

Ph.D. Thesis

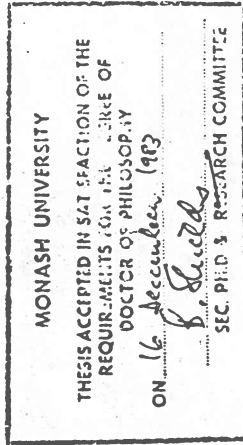
Name of candidate: PETