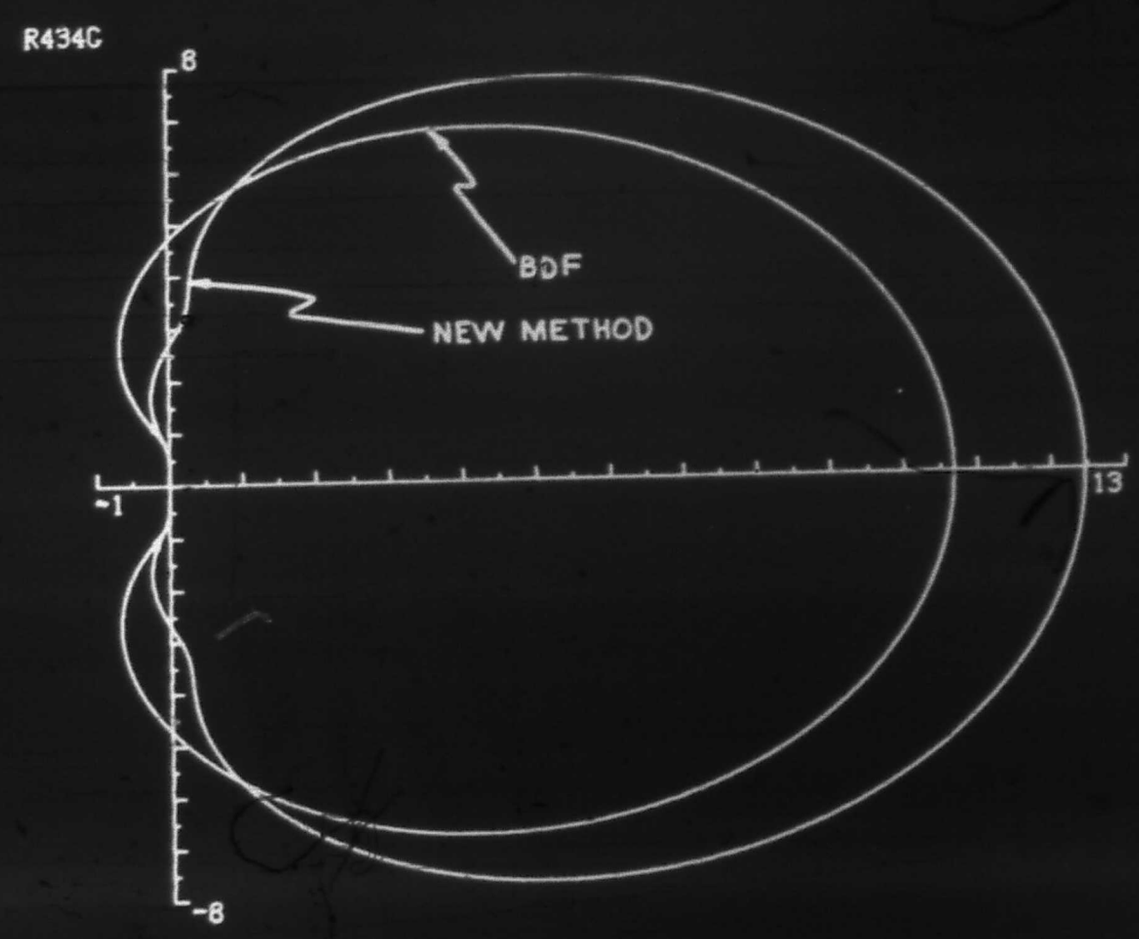
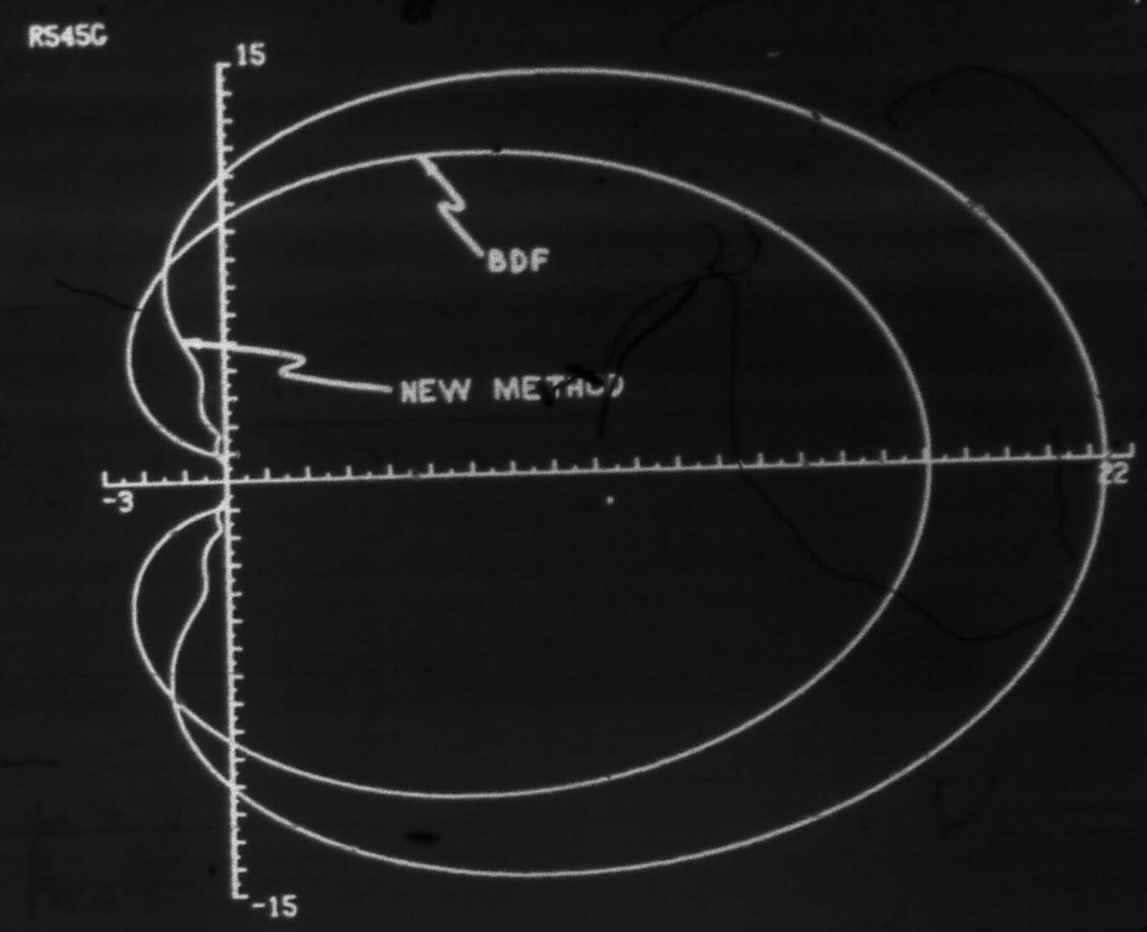


Figure 4.2: Comparison of Lambda Loci Between New Methods and BDF for Order Three Through Six



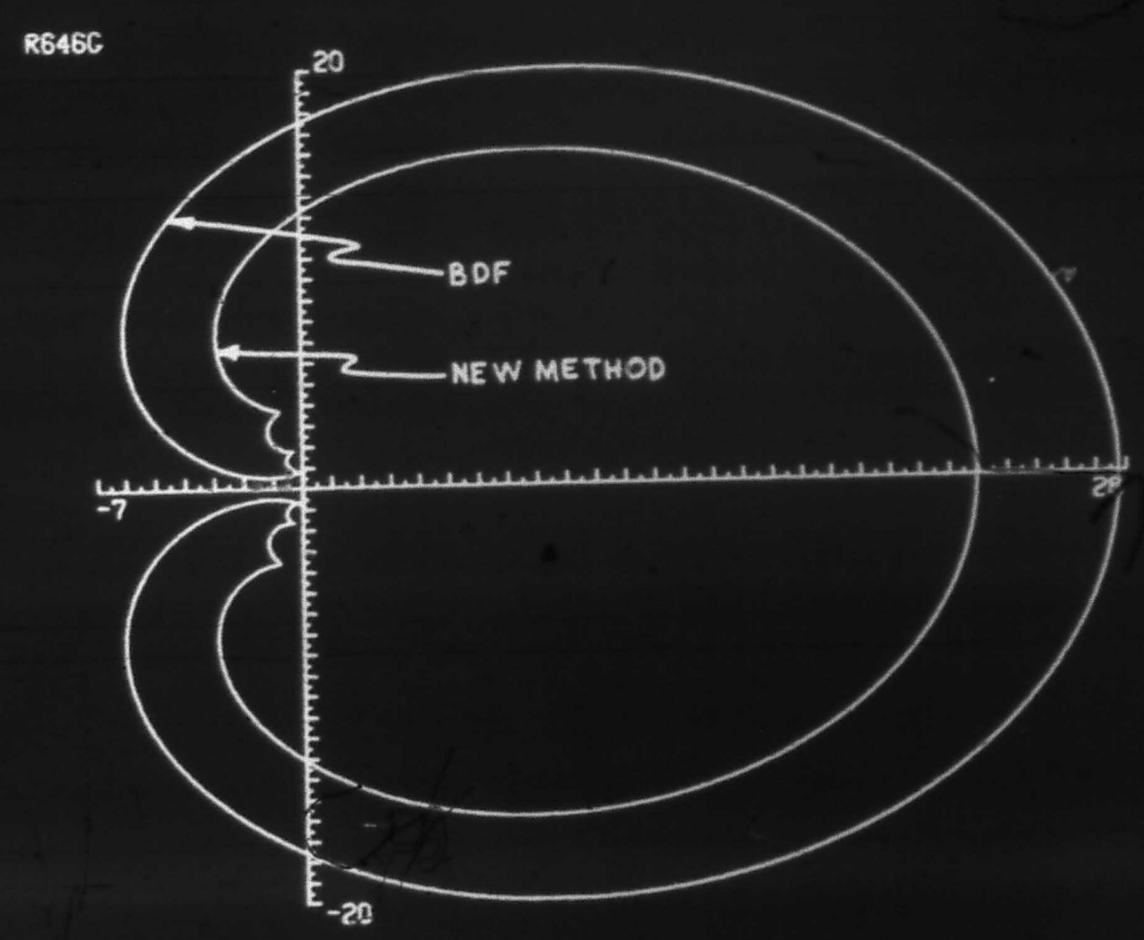


Figure 4.2: Comparison of Lambda Loci Between New Methods and BDF for Order Three Through Six (continued)

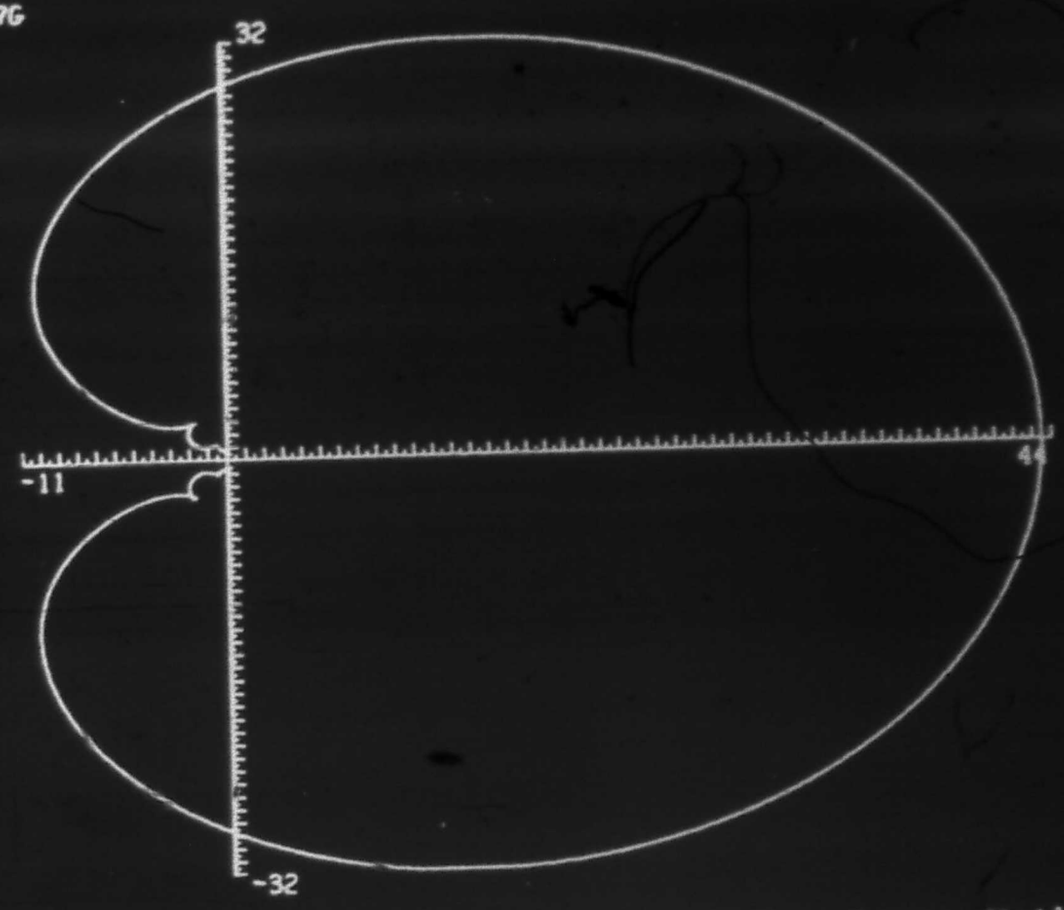


Figure 4.3: Lambda Locus for New Order Seven Method

differentiation formulas of order 1 and 2 form the basis for a new variable order, variable step algorithm described in the next chapter.

CHAPTER V

A NEW VARIABLE ORDER/VARIABLE STEP-SIZE INTEGRATION ALGORITHM

5.1 Introduction

A computer program embodying the cyclic composite linear multistep methods described in Chapter IV has been written for solving a system of first order ordinary differential equations. In the following a description of the computer program organization and logic as well as results obtained from some sample problems are described.

5.2 Computer Program Organization and Logic

The computer program, called DIFJMT, was written as a subroutine to be called by another program. DIFJMT, modeled after Gear's algorithm DIFSUB [GEA71a], is a variable order, variable step-size algorithm. The step-size and order—from 1 through 7—used at each step are chosen in such a manner as to achieve as large a step-size as possible while not introducing a single step error greater than a user specified value.

The computer program has four major sections which will be discussed individually. They are:

i) Starting procedure

ii) Implicit equation solution

iii) Error control

iv) Step-size and order selection logic

An organizational flow chart of DIFJMT is shown in Figure 5.1. A detailed flow chart and a computer program listing can be found in Appendix C.

The k-th order composite linear multistep method is given as

$$\sum_{j=-k+i}^{i} (\alpha_{i,j} y_{n\ell+j} - h\beta_{i,j} \dot{y}_{n\ell+j}) = 0, \quad \text{with } i=1, \ldots, \ell, \qquad (5.1)$$

where the $\alpha_{i,j}$'s and $\beta_{i,j}$'s are as in (4.7) and (4.8) for $k = 1$ and $k = 2$, respectively, and for $k = 3, \ldots, 7$ are as contained in Table 4.2.

The process proceeds as follows: The backpoints $y_{n\ell-k+1}, \ldots, y_{n\ell}$ and their first derivatives are assumed known. On entry to DIFJMT the index i is set to 1 and (5.1) is used to solve for $y_{n\ell+1}$ and $\dot{y}_{n\ell+1}$; i is incremented by 1 and (5.1) is used again. The cycle is repeated ℓ times—ℓ being a function of the order of the method used. The error test is then performed on the last term computed—$y_{n\ell+\ell}$—and if the estimated error is less than a user specified value the cycle is accepted; otherwise it is rejected. In either case control is transferred to the step-size and order selection section wherein the largest step-sizes,

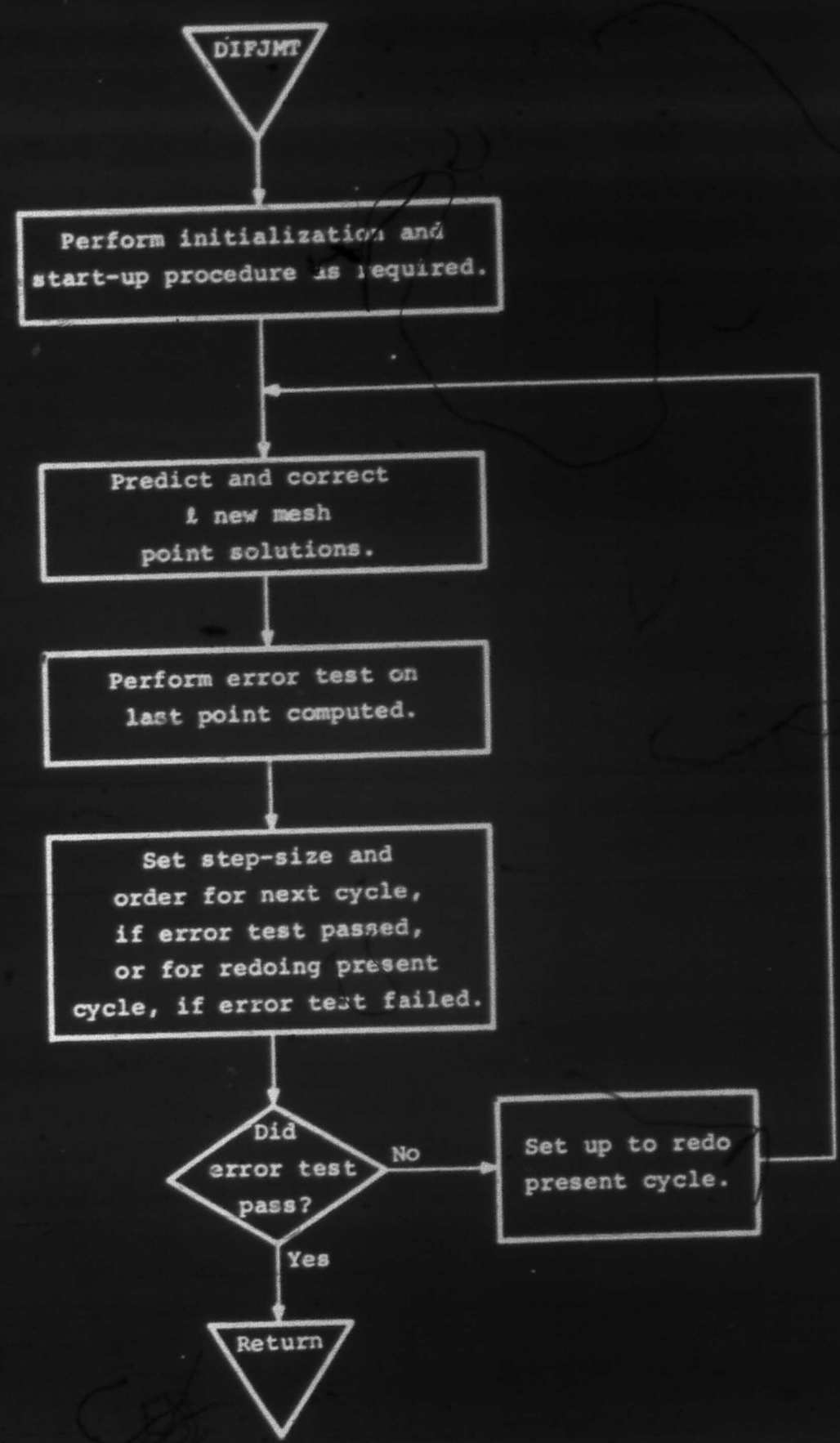


Figure 5.1: Organizational Flow Chart for Subroutine DIFJMT

consistent with a user specified maximum allowable local error, at the current order, one order lower, and one order higher* are determined. If the error test indicates failure the entire cycle is repeated at the largest of these step-sizes and the corresponding order. Otherwise control is returned to the calling program and on the next entry the largest step-size and corresponding order previously computed is used for the new cycle.

The following sub-sections describe in detail the various segments of DIFJMT. The solution of the implicit equations (5.1) is treated first. Subsequent sub-sections describe the other main segments of the computer program.

5.2.1 Solution of the Implicit Equations†

Each of the equations in (5.1) is implicit. The conventional iteration for solving (5.1) uses Picard's successive substitution method which proceeds as follows: If the predicted value for $y_{n\ell+i}$ is $y^{(0)}_{n\ell+i}$ the iteration

---

*If the error test indicated failure no increase in order is attempted. Logic is also built in to limit the order to a maximum value of 7 or a user specified lower value, namely, the value of the parameter MAXDER in the variable list of the calling sequence for DIFJMT as given in Appendix C.

†The treatment in the first part of this sub-section is similar to that of Gear [GEA67].

$$\alpha_{i,i} y_{n\ell+i}^{(m+1)} = h\beta_{i,i} f(y_{n\ell+i}^{(m)}, t_{n\ell+i}) - \sum_{j=-k+i}^{i-1} (\alpha_{i,j} y_{n\ell+j} - h\beta_{i,j} \dot{y}_{n\ell+j})$$

$$m=1, 2, \ldots, \qquad (5.2)$$

is performed, where $\dot{y} = f(y,t)$ is the differential equation being solved. This process will converge if

$$\left\| h \frac{\beta_{i,i}}{\alpha_{i,i}} \frac{\partial f}{\partial y} \right\| < 1, \qquad (5.3)$$

for any norm, in a neighborhood of the solution. Unless h is of the order of the reciprocal of the largest eigenvalue of $\partial f/\partial y$, (5.3) will not be satisfied. This clearly restricts the maximum allowable step-size that can be used.

A Newton-Rhapson iteration can be performed to solve (5.1) without restricting the maximum size of h. The procedure is described below.

Equation (5.1) for each i may be written as

$$y_{n\ell+i} - h\frac{\beta_{i,i}}{\alpha_{i,i}} f(y_{n\ell+i}, t_{n\ell+i}) = -\frac{1}{\alpha_{i,i}} \sum_{j=-k+i}^{i-1} (\alpha_{i,j} y_{n\ell+j} - h\beta_{i,j} \dot{y}_{n\ell+j}), \qquad (5.4)$$

where the right hand side is a sum of known values and hence is a constant. Equivalently, a value for $y_{n\ell+i}$ is desired such that

$$G(y_{n\ell+i}) = C, \tag{5.5}$$

where

$$C \equiv -\frac{1}{\alpha_{i,i}} \sum_{j=-k+i}^{i-1} (\alpha_{i,j} y_{n\ell+j} - h\beta_{i,j} \dot{y}_{n\ell+j}) \tag{5.6a}$$

is a constant and

$$G(y_{n\ell+i}) \equiv y_{n\ell+i} - h\frac{\beta_{i,i}}{\alpha_{i,i}} f(y_{n\ell+i}, t_{n\ell+i}). \tag{5.6b}$$

Assuming $y_{n\ell+i}^{(m)}$ is an estimate of the solution of (5.5), $y_{n\ell+i}^{(m+1)}$ is to be sought such that $G(y_{n\ell+i}^{(m+1)}) = C$. Upon expanding (5.5) in a Taylor series about $y_{n\ell+i}^{(m)}$ and keeping the first two terms, it is found that

$$G(y^{(m)}) + W_m[y^{(m+1)} - y^{(m)}] \simeq G(y^{(m+1)}) = C, \tag{5.7}$$

where the subscripts "$n\ell+i$" have been dropped for notational simplicity and where

$$W_m \equiv \left.\frac{\partial G}{\partial y}\right|_{y=y^{(m)}} \tag{5.8a}$$

or, from (5.6b),

$$W_m = I - h\frac{\beta_{i,i}}{\alpha_{i,i}} J_m, \qquad (5.\ b)$$

I being the identity matrix and

$$J_m \equiv \left.\frac{\partial f(y,t)}{\partial y}\right|_{y=y^{(m)}} . \qquad (5.9)$$

Substituting (5.6) and (5.8b) into (5.7) and re-arranging terms results in

$$[I - h\frac{\beta_{i,i}}{\alpha_{i,i}} J_m]y^{(m+1)} = h\frac{\beta_{i,i}}{\alpha_{i,i}}[f(y^{(m)},t) - J_m y^{(m)}] + C. \qquad (5.10)$$

Note: If (5.10) converges it will converge to the correct solution. Hence, the Jacobian matrix J_m need not be precisely known; in fact, a very poor approximation may be adequate. It can therefore be correctly inferred that J_m need not of necessity be recomputed at each iteration.

Now, subtracting (5.10) with m - 1 replacing m, from (5.10) and letting $J_{m-1} = J_m = J$ yields, after re-arranging terms, the desired result; namely,

$$y^{(m+1)} = y^{(m)} + \frac{\beta_{i,i}}{\alpha_{i,i}} W^{-1}[hf(y^{(m)},t) - d^{(m)}], \qquad (5.11)$$

where

$$d^{(m)} \equiv hf(y^{(m-1)},t) + hJ[y^{(m)} - y^{(m-1)}]. \quad (5.12)$$

In order to obtain a recurrence relation for $d^{(m)}$, substitute (5.11) into (5.12) with m + 1 replacing m in the latter; thus,

$$d^{(m+1)} = hf(y^{(m)},t)$$

$$+ h\frac{\beta_{i,i}}{\alpha_{i,i}} J\{W^{-1}[hf(y^{(m)},t) - d^{(m)}]\}. \quad (5.13)$$

Then, adding and subtracting $W^{-1}[hf(y^{(m)},t) - d^{(m)}]$ and rearranging terms gives the desired result; namely,

$$d^{(m+1)} = d^{(m)} + W^{-1}[hf(y^{(m)},t) - d^{(m)}]. \quad (5.14)$$

In order to start the recursion process—obtain $y^{(0)}$ and $d^{(0)}$—consider the k-step explicit formula

$$\alpha^*_{i,i}\, y_{n\ell+i} + \sum_{j=-k+i}^{i-1} (\alpha^*_{i,j}\, y_{n\ell+i} - h\beta^*_{i,j}\, \dot{y}_{n\ell+i}) = 0,$$

$$\text{with } i=1, \ldots, \ell. \quad (5.15)$$

Now, $y^{(0)}_{n\ell+i}$ may be taken as

$$y^{(0)}_{n\ell+i} = -\frac{1}{\alpha^*_{i,i}} \sum_{j=-k+i}^{i-1} (\alpha^*_{i,j}\, y_{n\ell+i} - h\beta^*_{i,j}\, \dot{y}_{n\ell+i}). \quad (5.16)$$

$d^{(0)}_{n\ell+i}$ is chosen so as to make (5.11) correct for $m = 0$, "correct" meaning $y^{(1)}_{n\ell+i} = y^{(0)}_{n\ell+i}$. Adding $-[I - h\frac{\beta_{i,i}}{\alpha_{i,i}} J]y^{(0)}_{n\ell+i}$ to both sides of (5.10) with $m = 0$ and setting the left hand side to zero, as the condition $y^{(1)}_{n\ell+i} = y^{(0)}_{n\ell+i}$ is being imposed, results in

$$0 = \frac{\beta_{i,i}}{\alpha_{i,i}}[hf(y^{(0)}_{n\ell+i}, t_{n\ell+i}) - \frac{\alpha_{i,i}}{\beta_{i,i}}\, y^{(0)}_{n\ell+i}$$

$$- \frac{1}{\beta_{i,i}} \sum_{j=-k+i}^{i-1} (\alpha_{i,j}\, y_{n\ell+j} - h\beta_{i,j}\, \dot{y}_{n\ell+j})]. \quad (5.17)$$

By substituting (5.16) and then examining (5.11) with $m = 0$ and $y^{(m+1)} = y^{(m)}$, it is seen that choosing

$$d^{(0)}_{n\ell+i} = \frac{1}{\alpha^*_{i,i}\beta_{i,i}} \sum_{j=-k+i}^{i-1} [(\alpha^*_{i,i}\, \alpha_{i,j} - \alpha^*_{i,j}\, \alpha_{i,i})\, y_{n\ell+i}$$

$$- h(\alpha^*_{i,i}\, \beta_{i,j} - \beta^*_{i,j}\, \alpha_{i,i})\, \dot{y}_{n\ell+i}] \quad (5.18)$$

makes (5.11) "correct" for $m = 0$. Note: Both $y^{(0)}_{n\ell+i}$ and $d^{(0)}_{n\ell+i}$ are linear combinations of earlier—known—function values and derivatives. Further, if $y^{(0)}_{n\ell+i}$ as given by (5.16) satisfies (5.1), then, from (5.11), $d^{(0)}_{n\ell+i} = hf(y^{(0)}_{n\ell+i}, t_{n\ell+i}) = h\dot{y}_{n\ell+i}$. Therefore, $d^{(0)}_{n\ell+i}$ can be identified as the "predicted

value of $h\dot{y}_{n\ell+i}$". Moreover, when convergence is achieved $d^{(m)}_{n\ell+i} = h\dot{y}_{n\ell+i}$—that is, just as $y^{(m)}_{n\ell+i}$ is considered the m-th iterate for $y_{n\ell+i}$ so too can $d^{(m)}_{n\ell+i}$ be considered the m-th iterate for $h\dot{y}_{n\ell+i}$.

Summarizing, the procedure used in solving (5.1) is, for $i=1, \ldots, \ell$:

1. A predictor formula for suitably chosen $\alpha^*_{i,j}$ and $\beta^*_{i,j}$ is used to obtain $y^{(0)}_{n\ell+i}$, as in (5.16), and $d^{(0)}_{n\ell+i}$, as in (5.18).
2. The Jacobian matrix J is evaluated and used to compute
$$W^{-1} = [I - h\frac{\beta_{i,i}}{\alpha_{i,i}} J]^{-1}.$$
3. For m = 1, 2, ... (5.11) and (5.14) are recursively used to solve for $y_{n\ell+i}$ and $d_{n\ell+i}$, the latter being taken as $h\dot{y}_{n\ell+i}$.

As the predictor formula used to obtain $y^{(0)}_{n\ell+i}$, and hence $d^{(0)}_{n\ell+i}$, does not affect the stability properties of the method—since each of the formulas (5.1) is iterated to convergence—the same form of (5.15) is used for each i. More precisely, in (5.15) for $i = 2, \ldots, \ell$ and $j = -k+i, \ldots, i$,

$$\alpha^*_{i,j} = \alpha^*_{i-1,j-1}, \tag{5.19a}$$

and

$$\beta^*_{i,j} = \beta^*_{i-1,j-1} \,. \tag{5.19b}$$

Then, for i = 1—the initial predictor—(5.15) is chosen to be of order k and of the form

$$\sum_{j=-k+1}^{1} \alpha^*_{1,j}\, y_{n\ell+j} - h\beta^*_{1,0}\, \dot{y}_{n\ell} = 0. \tag{5.20}$$

Note: To within a multiplicative constant the order condition serves to fully specify the $\alpha^*_{1,j}$ and $\beta^*_{1,0}$. Also an order k formula was chosen so that the predictor might enhance the convergence properties of (5.1).*

The calculation of W^{-1}—required for (5.11) and (5.14)—is time consuming for a large system of differential equations. Therefore, it is only re-evaluated when the Newton iteration fails to converge. It has also been found worthwhile to re-evaluate W^{-1} whenever an order change is made. However, if only the step-size is changed the convergence properties do not seem to be significantly affected and

---

*The order 1 and 2 predictor formulas are, respectively,

$$y_{n+1} = y_n + h\,\dot{y}_n,$$

and

$$y_{n+1} = y_{n-1} + h\,\dot{y}_n.$$

For k = 3, ..., 7 the predictor formulas used correspond to those found in the second row from the bottom of Table 4.1 for each of the corresponding orders.

W^{-1} is not re-evaluated.*

Convergence of (5.11) and (5.14) is determined as follows: If, form m = 1,†

$$\delta^{(m)} \equiv ||W^{-1}[hf(y^{(m)}, t) - d^{(m)}]|| \leq \text{BND}, \quad (5.21)$$

where BND is chosen as§ $\varepsilon/(2k + 4)$, ε being the user specified accuracy, then (5.1) is said to have converged. For m > 1 the inequality

$$\delta^{(m)} \times \min\{1, 2\delta^{(m)}/\delta^{(m-1)}\} \leq \text{BND} \quad (5.22)$$

is checked and if satisfied it is assumed that either convergence has been achieved on this iteration or would be achieved on the next iteration. In either case the process is terminated. Note: The look-ahead feature incorporated into (5.22) assumes linear convergence. The factor of 2 is applied to insure that on the next iteration, if it were performed, (5.21) would be more than satisfied. If such is the case the next iteration is not performed so as to save

---

*There seems to be no explanation for this anomaly. However, this backs up the experience of Gear [GEA68].

†The Euclidean norm is used, as in (5.32).

§This choice was made on the recommendation of Gear [private communication] and works satisfactorily.

function evaluations.*

In order to save computer time a maximum of 3 iterations of (5.11) and (5.14) is performed. If convergence has not been achieved by the end of the third iteration it is assumed that (5.11) and (5.14) are not going to converge. At this point W^{-1} is re-evaluated. If the iteration fails a second time at the same mesh point† the step-size is decreased by a factor of 4 and the entire cycle is repeated. There is no theoretical justification for the choice of a maximum of 3 iterations. However, numerical results indicate that, if 4 or more iterations are permitted, the number of times the corrector fails to converge is not appreciably less. If only 2 iterations are permitted, the number of times convergence failure occurs is considerably greater, resulting in significantly more evaluations of W^{-1} and more function evaluations.

5.2.2 Error Control

With reference to (2.24), $L_i[y(t_{n\ell+i}), h]$ is the amount by which the solution of the differential equation fails to satisfy the i-th equation of a composite linear

---

*This look-ahead feature is a variation of a suggestion made by Gear [private communication].

†Note: $\ell \geq 1$ mesh points are computed for each call to DIFJMT.

multistep method. As such it is a measure of the error introduced by the i-th formula. Since the solution of the i-th formula of (5.1) equals the truncated series of the true solution through terms in h^k, it is known as the local truncation error.*

Now, for (5.1)

$$L_i[y(t_{n\ell+i}), h] \approx C_{i,k+1}\, h^{k+1}\, y^{(k+1)}(t_{n\ell+i}), \quad (5.23)$$

where $\approx$ means to within terms of $O(h^{k+2})$ and where

$$C_{i,k+1} = \frac{1}{(k+1)!} \sum_{j=-k+i}^{i} (j^{k+1}\alpha_{i,j} - (k+1)j^k\beta_{i,j}). \quad (5.24)$$

Further, $D_{i,k+1}$, the local discretization error constant for the i-th formula of (5.1), is given as

$$D_{i,k+1} = C_{i,k+1}/\alpha_{i,i}, \quad i = 1, \ldots, \ell. \quad (5.25)$$

Now, for $i = 1, \ldots, \ell$,

$$y(t_{n\ell+i}) - y_{n\ell+i} \approx D_{i,k+1}\, h^{k+1}\, y^{(k+1)}(t_{n\ell+i}). \quad (5.26)$$

---

*Strictly speaking this is true if, for a k-th order method, the true solution is (k + 1) times continuously differentiable and $y_{n\ell+j+i} = y(t_{n\ell+j+i}) + O(h^{k+1})$ for $j=-k, \ldots, -1$. Although in general this cannot be shown the technique used for error control to be outlined, based on the local truncation error, appears to work successfully.

Equation (5.26) forms the basis for the error criterion used in DIFJMT. It proceeds as follows: At the completion of each cycle $\nabla^{k+1}y_{n\ell+\ell}$ is computed, where ∇ is the backward difference operator defined as

$$\nabla^j x_m \equiv \nabla^{j-1}x_m - \nabla^{j-1}x_{m-1}, \text{ for } j = 1, \ldots, \tag{5.27a}$$

and

$$\nabla^0 x_m \equiv x_m. \tag{5.27b}$$

As

$$\nabla^{k+1}y_{n\ell+\ell} = h^{k+1}y^{(k+1)}(\tau) \tag{5.28}$$

for some $\tau \in [t_{n\ell+\ell-k+1}, t_{n\ell+\ell}]$, the quantity $|\nabla^{k+1}y_{n\ell+\ell}|$ is compared with ε/D_{k+1}, where ε is the user specified maximum single step error permitted and

$$D_{k+1} = \max_{i=1,\ldots,\ell} \{|D_{i,k+1}|\} \quad . \tag{5.29}$$

If

$$|\nabla^{k+1} y_{n\ell+\ell}| \leq \varepsilon/D_{k+1} \tag{5.30}$$

the cycle is accepted.* Otherwise the cycle is rejected, the

---

*To obtain (5.30), it was tacitly assumed that the (k+1)-st derivative remains constant over the interval $[t_{n\ell+\ell-k+1}, t_{n\ell+\ell}]$; in general this is not true. However, $\nabla^{k+1}y_{n\ell+\ell}$ is a reasonable estimate for $h^{k+1}y^{(k+1)}(t)$ anywhere in the interval $[t_{n\ell+\ell-k+1}, t_{n\ell+\ell}]$. As the ℓ future points

step-size is decreased as outlined in the next section, and the cycle repeated. If the error test fails on three successive tries the order is set to 1 and the cycle repeated being treated as if it is the first call to DIFJMT.* This procedure is followed for two reasons as follows:

1. It is possible that the error test failed due to the eigenvalues of the Jacobian matrix associated with the differential equation being in an unstable region of the λ-plane. Since the order 1 method is A-stable—as can be seen from Figure 4.1—it is used.

2. The error test may have failed due to accumulated round-off error in $y_{n\ell+j}$, $j = -k+1, \ldots, 0$. Hence the integration process is restarted.

If the error test failed and the step-size cannot be decreased† then the order is immediately set to 2 if it is greater than

---

solved for by (5.1)—for all order methods—span $[t_{n\ell+1}, t_{n\ell+\ell}] \subset [t_{n\ell+\ell-k+1}, t_{n\ell+\ell}]$, using D_{k+1} as defined by (5.29), the the error test (5.30) yields a reasonable estimate of the maximum error introduced at any of the ℓ mesh points $t_{n\ell+i}$, $i = 1, \ldots, \ell$.

*Special procedures followed during start-up are described in Section 5.2.4.

†The step-size is not decreased below a user specified minimum value, namely, the value of the parameter HMIN in the variable list of the calling sequence for DIFJMT as given in Appendix C.

2, or to 1 if it is already 2.* If the order is already set to 1 the cycle is accepted and the user notified.

As DIFJMT can be used to solve a system of N first order differential equations the error test performed is not really that given by (5.30) but rather that given as

$$D_{k+1}||\nabla^{k+1} \underset{\sim}{y}_{n\ell+\ell}|| \leq \varepsilon, \tag{5.31}$$

where†

$$||\nabla^{k+1} \underset{\sim}{y}_{n\ell+\ell}||^2 = \sum_{j=1}^{N} [(\nabla^{k+1} \underset{\sim}{y}_{n\ell+\ell})_j/\omega_j]^2, \tag{5.32}$$

$(\nabla^{k+1}\underset{\sim}{y}_{n\ell+\ell})_j$ being the j-th component of the N-dimensional vector $\nabla^{k+1}\underset{\sim}{y}_{n\ell+\ell}$ and ω_j being a weighting function for the j-th component.§

---

*Note: The order 1 and order 2 methods are A-stable.

†Any other suitable norm could have been used but (5.32) is easily implemented in a FORTRAN program. Note: If (5.32) is used it is only necessary to form $||\nabla^{k+1}\underset{\sim}{y}_{n\ell+\ell}||^2$ and compare it to $(\varepsilon/D_{k+1})^2$.

§Note: In the computer program listing given in Appendix C, ω_j, j=1, ..., N, is set to the maximum of ω_j and $|(\underset{\sim}{y}_{n\ell+i})_j|$, i=1, ..., ℓ, at the end of each cycle. Initializing ω_j to $\max\{1,|(\underset{\sim}{y}_o)_j|\}$, for j=1, ..., N, provides an error test relative to the largest value in the history of $|(\underset{\sim}{y})_j|$ unless the user overrides it before each call. The YMAX parameter in the calling sequence corresponds to the weighting array $\underset{\sim}{\omega}$ described here.

The discretization error constants for the methods used in DIFJMT are given in Table 5.1.

5.2.3 Step-Size and Order Selection Logic

The step control mechanism is activated in each cycle following solution of the algebraic equations (5.1) and application of the error test (5.31). The step-size to use in the next cycle or when repeating a rejected cycle is estimated to be ηh, h being the current step-size. The value η to be used is determined as the maximum of η_{k-1}, η_k, and η_{k+1}, where

$$\eta_j = \gamma \left(\frac{\varepsilon}{D_{j+1} \| \nabla^{j+1} \underset{\sim}{y}_{n\ell+\ell} \|} \right)^{1/(j+1)} , \quad \text{with } j=k-1,\ k,\ k+1, \qquad (5.33)$$

Note: γ is a constant such that $0 < \gamma \leq 1$, and k is the current order. Also note that if the step-size $\eta_j h$ were used with an order j method and if the error were exactly proportional to h^{j+1}—that is, $\|\nabla^{j+1} \underset{\sim}{y}_{n\ell+\ell}\|$ remained constant—the error test (5.31) would be satisfied, and satisfied exactly if γ were 1. As $\|\nabla^{j+1} \underset{\sim}{y}_{n\ell+\ell}\|$ is not usually constant the factor γ is used to provide a "safety" margin.*

---

*A value of $\gamma = 1/1.2$ is used in DIFJMT.

Order	$D_{1,k+1}$	$D_{2,k+1}$	$D_{3,k+1}$	$D_{4,k+1}$	D_{k+1}
1	-1/2				1/2
2	-2/9				2/9
3	-3/22	-3/22	-1/6		1/6
4	-12/125	-12/125	-10/93		10/93
5	-10/137	-10/137	-99/1373	-239/2316	239/2316
6	-20/343	-121/2051	-591/10304	-1699/16093	1699/16093
7	-35/726	-35/726	-503/8008	-1319/19496	1319/19496

Table 5.1: Discretization Error Constants for Methods Used in DIFJMT

Now, given η_j for $j=k-1$, k, and $k+1$, the next step-size and order would be those associated with the largest of the three step-size ratios.* The computer program listing given in Appendix C does not increase the step or change the order—assuming the error test passed—if $\eta < 1.1$ since it is felt the increase is not worth the extra computer time required to interpolate the mesh point values at the new step-size.

As the step control mechanism requires $\nabla^{k+2} y_{n\ell+\ell}$ in order to determine η_{k+1}—requiring information from k+3 different time points in its evaluation—and as only $y_{n\ell+j}$, $j=-k+1, \ldots, 0$—information at only k different time points—is available at the start of DIFJMT, the solution at a minimum of three mesh points is computed at each call.† Moreover, as both the error control and step-size selection logic are dependent on the backward differences of $y_{n\ell+\ell}$, the i-th equation of (5.1) is formulated§ in terms of $\nabla^{\sigma} y_{n\ell+i}$ instead

---

*If the error test failed η_{k+1} is not computed and no attempt is made to increase the order.

†If an order 1 or 2 method is used the same equation—(4.7) or (4.8), respectively—is used for all three mesh points. For higher order methods $\ell \geq 3$.

§In terms of backward differences (5.1) can be written as

$$\sum_{\sigma=0}^{k} (\hat{\alpha}_{i,\sigma} \nabla^{\sigma} y_{n\ell+i} - h\hat{\beta}_{i,\sigma} \dot{y}_{n\ell+i-\sigma}) = 0,$$

with $i=1, \ldots, \ell$,

of $y_{n\ell+i-\sigma}$ for $\sigma=0, \ldots, k$.

When the step-size is changed the present value of time $t_{n\ell}$ remains fixed, but the past values of time $t_{n\ell+j}$, $j=-\hat{k}+1, \ldots, -1$, must be adjusted to reflect the new step-size $\hat{h} = \eta h$ by which they differ. Note: $\hat{k}$ denotes the possibly new order. Thus, it also follows that it is necessary to compute $\hat{k} - 1$ new values of the solution, $\hat{y}_{n\ell+j}$, $j=-\hat{k}+1, \ldots, -1$. The value of $\hat{h}\dot{\hat{y}}_{n\ell}$ needed for the predictor is trivially adjusted by introducing η as a multiplicative factor; thus

$$\hat{h}\dot{\hat{y}}_{n\ell} = \eta(h\dot{y}_{n\ell}). \tag{5.34}$$

---

where, for $\sigma=0, \ldots, k$ and $i=1, \ldots, \ell$,

$$\hat{\alpha}_{i,\sigma} = (-1)^{\sigma} \sum_{\nu=\sigma}^{k} \binom{\nu}{\sigma} \alpha_{i,i-\sigma},$$

and

$$\hat{\beta}_{i,\sigma} = \beta_{i,i-\sigma}.$$

The $\alpha_{i,j}$'s and $\beta_{i,j}$'s ($i=1, \ldots, \ell$ and $j=-k+i, \ldots, i$) are as in (5.1). Formulating (5.1) in terms of backward differences does not affect the solution of the implicit equations as described in Section 5.2.1 except that $\nabla^{\sigma} y_{n\ell+i}$, $\sigma=1, \ldots, k$, must now also be corrected. Note: $\nabla^{0} y_{n\ell+i} = y_{n\ell+i}$. In order to save computer time, the computation of $\nabla^{\sigma} y_{n\ell+i}$, $\sigma > 0$, is performed only once convergence has been achieved.

Two cases need be considered:

Case 1. The error test passed on the previous cycle and $\eta > 1.1$.

Case 2. The error test failed and the current cycle is to be redone.

Case 1: A k-th order interpolating polynomial is used to compute the $\hat{k}$-1 new values of the solution $\hat{y}_{n\ell+j}$, $j=-\hat{k}+1, \ldots, -1$. In order to keep the error introduced by the interpolation procedure to a minimum the k+1 values required to define the interpolating polynomial are chosen at k+1 consecutive mesh points. These mesh points, selected from among the $k+\ell$ mesh points available from the previous cycle, are chosen in such a manner that, for a given index j, they bracket $t_{n\ell} + j\hat{h}$. More precisely, an index ν is chosen such that, for a given j, $j=-\hat{k}+1, \ldots, -1$,

$$t_{n\ell} + j\hat{h} \in [t_{n\ell} - \nu h - kh,\ t_{n\ell} - \nu h] \equiv I_\nu, \quad (5.35a)$$

for some $\nu = \{0, \ldots, \ell-1\}$.* Equivalently, an index ν is chosen such that for a given j, $j=-\hat{k}+1, \ldots, -1$,

$$\nu \leq -j\eta \leq \nu + k. \quad (5.35b)$$

---

*At the start of DIFJMT only the values $y_{(n-1)\ell+j}$, $j=-k+1, \ldots, \ell$ are available from the previous call.

Note: Using this procedure, for differing values of j, ν may assume a different index. Using Newton's backward difference formula* [HEN64] an interpolating polynomial which interpolates $y_{n\ell-\nu+\mu}$, $\mu=-k, \ldots, 0$, is used to compute $\hat{y}_{n\ell+j}$. Note: If (5.35) is satisfied then†

$$\hat{y}_{n\ell+j} = y(t_{n\ell} + j\hat{h}) + o(h^{k+1}). \tag{5.36}$$

Requiring (5.35) to be satisfied restricts the maximum allowable value of η to

$$\eta_{max} = (\ell + k - 1)/(\hat{k} - 1). \tag{5.37}$$

Table 5.2 shows the maximum allowable values for η_{k-1}, η_k, and η_{k+1} subject to (5.37).

Summarizing, the procedure for increasing the step-size is as follows:

<u>Step 1</u>: The factor by which the step-size is changed, η, is set equal to $\max\{\eta_j\}$ as given by (5.33).

---

*As DIFJMT operates on $\nabla^\sigma y_{n\ell+i}$, $\sigma=0, \ldots, k$, $i=1, \ldots, \ell$, this form of the interpolating polynomial is used.

†Equation (5.36) assumes that the values $y_{n\ell-\nu+\mu}$ are exact or at least accurate to order k+1 in h, and that y(t) is (k+1) times continuously differentiable for $t \in I_\nu$. As the error test was passed on the previous cycle the values $y_{n\ell-\nu+\mu}$ are, at least locally, accurate to order k+1 in h.

k	ℓ	η_{k-1}	η_k	η_{k+1}
1	3	*	$\infty^{\dagger}$	3
2	3	$\infty^{\dagger}$	4	2
3	3	5	5/2	5/3
4	3	3	2	3/2
5	4	8/3	2	8/5
6	4	9/4	9/5	3/2
7	4	2	5/3	*

*These values are not computed.

†Program limited to 10^4.

Table 5.2: Maximum Allowable Step-Size Changes

If $\eta < 1.1$ the step-size and order are not changed and η is reset to the value of 1. Otherwise, the new order $\hat{k}$ is set accordingly and η is limited as given by (5.37).

Step 2: Using (5.34) $\hat{h}\hat{y}_{n\ell}$ is computed.

Step 3: The index j is set to -1 and, as $\hat{h} > h$, ν is set to 1.

Step 4: If ν does not satisfy (5.35b) then ν is set to ℓ-1.*

Step 5: The Newton backward interpolation formula, interpolating $y_{n\ell-\nu+\mu}$, $\mu=-k, \ldots, 0$ is used to compute $\hat{y}_{n\ell+j}$.

Step 6: The index j is decremented by 1 and if $j \geq -\hat{k}+1$ Steps 4 and 5 are repeated. Otherwise the process terminates.

Case 2: The procedure followed when h is decreased is similar to Case 1—an interpolating polynomial is used to compute $\tilde{y}_{n\ell+j}$ for $j=-\tilde{k}+1, \ldots, -1$. Note: Here, ~ denotes values to be established for repeating a cycle, and ^ denotes values established prior to performing the first of the rejected cycles.† The solution values used to form the interpolating

---

*An examiniation of Table 5.2 will show that if $\nu = 1$ does not satisfy (5.35b) then $\nu = \ell-1$ will, for all k.

†Note: The ^ values reflect any interpolation due to a possible step-size increase.

polynomial are those available prior to the first rejected cycle*—that is, the ^ values. The order of the interpolating polynomial used is $\hat{k}$. Note: $\tilde{k} \leq \hat{k}$. If no step-size increase was performed then the $\nabla^{\sigma}\hat{y}_{n\ell}$, $\sigma=0, \ldots, \hat{k}$, are available to use in Newton's backward difference formula, as in Case 1. If the step-size was initially increased then only the $\nabla^{\sigma}\hat{y}_{n\ell}$, $\sigma=0, \ldots, \hat{k}-1$, are available. In this case, together with $\hat{h}\dot{\hat{y}}_{n\ell}$, a $\hat{k}$-th order interpolating polynomial is constructed. It is readily verified that such an interpolating polynomial is

$$\phi(\mu) = a_0 + a_1\mu + a_2\mu(\mu + 1) + \cdots + a_{\hat{k}}\mu(\mu + 1)\cdots(\mu + \hat{k} - 1), \quad (5.38)$$

where

$$\mu \equiv (t - t_{n\ell})/\hat{h}, \quad (5.39a)$$

$$a_{\sigma} = \frac{1}{\sigma!}\nabla^{\sigma}\hat{y}_{n\ell}, \text{ for } \sigma=0, \ldots, \hat{k}-1, \quad (5.39b)$$

and

$$a_{\hat{k}} = [h\dot{y}_{n\ell} - \sum_{\nu=1}^{\hat{k}-1} (\nu - 1)!a_{\nu}]/(\hat{k} - 1)!. \quad (5.39c)$$

Note: $\phi(\mu)$ satisfies

---

*These values are retained in a special work area for this purpose.

$$\phi(j) = \hat{y}_{n\ell+j}, \text{ with } j = -\hat{k}+1, \ldots, 0, \tag{5.40a}$$

and

$$\dot{\phi}(0) = \hat{h}\hat{\dot{y}}_{n\ell}. \tag{5.40b}$$

Hence, $\phi(\mu)$, with $\mu = -\tilde{k}+1, \ldots, -1$, can be used to compute $\hat{y}_{n\ell+\mu}$ with the resulting values accurate to order $\hat{k}+1$ in $\hat{h}$.*

5.2.4 Start-Up Procedure

Starting is almost automatic. The value of $h\dot{y}_0$ is computed from a user specified initial condition y_0, a user specified initial step-size h, and a user supplied subroutine to evaluate f(y, t)—the right hand side of the differential equation. This is sufficient to allow the first order method to be used. From there on the error control process and the step-size/order selection logic, functioning as previously described, maintain the normal cycle processes.

---

*The assumption that the values $\hat{y}_{n\ell-\sigma}$, $\sigma=0, \ldots, \hat{k}-1$ and $\hat{h}\hat{\dot{y}}_{n\ell}$ are accurate to at least order $\hat{k}+1$ in $\hat{h}$, and that y(t) is $(\hat{k}+1)$ times continuously differentiable for $t \in [t_{n\ell} - (\hat{k}-1)\hat{h}, t_{n\ell}]$ has been made. Note: If the order was not initially increased then the step-size/order selection logic and the interpolation procedure insure that the values $\hat{y}_{n\ell-\sigma}$ and $\hat{h}\hat{\dot{y}}_{n\ell}$ are accurate, at least locally, to order $\hat{k}+1$ in $\hat{h}$. Though no such statement can be made if the order was initially increased, the procedure, however, does work satisfactorily.

5.3 Sample Problems and Comparisons

In this section five problems are presented which were solved using DIFJMT. For comparative purposes these problems were also solved using a computer program which was a slight modification* of Gear's algorithm DIFSUB [GEA71a]. Gear's algorithm is based on the backward differentiation formulas [HEN62] of order 1 through 6 and uses the Nordsieck vector [NOR62] as the means of storing information. The Nordsieck vector $\underset{\sim}{a}_n$ is defined as

$$\underset{\sim}{a}_n \equiv \left(y_n,\ h\dot{y}_n,\ \frac{h^2\ddot{y}_n}{2!},\ \ldots,\ \frac{h^k y_n^{(k)}}{k!}\right), \qquad (5.41)$$

where h is the current step-size and k is the current order. This choice was motivated by the ease with which one can predict y_{n+1} and can change the step-size h. However, it has been observed that the choice of the Nordsieck vector, as implemented in Gear's algorithm, does affect stability properties when h varies rapidly [BRA72a]. This undesirable

---

*The modification involved removing all coding in DIFSUB that corresponded to the MF = 0 and MF = 2 options in the calling sequence. The former permitted DIFSUB to use a variable order, variable step Adams-Moulton algorithm. Note: Adams type methods are not stiffly stable. The latter option permitted DIFSUB to evaluate the Jacobian matrix by numerical methods rather than calling a subroutine to supply it. As neither of these options affect the basic properties of the method but only add to computer run time, they were removed.

property was diminished in a third computer program which was a slight modification of DIFJMT—referred to as DIFBDF. This latter program is identical to DIFJMT except that all ℓ formulas in (5.1) are identical* and correspond to the backward differentiation formulas. Note: As the order seven BDF is not stiffly stable the maximum order was limited to six in DIFBDF—as in DIFSUB.

The five problems and their solutions are contained in Tables 5.3 through 5.7. In addition, each table includes the properties of the solution obtained by each of the three algorithms as a function of the local error requested. The maximum global errors listed were established for each solution as the maximum of the computed local errors at all mesh points, where, for a system of N first order differential equations,†

$$(\text{local error})_n \equiv \max_{i=1,\ldots,N} \left| \frac{(\underset{\sim}{y}_n)_i - (\underset{\sim}{y}(t_n))_i}{(\underset{\sim}{\omega}_n)_i} \right|, \quad (5.42a)$$

and

$$(\underset{\sim}{\omega}_n)_i \equiv \max\{1, \max_{j=0,\ldots,n} |(\underset{\sim}{y}_j)_i|\}. \quad (5.42b)$$

---

*Observe from Table 4.2, for i = 1 the formulas of the new methods, for all orders, are the BDF of the corresponding order. Hence the modification was readily implemented.

†Recall: All three algorithms <u>estimate</u> the local error using the norm as defined by (5.32).

DIFFERENTIAL EQUATION	SOLUTION
$\dot{y} = -y$ with $y(0) = 1$	$y(t) = e^{-t}$

PROPERTIES OF THE NUMERICAL SOLUTION							
Local Error Bound	10^{-5}	10^{-7}	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
SOLUTION VIA DIFJNT							
Maximum Global Error	5.05×10^{-6}	7.47×10^{-7}	9.76×10^{-8}	19.2×10^{-9}	7.83×10^{-10}	10.6×10^{-11}	7.33×10^{-12}
# of Time Steps	73	97	120	161	158	187	254
# of Function Evaluations	182	191	301	409	311	390	528
# of Jacobian Evaluations	17	10	19	35	11	11	15
Final Step-Size (sec)	0.394	0.334	0.265	0.210	0.168	0.126	0.0954
CPU Time (min $\times 10^{-3}$)	9	8	12	18	12	17	27
SOLUTION VIA DIFSUB							
Maximum Global Error	1.54×10^{-6}	2.50×10^{-7}	2.10×10^{-8}	3.31×10^{-9}	6.03×10^{-10}	4.81×10^{-11}	6.72×10^{-12}
# of Time Steps	67	85	106	135	184	235	317
# of Function Evaluations	154	178	252	293	397	494	662
# of Jacobian Evaluations	8	6	6	6	6	6	6
Final Step-Size (sec)	0.585	0.369	0.282	0.187	0.148	0.103	0.0722
CPU Time (min $\times 10^{-3}$)	8	7	8	14	16	23	27
SOLUTION VIA DIFBDF							
Maximum Global Error	3.06×10^{-6}	3.15×10^{-7}	1.35×10^{-8}	2.23×10^{-9}	3.59×10^{-10}	4.75×10^{-11}	6.68×10^{-12}
# of Time Steps	69	89	111	140	181	235	310
# of Function Evaluations	138	183	206	264	354	473	628
# of Jacobian Evaluations	9	8	6	6	10	6	6
Final Step-Size (sec)	0.554	0.326	0.246	0.203	0.148	0.114	0.0845

Table 5.3: Problem 1

DIFFERENTIAL EQUATION

$$\dot{\underline{y}} = U'BU\underline{y} + U'\underline{w}$$

with

$$\underline{y}(0) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

where

$$U = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1000 & 0 & 0 & 0 \\ 0 & -800 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & -0.001 \end{bmatrix}$$

$$\underline{w} = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{bmatrix} \text{ and } \underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = U\underline{y}$$

SOLUTION

$$\underline{y}(t) = U'\underline{x}(t)$$

where

$$\underline{x}(t) = \begin{bmatrix} \dfrac{1000}{1-1001e^{1000t}} \\ \dfrac{800}{1-801e^{800t}} \\ \dfrac{-10}{1+9e^{-10t}} \\ \dfrac{0.001}{1-1.001e^{0.001t}} \end{bmatrix}$$

PROPERTIES OF THE NUMERICAL SOLUTION

Local Error Bound	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
SOLUTION VIA DIFJMT							
Maximum Global Error	9.93×10^{-6}	10.4×10^{-7}	6.22×10^{-8}	14.2×10^{-9}	13.8×10^{-10}	17.5×10^{-11}	27.6×10^{-12}
# of Time Steps	293	378	496	690	825	1123	1641
# of Function Evaluations	658	918	1184	1724	1943	2631	4003
# of Jacobian Evaluations	60	74	85	114	103	118	200
Final Step-Size (sec)	1.28×10^{3}	983	518	621	218	385	246
CPU Time (min $\times 10^{-3}$)	139	213	252	380	412	558	856
SOLUTION VIA DIFSUB							
Maximum Global Error	2.48×10^{-6}	3.98×10^{-7}	2.08×10^{-8}	6.35×10^{-9}	4.78×10^{-10}	*	8.97×10^{-12}
# of Time Steps	301	425	530	665	841	*	1492
# of Function Evaluations	733	1059	1286	1646	2126	*	3651
# of Jacobian Evaluations	33	38	40	36	38	*	40
Final Step-Size (sec)	2.96×10^{3}	645	679	307	365	*	220
CPU Time (min $\times 10^{-3}$)	121	183	233	282	374	*	631
SOLUTION VIA DIFBDF							
Maximum Global Error	2.10×10^{-6}	3.19×10^{-7}	2.48×10^{-8}	3.97×10^{-9}	4.04×10^{-10}	4.62×10^{-11}	10.7×10^{-12}
# of Time Steps	270	366	488	643	853	1122	1497
# of Function Evaluations	603	804	1056	1394	1836	2416	3179
# of Jacobian Evaluations	42	56	54	56	53	56	55
Final Step-Size (sec)	1.15×10^{3}	1.15×10^{3}	828	498	424	310	211

*DIFSUB aborted after 6 time steps indicating no solution was possible.

Table 5.4: Problem 2

DIFFERENTIAL EQUATION	SOLUTION
$\dot{\underline{y}} = U'BU\underline{y} + U'\underline{w}$ with $\underline{y}(0) = \begin{bmatrix} 0 \\ -2 \\ -1 \\ -1 \end{bmatrix}$ where $U = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$ $B = \begin{bmatrix} 10 & 10 & 0 & 0 \\ -10 & 10 & 0 & 0 \\ 0 & 0 & -1000 & 0 \\ 0 & 0 & 0 & -0.001 \end{bmatrix}$ $\underline{w} = \begin{bmatrix} x_1^2/2 - x_2^2/2 \\ x_1x_2 \\ x_3^2 \\ x_4^2 \end{bmatrix}$ and $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = U\underline{y}$	$\underline{y}(t) = U'\underline{x}(t)$ where $\underline{x}(t) = \begin{bmatrix} \frac{-20(z_1+z_2)}{z_1^2+z_2^2} \\ \frac{-20(-z_1+z_2)}{z_1^2+z_2^2} \\ \frac{1000}{1-1001e^{1000t}} \\ \frac{0.001}{1-1.001e^{0.001t}} \end{bmatrix}$ $z_1 = 1 + e^{-10t}(9\cos 10t + 10\sin 10t)$ and $z_2 = e^{-10t}(-9\sin 10t + 10\cos 10t)$

PROPERTIES OF THE NUMERICAL SOLUTION							
Local Error Bound	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
SOLUTION VIA DIFJMT							
Maximum Global Error	7.11×10^{-6}	10.1×10^{-7}	20.2×10^{-8}	25.8×10^{-9}	87.3×10^{-10}	58.4×10^{-11}	66.8×10^{-12}
# of Time Steps	325	434	588	725	1043	1377	1768
# of Function Evaluations	775	1036	1353	1632	2477	3161	4022
# of Jacobian Evaluations	70	71	73	83	124	114	163
Final Step-Size (sec)	2.26×10^{3}	1.04×10^{3}	978	951	579	425	313
CPU Time (min $\times 10^{-3}$)	139	213	299	356	537	669	859
SOLUTION VIA DIFSUB							
Maximum Global Error	7.47×10^{-6}	5.07×10^{-7}	8.17×10^{-8}	13.0×10^{-9}	19.2×10^{-10}	27.7×10^{-11}	40.0×10^{-12}
# of Time Steps	1017	466	617	815	1584	1622	1978
# of Function Evaluations	2687	1226	1592	2148	3919	4420	5203
# of Jacobian Evaluations	37	34	42	43	43	40	48
Final Step-Size (sec)	1.63×10^{3}	1.97×10^{3}	1.53×10^{3}	484	339	270	281
CPU Time (min $\times 10^{-3}$)	431	207	275	382	703	740	883
SOLUTION VIA DIFBDF							
Maximum Global Error	4.14×10^{-6}	5.03×10^{-7}	5.82×10^{-8}	11.0×10^{-9}	16.2×10^{-10}	22.5×10^{-11}	35.4×10^{-12}
# of Time Steps	335	456	606	811	1111	1554	2038
# of Function Evaluations	745	1010	1331	1750	2384	3266	4283
# of Jacobian Evaluations	48	53	61	60	62	55	54
Final Step-Size (sec)	914	1.98×10^{3}	1.35×10^{3}	605	452	317	258

Table 5.5: Problem 3

DIFFERENTIAL EQUATION	SOLUTION
$\dot{y}(t) = By(t)$ with $y(0) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ where $B = \begin{bmatrix} -\sigma_1 & -\omega_1 & 0 \\ \omega_1 & -\sigma_1 & 0 \\ 0 & 0 & -\sigma_2 \end{bmatrix}$, $\sigma_1 = 10$, $\omega_1 = \sigma_1 \tan(55°) = 14.3$ and $\sigma_2 = 0.1$	$y(t) = \begin{bmatrix} e^{-\sigma_1 t} \cos \omega_1 t \\ e^{-\sigma_1 t} \sin \omega_1 t \\ e^{-\sigma_2 t} \end{bmatrix}$

PROPERTIES OF THE NUMERICAL SOLUTION							
Local Error Bound	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
SOLUTION VIA DIFJMT							
Maximum Global Error	9.03×10^{-6}	11.0×10^{-7}	11.3×10^{-8}	16.3×10^{-9}	24.8×10^{-10}	19.7×10^{-11}	68.8×10^{-12}
# of Time Steps	255	245	423	518	709	847	988
# of Function Evaluations	616	553	926	1134	1758	2046	2348
# of Jacobian Evaluations	45	38	45	61	137	135	134
Final Step-Size (sec)	1.2×10^{3}*	1.2×10^{3}*	215	519	869	244	1.2×10^{3}*
CPU Time (min $\times 10^{-3}$)	73	65	112	137	199	258	266
SOLUTION VIA DIFSUB							
Maximum Global Error	2.32×10^{-6}	2.58×10^{-7}	2.58×10^{-8}	3.99×10^{-9}	5.79×10^{-10}	8.59×10^{-11}	11.7×10^{-12}
# of Time Steps	232	273	23,551	26,619	23,692	23,813	23,987
# of Function Evaluations	568	664	68,704	60,410	69,203	67,842	69,987
# of Jacobian Evaluations	27	23	27	9	24	11	29
Final Step-Size (sec)	183	193	4.30×10^{-2}	4.19×10^{-2}	4.29×10^{-2}	4.30×10^{-2}	3.74×10^{-2}
CPU Time (min $\times 10^{-3}$)	52	59	5589	5429	5654	5648	5728
SOLUTION VIA DIFSDP							
Maximum Global Error	2.93×10^{-6}	2.97×10^{-7}	2.99×10^{-8}	4.21×10^{-9}	6.12×10^{-10}	8.62×10^{-11}	12.1×10^{-12}
# of Time Steps	191	255	366	4358	941	11,058	12,196
# of Function Evaluations	402	545	772	10,947	2025	31,881	31,237
# of Jacobian Evaluations	26	30	34	17	34	14	9
Final Step-Size (sec)	388	1.2×10^{3}*	275	0.245	261	9.44×10^{-2}	8.64×10^{-2}

*Maximum Step-Size Permitted.

Table 5.6: Problem 4

DIFFERENTIAL EQUATION	SOLUTION
$\dot{y}(t) = U'BUy(t)$ with $y(0) = \begin{bmatrix} 1/3 \\ 4/3 \\ 1/3 \end{bmatrix}$ where $U = \frac{1}{3}\begin{bmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ 2 & 2 & -1 \end{bmatrix}$ $B = \begin{bmatrix} -\sigma_1 & -\omega_1 & 0 \\ \omega_1 & -\sigma_1 & 0 \\ 0 & 0 & -\sigma_2 \end{bmatrix}$ $\sigma_1 = 10$ $\omega_1 = \sigma_1 \tan(55°) = 14.3$ and $\sigma_2 = 0.1$	$y(t) = U'x(t)$ where $x(t) = \begin{bmatrix} e^{-\sigma_1 t} \cos \omega_1 t \\ e^{-\sigma_1 t} \sin \omega_1 t \\ e^{-\sigma_2 t} \end{bmatrix}$

PROPERTIES OF THE NUMERICAL SOLUTION							
Local Error Bound	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}
SOLUTION VIA DIFJMT							
Maximum Global Error	10.4×10^{-6}	7.66×10^{-7}	12.8×10^{-8}	67.2×10^{-9}	20.1×10^{-10}	64.2×10^{-11}	24.1×10^{-12}
# of Time Steps	186	284	575	523	554	799	1054
# of Function Evaluations	433	638	1227	1225	1295	1885	2575
# of Jacobian Evaluations	36	47	48	70	75	107	160
Final Step-Size (sec)	385	275	1.07×10^{3}	173	$1.2 \times 10^{3*}$	359	474
CPU Time (min $\times 10^{-3}$)	63	100	176	168	191	268	364
SOLUTION VIA DIFSUB							
Maximum Global Error	1.95×10^{-6}	2.01×10^{-7}	2.74×10^{-8}	3.94×10^{-9}	5.62×10^{-10}	8.19×10^{-11}	13.3×10^{-12}
# of Time Steps	942	308	952	1162	2610	2700	2824
# of Function Evaluations	2540	765	2269	2805	7264	7303	7937
# of Jacobian Evaluations	24	26	28	30	28	26	30
Final Step-Size (sec)	191	86.3	63.5	43.6	229	77.6	36.9
CPU Time (min $\times 10^{-3}$)	281	98	277	343	795	812	875
SOLUTION VIA DIFBDF							
Maximum Global Error	2.58×10^{-6}	2.66×10^{-7}	2.74×10^{-8}	4.08×10^{-9}	6.06×10^{-10}	8.20×10^{-11}	11.8×10^{-12}
# of Time Steps	188	24,531	347	963	743	10,906	4961
# of Function Evaluations	400	52,744	716	1973	1524	31,969	10,597
# of Jacobian Evaluations	27	11	35	34	34	13	22
Final Step-Size (sec)	$1.2 \times 10^{3*}$	4.09×10^{-2}	$1.2 \times 10^{3*}$	687	1.13×10^{3}	9.58×10^{-2}	0.258

*Maximum step-size permitted.

Table 5.7: Problem 5

No CPU time is listed for the numerical solutions obtained using DIFBDF as this computer program was used only in a diagnostic mode making any time comparisons meaningless. The conditions for which the numerical solutions were obtained are contained in Table 5.8. Data tabulated in Tables 5.3 through 5.7 are for the mesh point solutions obtained up to and including the first mesh point beyond the final time specified in Table 5.8.*

Problem 1, whose solution is a decaying exponential, was chosen in order to obtain comparative data for the three algorithms without any interfering factors such as stiffness and non-linearities. As can be seen from Table 5.3, the numerical solutions as obtained from DIFSUB and DIFBDF are comparable in accuracy, number of function evaluations and number of Jacobian evaluations. The numerical solution as obtained using DIFJMT, however, is not as accurate, for a given requested local error. Moreoever, for local error bounds less than 10^{-10}, DIFJMT took more time steps and correspondingly more function evaluations. For higher error bounds, the opposite is true. In all cases DIFJMT required more Jacobian evaluations than both DIFSUB and DIFBDF, a point to be discussed later.

---

*For DIFJMT and DIFBDF the final mesh point included in the summaries may be 2 or 3 points beyond the final time, since, for each call, the solution at 3 or 4 mesh points is computed.

Problem	1	2	3	4	5
Initial Step-Size (sec)	2×10^{-6}	2×10^{-6}	2×10^{-6}	10^{-8}	10^{-8}
Minimum Step-Size Permitted (sec)	10^{-6}	10^{-10}	10^{-10}	10^{-10}	10^{-10}
Maximum Step-Size Permitted (sec)	4	10^{4}	10^{4}	1.2×10^{3}	1.2×10^{3}
Final Time (sec)	10	10^{4}	10^{4}	10^{3}	10^{3}

Table 5.8: Conditions Used For the Numerical Solutions

Problem 2, suggested by Krogh [KRO70], is both non-linear and stiff. The Jacobian matrix J is given as

$$J = U' \times \begin{pmatrix} -1000 + 2x_1 & 0 & 0 & 0 \\ 0 & -800 + 2x_2 & 0 & 0 \\ 0 & 0 & 10 + 2x_3 & 0 \\ 0 & 0 & 0 & -0.001 + 2x_4 \end{pmatrix} \times U, \quad (5.43)$$

where U and $\underset{\sim}{x}$ are as in Table 5.4. The Jacobian matrix at $t = 0$ is

$$J_o = U' \times \begin{pmatrix} -1002 & 0 & 0 & 0 \\ 0 & -802 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & -2.001 \end{pmatrix} \times U, \quad (5.44)$$

and as $t \to \infty$ is

$$J_\infty = U' \times \begin{pmatrix} -1000 & 0 & 0 & 0 \\ 0 & -800 & 0 & 0 \\ 0 & 0 & -12 & 0 \\ 0 & 0 & 0 & -0.001 \end{pmatrix} \times U. \quad (5.45)$$

Again, the numerical solutions obtained using DIFSUB and DIFBDF are comparable* with DIFSUB requiring more function evaluations but fewer Jacobian evaluations. Using DIFJMT

---

*For an unexplainable reason DIFSUB aborted when a local error bound of 10^{-11} was requested.

the number of time steps required was comparable but the number of function and Jacobian evaluations were significantly higher. The maximum global errors for DIFJMT are worse by approximately a factor of three for all test cases run. As the eigenvalues of the Jacobian matrix are all real it was expected that the numerical results would be comparable for all three algorithms. Note: The extra function and Jacobian evaluations used by DIFJMT will be discussed later.

Problem 3, also suggested by Krogh [KRO70], is similar to Problem 2 except that the Jacobian matrix has a pair of complex eigenvalues. The Jacobian matrix, J, is given as

$$J = U' \times \begin{pmatrix} 10 + x_1 & 10 - x_2 & 0 & 0 \\ -10 + x_2 & 10 + x_1 & 0 & 0 \\ 0 & 0 & -1000 + 2x_3 & 0 \\ 0 & 0 & 0 & -0.001 + 2x_4 \end{pmatrix} \times U, \quad (5.46)$$

where U and $\underset{\sim}{x}$ are as in Table 5.5. As t increases from 0 the Jacobian matrix goes from J_o to J_∞ where

$$J_o = U' \times \begin{pmatrix} 8 & 10 & 0 & 0 \\ -10 & 8 & 0 & 0 \\ 0 & 0 & -1002 & 0 \\ 0 & 0 & 0 & -2.001 \end{pmatrix} \times U, \quad (5.47)$$

and

$$J_\infty = U' \times \begin{pmatrix} -10 & -10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 0 & 0 & -1000 & 0 \\ 0 & 0 & 0 & -0.001 \end{pmatrix} \times U. \qquad (5.48)$$

As DIFJMT is more ideally suited to the numerical integration of differential equations with complex eigenvalues than DIFSUB and DIFBDF* "better" results obtained with DIFJMT were expected. Again, the maximum global errors contained in Table 5.5 are higher for DIFJMT than for the other two algorithms. Moreoever, the average number of Jacobian and function evaluations per time step are higher for DIFJMT.

Problem 4 was specifically chosen to exhibit the improved stability properties of the new methods over the backward differentiation formulas. The Jacobian matrix has a pair of complex eigenvalues at $-\sigma_1 \pm j\omega_1$ and a negative real eigenvalue at $-\sigma_2$, where σ_1, ω_1, and σ_2 are as in Table 5.6. As the ratio $\sigma_1/\sigma_2 = 100$ the problem is stiff. Moreoever, since $\tan^{-1}(\omega_1/\sigma_1) = 55°$ only those methods that are A(α)-stable for $\alpha > 55°$ are asymptotically stable for all step-sizes. For this problem, and for the new methods presented only the order 7 method is not asymptotically

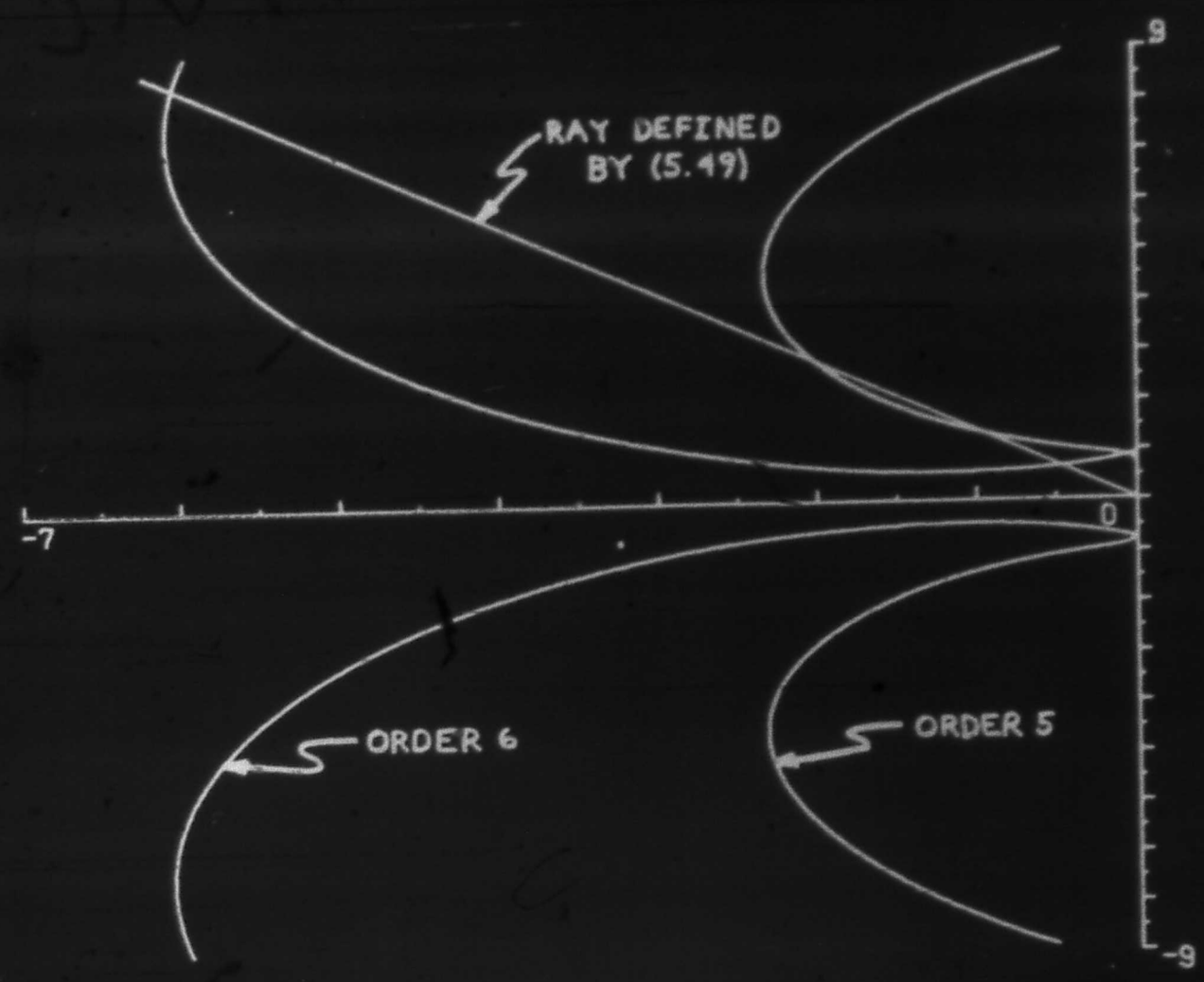
---

*The Lambda Loci for both of these algorithms are identical.

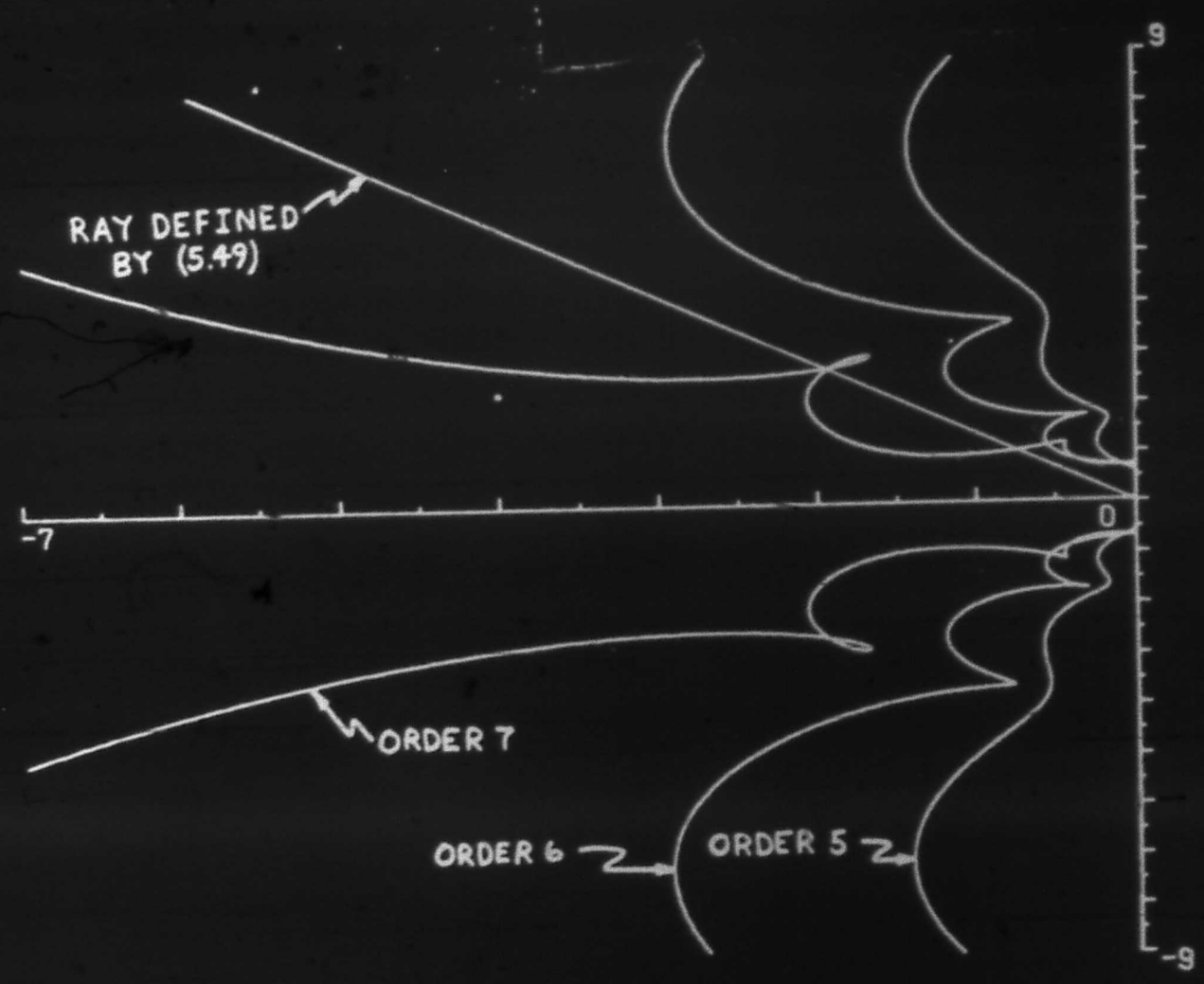
stable for all step-sizes. However, neither the order 5 nor the order 6 backward differentiation formulas are asymptotically stable for all h for this problem. For a given step h, the point in the λ-plane corresponding to the eigenvalue $-\sigma_1 + j\omega_1$ is, with $\sigma = \mathrm{Re}\{\lambda\}$ and $\omega = \mathrm{Im}\{\lambda\}$, $\sigma + j\omega = -\sigma_1 h + j\omega_1 h$. The locus of points for $h \in (0, \infty)$ is the open ray in the second quadrant of the complex plane defined by

$$\omega = -\frac{\omega_1}{\sigma_1}\sigma. \tag{5.49}$$

This ray together with the Lambda Loci for the order 5 and order 6 BDF and the Lambda Loci for the new methods of orders 5, 6, and 7, is shown in Figure 5.2. As the step-size increases the operating point in the λ-plane "moves out" along the ray defined by (5.49). If, for a given method and a given step-size h, $\sigma_1 h + j\omega_1 h$ is in an unstable region of the λ-plane—that is, the local error would increase due to numerical instability—the algorithm should detect the condition and either decrease the step, descrease the order, or both to satisfy the error criterion. From Figure 5.2, for the order 6 BDF the critical step-size for Problem 4 is approximately 0.04 whereas for the order 5 BDF it is approximately 0.1. Table 5.6 indicates that for local error bounds greater than 10^{-7} the final step-size used by DIFSUB is approximately 0.04. Though not shown in Table 5.6, for these



a) Backward Differentiation Formulas



b) New Methods

Figure 5.2: Lambda Loci and Ray Defined by (5.49) for Problem 4

local error bounds an order 6 method was in use by DIFSUB at the final time—that is, DIFSUB was "locked in", properly detecting that an increase in h would cause the local error to increase beyond acceptable limits. It did not, however, recognize that the step could be increased with a decrease in order. Similarly, DIFBDF got "locked in" with an order 5 method for local error bounds specified at 10^{-11} and 10^{-12}.* However, in all of the test cases run, DIFJMT was able to successfully increase h after the transient effects due to the eigenvalues at $-\sigma_1 \pm j\omega_1$ decayed. Note: Though the new order 7 method suffers the same disadvantage as the order 5 and 6 backward differentiation formulas, DIFJMT was not "caught". This is partially due to the order selection logic integral to DIFJMT. It is this same logic that most likely accounts for DIFBDF not getting "caught" with an order 6 method, though it did fail to decrease the order below 5.

As expected, the numerical solution for Problem 4 was obtained in significantly fewer time steps using DIFJMT than with DIFSUB. An improvement over DIFBDF is also apparent, though it is not as dramatic. As in other problems, the maximum global errors were not as low for solutions obtained by DIFJMT, when compared to the other two algorithms. Again, the average number of function and Jacobian evaluations per

---

*DIFBDF also got temporarily "locked in" when a local error bound of 10^{-9} was specified but was able to eventually increase h by decreasing the order.

time point were higher with DIFJMT.

Problem 5 is similar to Problem 4 except that the system is now coupled—that is, the Jacobian matrix J is given as

$$J = U' \times \begin{bmatrix} -\sigma_1 & -\omega_1 & 0 \\ \omega_1 & -\sigma_1 & 0 \\ 0 & 0 & -\sigma_2 \end{bmatrix} \times U, \qquad (5.50)$$

where U, σ_1, ω_1, and σ_2 are as in Table 5.7. Note: σ_1, ω_1, and σ_2 are the same as for Problem 4. The numerical solution was obtained in fewer time steps using DIFJMT than with DIFSUB or DIFBDF. The improvement, though, is not as dramatic as for Problem 4.

The following conclusions can be drawn from the results obtained for the five problems tested:

1. For non-stiff problems and stiff problems wherein the eigenvalues of the Jacobian matrix are real, numerical results obtained using the three algorithms require a comparable number of solution points. For stiff problems wherein the eigenvalues of the Jacobian matrix are complex with negative real parts DIFJMT requires fewer solution points.

2. DIFJMT requires more function and Jacobian evaluations per solution point than do either DIFSUB or DIFBDF.

3. Numerical solutions obtained using DIFJMT have a somewhat higher global error than do numerical solutions obtained with either DIFSUB or DIFBDF.

4. The means used for storing information has an effect on the properties of the numerical solution.

It is felt that the higher number of function evaluations and Jacobian evaluations per mesh point required by DIFJMT to numerically integrate a differential equation are due to the predictor formulas used. Though the predictor does not affect stability—as the corrector is iterated to convergence—it does affect the convergence properties of the corrector. The modified Newton-Rhapson iteration described in Section 5.2.1 for solving the implicit equations is dependent on a predicted value that is "close" to the true solution. Using the same predictor for all ℓ equations in (5.1) may not be the optimal strategy. A study of diagnostic output obtained from DIFJMT revealed that the corrector failed to converge significantly more often than for DIFSUB and DIFBDF* resulting in more Jacobian evaluations and correspondingly more function evaluations. In turn, this detracted from the results tabulated for DIFJMT.†

---

*Both DIFSUB and DIFBDF use the same scheme as DIFJMT for solving the implicit equations.

†Note: The step-size is decreased by a factor of 4 if the corrector fails to converge at the same mesh point on two successive attempts. Hence the number of solution points required is also increased if this condition occurs.

The author has not been able to explain the higher maximum global errors observed when DIFJMT was used as compared to the numerical results obtained using either DIFSUB or DIFBDF. On the average, the maximum global errors observed were worse by a factor of approximately 3½ for DIFJMT.

That numerical results obtained from DIFSUB and DIFBDF—two algorithms based on the same methods, namely the backward differentiation formulas—are not comparable for all test cases implies that the means for storing information affects the properties of the numerical solution. Note: DIFSUB uses the Nordsieck vector whereas DIFBDF stores information in the form of backward differences.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

New techniques were obtained by which the stability properties of composite linear multistep methods may be accurately characterized. The relationship of the Lambda and Zeta Loci to those stability properties was established. A computer program for plotting both the Lambda and Zeta Loci was presented and described. Using an interactive version of this program new and efficient cyclic composite linear multistep methods of orders three through seven were obtained. These new methods form the basis for a new variable step, variable order integration algorithm described.

This new algorithm, designed for the numerical integration of stiff systems of first order ordinary differential equations, exhibits better stability properties than available algorithms based on the backward differentiation formulas. It is ideally suited for the numerical solution of stiff systems wherein the Jacobian matrix has complex eigenvalues. Numerical results to support this claim were presented, including a comparison of the new algorithm with Gear's popular computer program DIFSUB and with a third algorithm based on the backward differentiation formulas.

The new algorithm, however, exhibits a slightly higher maximum global error than the two computer programs

used as a baseline. Moreover, the predictor formulas used in the new algorithm affect the convergence properties of the component formulas and detract from the results obtained. Further research is required to obtain "compatible" predictor formulas.

As the new algorithm presented herein must interpolate new mesh points each time the integration step is changed--a cost that goes up linearly with the number of differential equations being solved and the order method being used--effort should be expended to reformulate these new methods in terms of modified backward differences (equivalently, divided differences) wherein interpolation is not required each time the integration step-size is changed--an approach used by Brayton, Gustavson, and Hachtel [BRA72a] and also by Krogh [KRO72].

Stability properties of a composite linear multistep method were considered only in terms of a fixed step-size. As this is rarely the situation research is required to investigate the stability characteristics when the step-size varies--see, for example, the recent paper by Brayton and Conley [BRA72b].

The new methods presented were obtained solely on the basis of optimizing the Widlund wedge angle associated with the method. It is felt that attention should also be focused on simultaneously minimizing the local discretization

error. In addition, higher order stiffly stable methods should be derived.

The test for stiff stability of a composite linear multistep method presented herein is entirely graphical. For linear multistep method--that is, multistep methods with $\ell = 1$--Nørsett [NOR69] has presented an analytic condition that can be tested to determine if a method is $A(\alpha)$-stable for some $\alpha \varepsilon (0,\pi/2)$. Rubin and Bickart [RUB72] reported an analytic condition for testing whether or not a composite linear multistep method is A-stable. An expanded treatment of that test and of related topics can be found in [RUB73]. Research is required to obtain analytic conditions by which to test for stiff stability.

APPENDIX A

COMPUTER PROGRAM LISTING FOR THE BASIC PLOTTING PROGRAM

This appendix contains a computer program listing for the basic plotting program described in Section 3.5. The particular version of subroutine ACCEPT listed reads the characteristic equation for a given method directly from cards. The input card format is described with the listing of ACCEPT (deck identification "ACPCD").

The program was written for execution on an XDS SIGMA-5 computer. The program is in non-standard FORTRAN making full use of the power of XDS EXTENDED FORTRAN-IV. In addition, it was compiled using the "ADP" option wherein all variables that would normally be typed real are automatically considered double precision.* Similarly, variables that are declared complex are automatically considered double precision. Under the "ADP" option, the compiler automatically sets up the calls to the double precision FORTRAN intrinsic functions rather than the single precision intrinsics. As an example, calls to the double precision sine function can be coded as either SIN or DSIN.

---

*On the XDS SIGMA-5 computer double precision corresponds to 56 binary bits of mantissa, or approximately 16.8 decimal digits of accuracy.

If it is desired to implement Corollary 3.12 to test if a method is stiffly stable for a particular value of γ of Definition 3.6, card number 77 of the MAIN program segment (deck identification "MAIN") should be changed to read "VAR = (γ, 0.)". No other changes are required.

```
C
C     10JUNE1972
C
      DIMENSION CHAR(0/10,0/10)
      DIMENSION ILOCUS(300)
      COMPLEX POLY(0/17), W88T(1510) ,  VAR, CTEMP
      DOUBLE PRECISION ID
      COMMON /METHOD/ICHAR(0/10,0/10), NZ, NL, ID, INCRP, INCR, ITIME
      REAL ICHAR
      COMMON RHOT, ILOCUS, IFIRST
      COMMON /WORKAREA/  WORK(75), CHAR, POLY
      COMMON /OVERLAY/ IDUMMY(48)
      COMMON /LSERINFO/ JDUMMY(8)
C
          DIMENSION XVAR(2)
          EQUIVALENCE (XVAR(1), VAR), (Y, XVAR(2))
C
      DATA PI/3.141592653589793/
      DATA EPS /1.3D-17/
      DATA AI, AF /0.01, 100.0/
C
C             FOLLOWING EQUIVALENCE STATEMENTS ARE USED
C             SOLELY TO CONSERVE SPACE
      EQUIVALENCE (WORK(30), VAR), (WORK(32), DELTA), (WORK(33), THETA),
     1          (WORK(34), CTEMP), (WORK(36), B(0), (WORK(37), T)
      EQUIVALENCE (WORK(40), IRETURN), (WORK(41),IBACK), (WORK(42), IOR)
      EQUIVALENCE (WORK(43), I), (WORK(44), J), (WORK(45), IER),
     1          (WORK(46), NDEGREE), (WORK(47), ICRM), (WORK(48), ISTART)
      EQUIVALENCE (WORK(49), IREPEAT), (WORK(50), ILREPEAT),
     1          (WORK(51), IRESTART), (WORK(52), INEW),
     2          (WORK(53), JRROT), (WORK(54), JSROOT), (WORK(55),INDROOT)
      EQUIVALENCE (WORK(56), IFRL), (WORK(57), INCRPLOT)
C
C         INITIALIZE PLOTTER
C
      IFIRST = 0
      IPLOT = 0
C
C     ******************************************************************
C     *                                                                *
C     *   READ IN METHOD AS AS INTEGER CHARACTERISTIC EQUATION         *
C     *   INTO COMMON BLOCK METHOD WHERE...                            *
C     *   CHARACTERISTIC EQUATION IS                                   *
C     *              NZ   NL                                           *
C     *   P(Z,L) =  SUM  SUM  ICHAR(I,J)*(L**I)*(Z**J)                 *
C     *              I=0  J=0                                          *
C     *   IPLOT HAS THE FOLLOWING VALUES...                            *
C     *     1  PLOT LAMDA LOCUS, I.E.. LET Z=EXP(I*PHI),               *
C     *          PHI=.PI(P)/INCR)0                                     *
C     *     2  PLOT ZETA LOCUS, I.E.. LET LAMDA=(0,V),                 *
C     *          V=100(100/INCR)100                                    *
C     *   ID IS AN INTEGER 2 BY 1 ARRAY (2A4 FORMAT) USED AS A PLOT ID *
C     *                                                                *
C     ******************************************************************
C
C         ISPACE SHOULD BE SET TO THE DIMENSION OF ARRAY ROOT
C         IF THE DIMENSION OF ARRAY ROOT IS CHANGED, CHANGE NEXT CARD
   10 ISPACE = 1500
      CALL INPUT (ISPACE)
      INCRPLOT = INCR
C
C         SET UP REAL CHARACTERISTIC EQUATION MATRIX
      GO TO (20,30), IPLOT
C
C         LAMDA LOCUS
   20 NROOTS = NL
      NDEGREE = NZ
      VAR = (1.0, 0.0)
      DELTA = PI/INCR
      THETA=-PI
      PRINT 25
   25 FORMAT (////9LAMDA LOCUS BEING PLOTTED0)
      GO TO 40
C
C         ZETA LOCUS
   30 NROOTS = NZ
      NDEGREE = NL
      VAR = (0.0, 0.0)
      THETA = 1./INCR
      DELTA = (AF/AI)**THETA
      PRINT 35
   35 FORMAT (////9ZETA LOCUS BEING PLOTTED0)
C
C         SET UP CHARACTERISTIC EQUATION
C         AS A FUNCTION OF PLOT VARIABLE
   40 DO 70 I=0,NDEGREE
        DO  70 J=0,NROOTS
          GO TO (50, 60), IPLOT
   50     CHAR(I,J) = ICHAR(J,I)
          GO TO 70
   60     CHAR(I,J) = ICHAR(I,J)
   70     CONTINUE
      PRINT 71, ID
   71 FORMAT (0PLOT IDENTIFICATION = '0A8'0)
      PRINT 73, NROOTS, Z=INCRPLOT0
   72 FORMAT (X8I0I 0LOCI; 0I 0POINTS/LOCUS10)
   73
C
C         COMPUTE STARTING POINT OF PLOT FOR EACH LOCUS
      ISTART = 0
      ASSIGN 76 TO IRETURN
      GO TO 1000
   76 GO TO (78, 77), IPLOT
   77 Y = AI
      ASSIGN 80 TO IRETURN
   79 CALL CSMROOTS (POLY, WORK, NROOTS, ROOT, IER)
C
C         CHECK ON START-UP PROCEDURE
C         IER CAN ONLY ASSUME THE VALUES 0, 3, 4
C           IER = 0 ---> RESULT CORRECT
C           IER = 3 ---> S/R CMROOTS DID NOT CONVERGE
C           IER = 4 ---> POLYNOMIAL NOT OF FULL DEGREE
C         PARAMETER ISTART USED AS COUNTER FOR # OF TRIES ATTEMPTED
C           ISTART = 0 ---> 1ST ATTEMPT
C           ISTART = 1 ---> RESTART WAS ALREADY ATTEMPTED
C           ISTART = 2 ---> 2 RESTARTS ATTEMPTED (ABORT)
C         IFRL USED TO KEEP IER FOR LAST RESTART (ONLY IF ISTART = 2)
```

```
      IF (IER.E3) 99, 86, 81
C
C        START-UP PROCEDURE FAILED DUE TO POLYNOMIAL NOT OF FULL DEGREE
C
 81   IF ((ISTART.EQ.1) .OR. (IPLOT.EQ.2)) GO TO 8105
      DO 8101 I=0,NDEGREE
        IF (CHAR(I,NROOTS).NE.(0.,0.)) GO TO 8105
 8101   CONTINUE
      IF ((NROOTS - NROOTS-1).EQ.0) GO TO 86
      INCRPLOT = ((SPACE - NROOTS)/NROOTS
      PRINT 8102
8102 FORMAT (X,NOTE: HIGHEST ORDER POLYNOMIAL IN LAMDA ALL ZEROS /
     1       7X,LAMDA PLOT BEING ADJUSTED ACCORDINGLY.)
      GO TO 72
8105  IF ((ISTART.EQ.2) .OR. ((ISTART.EQ.1).AND.(IERL.EQ.4))) GO TO 86
      ISTART = ISTART+1
      IERL = 4
      PRINT 82
 82   FORMAT(/,5X,R CHARROTS INDICATES P(ZETA,LAMDA) NOT OF FULL DEGREE
     1          AT STARTING VALUE.)
      PRINT 83
 83   FORMAT (X,START-UP PROCEDURE BEING ADJUSTED.)
      INCRPLOT = INCRPLOT+1
      GO TO (900, 950), IPLOT
C
C        START UP PROCEDURE FAILED---ABORT
 86   PRINT 87, IER
 87   FORMAT (X,*****RESTART ATTEMPT FAILED(S/R CHARROTS INDICATES IER=,I
     1        =I3)/X,*****RUN ABORTED*****.)
      GO TO 10
C
C        START-UP PROCEDURE FAILED DUE TO NO CONVERGENCE
C          CHECK TO SEE IF WE HAVE ALREADY RESCALED
 86   IF ((ISTART.EQ.2) .OR. ((ISTART.EQ.1).AND.(IERL.EQ.3))) GO TO 86
      BIG = 0
      DO 89 I=0,NDEGREE
        DO 89 J=0,NROOTS
          IF ((Y=ABS(CHAR(I,J))).GT.BIG) BIG = Y
 89       CONTINUE
      DO 90 I=0,NDEGREE
        DO 90 J=0,NROOTS
 90       CHAR(I,J) = CHAR(I,J)/BIG
      PRINT 91, BIG
 91   FORMAT (/,5X,R CHARROTS FAILED TO CONVERGE AT STARTING VALUE./
     1          X,P(ZETA,LAMDA) RESCALED BY ,G23.17)
      PRINT 83
      IF ((ISTART.EQ.0) Y=0.0
      ISTART = ISTART + 1
      IERL = 3
      GO TO 1000
C
C        REFINE START-UP VALUES
 92   DO 93 J=1,NROOTS
 93     CALL REFCROOT (POLY, ROOT(J), NROOTS, ROOT(J), EPS, IER)
C
C        PRINT OUT STARTING VALUES FOR EACH LOCUS
      PRINT 94
 94   FORMAT(/,STARTING VALUES FOR EACH LOCUS.)
      INDROOT = 1
      ASSIGN 95 TO IRACK
      GO TO 2000
C
C
C        COMPUTE REMAINDER OF LOCUS
 95   INEW = 1
      ILREPEAT = 0
      ASSIGN 100 TO IRETURN
      DO 151 ICOMP,INCRPLOT+1
C
C        EVALUATE POLY(J) AT NEXT PLOT POINT
      GO TO (900, 950), IPLOT
C
C        CHECK IF PREVIOUS POINT HAS MULTIPLE ROOTS (COMMON LOCUS)
 100  IREPEAT = 0
      JROOT=INDROOT
      JSROOT = INDROOT+NROOTS
      IF (NROOTS.EQ.1) GO TO 148
      DO 110 I=INDROOT, INDROOT+NROOTS-2
        CTEMP = ROOT(I)
        DO 105 J=I+1, INDROOT+NROOTS-1
          IF (CABS(ROOT(J)-CTEMP).LT.EPS) IREPEAT=IREPEAT+1; GO TO 110
 105    CONTINUE
 110  CONTINUE
      IF (IREPEAT.EQ.0) IGO=1; ILREPEAT=IRESTART=0; GO TO 148
      IF (IREPEAT.EQ.ILREPEAT) IF (IRESTART=0) 115, 115, 148
      ILREPEAT = IREPEAT
      IRESTART = 0
      IGO = 1
 115  IRESTART = IRESTART +1
      IF (INEW.GT.900)GO TO 120
      ILOCUS(INEW) = ICOM
      INEW = INEW+1
      GO TO 130
 120  PRINT 125
 125  FORMAT (/,PLOTS MAY NOT SHOW RESTARTS FROM THIS POINT ONW.)
      ASSIGN 130 TO IBACK
      GO TO 2000
C
C        RESTART SEQUENCE
 130  CALL CHARROTS (POLY, WORK, NROOTS, ROOT(JSROOT), IER)
      PRINT 132, ICOM
 132  FORMAT (*****RESTART ATTEMPTED AT PLOT POINT , I, ,X, ,*****.)
      JROOT = INDROOT+NROOTS
      IF (IER.EQ.0) GO TO 148
      PRINT 135, ICOM, IER
 135  FORMAT (/,*****S/R CHARROTS FAILED DURING RESTART ON ITERATION ,
     1          I, ,(IER = , I, ,)/,*****RUN TERMINATED.)
      GO TO 10
 140  GO TO (145, 148), IGO
 145  IGO = 2
      PRINT 146, ICOM
 146  FORMAT (/,RESTART ABANDONED AT ITERATION ,I ,)
C
C        USE NEWTON-RAPSON ITERATION
C
 148  DO 150 J=0,NROOTS-1
        CALL REFCROOT
     1          (POLY, ROOT(J+JROOT), NROOTS, ROOT(J+JROOT), EPS, IER)
        IF (IER.EQ.0) GO TO 150
        PRINT 149, IER, ICOM, J+1
```

```
149     FORMAT(///6***S/R BEFCROOT FAILED WITH IERR=,I5,
     1          *AFTER *, I, *ITERATIONS ON LOCUS *, I5
        ASSIGN 190 TO IBACK
        GO TO 2000
150     CONTINUE
      INDROOT = INDROOT+NROOTS
151   CONTINUE
C
C       PRINT OUT MID-POINT VALUES FOR EACH LOCUS
      PRINT 152
152   FORMAT(/*MID-POINT VALUES FOR EACH LOCUS*)
      ASSIGN 190 TO IBACK
      GO TO 2000
C
C       PLOT LOCI
190   IF (INEW.LE.300) ILOCUS(INEW)=0
      CALL PLOTTER (NROOTS, ICOM,1, ID, IPLOT)
      GO TO 10
C     ***********************************************************
C     *                                                         *
C     *       THIS SECTION COMPUTES THE NEXT PLOT POINT (VAR)   *
C     *                                                         *
C     ***********************************************************
C
C       LAMDA LOCUS---Z=EXP(I*PHI)
900   THETA = THETA + DELTA
      VAR = CMPLX (COS(THETA), SIN(THETA))
      GO TO 1000
C
C       ZETA LOCUS
950   Y = Y+DELTA
C     ***********************************************************
C     *                                                         *
C     *   THIS SECTION FORMS THE POLYNOMIAL                     *
C     *       NROOTS                                            *
C     *         SUM  POLY(J)*(Y**J)                             *
C     *         J=0                                             *
C     *   WHERE                                                 *
C     *                     NDEGREE                             *
C     *         POLY(J) =    SUM   CHAR(I,J)*(VAR**I)           *
C     *                      I=0                                *
C     *   ON COMPLETION CONTROL IS RETURNED TO STATEMENT NO. IRETURN
C     *                                                         *
C     ***********************************************************
1000  DO 1020 J=0,NROOTS
        CTEMP= CHAR(NDEGREE,J)
        DO 1010 I=NDEGREE-1,0,-1
1010      CTEMP = CTEMP*VAR + CHAR(I,J)
1020    POLY(J) = CTEMP
      GO TO IRETURN, (70, 80, 100)
C     ***********************************************************
C     *                                                         *
C     *       THIS SECTION PRINTS OUT THE VALUE OF              *
C     *       ROOT(INDROOT) THRU ROOT(INDROOT+NROOTS-1)         *
C     *       AND RETURNS TO IBACK
```

```
C     *                                                         *
C     ***********************************************************
C
2000  PRINT 2010
2010  FORMAT (T10, *LOCUS*, T19, *REAL PART*, T36, *IMAGINARY PART*)
      DO 2020 I=INDROOT, INDROOT+NROOTS-1
2020      PRINT 2030, I-INDROOT+1, ROOT(I)
2030    FORMAT (T10, I3, T17, F14.8, T34, E14.8)
      GO TO IBACK, (100, 130, 190)
C-----------------------------------------------------------------
      END
```

```
      SUBROUTINE PLOTTER (NLOCUS, NPTS, ID, IPLOT)
C
C     12JUNE1972
C
C     ------------------------------------------------------------------
C         THIS S/R PLOTS THE LOCI AS GIVEN IN COMMON BLOCK LOCUS
C
C     VARIABLES IN ARGUMENT LIST HAVE FOLLOWING MEANING...
C      NPTS    NUMBER OF PTS PER LOCUS TO BE PLOTTED
C      NLOCUS  NUMBER OF LOCI TO BE PLOTTED
C      ID      A ALPHANUMERIC CHARACTER USED FOR PLOT IDENTIFICATION
C      IPLOT   DISCRETE INDICATING TYPE OF PLOT DESIRED HAVING
C              FOLLOWING VALUES
C        1       PLOT LAMDA LOCUS (ONLY IN RANGE -20. TO +20.
C                  IN BOTH REAL AND IMAGINARY PARTS)
C        2       PLOT ZETA LOCUS (ONLY FOR THOSE PTS WHOSE MAGNITUDE
C                  IS LESS THAN 1.5 ARE PLOTTED)
C
C     THE LOCUS IS PLOTTED FROM THOSE PTS IN BLANK COMMON
C     POINTS ARE STORED IN ARRAY ROOT DIMENSION (IN MAIN PROGRAM)
C     AS ROOT(NLOCUS,NPTS).   ROOT(I,J) IS THE I.TH POINT OF LOCUS J.
C     ROOT IS A COMPLEX ARRAY
C
C     TWO SENSE LIGHTS ARE USED AS FOLLOWS...
C         SENSE LIGHT 1 ON OVERRIDES AUTOMATIC AXIS SCALING
C                AND USES XMAX, XMIN, YMAX AS GIVEN IN COMMON /OVERLAY/
C                IF SENSE LIGHT 1 IS OFF, THEN A NEW PAGE IS
C                UNCONDITIONALLY STARTED FOR THE PLOT AND THE PROGRAM
C                AUTOMATICALLY SETS THE AXIS SCALING.  THIS OPTION
C                CAN ONLY BE EXERCISED IF IPLOT = 1 (LAMDA LOCUS).
C                THE STATUS OF SENSE LIGHT 1 IS NEVER CHANGED BY
C                SUBROUTINE PLOTTER.
C         SENSE LIGHT 2 ON CAUSES THE CURRENT PLOT TO BE PLOTTED ON A
C                NEW PAGE.  SENSE LIGHT 2 IS ONLY CHECKED IF SENSE
C                LIGHT 1 IS ON.  IF SENSE LIGHT 1 IS ON AND SENSE
C                LIGHT 2 IS OFF, THE CURRENT PLOT WILL BE PLOTTED
C                ON TOP OF THE PREVIOUS PLOT.  ON RETURN TO THE
C                CALLING PROGRAM, SENSE LIGHT 2 WILL BE OFF, REGARDLESS
C                OF ITS SETTING ON ENTRY. IF IPLOT = 1 (LAMDA LOCUS).
C                IF IPLOT = 2 (ZETA LOCUS), THEN SENSE LIGHT 2 WILL BE SET
C                ON RETURN TO THE CALLING PROGRAM.
C         THE FIRST TIME SUBROUTINE PLOTTER IS CALLED, SENSE LIGHT 1 IS
C                NOT INTERROGATED, BUT ACTION PROCEEDS AS IF IT WAS OFF,
C                I.E., SUBROUTINE PLOTTER AUTOMATTICALLY PERFORMS
C                AXIS SCALING AND STARTS A NEW PAGE.
C         IF A LAMDA LOCUS HAS PLOTTED (IPLOT = 1), THEN THE JUST USED
C                VALUES FOR XMIN, XMAX, AND YMAX ARE RETURNED IN
C                COMMON BLOCK /OVERLAY/ IN THE ORDER JUST GIVEN.
C         TWO INTERNAL SWITCHES ARE SET TO REFLECT THE PLOT TYPE AS
C         FOLLOWS....
C                ISCALE = 1(2) TO INDICATE THAT AUTOMATIC AXIS
C                  SCALING ARE (NOT) TO BE PERFORMED.
C                NEWPAGE = 1(2) TO INDICATE A NEW PAGE IS (NOT)
C                  TO BE STARTED THIS TIME.
C
C         IF BOTH SENSE LIGHTS 1 AND 2 ARE SET, BUT THE PREVIOUS VALUES OF
C             XMIN, XMAX AND YMAX ARE NOT INTEGER VALUES, THEN THE AXIS ARE
C             DRAWN, BUT NO AXIS.LABELING IS WRITTEN, NOR ARE ANY AXIS
C             MARKERS DRAWN.  IN ALL CASES THE PLOTTER SCALING USED FOR
C             THE AXIS ARE PRINTED ON THE LINE PRINTER
C
C         IF A LAMDA PLOT IS PLOTTED (IPLOT=1) THEN THE STABILITY MEDGE,
C             IN THE SENSE OF WIDLUND, I.E., A(ALPHA) STABILITY, IS
C             IS PRINTED ON THE LINE PRINTER.  ALPHA IS PRINTED IN
C             DEGREES.  THE POINT AT WHICH IT WAS DETERMINED IS ALSO PRINTED
C     ------------------------------------------------------------------
C
      DIMENSION X(2,1), ZPRINT(2)
      DIMENSION ILOCUS(300)
      COMPLEX PRINT
      DOUBLE COMPLEX Z(1910), ROOT(1)
      DOUBLE PRECISION X, ID
      EQUIVALENCE (Z, X, ROOT)
      COMMON Z, ILOCUS, IFIRST
      EQUIVALENCE (PRINT, ZPRINT(1), U1, (ZPRINT(2), V)
      COMMON /WORKAREA/ WORK(109)
      COMMON /USERINFO/ XMIN1, XMAX1, YMAX1, ALPHA
C
      DATA PI /3.141592653589793/
      COMPLEX CTEMP
C
C                 FOLLOWING EQUIVALENCE STATEMENTS ARE USED
C                 SOLELY TO CONSERVE SPACE
C
      EQUIVALENCE (WORK(1), POINT), (WORK(9), ANGLE),
     1         (WORK(4), BIG), (WORK(5), SMALL),
     2         (WORK(6), XS, XSS), (WORK(7), XB, XBB),
     3         (WORK(8), YS), (WORK(9), YB),
     4         (WORK(10), XMIN), (WORK(11), XMAX),
     5         (WORK(12), YMIN),(WORK(13), YMAX)
      EQUIVALENCE (WORK(14), XP), (WORK(15), YP),
     1         (WORK(16), XA), (WORK(17), YA), (WORK(18), FCTR),
     2         (WORK(19), TC), (WORK(20), CTEMP)
      EQUIVALENCE (WORK(22), T), (WORK(23), XALPHA), (WORK(24), YALPHA)
      EQUIVALENCE (WORK(26), XMIN), (WORK(27), XMAX), (WORK(28),YMAX)
      EQUIVALENCE (WORK(30), I), (WORK(31), J), (WORK(32), N),
     1         (WORK(33), NSMALL), (WORK(34), ITEM), (WORK(35), IREPEAT)
      EQUIVALENCE (WORK(40), IFORMAT), (WORK(41),LABEL),
     1         (WORK(42), IRETURN), (WORK(43), ICONTINU),
     2         (WORK(44), ISCALE), (WORK(45), NEWPAGE)
C
C         INITIALIZE PLOTTER FOR EITHER 1-ST OR SUBSEQUENT CALLS
C
      IF (IPLOT.EQ.1) ALPHA = PI/2.0; XALPHA = YALPHA = 0.0
      N = NPTS*NLOCUS
      IF (IFIRST.NE.0) GO TO 5
C
C         FIRST CALL THIS JOB
C
      CALL PLTMED
      CALL PLOT (3.0, -1.0, -3)
      CALL PLOT (0.0, 8.5, 2)
      IFIRST = 1
      CALL SLITET (2)
      NEWPAGE = 1
      ISCALE = 2
      GO TO 20
C
C         NOT FIRST CALL THIS JOB---INITIALIZE DUE TO OVERLAY
```

```
    CALL LIMITS (-PZ.0, 22.0)
    CALL PLOT (0.0, 3.0, 2)
    CALL PLOT (0.0, 0.0, 3)
    GO TO (7, 10), IPLOT
7   CALL SLITET (1, ISCALE)
    GO TO (8, 11), ISCALE
8   SENSE LIGHT 1
    XMAX = XMAX1
    XMIN = XMIN1
    YMAX = YMAX1
    CALL SLITET (2, NEWPAGE)
    GO TO (70, 9), NEWPAGE
9   CALL PLOT (-11.0, 0.0, -3)
    GO TO 70
10  ISCALE = 2
11  NEWPAGE = 1
C
C       DETERMINE XMIN,XMAX,YMIN,YMAX AND SET PLOTTER PARAMETERS
20  CALL SLITET (2)
    GO TO (30, 80), IPLOT
C
C       LAMDA LOCUS
30  XMIN=XMAX=YMAX=0.0
    DO 40 I=1,N
      IF (X(1,I).LT.XMIN) XMIN=X(1,I)
      IF (X(1,I).GT.XMAX) XMAX=X(1,I)
      IF (ABS(X(2,I)).GT.YMAX) YMAX=ABS(X(2,I))
40    CONTINUE
    IF (XMAX.GE.19.0) XMAX = 20.0; GO TO 50
    IF (XMAX.NE.0.0) XMAX = AINT(XMAX) + 1.0
50  IF (XMIN.LE.-19.0) XMIN = -20.0; GO TO 60
    IF (XMIN.NE.0.0) XMIN = AINT(XMIN) - 1.0
60  IF (YMAX.GE.19.0) YMAX = 20.0; GO TO 70
    YMAX = AINT(YMAX) + 1.0
70  XSS = 8.0
    XDD = 1.5
    GO TO 130
C
C       ZETA LOCUS
80  SENSE LIGHT 2
    DO 100 I=1,N
      IF (X(1,I).EQ.0.0) U = ABS(X(2,I)); GO TO 90
      IF (X(2,I).EQ.0.0) U = ABS(X(1,I)); GO TO 40
      U = CABS(Z(I))
90    IF (U.GT.1.0) GO TO 110
100   CONTINUE
    XMAX = YMAX = 1.5
    XMIN = -1.0
    GO TO 120
110 XMAX = YMAX = 1.5
    XMIN = -1.5
120 XSS = 6.5
    XDD = 2.25
C
C       COMMON LEG---SET PLOTTER SCALING
130 XS = (XMAX-XMIN)/XSS
    YS = YMAX/3.25
    XB = XMIN - XDD*XS
    YB = -(YMAX +1.0*YS)
```

```
PLOT 117        CALL OFFSET (XB, XS, YB, YS)
PLOT 118        YMIN = -YMAX
PLOT 119  C
PLOT 120        GO TO (137, 230), NEW PAGE
PLOT 121  137   GO TO (138, 140), ISCALE
PLOT 122  138   IF ((AINT(XMIN).EQ.XMIN) .AND. (AINT(XMAX).EQ.XMAX) .AND.
PLOT 123      1     (AINT(YMAX).EQ.YMAX)) GO TO 145
PLOT 124        CALL PLOT (XMIN, 0.0, 13)
PLOT 125        CALL PLOT (XMAX, 0.0, 12)
PLOT 126        CALL PLOT (0.0, YMIN, 13)
PLOT 127        CALL PLOT (0.0, YMAX, 12)
PLOT 128        GO TO 230
PLOT 129  140   GO TO (145, 150), IPLOT
PLOT 130  145   BIG = 1.0
PLOT 131        NSMALL = 1
PLOT 132        SMALL = 0.5
PLOT 133        ASSIGN 1000 TO IFORMAT
PLOT 134        GO TO 155
PLOT 135  150   BIG = 0.5
PLOT 136        NSMALL = 4
PLOT 137        SMALL = 0.1
PLOT 138        ASSIGN 1010 TO IFORMAT
PLOT 139  C
PLOT 140  C         PLOT X-AXIS
PLOT 141  155   CALL PLOT (XMIN, 0.0, 13)
PLOT 142        CALL WHERE (XA, YA, FCTR)
PLOT 143  160   ENCODE (4,IFORMAT, LABEL), XMIN
PLOT 144        CALL SYMBOL (XA, YA-0.2, 0.14, LABEL, 0.0, 4)
PLOT 145        CALL PLOT (XA, YA-0.1, 3)
PLOT 146        CALL PLOT (XA, YA, 2)
PLOT 147        DO 180 XP=XMIN+BIG,XMAX,BIG
PLOT 148          DO 170 I=1,NSMALL
PLOT 149            CALL PLOT (XP-(NSMALL-I+1)*SMALL, 0.0, 12)
PLOT 150            CALL WHERE (XA, YA, FCTR)
PLOT 151            CALL PLOT (XA, YA-0.05, 2)
PLOT 152  170       CALL PLOT (XP, YA, 2)
PLOT 153          CALL PLOT (XP, 0.0, 12)
PLOT 154          CALL WHERE (XP, YA, FCTR)
PLOT 155          CALL PLOT (XA,YA+0.1, 2)
PLOT 156  180     CALL PLOT (XP, YA, 2)
PLOT 157        ENCODE (4, IFORMAT, LABEL), XMAX
PLOT 158        CALL SYMBOL (XA-0.25, YA-0.2, 0.14, LABEL, 0.0, 4)
PLOT 159  C
PLOT 160  C         PLOT Y-AXIS
PLOT 161        CALL PLOT (0.0, YMIN, 13)
PLOT 162        CALL WHERE (XA, YA, FCTR)
PLOT 163        ENCODE (4, IFORMAT, LABEL), YMIN
PLOT 164        CALL SYMBOL (XA+0.15, YA-0.15, 0.14, LABEL, 0.0, 4)
PLOT 165        CALL PLOT (XA+0.1, YA, 3)
PLOT 166        CALL PLOT (XA, YA, 2)
PLOT 167        DO 200 YP=YMIN+BIG,YMAX,BIG
PLOT 168          DO 190 I=1,NSMALL
PLOT 169            CALL PLOT (0.0, YP-(NSMALL-I+1)*SMALL, 12)
PLOT 170            CALL WHERE (XA, YA, FCTR)
PLOT 171            CALL PLOT (XA+0.05, YA, 2)
PLOT 172  190       CALL PLOT (XA, YA, 2)
PLOT 173          CALL PLOT (0.0, YP, 12)
PLOT 174          CALL WHERE (XA, YA, FCTR)
PLOT 175          CALL PLOT (XA+0.1, YA, 2)
```

```
PLOT 176
PLOT 177
PLOT 178
PLOT 179
PLOT 180
PLOT 181
PLOT 182
PLOT 183
PLOT 184
PLOT 185
PLOT 186
PLOT 187
PLOT 188
PLOT 189
PLOT 190
PLOT 191
PLOT 192
PLOT 193
PLOT 194
PLOT 195
PLOT 196
PLOT 197
PLOT 198
PLOT 199
PLOT 200
PLOT 201
PLOT 202
PLOT 203
PLOT 204
PLOT 205
PLOT 206
PLOT 207
PLOT 208
PLOT 209
PLOT 210
PLOT 211
PLOT 212
PLOT 213
PLOT 214
PLOT 215
PLOT 216
PLOT 217
PLOT 218
PLOT 219
PLOT 220
PLOT 221
PLOT 222
PLOT 223
PLOT 224
PLOT 225
PLOT 226
PLOT 227
PLOT 228
PLOT 229
PLOT 230
PLOT 231
PLOT 232
PLOT 233
PLOT 234
```

```
 230    CALL PLOT (XA, YA, 2)
      ENCODE (4, IFORMAT, LABEL), YMAX
      CALL SYMBOL (XA+0.15, YA+0.01, 0.14, LABEL, 0.0, 6)
C
C          PLOT UNIT CIRCLE IF IPLOT=2
      GO TO (230, 210), IPLOT
 210  CALL PLOT (0.0, 1.0, 13)
      DO 220 I=1,3
      DO 220 BIG=99.0, 450.0, 2.0
        ANGLE=BIG*PI/180.0
        CALL PLOT (COS(ANGLE), SIN(ANGLE), 1P)
 220
C          PLOT POINTS
C
 230  XPMIN = XPMAX = YPMAX = 0.0
      DO 285 I=1,NLOCUS
        ASSIGN 240 TO ICONTINU
        INEW = 1
        J=1
        IC = 13
        ASSIGN 235 TO IRETURN
        GO TO 500
 235    ASSIGN 250 TO IRETURN
        DO 250 J=2,NPTS
          GO TO ICONTINU, (240, 245)
 240      IF (ILOCUS(INEW).NE.J) GO TO 245
          IF (IC.NE.13) GO TO 242
          CALL WHERE (XA, YA, FCTR)
          CALL PLOT (XA, YA, 2)
 242      IC = 13
          INEW = INEW+1
          IF (INEW.GT.300) ASSIGN 245 TO ICONTINU
          GO TO 500
 245      IC =12
          GO TO 500
 250      CONTINUE
        J = NPTS
        IC = 13
        ASSIGN 255 TO IRETURN
        GO TO 505
 255    IF (INEW.EQ.1) ASSIGN 275 TO ICONTINU; GO TO 260
        INEW = INEW+1
        ASSIGN 265 TO ICONTINU
        ASSIGN 280 TO IRETURN
 260    DO 280 J=NPTS-1,1,-1
          GO TO ICONTINU, (265, 275)
 265      IF (ILOCUS(INEW).NE.J+1) GO TO 275
          IF (IC.NE.13) GO TO 270
          CALL WHERE (XA, YA, FCTR)
          CALL PLOT (XA, YA, 2)
 270      IC = 13
          INEW = INEW+1
          IF (INEW.EQ.0) ASSIGN 275 TO ICONTINU
          GO TO 500
 275      IC = 12
          GO TO 505
 280      CONTINUE
 285    CONTINUE
C
      GO TO (310, 320), NEW PAGE
C          PLACE PLOTTER ID ON PLOT---UPPER LEFT-HAND CORNER
C
 310  CALL SYMBOL (1.0, 7.7, 0.14, ID, 0.0, 8)
C
C          FINISH PUTTING BORDERS AROUND PLOTTER PAGE
      CALL PLOT (0.0, 4.5, 3)
      CALL PLOT (11.0, 8.5, 2)
      CALL PLOT (11.0, 0.0, 2)
 320  CALL PLOT (11.0, 0.0, 3)
C
      GO TO (330, 340), IPLOT
 330  XMAX1 = XMAX
      XMIN1 = XMIN
      YMAX1 = YMAX
      PRINT 335, 180.0*ALPHA/PI, XALPHA, YALPHA
 335  FORMAT (/'METHOD IS A('F.2,'*X')+BTABLE AT ('F.2,'*X', 'F.0,'*X')')
 340  PRINT 350
      PRINT 370, XPMAX, XPMIN, YPMAX
      PRINT 360
      PRINT 370, XMAX, XMIN, YMAX
 350  FORMAT (/'PLOT EXTREMES...')
 360  FORMAT (/'PLOTTER SCALING...')
 370  FORMAT (' XMAX '011.4' XMIN '011.4' YMAX '011.4)
      RETURN
C
C
C          COMPUTE PLOT POINT
 500  POINT = ROOT(NLOCUS*(J-1)+I)
      IF (IPLOT.EQ.1) IF (U.LT.0.0) IF ((T+ABS (ATAN (V/U))).LT.ALPHA)
     1                    ALPHA = T; XALPHA = U; YALPHA = V
      IF (U.LT.XPMIN) XPMIN = U
      IF (U.GT.XPMAX) XPMAX = U
      IF ((T = ABS(V)).GT. YPMAX) YPMAX = T
      GO TO 510
 505  POINT = CONJG(ROOT(NLOCUS*(J-1)+I))
C
 510  XP = AMIN(U,XMAX)
      XP = AMAX(XP, XMIN)
      YP = AMIN(V, YMAX)
      YP = AMAX(YP,YMIN)
      CALL PLOT (XP, YP, IC)
      GO TO IRETURN, (235, 250, 255, 260)
C
 1000 FORMAT(I)
 1010 FORMAT (F6.1)
C-----------------------------------------------------------------------
      END
```

```
      SUBROUTINE INPUT (ISPACE)

      3 MAY 1972

C     ******************************************************************
C     *
C     *THIS S/R CALLS BY SUBROUTINE ACCEPT TO SUPPLY THE CHARACTERISTIC
C     *  EQUATION OF A METHOD AND OTHER SYSTEM PARAMETERS;  SPECIFICALLY
C     *    ICHAR, NZ, NL, ID AND IPLOT
C     *WHERE...
C     *    IPLOT  DISCRETE INDICATING TYPE OF PLOT DESIRED
C     *       1  LAMDA PLOT
C     *       2  ZETA PLOT
C     *       3  BOTH PLOTS (LAMDA LOCUS FIRST THEN ZETA LOCUS)
C     *    NL    DEGREE OF CHARACTERISTIC POLYNOMIAL IN LAMDA (L)
C     *    NZ    DEGREE OF CHARACTERISTIC POLYNOMIAL IN ZETA (Z)
C     *    ID    A ALPHANUMERIC CHARACTER USED AS PLOT IDENTIFIER
C     *    ICHAR(I,J)  COEFFICIENT OF CHARACTERISTIC EQUATION
C     *                MULTIPLYING (LAMDA**I)*(ZETA**J)
C     *
C     *THE ROOTS OF THE CHARACTERISTIC EQUATION AT H = 0 AND AT
C     *  H = INFINITY ARE COMPUTED AND PRINTED.
C     *
C     *THE MAXIMUM NUMBER OF POINTS PER LOCUS IS COMPUTED AS...
C     *    NPTS = 2*(ISPACE-NLOCUS)/NLOCUS> + 1
C     *    WHERE...
C     *          NPTS      # OF POINTS PER LOCUS TO BE PLOTTED
C     *          ISPACE    DIMENSION OF ARRAY IN  (IN MAIN PROGRAM)
C     *          NLOCUS    # OF LOCI TO BE PLOTTED
C     *          <X>       INTEGER PORTION OF X
C     *
C     ******************************************************************
      DOUBLE PRECISION ID
      COMMON /METHOD/ICHAR(0/10,0/10), NZ, NL, ID, IPLOT, INCR, ITIME
      REAL ICHAR
      COMMON /WORKAREA/ WORK(109)

C     COMPLEX CWORK(1)
      DIMENSION XWORK(2,1)
      EQUIVALENCE (WORK, CWORK, XWORK)

C     DATA PI /3.141592653589793/
      DATA EPS /1.0D-10/

C                 FOLLOWING EQUIVALENCE STATEMENTS ARE USED
C                 SOLELY TO CONSERVE SPACE
      EQUIVALENCE (WORK(4), RTEMP)
      EQUIVALENCE (WORK(70), I), (WORK(71), J),
     1            (WORK(72), IR, IER, NLOCUS), (WORK(73), ISTABLE),
     2            (WORK(74), IRETURN)

C     IF (IPLOT.EQ.0) GO TO 2
      IF (ITIME.EQ.0) GO TO 4
      IPLOT=2
      INCR = (ISPACE-NZ)/NZ
      ITIME=0
      RETURN
2     ITIME=0
4     CALL ACCEPT
20    IF (IPLOT.NE.3) GO TO 25
      IPLOT=1
      ITIME=1
25    PRINT 30, ID, NL
30    FORMAT (5X,CHARACTERISTIC EQUATION FOR '0 A8,   0'0//
     1        T19, I/ 0 P(ZETA,LAMDA) = SUM  P(I)(ZETA)*(LAMDA**I)0/
     2    T18,I=0,0/ 0   WHERE...0/)
      DO 35 I=0,NL
35      PRINT 40, I, ICHAR(I,NZ), NZ, NZ, (ICHAR(I,J), J= J=NZ-1,0,-1)
40    FORMAT (T4 0P(0I-X0) * 0023-17 0 ZETA**0I/
     1        NI(T4 0+ 0323-17, 0 ZETA**0 I /I/)
      ISTABLE = 0
      PRINT 50, ID
50    FORMAT (0)ROOTS FOR '0 A8, 0: AS H TENDS TO ZERO0)
      IRETURN = 1
      I = 0
      DO T= 100
55    PRINT 65
65    FORMAT (4/0 ROOTS AS H TENDS TO INFINITY0)
      IRETURN = 2
      I = NL
      DO T= 100
66    IF ((I-I-1)*EQ.0) GO TO 68
      PRINT 67, I
67    FORMAT (K0****USING P(0I-X0) TO DETERMINE ROOTS AS H TENDS TO 0
     1        0INFINITY0)
      GO TO 100
68    PRINT 69
69    FORMAT (2/0****RUN ABORTED0)
      GO TO 2

C         PRINT OUT WHETHER METHOD IS AT LEAST STIFFLY-STABLE
C             (ISTABLE=0) OR NOT (ISTABLE =1)
C     PRINT 72, ISTABLE, MOD(ISTABLE+1, 2)
72    FORMAT (4/X0-METHOD IS 0, N(0NOT EVEN0), N(0AT LEAST0),
     1        0 STIFFLY STABLE0)

C         COMPUTE NUMBER OF POINTS/LOCUS
C     IF (IPLOT.EQ.1) NLOCUS = NL; GO TO 80
      NLOCUS = NZ
80    INCR = (ISPACE-NLOCUS)/NLOCUS
      PRINT 90
90    FORMAT (1H1)
      RETURN

C     ******************************************************************
C     *
C     *    THIS SECTION SOLVES FOR AND PRINTS
C     *    THE ROOTS OF THE POLYNOMIAL
C     *                    NZ
C     *    P(ZETA) =  SUM  (CHAR(I,J)*(ZETA**J)
C     *                    J=0
C     *    WHERE I IS SET IN THE MAIN PROGRAM TO EITHER
C     *          0 (STABILITY AS H GOES TO ZERO) OR TO
C     *          NL (STABILITY AS H GOES TO INFINITY)
C     *
```

```
C     ***********************************************************************INPUT117
100   DO 101 J=0,NZ                                                           INPUT118
101     CWORK(J+1) = CMPLX (ICHAR(I,J), 0.0)                                  INPUT119
      IFIRST = 0                                                              INPUT120
      DO 112 J=NZ,1,-1                                                        INPUT121
10P     CALL CWROOTS (CWORK, CWORK(22), J, CWORK(12), IER)                    INPUT122
        IF (IER.GT.3) 113, 103, 110                                           INPUT123
103     IF (IFIRST.EQ.0) GO TO 105                                            INPUT124
        PRINT 104, IER                                                        INPUT125
104     FORMAT (8H****S/R CWROOTS INDICATES ERROR, IER=,I                     INPUT126
        GO TO (55, 70), IRETURN                                               INPUT127
105     BIG = 0                                                               INPUT128
        DO 106 JJ=0,J                                                         INPUT129
          IF ((T=ABS(ICHAR(I,JJ))).GT.=BIG) BIG = T                           INPUT130
106     CONTINUE                                                              INPUT131
        PRINT 107, BIG                                                        INPUT132
107     FORMAT (8HORIGINAL POLYNOMIAL BEING NORMALIZED BY =0P23.17)           INPUT133
        II = 1                                                                INPUT134
        IFIRST = 1                                                            INPUT135
108     DO 109 JJ=0,J                                                         INPUT136
109       XWORK(1,JJ+1) = ICHAR(I,JJ)/BIG                                     INPUT137
        GO TO (102, 112), II                                                  INPUT138
110     IF (IFIRST.NE.1) IFIRST = 0; BIG = 1.0; II = 2; GO TO 108             INPUT139
112     CONTINUE                                                              INPUT140
      IF (XWORK(1,1).EQ.0.0) GO TO (55, 66), IRETURN                          INPUT141
      ISTABLE = 1                                                             INPUT142
      PRINT 1125                                                              INPUT143
1125  FORMAT (8HALL ROOTS ARE AT INFINITY =)                                  INPUT144
      GO TO (55, 70), IRETURN                                                 INPUT145
119   IF ((IR = NZ-J).EQ.0) GO TO 1135                                        INPUT146
      PRINT 1131, IR, J                                                       INPUT147
1131  FORMAT (XI =ROOTS ARE AT INFINITY, =                                    INPUT148
     1          XHREMAINING =I = ROOTS ARE LISTED BELOW,=)                    INPUT149
      ISTABLE = 1                                                             INPUT150
1135  DO 115 I=1,J                                                            INPUT151
        CALL REFCROT (CWORK, CWORK(11+I), J, CWORK(11+I), EPS, IER)           INPUT152
        IF (IER.EQ.0) GO TO 115                                               INPUT153
        PRINT 114, IER, I                                                     INPUT154
114     FORMAT (8H****S/R REFCROOT INDICATES IER=,I =FOR ROOT NUMBER =I)      INPUT155
115     CONTINUE                                                              INPUT156
      PRINT 116                                                               INPUT157
116   FORMAT (T4,'REAL PART', T31, 'IMAGINARY PART', T58 'MODULUS',           INPUT158
     1          T85 'ANGLE')                                                  INPUT159
      DO 130 IR=12,11+J                                                       INPUT160
        IF ( (WORK(11 = CABS (CWORK(IR))) .GT. 1.0) ISTABLE = 1               INPUT161
        IF (XWORK(2,IR).EQ.0.) GO TO 120                                      INPUT162
        PRINT 140, XWORK(1,IR), XWORK(2,IR), WORK(1),                         INPUT163
     1                          ATAN2 (XWORK(2,IR), XWORK(1,IR))*180./PI      INPUT164
        GO TO 130                                                             INPUT165
120     PRINT 140, XWORK(1,IR)                                                INPUT166
130     CONTINUE                                                              INPUT167
140     FORMAT (X, 3(E22.16, 5X) F8.3)                                        INPUT168
      GO TO (55, 70), IRETURN                                                 INPUT169
C     ***********************************************************************INPUT170
      END                                                                     INPUT171
```

```
      SUBROUTINE ACCEPT                                                       ACPCD  1
                                                                              ACPCD  2
C     ***********************************************************************ACPCD  3
C     *THIS S/R ACCEPTS THE CHARACTERISTIC EQUATION IN FOLLOWING FORMAT *ACPCD  4
C     *    CARD 1     NR, NC, IFORM (COL 81, ID (COL 71-78)             *ACPCD  5
C     *    CARD 2     P(0,0), .... P(0,NC)                              *ACPCD  6
C     *     ...        ...                                              *ACPCD  7
C     *    CARD 2+NR  P(NR,0), .... P(NR,NC)                            *ACPCD  8
C     *WHERE...                                                          *ACPCD  9
C     *    ID     8 ALPHANUMERIC CHARACTERS USED AS AN IDENTIFIER ON ALL *ACPCD 10
C     *           OUTPUT                                                 *ACPCD 11
C     *  IF IFORM = BLANK CHARACTER THEN...                               *ACPCD 12
C     *    NR     DEGREE OF CHARACTERISTIC EQUATION IN LAMDA             *ACPCD 13
C     *    NC     DEGREE OF CHARACTERISTIC EQUATION IN ZETA              *ACPCD 14
C     *    P(I,J) COEFFICIENT OF CHARACTERISTIC EQUATION MULTIPLYING     *ACPCD 15
C     *           (LAMDA**I)*(ZETA**J)                                    *ACPCD 16
C     *  IF IFORM = T (THE LETTER T) THEN...                              *ACPCD 17
C     *    NR     DEGREE OF CHARACTERISTIC EQUATION IN ZETA              *ACPCD 18
C     *    NC     DEGREE OF CHARACTERISTIC EQUATION IN LAMDA             *ACPCD 19
C     *    P(I,J) COEFFICIENT OF CHARACTERISTIC EQUATION MULTIPLYING     *ACPCD 20
C     *           (ZETA**I)*(LAMDA**J)                                    *ACPCD 21
C     *                                                                   *ACPCD 22
C     *                                                                   *ACPCD 23
C     *THE SYSTEM PARAMETER IPLOT IS SET TO 3 TO INDICATE THAT BOTH A    *ACPCD 24
C     *  LAMDA AND A ZETA PLOT ARE DESIRED                                *ACPCD 25
C     ***********************************************************************ACPCD 26
      COMMON /METHOD/ICHAR(0/10,0/10), NZ, NL, ID, IPLOT, INCR, ITIME     ACPCD 27
      DOUBLE PRECISION ID                                                 ACPCD 28
      REAL ICHAR                                                          ACPCD 29
                                                                          ACPCD 30
      READ (105, 5, END=100), P, NROWS, NCOL, IFORM, ID                   ACPCD 31
C                                                                         ACPCD 32
    5 FORMAT (5I, T8, A1, T71, A8)                                        ACPCD 33
      IPLOT = 3                                                           ACPCD 34
      IF (IFORM.EQ.1HT) GO TO 15                                          ACPCD 35
C                                                                         ACPCD 36
C         MATRIX IS IN NORMAL FORM                                        ACPCD 37
C                                                                         ACPCD 38
      NL = NROWS                                                          ACPCD 39
      NZ = NCOL                                                           ACPCD 40
      DO 10 I=0,NL                                                        ACPCD 41
   10   READ 5, NZ+1, (ICHAR(I,J), J=0,NZ)                                ACPCD 42
      GO TO 20                                                            ACPCD 43
C                                                                         ACPCD 44
C         MATRIX IS IN TRANSPOSE FORMAT                                   ACPCD 45
C                                                                         ACPCD 46
   15 NL = NCOL                                                           ACPCD 47
      NZ = NROWS                                                          ACPCD 48
      DO 17 J=0,NZ                                                        ACPCD 49
   17   READ 5, NL+1, (ICHAR(I,J), I=0,NL)                                ACPCD 50
   20 RETURN                                                              ACPCD 51
  100 PRINT 110                                                           ACPCD 52
  110 FORMAT (8HEND OF CARD DATA8)                                        ACPCD 53
      STOP                                                                ACPCD 54
      END                                                                 ACPCD 55
```

```
      SUBROUTINE CMROOTS (XCOF, COF, M, ROOT, IER)
C     29MAY1972
C
C     ******************************************************************
C     *                                                                *
C     *   COMPUTES COMPLEX ROOTS OF A COMPLEX POLYNOMIAL               *
C     *   CALLING SEQUENCE:                                            *
C     *   CALL CMROOTS (XCOF, COF, M, ROOT, IER)                       *
C     *   WHERE:                                                       *
C     *         XCOF = M+1 VECTOR OF COMPLEX COEFFICIENTS              *
C     *                IN ASCENDING ORDER                              *
C     *         COF  = M+1 VECTOR USED AS WORK AREA                    *
C     *         M    = DEGREE OF POLYNOMIAL                            *
C     *         ROOT = M VECTOR WHERE ROOTS WILL BE STORED             *
C     *         IER  = ERROR INDICATOR AS FOLLOWS                      *
C     *                0 = NO ERROR                                    *
C     *                1 = M.LE.0                                      *
C     *                2 = CONVERGENCE FAILED AFTER 2000 ITERATIONS WITH *
C     *                    5 DIFFERENT STARTING VALUES                 *
C     *                3 = HIGHEST ORDER COEFICIENT IS ZERO            *
C     *   XCOF, COF AND ROOT ARE ALL COMPLEX                           *
C     *   POLYNOMIAL FROM AS FOLLOWS                                   *
C     *         P(X) = XCOF(1) + XCOF(2)*X + ... +XCOF(M+1)*X**M       *
C     *                                                                *
C     ******************************************************************
C
      COMPLEX XCOF(1), COF(1), ROOT(1)
      COMPLEX Z, DZ, P, DP, ZPR, CTEMP1, CTEMP2
      DIMENSION XX(2), DXX(2), PRI(2), DPRI(2)
      EQUIVALENCE (Z, XX, X), (XX(2), Y), (DZ, DXX, DX), (DXX(2), DY)
      EQUIVALENCE (PRI, P, PREAL), (PRI(2), PIMAG)
      EQUIVALENCE (DPRI, DP, DPREAL), (DPRI(2), DPIMAG)
C
      DATA EPS /1.0E-10/
C
C                 FOLLOWING EQUIVALENCE STATEMENTS ARE USED
C                 SOLELY TO CONSERVE SPACE
C     EQUIVALENCE (RTEMP1, CTEMP1), (DP, CTEMP2, RTEMP2), (DZ, P)
C
      IF (M.GT.0) GO TO 5
      IER=1
      RETURN
    5 IF (XCOF(M+1).NE.(0.,0.)) GO TO 10
      IER=2
      RETURN
   10 NM=M
      M=M-1
      IER=0
C
C         REMOVE ZERO ROOTS
C
   15 IF (XCOF(NM).NE.(0.,0.)) GO TO 20
      ROOT(NM) = (0.,0.)
      NM=NM-1
      IF (NM.NE.M) GO TO 15
      RETURN
C
C         CHECK IF EXACT SOLUTION POSSIBLE
C
   20 NZ=NM-1
      IF (NZ.NM-2) 25,35,35
C
C         ONE ROOT LEFT
C
   25 ROOT(NM)=-XCOF(M)/XCOF(M+1)
      RETURN
C
C         TWO ROOTS LEFT
C
   35 CTEMP1 = -XCOF(M)/((2.,0.)*XCOF(M+1))
      CTEMP2 = CSQRT(CTEMP1**2 - XCOF(M-1)/XCOF(M+1))
      ROOT(NM)=CTEMP1+CTEMP2
      ROOT(NM-1)=CTEMP1-CTEMP2
      RETURN
C
C         ITERATION NECESSARY.SET UP COF ARRAY
C
      DO 45 I=1,M+1
   45   COF(I)=XCOF(NM+I-1)
      IFIRST=0
C
C         START ITERATION FOR ROOT
C
   45 Z=(0.00500101,0.01000101)
      ITIME=0
      IFIT=0
      NC=NM-M+1
   50 RTEMP1 = X
      ** 1/M.Y
      Y=1/M.*RTEMP1
   53 ITIME=ITIME+1
      ICNT=0
C
C         EVALUATE F.N AND ITS DERIVATIVE
C
   55 P = COF(NC)
      DP = P
      DO 57 I=NC+1,M+1
        P = P*Z + COF(I)
   57   DP = DP*Z + P
      P = P*Z + COF(1)
C
      IF (ABS(DPREAL)+ABS(DPIMAG).LT.EPS) GO TO 69
      IP=1
      IF (ABS(PREAL)+ABS(PIMAG).LT.EPS) GO TO 70
      DZ=P/DP
      Z=Z-DZ
      RTEMP1 = ABS(X)+ABS(Y)
      RTEMPP = ABS(DX)+ABS(DY)
      IF (RTEMP1.LT.1.) GO TO 59
      IF (RTEMP1/RTEMPP.LT.EPS) GO TO 70
      GO TO 61
   59 IF (RTEMPP.LT.EPS) GO TO 70
   61 ICNT=ICNT+1
      IF (ICNT.LT.2000) GO TO 55
```

```
      IF (IFIT.EQ.1) GO TO 64
   62 IF (ITIME.NE.1) GO TO 64
      IF (IFIRST.EQ.0) GO TO 50
      Z = (C*10000131*D*0500(01)
      GO TO 53
   64 IF (ITIME.LE.5) GO TO 50
      IERR=3
      RETURN
C
C        DP=0
C
   65 NZ=Z
      IF (ABS(PREAL)+ABS(PIMAG).LT.EPS) GO TO 70
      IF (IFIT.EQ.3) GO TO 62
      NZ=1
   68 Z=ZPR
      GO TO 79
   70 IF (IFIT.EQ.0) GO TO 105
   75 DO 80 I=1,NC
      CTEMP1=CDF(I)
      CDF(I)=XCDF(NZ+I)
   80 XCDF(NZ+I)=CTEMP1
   85 IF (ABS(X).LT.EPS) X=0.0
      IF (ABS(Y).LT.EPS) Y=0.0
C
C        STORE NEW ROOT JUST FOUND
C        AND ELIMINATE IT FROM REDUCED POLYNOMIAL *** CDF
C
   86 ROOT(NR) = Z
      IF (N.EQ.NR+1) RETURN
      CTEMP1=CDF(N+NR)
      CDF(N+NR)=CDF(N+NR+1)
      DO 90 I=N+NR-1,1,-1
        CTEMP2=CDF(I)
        CDF(I)=CTEMP1+Z*CDF(I+1)
        CTEMP1=CTEMP2
   90 NR=NR+1
      NP=NP-1
      IF (NP.EQ.1) GO TO 86
      IFIRST=1
      NC=N-NP+1
      IF (N+NR-2) 95, 100, 92
C
C        MORE THAN TWO ROOTS LEFT TO EXTRACT
C
   92 ICNT=0
      ITIME=0
      IFIT=0
      Z = ZPR
      GO TO 55
C
C        ONE ROOT LEFT TO BE FOUND
C
   95 Z = -CDF(1)/CDF(2)
      GO TO 105
C
C        TWO ROOTS LEFT TO EXTRACT
C
  100 CTEMP1=-CDF(2)/(2.0*1.0*CDF(3))
      Z=CSQRT(CTEMP1**2-CDF(1)/CDF(3)) + CTEMP1
  105 ZPR=Z
      DO 110 I=1, N-NZ
        CTEMP1=CDF(I)
        CDF(I)=XCDF(NZ+I)
  110   XCDF(NZ+I)=CTEMP1
      IFIT=1
      ICNT=0
      NC=N-Z
      GO TO 55
C-----------------------------------------------------------------------
      END
```

```
      SUBROUTINE REFCRRT(XCOF, GUESS, M, RRRT, EPSILON, IER)
C
C     8 MAY 1972
C
C     ******************************************************************
C     *
C     *   REFINES AN INITIAL GUESS OF A ROOT OF A COMPLEX POLYNOMIAL
C     *   CALLING SEQUENCE:
C     *   CALL REFCRRT (XCOF, GUESS, M, RRRT, EPSILON, IER)
C     *   WHERE
C     *        XCOF = MAX VECTOR OF COMPLEX COEFFICIENTS
C     *             IN ASCENDING ORDER
C     *        GUESS = INITIAL GUESS AT RRRT
C     *        M = DEGREE OF POLYNOMIAL
C     *        RRRT = REFINED VALUE OF RRRT
C     *        EPSILON = STOPPING CRITERIA, I.E., METHOD STOPS WHEN
C     *             ABS (RRRT(N+1)-RRRT(N)) < EPSILON
C     *             WHERE
C     *                  RRRT(N) IS NTH ITERATION GUESS AT RRRT
C     *                  AND ABS(X) = ABS(RE(X))+ABS(IM(X))
C     *        IER = ERROR INDICATOR AS FOLLOWS
C     *             0 = NO ERROR
C     *             1 = M.LE. 0
C     *             2 = CONVERGENCE FAILS SINCE DERIVATIVE = 0
C     *             3 = CONVERGENCE FAILED WITH 1000 ITERATIONS
C     *             4 = HIGHEST ORDER COEFICIENT IS ZERO
C     *
C     ******************************************************************
C
      COMPLEX XCOF(1),GUESS,RRRT
      COMPLEX Z,DZ,P,DP,CMP1,C1
      DIMENSION XX(2), DXX(2), PRI(2), DPRI(2)
      EQUIVALENCE (XX, X, Z1, (XX(2), Y), (DXX, DX, DZ), (DXX(2), DY)
      EQUIVALENCE (PRI, P, PREAL), (PRI(2), PIMAG),
     1            (DPRI, DP, DPREAL), (DPRI(2), DPIMAG)
C
C                 FOLLOWING EQUIVALENCE STATEMENTS ARE USED
C                 SOLELY TO CONSERVE SPACE
C        EQUIVALENCE (DZ, P1), (DP, RTEMP1), (C1, RTEMP2)
C
      IF (M.GT.0) GO TO 5
      IER=1
      RETURN
    5 IF (XCOF(M+1).NE.(0.,0.)) GO TO 10
      IER=4
      RETURN
   10 IER=0
      DO 15 I=1,M
        IF (XCOF(I).NE.(0.,0.)) GO TO 30
   15 CONTINUE
      Z = (0.,0.)
      GO TO 60
   30 IF (I.NE.M) GO TO 35
      Z = -XCOF(I)/XCOF(M)
      GO TO 65
   35 Z=GUESS
      ICNT=0
   40 P = XCOF(M+1)
      DP = 0
      DO 45 I=M,2,-1
        P = P*Z + XCOF(I)
   45 DP = DP*Z + P
      P = P*Z + XCOF(1)
      IF (ABS(PREAL)+ABS(PIMAG).LT.EPSILON) GO TO 45
      IF (ABS(DPREAL)+ABS(DPIMAG).LT.EPSILON) GO TO 50
      DZ=-P/DP
      Z=Z+DZ
      RTEMP1 = ABS( X)+ABS( Y)
      RTEMP2 = ABS(DX)+ABS(DY)
      IF (RTEMP1.LT.1.) GO TO 47
      IF (RTEMP2/RTEMP1.LT.EPSILON) GO TO 45
      GO TO 49
   47 IF (RTEMP2.LT.EPSILON) GO TO 45
   49 ICNT=ICNT+1
      IF (ICNT.LT.1000) GO TO 40
      IER=3
      GO TO 60
   50 IER=2
   60 RRRT = Z
      RETURN
   65 IF (ABS(X).LT.EPSILON) X=0.
      IF (ABS(Y).LT.EPSILON) Y=0.
      GO TO 60
C-----------------------------------------------------------------------
      END
```

APPENDIX B

COMPUTER PROGRAM LISTING FOR INTERACTIVE VERSION OF SUBROUTINE ACCEPT

This appendix contains a brief description and a computer program listing of the interactive version of subroutine ACCEPT used in deriving the new methods contained in Chapter IV.

Though referred to as a single program the interactive version of ACCEPT consists of several different units. A block diagram showing the six main segments is given in Figure B.1. Each segment is a separate FORTRAN subroutine. NOTE: Though several of the subroutines shown in Figure B.1, and indeed the entire package, could have been combined into a single subroutine they were purposely written as independent units. This was done in order to provide the overall program with greater flexibility. Other versions of the plotting program use some of these units. As an example, another form of the plotting program uses a different ACCEPT subroutine, one which reads a method from cards and then calls the same FILL and DETERMIN subroutines shown in Figure B.1 to compute the characteristic polynomial.

The computer programs listed on the following pages were written in XDS EXTENDED FORTRAN-IV for execution on an

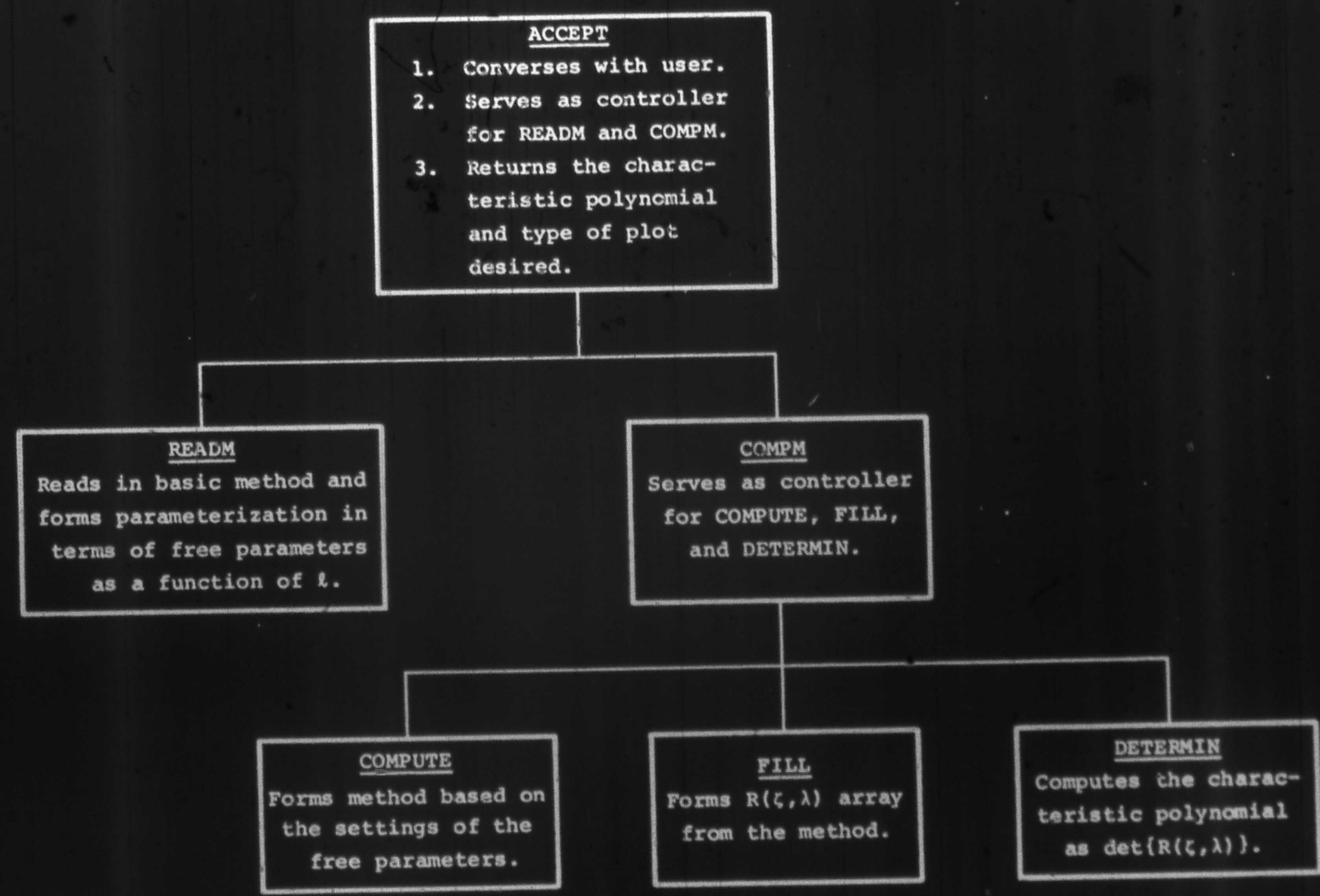


Figure B.1: Organizational Chart for Interactive ACCEPT

XDS SIGMA-5 digital computer. They were compiled using the ADP—automatic double precision—option discussed in Appendix A. Interactive communication was via a teletypewriter, the only device available for this purpose at the computer facility used.

```
      SUBROUTINE ACCEPT
C     WRITTEN AS JOINT EFFORT BY W. RUMIN AND J. TENDLER
C
C     10JUNE1972
C
C     LIST OF BRANCHES:
C
C     N       READ IN NEW METHOD FROM CARDS
C             NEW SERIAL NUMBER
C
C     S       GO (RETURN TO CALLING PROGRAM)
C     S       STOP
C
C     F       RESPECIFY FREE PARAMETERS
C     Q       QUERY:  DISPLAY FREE PARAMETERS
C
C     L       PLOT LAMDA LOCUS THIS TIME
C     Z       PLOT ZETA LOCUS THIS TIME
C     B       PLOT BOTH LOCI THIS TIME
C
C     R       MAGNIFY REAL AXIS BY FACTOR
C     I       MAGNIFY IMAGINARY AXIS BY FACTOR
C     O       RELOCATE ORIGIN
C     X       RESPECIFY XMAX
C     A       RESPECIFY XMIN
C     Y       RESPECIFY YMAX
C     E       DISPLAY AXIS LIMITS
C
C     T       ENTER OVERPLOT MODE
C     U       LEAVE OVERPLOT MODE
C     V       PLOT OVER LAST LAMDA PLOT
C     W       START NEW LAMDA PAGE
C
C     D       ENTER AUTOMATIC-RESCALE MODE
C     H       LEAVE AUTOMATIC-RESCALE MODE
C     C       COMPUTE SCALE
C     J       DO NOT COMPUTE SCALE
C
C     K       DISPLAY MIDLUNG WEDGE ANGLE
C
      COMMON/METHOD/ICHAR(0/10,0/10),N2,NL,ID,IPLOT
      REAL ICHAR
      DOUBLE PRECISION ID, TID
      COMMON /OVERLAY/ X(3), FP(0/49), NF, N, PLOT, SCALE
      COMMON /USERINFO/ XMIN, XMAX, YMAX, ALPHA
      LOGICAL PLOT,SCALE
      REAL PI/3.1415926535897932/
C
      IF (IPLOT .EQ. 0) GO TO 19
      N=N+1
C
C     INITIALIZE PLOTTER INSTRUCTIONS
      CALL SLITET(1)
      CALL SLITET(2)
      IF (SCALE) SENSE LIGHT 1
      IF (PLOT) SENSE LIGHT 2
C     PUT SERIAL NUMBER N INTO ID
   15 TID=ID
      FORMAT (A6,I3)) ENCODE (6,16,ID),TID,N
C
C     ACCEPT NEW LETTER AND BRANCH
      INTEGER   LETTER (25)/'N','M','O','S','P','R','L','Z','B','X',
     2'I','A','Y','A','V','E','T','U','V','W','D','H','C','J','K'/
    1 FORMAT (1X,'A1') ))) TYPE 1, ID
    2 FORMAT (A1)) ACCEPT 2,IN
      DO 4 I=1,25
    4 IF (IN .EQ. LETTER(I)) GO TO (20,25,30,40,55,57,60,61,62,70,72,
     273,74,76,77,78,80,82,84,86,90,92,94,96,99)) I
    5 FORMAT (1X'MAIN ERROR')) TYPE 5) GO TO 1
C
C     N       READ IN NEW METHOD FROM CARDS
   19 FORMAT (1X'N')) TYPE 19
   20 CALL READN (ID,NL,N2,FP,ICHAR)
      N=1
      IPLOT=1
      PLOT=.FALSE.; SCALE=.TRUE.
      CALL SLITET (1)
      SENSE LIGHT 2
      GO TO 19
C     S       NEW SERIAL NUMBER
   25 FORMAT (1X'LARGEST SERIAL NUMBER THAN ',I,'.X:I ')) TYPE 25, N
   26 FORMAT (I)) ACCEPT 26, I
      IF (I .LT. N) GO TO 5
      N=I)) GO TO 19
C     S       GO (RETURN TO CALLING PROGRAM)
C
   30 FORMAT (1X'1',A8//'PARAMETER VALUES'1/N(I3)' '022.16/1//)
      PRINT 30,ID,NF+1, (I, FP(I), I=0,NF)
      CALL COMMON (ILOSS)
      IF (ILOSS.EQ.3) N=N+1) GO TO 19
      IF (SENSE LIGHT 11) 32,33
   32 SENSE LIGHT 10
   33 IF (SENSE LIGHT 10) 34,35
   34 SENSE LIGHT 2
   35 IF (IPLOT .GT. 1) SENSE LIGHT 11
      RETURN
C     S       STOP
C
   49 FORMAT (1X'PROBABLE TERMINATION' 10X'R')) TYPE 49
      PRINT 49, STOP
C
C     F       RESPECIFY FREE PARAMETERS
   55 FORMAT (1X'SUBSCRIPT, AND VALUE; ')) TYPE 55
   56 FORMAT (2I)) ACCEPT 56, I, IN
      IF (I.EQ.0.OR.I.GT.NF) GO TO 1
      IF (0.LE.I.LE.NF) FP(I) = IN; GO TO 55
      GO TO 5
C
C     Q       QUERY:  DISPLAY FREE PARAMETERS
   57 FORMAT (1X'PARAMETER VALUES...')) TYPE 57
   58 FORMAT (1X' I4'' 'I)
      DO 59 I=0,NF
      IF (FP(I).NE.0) TYPE 58, I, FP(I)
```

```
 59   CONTINUE
      GO TO 1
C
C     L      PLOT LAMDA LOCUS THIS TIME
 60   FORMAT ('PLOT LAMDA LOCUS.');  TYPE 60
      IPLOT=1; GO TO 1
C
C     Z      PLOT ZETA LOCUS THIS TIME
 61   FORMAT ('PLOT ZETA LOCUS.');  TYPE 61
      IPLOT=2;GO TO 1
C
C     M      PLOT MATH LOCI THIS TIME
 62   FORMAT ('PLOT MATH LOCI.');  TYPE 62
      IPLOT=3; GO TO 1
C
C     R      MAGNIFY REAL AXIS BY FACTOR
 72   FORMAT ('MAGNIFY REAL AXIS BY FACTOR; ');  TYPE 72
 71   FORMAT (F)
      ACCEPT 71,A
      IF (A .LE. 0) GO TO 5
      XMAX = XMAX/A; XMIN = XMIN/A; GO TO 1
C
C     I      MAGNIFY IMAGINARY AXIS BY FACTOR
 72   FORMAT ('MAGNIFY IMAGINARY AXIS BY FACTOR; ');  TYPE 72
      ACCEPT 71; A
      IF (A .LE. 0) GO TO 5
      YMAX = YMAX/A; GO TO 1
C
C     O      RELOCATE ORIGIN
 73   FORMAT ('NEW ORIGIN POSITION; '), TYPE 73
      ACCEPT 71,A
      IF (A .LT. -1 .OR. A .GT. 1) GO TO 5
      I=XMAX-XMIN
      XMIN=(XMAX+AINT(I*(1.+A)/2.))-I
      SENSE LIGHT 10; GO TO 1
C
C     X      RESPECIFY XMAX
 74   FORMAT ('XMAX = ');  TYPE 74
      ACCEPT 71; A
      IF (A .LT. 0) GO TO 5
      IF (A .EQ. 0 .EQ. XMIN) GO TO 5
      XMAX=A
      SENSE LIGHT 10; GO TO 1
C
C     A      RESPECIFY XMIN
 76   FORMAT ('XMIN = ');  TYPE 76
      ACCEPT 71; A
      IF (A .GT. 0) GO TO 5
      IF (A .EQ. 0 .EQ. XMAX) GO TO 5
      XMIN=A
      SENSE LIGHT 10; GO TO 1
C
C     Y      RESPECIFY YMAX
 77   FORMAT ('YMAX = ');  TYPE 77
      ACCEPT 71; A
      IF (A .LE. 0) GO TO 5
      YMAX = A
      GO TO 1
C
C     F      DISPLAY AXIS LIMITS
 78   FORMAT ('XMAX =',G11.4,X,'XMIN =',G11.4,X,'YMAX =',G11.4)
      TYPE 78; XMAX, XMIN, YMAX
      GO TO 1
C
C     T      ENTER OVERPLOT MODE
 80   FORMAT ('ENTER OVERPLOT MODE.');  TYPE 80
      PLOT=.FALSE.
      IF (SENSE LIGHT 2) 93,83
C
C     U      LEAVE OVERPLOT MODE
 82   FORMAT ('LEAVE OVERPLOT MODE.');  TYPE 82
 83   PLOT=.TRUE.; GO TO 87
C
C     Z      PLOT OVER LAST LAMDA PLOT
 84   FORMAT ('PLOT NEW LAMDA PLOT ON TOP OF OLD.');  TYPE 84
 85   IF (SENSE LIGHT 2) 1,1
C
C     A      START NEW LAMDA PAGE
 86   FORMAT ('NEW LAMDA PLOT PAGE.');  TYPE 86
 87   SENSE LIGHT 2; GO TO 1
C     T      ENTER AUTOMATIC-RESCALE MODE
 90   FORMAT ('ENTER AUTOMATIC-RESCALE MODE.');  TYPE 90
      SCALE=.FALSE.
      IF (SENSE LIGHT 1) 83,83
C
C     U      LEAVE AUTOMATIC-RESCALE MODE
 92   FORMAT ('LEAVE AUTOMATIC-RESCALE MODE.');  TYPE 92
 93   SCALE=.TRUE.; GO TO 87
C
C     R      COMPUTE SCALE
 94   FORMAT ('AUTOMATIC RESCALE.');  TYPE 94
 95   IF (SENSE LIGHT 1) 1,1
C
C     J      DO NOT COMPUTE SCALE
 96   FORMAT ('DO NOT RESCALE.');  TYPE 96
 97   SENSE LIGHT 1; GO TO 1
C
C     K      DISPLAY WINLUND WEDGE ANGLE
 98   FORMAT ('ALPHA =',G11.4)
      TYPE 98; 180.0*ALPHA/PI
      GO TO 1
      END
```

```
      SUBROUTINE READM (ID, L, K, FP, IF)

C       THIS S/R IS USED TO READ IN METHODS OF THE 'SU' TYPE
C       'SU' METHODS ARE BASICALLY GEAR TYPE METHODS WITH THE ORDER
C         EQUAL TO THE NUMBER OF BACK POINTS.  THEY ARE CHARACTERIZED BY
C         HAVING ALL OF THEIR ROOTS AS M TENDS TO INFINITY ALL ZERO.
C         THE 'SU' METHODS ARE COMPOSITE GEAR TYPE METHODS WITH
C         THEIR ROOTS AS M ---> INFINITY ALL ZERO BUT FREE
C         PARAMETERS ARE AVAILABLE TO IMPROVE THE METHOD.  THE FREE
C         PARAMETERS ARE ALL SET INTO THE BACK DETAILS.
C       'SU' METHODS ARE CYCLIC IN THE SENSE OF DAHLQUIST AND MANSFIELD
C         THEY ARE NOT IN CANONICAL FORMAT, HOWEVER.  ALL OF THE BACK
C         B MATRICES ARE ZERO.  THE B=0 MATRIX AS WELL AS THE A=0
C         MATRIX ARE TRIANGULAR.

C       THE METHOD IS INPUTTED BY K+2 CARDS AS FOLLOWS
C       CARD 1---K (I FORMAT) ID (A8 FORMAT IN COL 71-75)
C       CARD 2 --- SA(1,K+1)   ...   SA(K+1,K+1)
C       CARD 3 --- SA(1,K)     ...   SA(K+1,K)
C                   ...                ...
C                  SA(1,1)     ...   SA(K+1,1)
C                  SB(1,K+1)   ...   SB(K+1,K+1)
C                   ...                ...
C                  SB(1,1)     ...   SB(K+1,1)
C       WHERE:
C       K  = # OF BACK POINTS PER BASIC GEAR EQUATION
C       NC = 2*(K+1)
C       THE (K+1)'ST # ON EAC
C       SA(K+1,J) AND SB(K+1,J) CORRESPONDS TO GEAR'S METHOD
C       SA(I,J) AND SB(I,J) CORRESPONDS TO THE PARAMETERIZATION
C         IN TERMS OF THE BETA(I-K+J) PARAMETER   (J=1, ---, K)

C        A TEMP RAD FILE IS USED---F12 IN ADDITION TO F11
C        THE ASSIGN CARD TO USE IS---
C             ASSIGN F12,(FILE,TEMP2),(RANDOM),(RSTORE,2),(OUTIN),(REL)
C
      DIMENSION AB(10,28), FAB(45,28), ISOURCE(45), IDEST(45), FP(10/45)
      DIMENSION SA(11,22), IFIELD(28), ABFAB(1540)
      COMMON A1, FAB, ISOURCE, IDEST, SID, SAB, IFIELD, IDUMMY(5)
      DOUBLE PRECISION ID, SID
      EQUIVALENCE (AB, ABFAB)
      REAL ICHAR(10/10,0/10)
      COMMON /METHOD/ ICHAR, NZ, NL, Q, IPLOT
      DOUBLE PRECISION Q
      INTEGER TWOKP1
C
      IF (IPLOT.EQ.0) GO TO 1000
C
C         ...AT FIRST TIME---READ IN SU METHOD FROM DISC F12
      READ DISC 2, Q, SID, K, SAB
      KP1 = K+1
    5 KP2 = KP1 + 1
      TYPE 10, SID
   10 FORMAT (8HENTER N8, 8H EQUATION FOR '8A50'1 8)
      ACCEPT 1030, L
      IF (L.EQ.0) GO TO 1000
      NC = 2*(KP1 + K+L)
      IF (L.LT.L.LE.KP1) IF (KPL.LT.1B) GO TO 30
      TYPE 20
   20 FORMAT (8HDMAIN ERROR 8)
      GO TO 5
   30 DECODE (A, 80, ID), K, L, K
   40 FORMAT (I*1321'0')
      LM1 = L-1
      L2K = L + 2*K
      TWOKP1 = 2*KP1
C
C         FILL IN ISOURCE ARRAY, IDEST ARRAY
C           AND COMPUTE NF (# OF FREE PARAMETERS)
      NF = 0
      DO 50 J=2,L
        DO 50 I=J,L
   50     IDEST((NF+NF+1)) = I
      DO 60 I=1,NF
   60   ISOURCE(I) = I
      TYPE 70, NF, ID
   70 FORMAT (I 8TOTAL FREE PARAMETERS FOR '8A50'8)
C
C         CLEAR AB AND FAB ARRAYS THEN FILL THEM IN
      DO 80 I=1,1540
   80   ABFAB(I) = 0.0
      DO 100 J=1,L
        JM1 = J-1
        DO 90 I=1,KP1
   90     AB(J,I+JM1) = SAB(KP1,I)
  100   AB(J,L+2*K+J) = SAB(KP1,TWOKP1)
      INF = 0
      DO 120 II=1,LM1
        DO 120 J=K,K+II-LM1,-1
          INF = INF + 1
          INDEX2 = (INDEX1 + II + K + J1 + KPL
          DO 110 I=1,KP1
            FAB(INF,INDEX1+I) = SAB(J,I)
  110       FAB(INF,INDEX2+I) = SAB(J,I+KP1)
  120   CONTINUE
C
C         PRINT OUT METHOD
      DO 150 I=1,NC
        BIG = 0.0
        DO 130 J=1,INF
          IF ((T+ABS(FAB(J,I))).GT.BIG) BIG = T
  130     CONTINUE
        DO 140 J=1,L
          IF ((T+ABS(AB(J,I))).GT.BIG) BIG = T
  140     CONTINUE
  150   ENCODE (20, 1030, IDUMMY, IFIELD(I)), BIG
      N1 = N2 = 1
      DO 155 I=1,KPL
        N1 = N1 + IFIELD(I) + 1
  155   N2 = N2 + IFIELD(I+KPL) + 1
      PRINT 160, ID, KPL, (IFIELD(I), I=K, I=1,KPL),
     1                    KPL, (IFIELD(KPL+I), I=K,I=1,KPL)
  160 FORMAT (8I  '8A50'8//X*(X,I*10 I 8 N(I*,X))
      PRINT 170, N1, N2
  170 FORMAT (X'(*,I'*,*N(I*,*))
      PRINT 180,    (KPL, (IFIELD(I), AB(J,I), I=1,KPL),
```

```
     1                 KPL, (IFIFLD(I), AB(J,I), I=KPL+1,NC), J=1,L)
      PRINT 170, N1, N2
      PRINT 180,     (KPL, (IFIFLD(I), FAB(J,I), I=1,KPL),
                     KPL, (IFIFLD(I), FAB(J,I), I=KPL+1,NC), J=1,NF)
     1
  180 FORMAT ((X,N(X,I4), ' : ' N(IN,X)))
      N2 = 0
  190 N1 = N2 + 1
      N2 = MIN (N1+19, NF)
      J = N2-N1+1
      PRINT 200, J, (I, I=N1,N2)
  200 FORMAT (/' FREE PARAM   'N15)
      PRINT 210, J, (ISOURCE(I), I=N1,N2)
  210 FORMAT (' MUL VECTOR    'N15)
      PRINT 220, J, (IDEST(I), I=N1,N2)
  220 FORMAT (' ADDS EQUAT    'N15)
      IF (N2.NE.NF) GO TO 190
C
C         INITIALIZE FREE PARAMETER VECTOR
      FP(0) = 1.0
      DO 230 I=1,INF
  230   FP(I) = 0.0
      WRITE DISC 1, 0, AB, FAB, ISOURCE, IDEST
      RETURN
C
C           READ IN NEW SU METHOD FROM CARDS (F:105)
 1000 READ (105, 1005, END=2000), NC, SID
 1005 FORMAT (I, T71, A5)
      K = (KP1 + NC/8) + 1
      READ 1010, (KP1, (SAB(J,I), J=1,KP1), I=1,NC)
 1010 FORMAT (:I)
      WRITE DISC 2, 0, SID, K, SAB
      DO 1040 I=1,NC
        BIG = 0.0
        DO 1020 J=1,KP1
          IF ((T=ABS(SAB(J,I))).GT.BIG) BIG = T
 1020   CONTINUE
      ENCODE (25, 1030, IDUMMY, IFIELD(I)), BIG
 1030   FORMAT (I)
 1040   CONTINUE
      PRINT 1050, SID, KP1, (IFIELD(I), I=K, I=1,KP1),
     1                  KP1, (IFIELD(KP1+I), I=K, I=1,KP1)
 1050 FORMAT (#1    '0A56' BASIC METHOD AS FOLLOWS...0//
     1                  X,N(X,I4), ' : ' N(IN,X)/)
      PRINT 1060, (KP1, (IFIELD(I), SAB(J,I), I=1,KP1),
     1                  KP1, (IFIELD(I), SAB(J,I), I=KP1+1,NC), J=1,KP1)
 1060 FORMAT ((X,N(X,I4), ' : ' N(IN,X)))
      GO TO 5
 2000 PRINT 2010
      TYPE 2020
      STOP
 2010 FORMAT (#1END OF CARD DATA#)
 2020 FORMAT (#END OF CARD DATA#)
      END
```

```
      SUBROUTINE COMPP (ILOSS)
      COMMON /OVERLAY/ X(2)
      DIMENSION Y(2), INT(4)
      EQUIVALENCE (INT, X), (INT(4), JLOSS)
      DO 5 I=1,2
    5 Y(I) = X(I)
      CALL COMPUTE
      CALL FILL
      CALL DETERMIN (DCD)
      ILOSS = JLOSS
      IF (JLOSS.NE.3) GO TO 20
      TYPE 10
   10 FORMAT (1LOSS OF SIGNIFICANCE (S/R DETERMIN)')
   20 DO 25 I=1,2
   25 X(I) = Y(I)
      RETURN
      END
```

```
      SUBROUTINE COMPUTE
C
C     3JUNE1972
C
      LOGICAL SLITET
      COMMON R(1200), IMETHOD(PR,10), FAB
      COMMON /OVERLAY/ X(3), FP(0/45), NF
      REAL 10CD, IMETHOD, ICHAR(0/10,0/10), LARGEI/16Z4FFFFFFFFFFFFFFFF/
      REAL AB(10,2P), FAB(45,2P), W(0/45)
      INTEGER ISOURCE(45), IDEST(45), TWOKPL
      EQUIVALENCE (R, AP), (R(P91), ISOURCE), (R(P91), IDEST)
      EQUIVALENCE (R(3711), W)
      COMMON /METHOD/ ICHAR, K, L
      EQUIVALENCE (FP0, W)
C
      READ DISC 1, 0, AB, FAB, ISOURCE, IDEST
      TWOKPL = 2**(KPL - K+L)
      T = IOCD (FP, W, NF+1)
      DO 15 I=0,NF
   15   W(I) = FP(I)/T
      DO 2^ LL=1,L
        DO 20 I=1,TWOKPL
          IF ((IMETHOD(I,LL)*FP0*AB(LL,I)).GT.LARGEI) SENSE LIGHT 5
   20   CONTINUE
        REPEAT 35, FOR IFP = (1,NF)
          IS = ISOURCE(IFP)
          ID = IDEST(IFP)
          DO 30 I=1,TWOKPL
            IF ((TW(IFP)*FAB(IS,I)).GT.LARGEI) SENSE LIGHT 5
            IF ((IMETHOD(I,ID) + IMETHOD(I,ID)*T).GT.LARGEI) SENSE LIGHT 5
   30       CONTINUE
   35   CONTINUE
        DO 50 LL=1,L
          T = IOCD (IMETHOD(1,LL), N, TWOKPL)
          DO 45 I=1,TWOKPL
   45       IMETHOD(I,LL) = IMETHOD(I,LL)/T
   50   CONTINUE
        IF (SLITET(5)) PRINT 65
   65   FORMAT (/'POSSIBLE LOSS OF SIGNIFICANCE (S/R COMPUTE)')
        RETURN
        END
```

```
      SUBROUTINE FILL
C
C       FILLS ARRAY R(ZETA,LAMDA) IN A FORM COMPATIBLE WITH DETERMIN.
C       METHOD IS PICKED UP FROM IMETHOD(28,10) WHERE METHOD IS GIVEN AS
C
C           L
C         SUM <A(I,LL)*X(N+I) + MUB(I,LL)*DX(N+I)> = 0     LL=1,....L
C        I=K+1
C             A(I,LL) STORED AT IMETHOD(I+K,LL)
C             B(I,LL) STORED AT IMETHOD(I+K+L,LL)
C                 (I=K+1,....L)  LL=1,....L)
C       METHOD IS PRINTED.
C       CHARACTERISTIC EQUATION IS CLEARED
C       ALL PARAMETERS FOR S/B DETERMIN ARE SET
C       EACH EQUATION OF THE METHOD IS ASSUMED IN GCD FORM
C         BUT EACH COLUMN OF R(ZETA,LAMDA) IS GCD'D.
C       IT IS ASSUMED THAT ALL POINTS ARE KEPT
C
      COMMON /METHOD/ ICHAR(0/10,0/10), NZ, NL, ID
      COMMON R(1P33), IMETHOD(28,10)
      REAL IMETHOD, ICHAR, ISCD, R(1), IC(121)
      DOUBLE PRECISION ID
      COMMON /OVERLAY/ NK, L, K, ILOSS
      INTEGER IFIELD(10), TWKPL
      EQUIVALENCE (IC, ICHAR, IFIELD), (R, IMETHOD)
C
      NW = ((K+NZ) + (L+NL) +1)/NL
      TWKPL = 2*(KPL = K+L)
C
C         PRINT OUT METHOD
      DO 20 LL=1,L
        BIG = 0
        DO 10 I=1,TWKPL
          IF ((T+ABS(IMETHOD(I,LL))) .GT. BIG) BIG = T
   10     CONTINUE
        ENCODE (20, 15, R, IFIFLD(LL)), BIG
   15   FORMAT (I)
   20   CONTINUE
      PRINT 25, L, (IFIELD(LL), LL, LL=1,L)
   25 FORMAT (/////'    I    ', N(2X,I4))
      PRINT 30, (I,K, L, (IFIELD(LL), IMETHOD(I,LL), LL=1,L), I=1,KPL)
   30 FORMAT (/ ('A(I,'I2')' N(2X,I4)) )
      PRINT 35,(I=K,L, (IFIELD(LL), IMETHOD(I+KPL,LL), LL=1,L), I=1,KPL)
   35 FORMAT (/ ('B(I,'I2')' N(2X,I4)) )
C
C         CLEAR BOTH R(ZETA,LAMDA) AND ICHAR
      DO 40 I=1,1200
   40   R(I) = 0
      DO 45 I=1,121
   45 IC(I) = 0
C
C         FILL IN R(ZETA,LAMDA)
      NTERMS = L*(NENTRY + 2*(KKP1 + NK+1))
      DO 70 LL=1,L
        IB = IBASE + IBB + (LL-1)*NENTRY + 1
        J = 1
        DO 50 I=1,KPL
          R(IB) = IMETHOD(I,LL)
          IF ((J+J-1).LE.L) IB = IB + NTERMS; GO TO 50
          J = 1
          IB = IBASE + IBASE + 1
   50     CONTINUE
        IB = IBASE + IBB + NKP1
        J = 1
        DO 60 I=KPL+1,TWKPL
          R(IB) = IMETHOD(I,LL)
          IF ((J+J-1).LE.L) IB = IB + NTERMS; GO TO 60
          J = 1
          IB = IBASE + IBASE + 1
   60     CONTINUE
   70   CONTINUE
        IB = 1
        DO 90 LL=1,L
          T = IGCD (R(IB), R, NTERMS)
          DO 80 IB= IB,IB+NTERMS-1
   80       R(IB) = R(IB)/T
   90   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DETERMIN (BCD)

C     12JUN01972

C        THIS S/R COMPUTES P(ZETA,LAMDA) AS THE DETERMINANT
C          OF R(ZETA,LAMDA)
C        INTERMEDIATE CALCULATIONS ARE CARRIED OUT WITH EACH
C          TERM MAINTAINED AS A POLYNOMIAL IN POWERS OF A BASE,
C          UP TO SIXTH POWER...HENCE NUMERICAL RANGE CAN BE EXTENDED.
C     REAL BASE/30030.0D0/
C          THE ABOVE BASE IS CHOSEN AS IT IS THE PRODUCT OF
C          THE FOLLOWING PRIME NUMBERS....
C             1, 2, 3, 5, 7, 11, 13
C        NUMBERS ARE RESTORED TO NORMAL FORM AFTER A CHECK
C          FOR A POSSIBLE LOSS OF SIGNIFICANCE IS MADE...IF SO
C          PARAMETER ILOSS IS SET TO 2 AND LOG (BASE 10) OF
C          THE LARGEST NO., IN MAGNITUDE, IS PRINTED.
C          NO OTHER I/O IS PERFORMED.
C        THE BCD OF THE CHARACTERISTIC EQUATION IS RETURNED.  THE
C          CHARACTERISTIC EQUATION IS RETURNED IN BCD FORM.

C        THE PARAMETER ILOSS IS SET AS FOLLOWS...
C             ILOSS      MEANING
C             -----      ------------------------------------
C               1        RESULT IS CORRECT
C               2        LOSS OF SIGNIFICANCE IN FINAL RESULT
C               3        LOSS OF SIGNIFICANCE IN INTERIM RESULT
C                        S/R DETERMIN ABORTED

C           SUBROUTINES AND FUNCTIONS REQUIRED...
C                 IPOLYVAL
C                 IGCD
C
C        COMMON R(1200), PRODUCT(726), SUM(726), TEMP(726)
C        COMMON /WRKAREA/ TERM(88), WRK(130)
C        COMMON /METHOD/ ICHAR, NT, NL, ID, IPLOT, INCR, ITIME
C        COMMON /OVERLAY/ NK, L, K, ILOSS
C        REAL ICHARID/10.1/10/, IPOLYVAL, IGCD
C        REAL LARGEST1 /1A2AEFFFFFFFFFFFFFF/
C        DOUBLE PRECISION ID
C        INTEGER INDEX(10), JINDEX(10)
C        EQUIVALENCE (INDEX(1), I1), (INDEX(2), I2), (INDEX(3), I3),
C       1            (INDEX(4), I4), (INDEX(5), I5), (INDEX(6), I6),
C       2            (INDEX(7), I7), (INDEX(8), I8), (INDEX(9), I9),
C       3            (INDEX(10), I10)

C        FOLLOWING CARDS USED SOLELY TO CONSERVE SPACE
C        DIMENSION INTS(100), REALS(80)
C        EQUIVALENCE (R(1101), REALS), (R(1151), INTS)

C             INTEGER EQUIVALENCE
C        EQUIVALENCE (INTS(1), INDEX), (INTS(11), JINDEX)
C        EQUIVALENCE (INTS(21), I1), (INTS(22), IE1), (INTS(23), IN),
C       1            (INTS(24), IP), (INTS(25), IS), (INTS(26), ISIGN),
C       2            (INTS(27), IPWKP1), (INTS(28), IZERO0
C        EQUIVALENCE (INTS(31), J1), (INTS(32), JP), (INTS(33), JPP),
C       1            (INTS(34), JPT), (INTS(35), JS), (INTS(36), J11),
C       2            (INTS(37), J2), (INTS(38), J3), (INTS(39), J22),
C       3            (INTS(40), J33)
C        EQUIVALENCE                 (INTS(52), KP), (INTS(53), KPP),
C       1            (INTS(54), KPT), (INTS(55), KP1), (INTS(56), K31),
C       2            (INTS(57), KP2), (INTS(58), K33), (INTS(59), KNP1
C        EQUIVALENCE (INTS(61), L11), (INTS(62), LPP), (INTS(63), L33)
C        EQUIVALENCE (INTS(71), N1), (INTS(72), NKP1), (INTS(73), NR1),
C       1            (INTS(74), NR1), (INTS(75), NTERMS)

C             REAL EQUIVALENCE
C        EQUIVALENCE (REALS(1), RIO)
C        EQUIVALENCE (REALS(11), S1), (REALS(12), SION)
C        EQUIVALENCE (REALS(21), T1), (REALS(22), TRUNCATE)

C
      ILOSS = 1
      KNP = 1 + (IZNKP1 + P*(NKP1 + NK+1))*(L+1)
      NTERMS = 6*(L+1)*((KP1*K+1))

C
C             CLEAR SUM
      DO 10 I=1,NTERMS,1,=1
   10   SUM(I5) = 0
      DO 901 I1=L,1,=1
        IF (L.EQ.1) GO TO 500
        DO 902 I2=L,1,=1
          I=1
          GO TO 1000
  150     DO 903 I3=L,1,=1
            I=2
            GO TO 1000
  160       DO 904 I4=L,1,=1
              I=3
              GO TO 1000
  170         DO 905 I5=L,1,=1
                I=4
                GO TO 1000
  180           DO 906 I6=L,1,=1
                  I=5
                  GO TO 1000
  190             DO 907 I7=L,1,=1
                    I=6
                    GO TO 1000
  200               DO 908 I8=L,1,=1
                      I=7
                      GO TO 1000
  210                 DO 909 I9=L,1,=1
                        I=8
                        GO TO 1000
  220                   DO 910 I10=L,1,=1
                          I=9
                          GO TO 1000
  230                     CONTINUE

C
C
C                 MAIN COMPUTATION STARTS HERE
  500                 DO 510 I=NTERMS,1,=1
  510                   PRODUCT(I) = 0
C                       J = 1
                        GO TO 2000
```

```
520         IF (IZERO.EQ.0) GO TO 526                                   DETER117
            DO 525 KP=1,NP,1                                            DETER118   543
              KPP = KP*KP1                                              DETER119
              KPT = KP*NKP1                                             DETER120
              DO 525 JP=NK,NP,1                                         DETER121
                JPP = 4*(JP*KPP)                                        DETER122
                JPT = 4*(JP*KPT)                                        DETER123   544
                DO 525 IP=4,1,-1                                        DETER124   545
                  PRODUCT(IP+JPP) = TERM(IP+JPT)                        DETER125
525         REPEAT 540, FOR J=(Z,L)                                     DETER126   C
526           GO TO 2000                                                DETER127   C
C                                                                       DETER128
C           TAKE OUTER PRODUCT OF TERM AND PRODUCT                      DETER129   546
530         IF (IZERO.EQ.0) GO TO 540                                   DETER130
            DO 532 I=NTERMS,1,-1                                        DETER131   542
              TEMP(I) = 0                                               DETER132   549
532           DO 534 L3=1,NP,1                                          DETER133   550
                L33 = L3*NKP1                                           DETER134
                DO 534 J3=L,L3,NP,1                                     DETER135
                  J33 = J3*KP1                                          DETER136
                  K33 = (L3*J3)*KP1                                     DETER137     1
                  DO 534 L2=NK,NP,1                                     DETER138   552
                    L22 = 1 + 4*(L2*L33)                                DETER139
                    DO 534 J2=L2,NP,1                                   DETER140
                      J22 = 1 + 4*(J2*J33)                              DETER141   555
                      K22 = 1 + 4*(J2*L2 + K33)                         DETER142
                      DO 534 L1=3,NP,1                                  DETER143     1
                        L11 = L1*L22                                    DETER144   559
                        DO 534 J1=4,NP,1                                DETER145   560
                          S = TERM(L11)*PRODUCT(J1+J22)                 DETER146     1
                          L1PJ1 = L1 + J1                               DETER147     2
                          REPEAT 533, WHILE L1PJ1.GT.5                  DETER148     3
                            S = S*BASE                                  DETER149
                            IF (ABS (S) .GT. LARGEST)                   DETER150
                              GO TO 565                                 DETER151   565
                          L1PJ1 = L1PJ1 + 1                             DETER152
533                       TEMP(L1PJ1 + K22)                             DETER153   C
                            = TEMP(L1PJ1 + K22) + S                     DETER154   910
534                     CONTINUE                                        DETER155   909
            DO 538 N=NTERMS,1,-6                                        DETER156   908
              S = TEMP(N)                                               DETER157   907
              DO 537 I=0,4                                              DETER158   906
                S = TEMP(N-I)                                           DETER159   905
                  + (S-(PRODUCT(5-I)*AMOD(S,BASE)))/BASE                DETER160   904
537           CONTINUE                                                  DETER161   903
              PRODUCT(N-5) = S                                          DETER162   902
538           CONTINUE                                                  DETER163   901
540         CONTINUE                                                    DETER164   C
C                                                                       DETER165   C
C                   COMPUTE SIGN OF TERM                                DETER166   C
            IF (L.EQ.1) GO TO 546                                       DETER167   C
            ISIGN = +1                                                  DETER168
C                                                                       DETER169
            DO 541 I=L,1,-1                                             DETER170
541           JINDEX(I) = INDEX(I)                                      DETER171
            DO 545 I=L,2,-1                                             DETER172
              REPEAT 542, FOR J=(1,I-1)                                 DETER173
                IF (JINDEX(J).EQ.I) GO TO 543                           DETER174
542             CONTINUE                                                DETER175
              GO TO 545                                                 DETER176
              DO 544 M=J+1,I                                            DETER177
544             ISIGN = -ISIGN                                          DETER178
                IT = JINDEX(M-1)                                        DETER179
                JINDEX(M-1) = JINDEX(M)                                 DETER180
                JINDEX(M) = IT                                          DETER181
                CONTINUE                                                DETER182
            CONTINUE                                                    DETER183
C                                                                       DETER184
C           ADD PRODUCT AND SUM                                         DETER185
            IF (ISIGN.LT.0) GO TO 548                                   DETER186
            DO 547 I=NTERMS,1,-1                                        DETER187
547           SUM(I) = SUM(I) + PRODUCT(I)                              DETER188
            GO TO 550                                                   DETER189
            DO 549 I=NTERMS,1,-1                                        DETER190
549           SUM(I) = SUM(I) - PRODUCT(I)                              DETER191
            DO 555 J=NTERMS-5,1,-6                                      DETER192
              S = SUM(J)                                                DETER193
              DO 552 I=0,4                                              DETER194
                S = SUM(J+I)                                            DETER195
                  + (S- (SUM(J+I)*AMOD(S,BASE)))/BASE                   DETER196
                CONTINUE                                                DETER197
              SUM(J-5) = S                                              DETER198
              IF (ABS(S) .GT. LARGEST) GO TO 559                        DETER199
              CONTINUE                                                  DETER200
            GO TO (901, 902, 903, 904, 905, 906, 907, 908,              DETER201
                   909, 910), L                                         DETER202
            PRINT 560, ALOG10(ABS(S)) + 5,*ALOG10(BASE)                 DETER203
            FORMAT (//'******LOSS OF SIGNIFICANCE IN '                  DETER204
                      'INTERIM RESULT (S/B DETERMIN)'                   DETER205
                      '***LOG (BASE 10) = 'F5.1/                        DETER206
                      '******RUN ABORTED')                              DETER207
            ILOSS = 3                                                   DETER208
            GO TO 970                                                   DETER209
            BIG = ABS (S)                                               DETER210
            GO TO 559                                                   DETER211
                CONTINUE                                                DETER212
              CONTINUE                                                  DETER213
            CONTINUE                                                    DETER214
          CONTINUE                                                      DETER215
        CONTINUE                                                        DETER216
      CONTINUE                                                          DETER217
    CONTINUE                                                            DETER218
  CONTINUE                                                              DETER219
CONTINUE                                                                DETER220
CONTINUE                                                                DETER221
CONTINUE                                                                DETER222
CONTINUE                                                                DETER223
C                                                                       DETER224
C         P(ZETA,LAMDA) IN EXPANDED FORM IN SUM                         DETER225
C         GCD IT FIRST                                                  DETER226
C         BREAK IT DOWN TO NORMAL FORM AND CHECK FOR OVERFLOW           DETER227
C                                                                       DETER228
          GCD = 1*2                                                     DETER229
          NR1 = (NZ+1)*(NL+1)+1                                         DETER230
          TEMP(NR1) = BASE                                              DETER231
          DO 915 I=NR1-1,1,-1                                           DETER232
915         TEMP (I) = SUM(6*I-5)                                       DETER233
920       IF ((T = IGCD (TEMP, PRODUCT, NR1) .EQ.1) GO TO 925           DETER234
          GCD = GCD*T                                                   DETER235
          DO 924 I=NR1-1,1,-1                                           DETER236
```

```
      S = SUM( (I6 + 6*I))
      DO 922 J=0,4
        S = SUM(I6+J+1) + (S + T*((SUM(I6-J) * AINT(S/T))))*BASE
922   TEMP(I) = SUM(I6+5) * S/T
      GO TO 920
C
925   PIO = LARGEST1
      DO 930 J5=NZ,0,-1
        DO 930 K5=NL,0,-1
          SIGN = +1.
          IF (MOD(K5,2).EQ.1) SIGN = -1.
          ICHAR(K5,J5)=SIGN*IPOLYVAL (SUM(I+6+(J5*KP1+K5)),6,BASE, IER)
          IF (IER.EQ.2) ILOSS = 2
930       CONTINUE
      GO TO (945, 940), ILOSS
940   DO 950 J5=NZ,0,-1
        DO 950 K5=NL,0,-1
          IF ( (T=ABS(ICHAR(K5,J5))) .GT. BIG) BIG = T
950       CONTINUE
      PRINT 960, ALOG10 (BIG)
960   FORMAT (//'*****LOSS OF SIGNIFICANCE (S/R DETERMIN) --- LOG OF MAX
     1 MA. = ' F5.1/'*****CONTINUING ANYWAY')
945   IF ( (T = IGCD ((ICHAR, PRODUCT, 121) )-EQ.1.0) GO TO 970
      IGCD = GCD*T
      DO 967 J=NZ,0,-1
        DO 967 I=NL,0,-1
967       ICHAR(I,J) = ICHAR(I,J)/T
      IF (ILOSS.EQ.2) IF (T.NE.1.0) PRINT 969, T
969   FORMAT (1*****MOTE: IGCD = F1)
970   RETURN
C
C         THIS SECTION CHECKS TO SEE IF INDEX(I) = INDEX(J) (J<I)
C         IF SO RETURN IS TO 902, ... 910***OTHERWISE RETURN
C         IS TO 150, ... 230 UNLESS L = I IN WHICH CASE RETURN IS TO
C         STATEMENT # 500.
C         NOTE: AS USED BELOW, I OF THE CODING CORRESPONDS TO I+1
C                   OF THE ABOVE COMMENTS.
C
1000  IN = INDEX(I+1)
      DO 1010 J=I,1,-1
1010    IF (INDEX(J).EQ.IN)
     1      GO TO (902, 903, 904, 905, 906, 907, 908, 909, 910), I
        IF (L.EQ.I+1) GO TO 500
        GO TO (150, 160, 170, 180, 190, 200, 210, 2201, I
C
C         THIS S/R TAKES THE POLYNOMIAL OF ZETA AND LAMDA
C         AT COLUMN 'J' AND ROW 'INDEX(J)'
C         AND BREAKS IT APART INTO TERM WHERE EACH ELEMENT OF
C         OF THE ORIGINAL POLYNOMIAL IS EXPRESSED AS A POLYNOMIAL
C         IN THE BASE UP TO 3RD ORDER
C         IZERO IS SET TO 0 (3) IF ALL #'S ARE (NOT) ZERO.
C
2000  IZERO = 0
      NB = KNN + IPNKP1*(INDEX(J) + J*L)
      DO 2040 J3=1,0,-1
        NB1 = NB + (J3) * J3*NKP1)
        DO 2030 J2=NY,0,-1
          JPP = NP(J2 + J3)
          IF ((T = R(NB1+JPP))-EQ.0) GO TO 2020
        DO 2010 J1=1,3
          T = (T * (TERM(J1+J22) * AMOD (T, BASE)))/BASE
2010    CONTINUE
        TERM(J22+4) = T
        IZERO = 3
        GO TO 2030
2020    TERM(J22+1) = TERM(J22+2) * TERM(J22+3) * TERM(J22+4) * 5.0
2030    CONTINUE
2040  CONTINUE
      IF (J.NE.1) GO TO 530
      GO TO 520
      END
```

```
      REAL FUNCTION IGCD (A, W, N)
C     16FEB1972
C
C        GREATEST COMMON DIVISOR OF THE N BY 1 ARRAY A IS RETURNED
C        W IS A WORK AREA OF AT LEAST N LOCATIONS
C        EUCLID'S ALGORITM IS USED
C        PSEUDO ALL-INTEGER ARITHMETIC IS PERFORMED
C
      DIMENSION A(1), W(1)
C
      IS = 0
      DO 10 I=1,N
        IF ( (T=ABS(A(I))) .EQ.0) GO TO 10
        IS = IS+1
        W(IS) = T
 10   CONTINUE
      IF (IS.GT.0) GO TO 20
      IGCD = 1
      RETURN
 20   IGCD = W(IS)
 25   IF (IS.EQ.1) RETURN
      DO 30 I=1,IS-1
        IF ( (T=W(I)) .LT.IGCD) IGCD = T
 30     CONTINUE
      IA = 0
      DO 40 I=1,IS
        IF ( (T=AMOD(W(I),IGCD)) .EQ.0) GO TO 40
        IA = IA+1
        W(IA) = T
 40     CONTINUE
      IS = IA+1
      W(IS) = IGCD
      GO TO 25
      END
```

```
      REAL FUNCTION IPOLYVAL (A, N, X, IER)
C     16FEB1972
C
C        USES HORNER'S METHOD TO EVALUATE
C          N
C         SUM  A(I)*X**(I-1)
C          I=1
C        IER HAS THE FOLLOWING MEANINGS....
C              1     ANSWER IS CORRECT
C              2     POSSIBLE LOSS OF SIGNIFICANCE IN RESULT
C        PSEUDO ALL-INTEGER ARITHMETIC IS PERFORMED
C
      DIMENSION A(1)
      REAL LARGEST1 /1E2HFFFFFFFFFFFFFFFF/
      IER = 1
      IF (X.EQ.0) IPOLYVAL = A(1); RETURN
      T = A(N)
      DO 10 I=N-1,1,-1
        IF (T.EQ.0) T = A(I); GO TO 10
        T = T*X
        IF (ABS (T).GT.LARGEST1) IER = 2
        T = T+A(I)
        IF (ABS (T).GT.LARGEST1) IER = 2
 10     CONTINUE
      IPOLYVAL = T
      RETURN
      END
```

APPENDIX C

FLOW CHART AND COMPUTER PROGRAM LISTING FOR DIFJMT

A detailed flow chart for DIFJMT is shown in Figure C.1. Unlike the other computer programs listed in Appendices A and B, DIFJMT is written in standard FORTRAN-IV. A computer program listing for DIFJMT can also be found in this appendix.

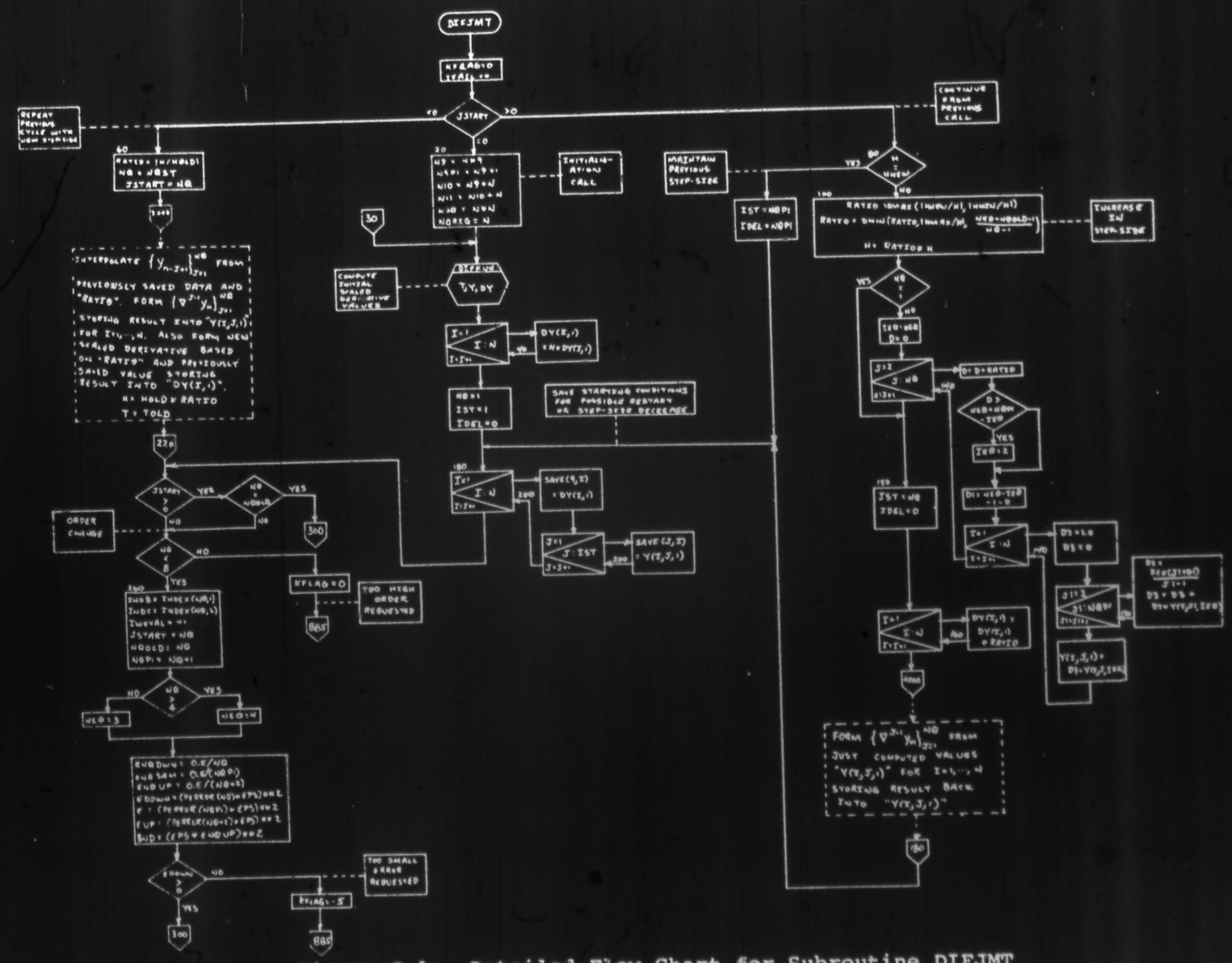


Figure C.1: Detailed Flow Chart for Subroutine DIFJMT (Sheet 1 of 3)

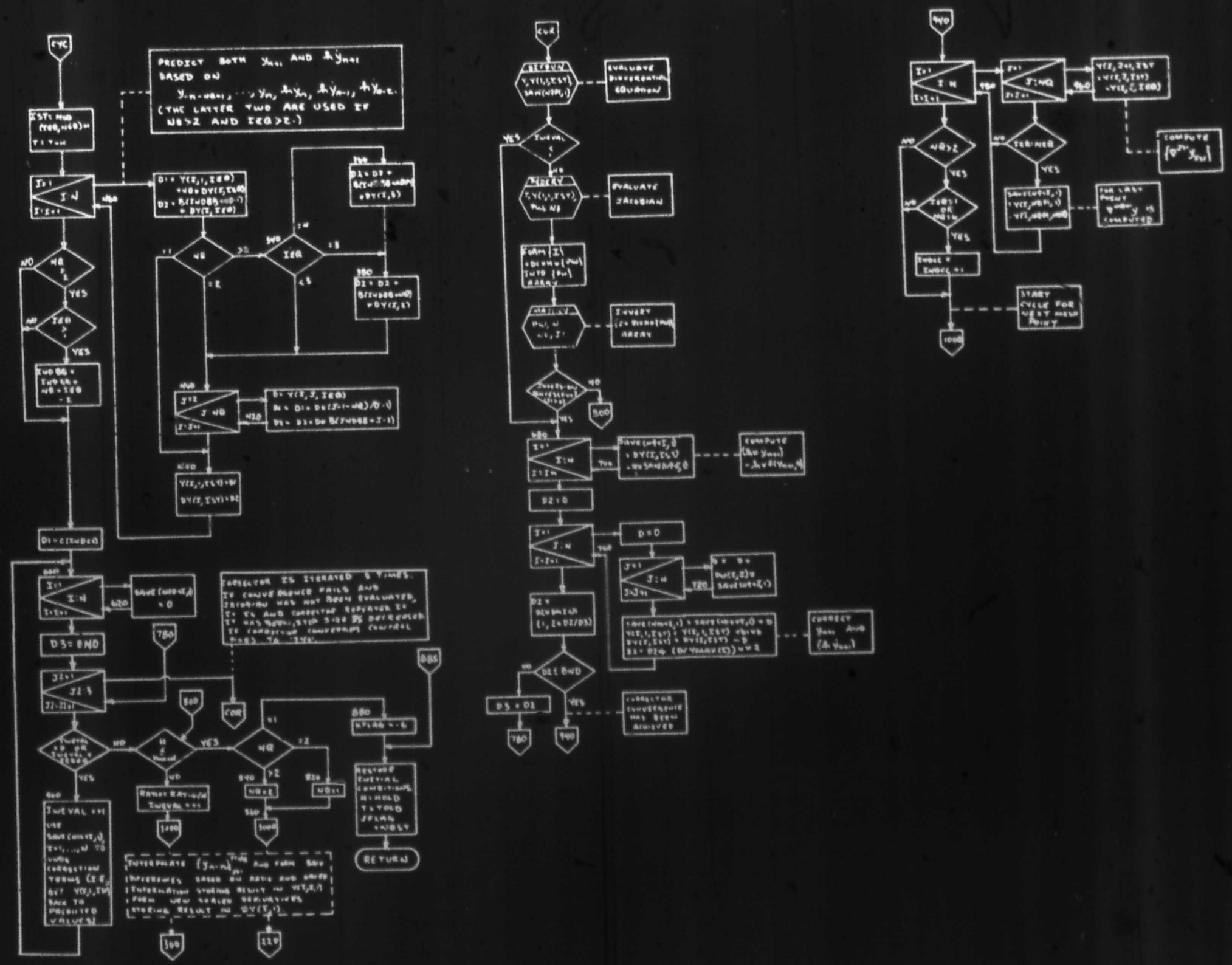


Figure C.1: Detailed Flow Chart for Subroutine DIFJMT
(Sheet 2 of 3)

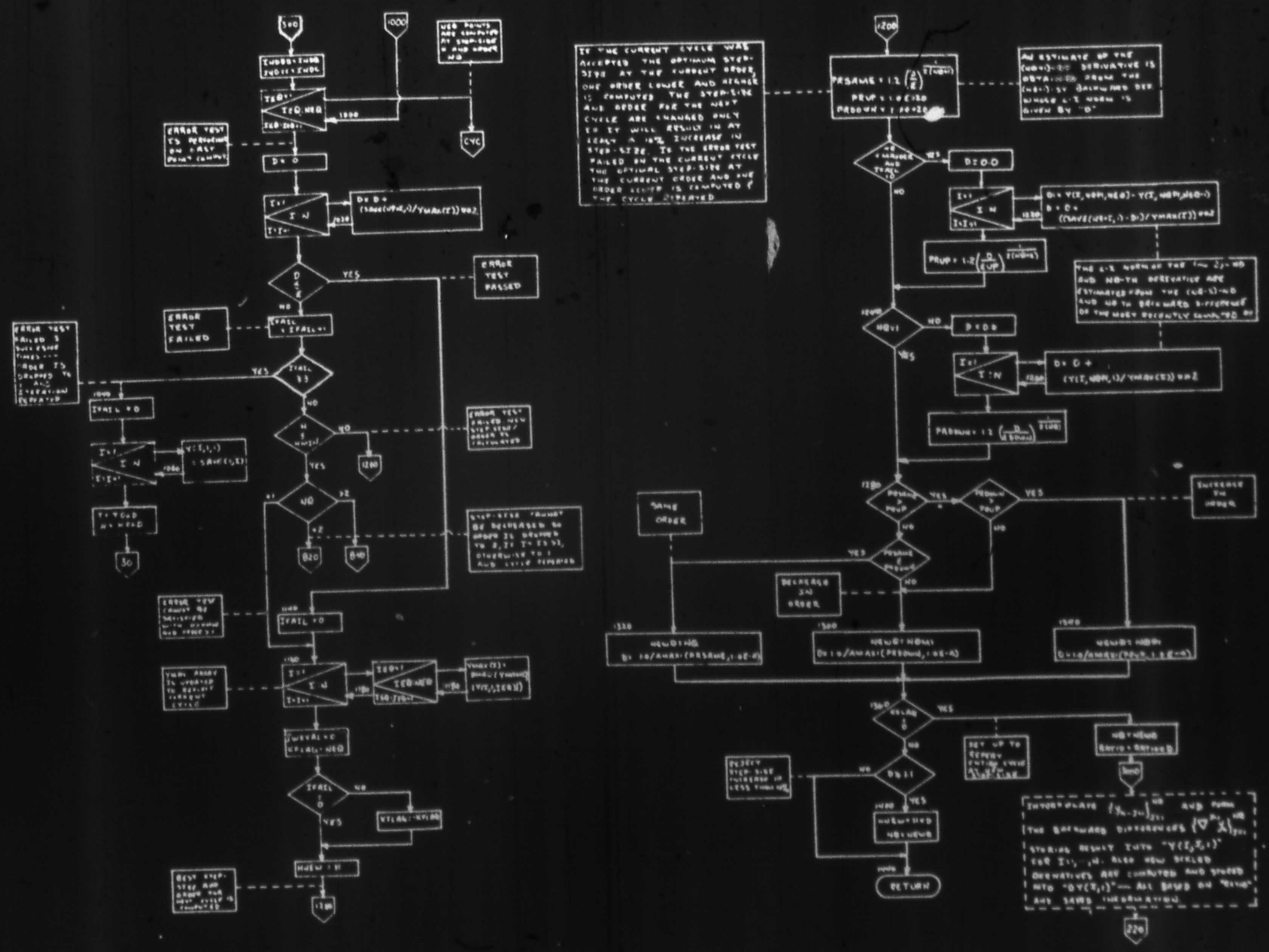


Figure C.1: Detailed Flow Chart for Subroutine DIFJMT
(Sheet 3 of 3)

```
      SUBROUTINE DIFJMT (N, T, Y, DY, SAVE, H, HMIN, HMAX, EPS, YMAX,
     *                   KFLAG, JSTART, MAXDER, PW)
C
C*********************************************************************
C*
C*    THIS PROGRAM INTEGRATES A SET OF N FIRST ORDER ORDINARY
C*  DIFFERENTIAL EQUATIONS.  THREE OR FOUR POINTS, AT A STEP SIZE H,
C*  ARE COMPUTED AT EACH CALL.  THE STEP SIZE MAY BE SPECIFIED BY THE
C*  USER AT EACH CALL, BUT IT MAY BE INCREASED OR DECREASED BY DIFJMT
C*  WITHIN THE RANGE HMIN TO HMAX IN ORDER TO ACHIEVE AS LARGE A STEP
C*  AS POSSIBLE WHILE NOT COMMITTING A SINGLE STEP ERROR WHICH IS
C*  LARGER THAN EPS IN THE L-2 NORM, WHERE EACH COMPONENT OF THE
C*  ERROR IS DIVIDED BY THE CORRESPONDING COMPONENT OF YMAX.
C*
C*    THE PROGRAM REQUIRES 3 SUBROUTINES NAMED:
C*        DIFFUN (T, Y, DY)
C*        PDERV (T, Y, PW, NO)
C*        MATINV (PW, N, NO, J)
C*    DIFFUN EVALUATES THE DERIVATIVES OF THE DEPENDENT VARIABLE Y(I)
C*  FOR I=1,...,N  AND T AND STORES THE RESULT IN THE ARRAY DY.
C*    PDERV COMPUTES THE PARTIAL DERIVATIVES OF THE DIFFERENTIAL
C*  EQUATIONS AT THE VALUES Y(I) FOR I=1,...,N AND T AND STORES THE
C*  RESULT IN THE ARRAY PW(NO,NO), WHERE PW(I,J)=PARTIAL DY(I) W.R.T.
C*  Y(J) FOR I,J=1,...,N (NO.GE.N).
C*    MATINV INVERTS THE N BY N MATRIX STORED IN THE ARRAY PW(NO,NO),
C*  NO.GE.N.  IF THE INVERSION IS SUCCESSFULL J SHOULD BE SET TO +1.
C*  OTHERWISE IT SHOULD BE SET TO -1.
C*
C*    THIS PROGRAM USES DOUBLE PRECISION FOR ALL FLOATING POINT
C*  VARIABLES EXCEPT THOSE STARTING WITH P, WHICH ARE SINGLE PRECISION
C*  TO SAVE TIME AND SPACE.
C*
C*    THE TEMPORARY STORAGE SPACE IS PROVIDED BY THE CALLER IN THE
C*  SINGLE PRECISION ARRAY PW AND THE DOUBLE PRECISION ARRAY SAVE.
C*  THE ARRAY PW IS USED TO HOLD THE MATRIX OF THE SAME NAME. SAVE IS
C*  USED TO HOLD SEVERAL ARRAYS PARTITIONED AS FOLLOWS...
C*
C*    SAVE(J,I)       1.LE.J.LE.N AND 1.LE.I.LE.N IS USED TO SAVE
C*                    Y(I,J) IN CASE A STEP HAS TO BE REPEATED.
C*                    (AT ENTRY AND AFTER ANY RESCALING).
C*
C*    SAVE(9,I)       1.LE.I.LE.N IS USED TO STORE THE DERIVATIVE OF THE
C*                    I-TH DEPENDENT VARIABLE SCALED BY H (AT ENTRY).
C*
C*    SAVE(N9+I,I)    IS USED TO STORE THE DERIVATIVES WHEN THEY ARE
C*                    EVALUATED BY DIFFUN FOR THE CORRECTOR.  IT IS ALSO
C*                    ACCESSED AS SAVE(NP1,I) AS A COMPLETE ARRAY.  IN
C*                    ADDITION IT IS USED TO CHECK IF THE ERROR CRITERION
C*                    IS MET.  (N9= 9*N AND NP1 = N+1)
C*
C*    SAVE(N10+I,I)   IS USE USED TO HOLD THE CORRECTION TERMS FOR THE
C*                    ENTIRE CORRECTOR ITERATION IN CASE THE CORRECTOR
C*                    WAS TO BE REPEATED.  (N10 = N+10)
C*
C*
C*  THE PARAMETERS IN THE CALLING SEQUENCE HAVE THE FOLLOWING MEANING:
C*
C*        N       THE NUMBER OF FIRST ORDER DIFFERENTIAL EQUATIONS TO BE
C*                  INTEGRATED.  N MAY BE DECREASED ON SUBSEQUENT CALLS IF
C*                  THE NUMBER OF ACTIVE EQUATIONS DECREASES, BUT IT MUST
C*                  NOT BE INCREASED WITHOUT CALLING WITH JSTART = 0.
C*
C*        T       THE INDEPENDENT VARIABLE.  ON ENTRY T IS THE CURRENT
C*                  SETTING OF THE INDEPENDENT VARIABLE.  ON RETURN TO THE
C*                  CALLING PROGRAM T CORRESPONDS TO THE SETTING OF THE
C*                  INDEPENDENT VARIABLE FOR THE MOST RECENT POINT.
C*
C*        Y       AN N BY 4 BY 4 ARRAY CONTAINING THE DEPENDENT VARIABLES
C*                  AND THEIR BACKWARD DIFFERENCES.  UP TO FOUR MESH POINTS
C*                  SOLUTIONS ARE RETURNED.  THE MOST RECENT IS ALWAYS AT
C*                  Y(I,J,1).  THE NEXT MOST RECENT IS RETURNED IN
C*                  Y(I,J,KFLAG).  ONLY Y(I,J,1), I=1,...,N NEED BE
C*                  SUPPLIED ON THE FIRST CALL (JSTART=0).  Y(I,J+1,K)
C*                  CONTAINS THE J-TH BACKWARD DIFFERENCE OF THE I-TH
C*                  DEPENDENT VARIABLE (FOR K=1,...,KFLAG).
C*                    IF IT IS DESIRED TO INTERPOLATE TO NON-MESH POINTS
C*                  THESE VALUES CAN BE USED.  IF THE CURRENT STEP SIZE IS
C*                  H AND THE VALUE AT T+E (E<H) IS NEEDED, FORM S=E/H AND
C*                  THEN COMPUTE
C*
C*                                   NO
C*                      Y(I)(T+E) = SUM Y(I,J+1,K)*B(J)
C*                                   J=0
C*                  WHERE K IS CHOSEN SO THAT T+E IS NEAREST THE CORRESPOND-
C*                  ING MESH POINT AND B(J+1) = B(J)*(J-S)/(J+1)
C*                  WITH B(0) = 1.
C*
C*        DY      AN N BY 4 ARRAY.  DY(I,K), 1.LE.I.LE.N, 1.LE.K.LE.KFLAG,
C*                  CONTAINS THE DERIVATIVE OF THE I-TH DEPENDENT VARIABLE
C*                  SCALED BY H.  THE DY ARRAY NEED NOT BE SUPPLIED AT THE
C*                  FIRST CALL (JSTART = 0).
C*
C*        SAVE    A BLOCK OF AT LEAST 12*N DOUBLE PRECISION FLOATING POINT
C*                  LOCATIONS.
C*
C*        H       THE STEP SIZE USED FOR THE JUST COMPLETED CYCLE.  IT MAY
C*                  BE ADJUSTED UP OR DOWN ON SUBSEQUENT INTEGRATION STEPS
C*                  IN ORDER TO ACHIEVE AN ECONOMICAL INTEGRATION.  THE
C*                  USER MAY RESPECIFY H ON ANY CALL.  IF HE DOES NOT THE
C*                  PROGRAM AUTOMATICALLY ADJUSTS THE STEP SIZE.  ON THE
C*                  FIRST CALL (JSTART=0) THE USER MUST SPECIFY AN INITIAL
C*                  STEP SIZE.  IF THE INITIAL STEP SIZE CHOICE DOES NOT
C*                  CAUSE AN ERROR GREATER THAN EPS, IN THE L-2 NORM, IT
C*                  WILL BE ACCEPTED AND SUBSEQUENTLY ADJUSTED FOR THE
C*                  LARGEST STEP SIZE POSSIBLE.
C*
C*        HMIN    THE MINIMUM STEP SIZE TO BE USED FOR THE CURRENT INTEG-
C*                  RATION STEP.  ON THE FIRST CALL (JSTART=0), HMIN SHOULD
C*                  BE CHOSEN SIGNIFICANTLY SMALLER THAN H SO AS TO AVOID
C*                  START-UP PROBLEMS IF THE ERROR CRITERION IS NOT MET ON
C*                  THE FIRST CALL.  IT MAY BE INCREASED ON SUBSEQUENT
C*                  CALLS.
C*
C*        HMAX    THE MAXIMUM STEP SIZE TO BE ATTEMPTED FOR THE CURRENT
C*                  INTEGRATION STEP.
C*
C*        EPS     THE ERROR TEST CONSTANT.  SINGLE STEP ERROR ESTIMATES
C*                  DIVIDED BY YMAX(I) MUST BE LESS THAN THIS IN THE
C*                  EUCLIDEAN NORM.  THE STEP SIZE AND/OR ORDER ARE ADJUSTED
C*                  TO ACHIEVE THIS.
C*
C*        YMAX    AN ARRAY OF N LOCATIONS WHICH CONTAINS THE MAXIMUM OF
C*                  EACH Y SEEN SO FAR.  IT SHOULD NORMALLY BE SET TO 1 IN
C*                  EACH COMPONENT BEFORE THE FIRST ENTRY.
C*
C*        KFLAG   A COMPLETION CODE.  IF KFLAG IS GREATER THAN 0 THEN
C*                  KFLAG POINTS HAVE BEEN COMPUTED.  IF KFLAG IS
C*                  =3 OR =4 THEN 3 OR 4 MESH POINTS, RESPECTIVELY,
C*                  HAVE BEEN COMPUTED WITH HMIN=H, BUT THE REQUESTED ERROR
C*                  WAS NOT ACHIEVED.  OTHER VALUES KFLAG CAN ASSUME ARE
C*                  AS FOLLOWS...
C*                    0  THE MAXIMUM ORDER SPECIFIED WAS TOO LARGE.
C*                   -5  THE REQUESTED ERROR WAS SMALLER THAN CAN BE HANDLED
C*
```

```
C*                FOR THIS PROBLEM.                                     *DIJMT117
C*             -6 CORRECTOR CONVERGENCE COULD NOT BE ACHIEVED FOR       *DIJMT118
C*                H. GT. HMIN.                                          *DIJMT119
C*    JSTART  AN INPUT INDICATOR WITH THE FOLLOWING MEANINGS...         *DIJMT120
C*             -1 REPEAT THE LAST STEP WITH A NEW H.                    *DIJMT121
C*              0 INITIALIZATION CALL.  (JSTART MUST BE SET TO 0 ON THE*DIJMT122
C*                FIRST CALL.)                                          *DIJMT123
C*             >0 CONTINUE FROM THE LAST STEP.                          *DIJMT124
C*            ON RETURN JSTART IS SET TO NO, THE MAXIMUM BACKWARD       *DIJMT125
C*            DIFFERENCE AVAILABLE IN THE Y ARRAY.  (THIS ALSO CORRES*DIJMT126
C*            PONDS TO THE ORDER OF THE METHOD USED TO COMPUTE THE      *DIJMT127
C*            POINTS.)                                                  *DIJMT128
C*    MAXDER  THE MAXIMUM ORDER THAT SHOULD BE USED IN THE CURRENT      *DIJMT129
C*            CYCLE.  (1.LE.MAXDER.LE.7)                                *DIJMT130
C*    PW      A BLOCK OF AT LEAST 4*N+2 SINGLE PRECISION FLOATING POINT *DIJMT131
C*            LOCATIONS.                                                *DIJMT132
C*************************************************************************DIJMT133
C                                                                        DIJMT134
      IMPLICIT REAL*8 (A-H, O-Z)                                         DIJMT135
      DIMENSION Y(N,8,4), DY(N,4), SAVE(N,1), YMAX(1), PW(1),            DIJMT136
     +          INDEX(7,2), B(82), C(16), PERROR(9)                      DIJMT137
C                                                                        DIJMT138
C*************************************************************************DIJMT139
C*   THE ARRAY INDEX HOLDS POINTERS AND CONSTANTS FOR THE VARIOUS       *DIJMT140
C*   ORDER METHODS.  FOR NO=1,....7 THE ENTRIES ARE AS FOLLOWS...       *DIJMT141
C*    INDEX(NO,1)   BASE INDEX FOR B ARRAY (HODY PREDICTOR).            *DIJMT142
C*    INDEX(NO,2)   BASE INDEX FOR C ARRAY (CORRECTOR).                 *DIJMT143
C*************************************************************************DIJMT144
      DATA INDEX/ 1, 2, 4, 11, 20, 34, 59,                               DIJMT145
     +            1, 2, 3,  5,  7, 10, 14/                               DIJMT146
C                                                                        DIJMT147
C*************************************************************************DIJMT148
C*   THE COEFFICIENTS APPEARING IN THE NEXT DATA STATEMENTS FOR THE     *DIJMT149
C*   B ARRAY SHOULD BE DEFINED TO THE MAXIMUM ACCURACY PERMITTED BY     *DIJMT150
C*   THE MACHINE. THEY ARE, IN THE ORDER SPECIFIED....                  *DIJMT151
C*    1                                                                 *DIJMT152
C*    -2, 1                                                             *DIJMT153
C*    -1/2, -5/4, 11/2                                                  *DIJMT154
C*    -1/2, -3/4, 17/2, 2                                               *DIJMT155
C*    -22/3, -8/3, -17/18, 25/3                                         *DIJMT156
C*    -9, -9/4, -7/8, 35/4, 9/4                                         *DIJMT157
C*    -175/12, -171/24, -71/36, -37/48, 137/12                          *DIJMT158
C*    -253/24, -179/120, -767/360, -193/180, 1979/120, 1/10             *DIJMT159
C*    -57/4, -25/8, -17/10, -169/240, 61/5, -1/20, 31/10                *DIJMT160
C*    -137/10, -117/20, -66/15, -191/120, -197/300, 147/10              *DIJMT161
C*    -353/24, -571/120, -1819/600, -79/40, -3019/6000, 1677/100, 7/20  *DIJMT162
C*    -3527/200, -1889/400, -1409/800, -3527/2400, -931/1500, 3079/200, *DIJMT163
C*        -3/20, 17/4                                                   *DIJMT164
C*    -393/20, -303/40, -253/60, -589/240, -101/75, -23/40, 983/20      *DIJMT165
C*    -264/15, -221/30, -749/180, -73/30, -803/400, -103/180, 547/30,   *DIJMT166
C*        1/2                                                           *DIJMT167
C*    -1315/72, -1151/150, -3589/900, -121/50, -4003/3000, -257/450,    *DIJMT168
C*        2737/150, -1/5, 1/2                                           *DIJMT169
C*************************************************************************DIJMT170
      DATA (B(I), I=1,82)/                                               DIJMT171
     +        1.0D0, -2.0D0, 3.0D0, -4.5D0, -1.2500, 5.400, -7.500,     DIJMT172
     +        -.7500, 4.500, 2.000, -7.3333333333333333,                DIJMT173
     +        -2.6666666666666667, -.9444444444444444,                  DIJMT174
     +        -.3333333333333333, -9.000, -2.2500, -.8740D0, 8.7500,    DIJMT175
     +        1.5500, -10.4166666666666667, -4.2083333333333333,        DIJMT176
     +        -1.9722222222222222, -.7708333333333333,                  DIJMT177
     +        11.4166666666666667, -10.4166666666666667,                 DIJMT178
     +        -4.1708333333333333, -1.4444444444444444, -.7687500,      DIJMT179
     +        11.4146666666666667, 3.100, -1.3500, -3.1250, -1.770,     DIJMT180
     +        -.750, 1.6666666666666667, 17.750, -.7500, 3.175, -13.700,DIJMT181
     +        -5.8500, -3.0666666666666667, -1.5916666666666667,        DIJMT182
     +        -.6566666666666666667, 14.750, -1.1250, -5.7130,          DIJMT183
     +        -3.0316666666666667, -1.5500, -.6531666666666667/         DIJMT184
      DATA (B(I), I=49,82)/                                              DIJMT185
     +        14.7750, 0.3500, -17.6450, -4.72950,                       DIJMT186
     +        -2.6916666666666667, -1.4495833333333333,                  DIJMT187
     +        -.8208666666666667, 15.9950, -5.1500, 3.400, -17.1500,     DIJMT188
     +        -7.57500, -4.2166666666666667, -2.5416666666666667,        DIJMT189
     +        -1.3666666666666667, -0.57500, 18.1500,                    DIJMT190
     +        -17.7333333333333333, -7.1666666666666667,                 DIJMT191
     +        -4.6161111111111111, -2.4333333333333333,                  DIJMT192
     +        -1.3383333333333333, -0.5722222222222222,                  DIJMT193
     +        18.2333333333333333, 0.500, -17.5666666666666667,          DIJMT194
     +        -7.6733333333333333, -4.0988888888888889, -2.4200,         DIJMT195
     +        -1.3383333333333333, -.5711111111111111112,                DIJMT196
     +        18.2.6666666666667, -0.200, 0.500/                         DIJMT197
C                                                                        DIJMT198
C*************************************************************************DIJMT199
C*   THE COEFFICIENTS APPEARING IN THE NEXT DATA STATEMENTS FOR THE     *DIJMT200
C*   C ARRAY SHOULD BE DEFINED TO THE MAXIMUM ACCURACY PERMITTED BY     *DIJMT201
C*   THE MACHINE.  THEY ARE, IN THE ORDER SPECIFIED....                 *DIJMT202
C*    1                                                                 *DIJMT203
C*    -2/3                                                              *DIJMT204
C*    -6/11, -2/3                                                       *DIJMT205
C*    -12/25, -16/31                                                    *DIJMT206
C*    -60/137, -600/1373, -100/193                                      *DIJMT207
C*    -20/49, -120/293, -75/184, -1200/2899                             *DIJMT208
C*    -140/363, -49/143, -1050/2437                                     *DIJMT209
C*************************************************************************DIJMT210
      DATA C/  -1.0D0, -.6666666666666666667, -.4545454545454545,        DIJMT211
     +         -.6666666666666666667, -.4800, -.4121203372580646452,     DIJMT212
     +         -.4379562043795620C4, -.3899997166785556,                 DIJMT213
     +         -.5181347150259067C, -.4281432853061224545,               DIJMT214
     +         -.4095563139931770C5, -.3767845995857391,                 DIJMT215
     +         -.5215866072208365665, -.4586743111279676659,             DIJMT216
     +         -.4195804195804195958, -.3085761151780879/                DIJMT217
C                                                                        DIJMT218
C*************************************************************************DIJMT219
C*   THE COEFFICIENTS IN THE PERROR ARRAY ARE USED TO CHECK IF THE      *DIJMT220
C*   ERROR CRITERION HAS BEEN MET AND FOR SELECTING THE STEP-SIZE       *DIJMT221
C*   AND ORDER FOR FOLLOWING STEPS. PERROR(I+1) = 1/2(I+1), I=1,....7, *DIJMT222
C*   WHERE D(I+1) = MAXIMUM DISCRETIZATION ERROR FOR ALL EQUATIONS      *DIJMT223
C*   OF ORDER I.  PERROR(1) AND PERROR(8) ARE DEFINED SOLELY FOR        *DIJMT224
C*   PROGRAMMING EASE; THEY ARE NOT USED.  THE DISCRETIZATION ERRORS    *DIJMT225
C*   FOR ORDER 1 THRU 7 ARE GIVEN AS...                                 *DIJMT226
C*        1/2, 2/9, 1/8, 10/99, 239/2916, 1699/16995, 1319/19998,       *DIJMT227
C*   ONLY SINGLE PRECISION VALUES ARE KEPT.                             *DIJMT228
C*************************************************************************DIJMT229
      DATA PERROR/ 1.0, 2.0, 4.5, 8.0, 9.3,                              DIJMT230
     +             9.693077, 9.472042, 14.785895, 1.0/                  DIJMT231
C                                                                        DIJMT232
      FFLAG = 0                                                          DIJMT233
      IFAIL = 0                                                          DIJMT234
```

```
      IF (JSTART) 60, 20, 80
C
C           INITIALIZATION CALL
   20 NO = N
      N9 = NO*9
      N9P1 = N9+1
      N10 = N9+NO
      N11 = N10+NO
      N12 = NO*NO
   30 CALL DIFFUN (T, Y, DY)
      DO 40 I=1,N
   40   DY(I,1) = HODY(I,1)
      NO = 1
      IST = 1
      IDEL = 0
      GO TO 180
C
C           USER WANTS TO REPEAT OLD STEP WITH NEW H
   60 RATIO = DABS (H/HOLD)
      NO = NOST
      JSTART = NO
      IRET = 1
      GO TO 3000
C
C           CONTINUE FROM PREVIOUS STEP
   80 IF (H.NE.HNEW) GO TO 100
      IST = NOP1
      IDEL = NOP1
      GO TO 180
C
C           NEW STEP SIZE---INTERPOLATE NEW POINTS
  100 RATIO = DMAX1 (DABS(HNEW/H), DABS(HMIN/H))
      RATIO = DMIN1 (RATIO, DABS(HMAX/H), DFLOAT(NEQ+NOP1)/DFLOAT(NO+1))
      H = RATIO*H
      IF (NO*NEQ.EQ.1) GO TO 150
      IEQ = NEQ
      D = 0.0D0
      DO 140 J=2,NO
        D = D + RATIO
        IF (D.GT.DFLOAT(NEQ+NOP1-IEQ)) IEQ = 2
        D1 = DFLOAT(NEQ-IEQ+1) * D
        DO 140 I=1,N
          D2 = 1.0D0
          D3 = 0.0D0
          DO 120 J1=2,NOP1
            D2 = D2*(DFLOAT(J1) * D1)/DFLOAT(J1-1)
  120       D3 = D3 + D2*Y(I,J1,IEQ)
  140     Y(I,J,1) = D3 + Y(I,1,IEQ)
  150 IST = NO
      IDEL = 0
      DO 160 I=1,N
  160   DY(I,1) = DY(I,1)*RATIO
      IRET = 3
      GO TO 4000
C
C           INITIALIZE SAVE ARRAY
  180 DO 200 I=1,N
        SAVE(9,I) = DY(I,1)
        DO 200 J=1,IST
  200     SAVE(J,I) = Y(I,J,1)
      NOST = NO
      RATIO = 1.0D0
      TOLD = T
      HOLD = H
C
C           CHECK IF THERE WAS AN ORDER CHANGE
C           AND IF SO SET IN APPROPRIATE PARAMETERS.
  220 IF ((JSTART.GT.0) .AND. (NO.EQ.NOOLD)) GO TO 300
      IF (NO.LT.8) GO TO 260
      KFLAG = 0
      GO TO 885
  260 INOB = INDEX(NO+1)
      INOC = INDEX(NO+2)
      NEQ = 3
      IF (NO.GT.4) NEQ = 4
      INEVAL = 1
      NOP1 = NO+1
      JSTART = NO
      NOOLD = NO
      NOP1 = NO+1
      D = DFLOAT (NO)
      ENOD2N = 0.5D0/D
      ENOSAM = 0.5D0/(D+1.0D0)
      ENOUP = 0.5D0/(D+2.0D0)
      EDWN = (PERRBR(NO)*EPS)**2
      E = (PERRBR(NOP1)*EPS)**2
      EUP = (PERRBR(NO+2)*EPS)**2
      BND = (ERRBR*EPS)**2
      IF (EDWN.GT.0.0D0) GO TO 300
      KFLAG = -5
      GO TO 885
C
C           START INTEGRATION CYCLE FOR NEQ POINTS
C
  300 INOBB = INOB
      INOCC = INOC
      DO 1000 IEQ=1,NEQ
        IST = NOP (IEQ, NEQ) + 1
        T = T + H
C
C           PREDICT BOTH Y AND DY FOR NEXT MESH POINT
        DO 460 I=1,N
          D = DY(I,IEQ)
          D1 = Y(I,1,IEQ) + D*D
          D2 = R(1,INOBB+NOP1)*D
          IF (NO-2) 440, 400, 340
  340     IF (NO-3) 400, 380, 360
  360     D2 = D2 + R(1,INOBB+NOP1+1)*DY(I,3)
  380     D2 = D2 + R(1,INOBB+NOP1+2)*DY(I,2)
  400     DO 420 J=2,NO
            D = Y(I,J,IEQ)
            D3 = DFLOAT (J+1)
            D1 = D1 + D*(D3*D1/D3)
            D2 = D2 + D*R(1,INOBB+J+2)
  420     Y(I,1,IST) = D1
  440     DY(I,IST) = D2
        IF (NO.GT.2) IF (IEQ.GT.1) INOBB = INOBB + NO + IEQ - 2
C
C           CORRECTOR IS ITERATED THREE TIMES.  CORRECTION TERMS
```

```
C           ARE ACCUMULATED IN  SAVE(N10+I,1) FOR REDRING ENTIRE
C           CORRECTOR IF IT DOES NOT CONVERGE AND JACOBIAN HAS NOT
C           BEEN EVALUATED.
      D1 = C(INDCC)
  600 DO 620 I=1,N
  620   SAVE(N10+I,1) = 0.0D0
      D3 = BIG
      DO 720 J=1,N
        CALL DIFRUN (T, Y(1,1,IST), SAVE(NOP1,1))
        IF (IEVAL.LT.1) GO TO 680
        CALL PEDERV (T, Y(1,1,IST), PW, NO)
        D = D1*D
        DO 640 I=1,NSQ
          PW(I) = PW(I)*D
  640   DO 660 I=1,NO
          PW(I+(NO+1)*NO) = 1.0 + PW(I+(NO+1)*NO)
        IWEVAL = -IEQ
        CALL MATINV (PW, N, NO, J1)
        IF (J1.LT.0) GO TO 800
  680   DO 700 I=1,N
  700     SAVE(N9+I,1) = DY(I,IST) - W*SAVE(N9+I,1)
        D2 = 0.0D0
        DO 760 I=1,N
          D = 0.0D0
          DO 720 J=1,N
  720       D = D + PW(I+(J-1)*NO)*SAVE(N9+J,1)
          SAVE(N10+I,1) = SAVE(N10+I,1) + D
          Y(I,1,IST) = Y(I,1,IST) + D1*D
          DY(I,IST) = DY(I,IST) + D
  760     D2 = D2 + (D/YMAX(I))**2
        D2 = D2*DMIN1(1.0D0, 2.0D0*D2/D3)
        IF (D2.LE.HND) GO TO 940
        D3 = D2
  780   CONTINUE
C           CORRECTOR FAILED TO CONVERGE OR MATINV INDICATES ERROR
C     IF (((IWEVAL.EQ.0) .OR. (IWEVAL.IEQ.NE.0)) GO TO 900
  800 IF (W.LE.(1.0D-1*HMIN)) IF (NO-2) 820, 820, 840
      RATIO = RATIO*0.5D0
      IWEVAL = -1
      IRET = 2
      GO TO 3000
  820 NO = 1
      GO TO 840
  840 NO = 2
  860 IFAIL = 0
      IRET = 1
      GO TO 3000
  880     KFLAG = -6
  885     J1 = NOST+1
          DO 890 I=1,N
            DY(I,1) = SAVE(9,I)
            DO 890 J=1,J1
              Y(I,J,1) = SAVE(J,I)
          H = HOLD
          T = TOLD
          JFLAG = NOST
          RETURN
  900   DO 920 I=1,N
```

```
          D = SAVE(N10+I,1)
          Y(I,1,IST) = Y(I,1,IST) - D1*D
  920     DY(I,IST) = DY(I,IST) - D
        IWEVAL = -1
        GO TO 800
C           CORRECTOR CONVERGENCE HAS BEEN ACHIEVED.  THE ND
C           BACKWARD DIFFERENCES AT THE NEW MESH POINT ARE NOW
C           COMPUTED.
  940   DO 990 I=1,N
          DO 960 J=1,ND
  960       Y(I,J+1,IST) = Y(I,J,IST) + Y(I,J,IFO)
            IF (IFO.EQ.NEO) SAVE(N9+I,1) = Y(I,NOP1,1) - Y(I,NOP1,NEO)
  980     CONTINUE
        IF (ND.GT.2) IF ((IEQ.GT.1) .OR. (ND.EQ.6)) INDCC = INDCC + 1
 1000   CONTINUE
C           ERROR TEST IS NOW MADE ON LAST TERM COMPUTED
      D = 0.0D0
      DO 1020 I=1,N
 1020   D = D + (SAVE(N9+I,1)/YMAX(I))**2
      IF (D.LE.E) GO TO 1140
      IFAIL = IFAIL + 1
      IF (IFAIL.GE.3) GO TO 1040
      IF (W.LE.(HMIN*1.0D0D1)) IF (ND-2) 1160, 820, 840
      GO TO 1200
C           START OVER WITH ORDER 1 METHOD RECOMPUTING DERIVATIVES
 1040 IFAIL = 0
      DO 1060 I=1,N
 1060   Y(I,1,1) = SAVE(1,2)
      T = TOLD
      H = HOLD
      GO TO 30
C           ERROR CRITERION HAS BEEN SATISFIED.
C           STEP-SIZE AND ORDER FOR NEXT CALL ARE NOW COMPUTED.
 1140 IFAIL = 0
 1160 DO 1180 I=1,N
        D1 = YMAX(I)
        DO 1170 IFO=1,NEO
 1170     D1 = DMAX1 (DABS (Y(I,1,IFO)), D1)
 1180   YMAX(I) = D1
      IWEVAL = 0
      KFLAG = NFO
      IF (IFAIL.NE.0) KFLAG = -KFLAG
      HNEW = H
 1200 PRSAME = 1.2*(D/E)**EXPSAM
      PRUP = 1.0E+30
      PRDWN = 1.0E+30
      IF ((ND.GE.MAXORD) .OR. (IFAIL.NE.0)) GO TO 1240
      D = 0.0D0
      DO 1220 I=1,N
        D1 = Y(I,NOP1,NEO) - Y(I,NOP1,NFO+1)
 1220   D = D + ((SAVE(N9+I,1) - D1)/YMAX(I))**2
      PRUP = 1.2*(D/EUP)**EXPUP
 1240 IF (ND.EQ.1) GO TO 1280
      D = 0.0D0
      DO 1260 I=1,N
```

```
1260   D = D + (Y(I,NOP1,1)/YMAX(I))**2                                 DIJMT471
     PRODWN = 1.0*(D/EDOWN)**EXDOWN                                     DIJMT472
1280 IF (PRSAME.GT.PROUP) IF (PROUP.PRODWN) 1340, 1300, 1300            DIJMT473
     IF (PRSAME.LE.PRODWN) GO TO 1320                                   DIJMT474
1300 NEWO = NOP1                                                        DIJMT475
     D = 1.0/(AMAX1(PRODWN, 1.0E-6))                                    DIJMT476
     GO TO 1380                                                         DIJMT477
1320 NEWO = NO                                                          DIJMT478
     D = 1.0/(AMAX1(PRSAME, 1.0E-6))                                    DIJMT479
     GO TO 1380                                                         DIJMT480
1340 NEWO = NOP1                                                        DIJMT481
     D = 1.0/(AMAX1(PROUP, 1.0E-6))                                     DIJMT482
1360 IF (KFLAG.NE.0) IF (D-1.100) 1440, 1400, 1400                      DIJMT483
     NO = NEWO                                                          DIJMT484
     RATIO = RATIO*D                                                    DIJMT485
     IRET = 1                                                           DIJMT486
     GO TO 3000                                                         DIJMT487
1400 HNEW = HNO                                                         DIJMT488
     NO = NEWO                                                          DIJMT489
1440 RETURN                                                             DIJMT490
C                                                                       DIJMT491
C              THIS SECTION IS USED TO REDUCE THE STEP SIZE             DIJMT492
C                                                                       DIJMT493
3000 RATIO = DMAX1 (DABS(HMIN/HHLD), DMIN1(RATIO, 1.0D0))               DIJMT494
     T = THLD                                                           DIJMT495
     IF (RATIO.LT.1.0D0) IF (IDEL) 3070, 3070, 3080                     DIJMT496
     DO 3020 I=1,N                                                      DIJMT497
       DY(I,1) = SAVE(9,I)                                              DIJMT498
       DO 3020 J=1,NO                                                   DIJMT499
3020     Y(I,J,1) = SAVE(J,I)                                           DIJMT500
     GO TO (220, 300), IRET                                             DIJMT501
3030 IDEL = NOST+1                                                      DIJMT502
     IF ((NOST.LT.2) .OR. (NO.LT.2)) GO TO 3080                         DIJMT503
     D = DFLOAT (NOST)                                                  DIJMT504
     DO 3060 I=1,N                                                      DIJMT505
       D1 = SAVE(9,I)                                                   DIJMT506
       DO 3040 J=2,NOST                                                 DIJMT507
3040     D1 = D1 + SAVE(J,I)/DFLOAT(J-1)                                DIJMT508
3060   SAVE(IDEL,I) = D*D1                                              DIJMT509
3080 DO 3140 I=1,N                                                      DIJMT510
       DY(I,1) = RATIO*SAVE(9,I)                                        DIJMT511
       IF (NO.LT.2) GO TO 3140                                          DIJMT512
       D1 = 0.0D0                                                       DIJMT513
       DO 3120 J=2,NO                                                   DIJMT514
         D1 = D1 + RATIO                                                DIJMT515
         D2 = 1.0D0                                                     DIJMT516
         D = 0.0D0                                                      DIJMT517
         DO 3100 J1=2,IDEL                                              DIJMT518
           D2 = D2*(DFLOAT(J1-2) + D1)/DFLOAT(J1-1)                     DIJMT519
3100       D = D + D2*SAVE(J1,I)                                        DIJMT520
3120     Y(I,J,1) = D + SAVE(1,I)                                       DIJMT521
3140   Y(I,1,1) = SAVE(1,I)                                             DIJMT522
     H = HHLD*RATIO                                                     DIJMT523
C                                                                       DIJMT524
C              THIS SECTION FORMS THE DIFFERENCES AFTER RESCALING       DIJMT525
C                                                                       DIJMT526
4000 IF (NO.LT.2) GO TO 4040                                            DIJMT527
     NOM1 = NO-1                                                        DIJMT528
     DO 4020 I=1,N                                                      DIJMT529
       DO 4020 J=1,NOM1                                                 DIJMT530
         DO 4020 J1=J+1,NO                                              DIJMT531
           J2 = NO-J1+J+1                                               DIJMT532
4020       Y(I,J2,1) = Y(I,J2-1,1) - Y(I,J2,1)                          DIJMT533
4040 GO TO (220, 300, 1801, IRET                                        DIJMT534
C-----------------------------------------------------------------------
     END
```

REFERENCES

[AHL66] L. V. Ahlfors, Complex Analysis, Second Edition, McGraw-Hill Book Company, New York, 1966.

[BIC71] T. A. Bickart, D. A. Burgess, and H. M. Sloate, "High Order A-Stable Composite Multistep Methods for Numerical Integration of Stiff Differential Equations", Proc. Ninth Annual Allertown Conference on Circuit and System Theory, University of Illinois, 1971.

[BIC72] T. A. Bickart, and Z. Picel, High Order Stiffly Stable Composite Multistep Methods for Numerical Integration of Stiff Differential Equations, presented at SIAM 20th Anniversary Meeting, Philadelphia, June 1972; also to appear in B.I.T.

[BJU70] G. Bjurel, G. Dahlquist, B. Lindberg, S. Linde, and L. Odén, Survey of Stiff Ordinary Differential Equations, Department of Information Processing, Royal Institute of Technology, Stockholm, Report NA70.11, 1970.

[BRA72a] R. K. Brayton, F. G. Gustavson, and G. D. Hachtel, "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Differentiation Formulas", Proceedings of the IEEE, 60, 1972, pp 98-108.

[BRA72b] R. K. Brayton, C. C. Conley, Some Results on the Stability and Instability of the Backward Differentiation Methods with Non-Uniform Time Steps, IBM Research Report, RC-3964, IBM Watson Research Center, Yorktown Heights, New York, July 1972.

[BUT64] J. C. Butcher, "Implicit Runge-Kutta Processes", Mathematics of Computation, 18, 1964, pp 50-64.

[CHE71] L. Chesler and S. Pierce, The Application of Generalized, Cyclic and Modified Numerical Integration Algorithms to Problems of Satellite Orbit Computation, System Development Corporation Document TM-4717, March 1971.

[DAH63] G. G. Dahlquist, "A Special Stability Problem for Linear Multistep Methods", B.I.T., 3, 1963, pp 27-43.

[DEJ67] B. Dejon, "Numerical Stability of Difference Equations with Matrix Coefficients", SIAM J. Numer. Anal., 4, 1967, pp 119-128.

[DON71] J. Donelson III, and E. Hansen, "Cyclic Composite Multistep Predictor-Corrector Methods", SIAM J. Numer. Anal., 8, 1971, pp 137-157.

[DYE72] J. Dyer, S. Pierce, R. F. Haney, and L. Chesler, Generalized Multistep Methods in Orbit Computation: Studies in Existence Theory, Efficiency, Optimization, Systems Development Corporation Document TM-4888, February 1972.

[EHL68] B. L. Ehle, "High Order A-Stable Methods for the Numerical Solution of Systems of Differential Equations", B.I.T., 8, 1968, pp 276-278.

[ENR72] W. Enright, Studies in the Numerical Solution of Stiff Ordinary Differential Equations, Dissertation, Department of Computer Science, University of Toronto, Technical Report No. 46, October 1972.

[GAN60] F. R. Gantmacher, The Theory of Matrices, Volume 1, Chelsea Publishing Company, New York, 1960.

[GEA67] C. W. Gear, Numerical Integration of Stiff Ordinary Differential Equations, Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 221, January 1967.

[GEA68] _____, "The Automatic Integration of Stiff Ordinary Differential Equations", Proceedings IFIP Congress - 1968, Booklet A (Mathematics), August 1968, pp A81-A85.

[GEA71a] _____, "DIFSUB for Solution of Ordinary Differential Equations", Algorithm 407, CACM, 14, 1971, pp 185-190.

[GEA71b] _____, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[HEN62] P. Henrici, Discrete Variable Methods in Ordinary Differential Equations, John Wiley and Sons, Inc., New York, 1962.

[HEN64] _____, Elements of Numerical Analysis, John Wiley and Sons, Inc., New York, 1964.

[JAI70] M. K. Jain, and V. K. Srivastava, High Order Stiffly Stable Methods for Ordinary Differential Equations, Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 394, April 1970.

[KRO70] F. T. Krogh, On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations, Jet Propulsion Laboratory, Pasadena, California, Technical Memorandum 217, October 1970.

[KRO72] _____, Changing Step-Size in the Integration of Differential Equations Using Modified Divided Differences, Jet Propulsion Laboratory, Pasadena, California, Technical Memorandum 312, October 1972.

[NOR62] A. Nordsieck, "On Numerical Integration of Ordinary Differential Equations", Math. of Comp., 16, 1962, pp 22-49.

[NOR69] S. P. Nørsett, "A Condition for A(α)-Stability of Linear Multistep Methods", B.I.T., 9, 1969, p 259.

[ODE71] F. Odeh and W. Liniger, "A Note on Unconditional Fixed-h Stability of Linear Multistep Formulae", Computing, 7, 1971, pp 240-253.

[RUB72] W. B. Rubin and T. A. Bickart, A-Stability of Composite Multistep Methods, presented at the SIAM-SIGNUM 1972 Fall Meeting, Austin, Texas, October 1972.

[RUB73] W. B. Rubin, A-Stability of Composite Multistep Methods for the Numerical Solution of Stiff Ordinary Differential Equations, Dissertation, Department of Electrical and Computer Engineering, Syracuse University, February 1973.

[SLO71a] H. M. Sloate, Simultaneous Implicit Formulas for the Solution of Stiff Systems of Differential Equations, Department of Electrical and Computer Engineering, Syracuse University, Technical Report TR-71-4, May 1971.

[SLO71b] H. M. Sloate, and T. A. Bickart, "A-Stable Composite Multistep Methods", presented at The 1971 SIAM National Meeting; also to appear in J. ACM.

[WAT71] H. A. Watts, A-Stable Block Implicit One-Step Methods, Dissertation, University of New Mexico, 1971.

[WID67] O. B. Widlund, "A Note on Unconditionally Stable Linear Multistep Methods", B.I.T., 5, 1967, pp 65-70.

BIOGRAPHICAL DATA

Name:	Joel Marvin Tendler
Date and Place of Birth:	May 18, 1943; New York, New York
Elementary School:	Rabbi Jacob Joseph School, New York, New York, Graduated 1956
High School:	Rabbi Jacob Joseph School, New York, New York, Graduated 1960
College:	The Cooper Union for the Advancement of Science and Art, New York, New York, B.E., 1964
Graduate Work:	Syracuse University, Syracuse, New York, Graduate Research Assistant, 1964-1966, National Science Foundation Trainee, 1966-1969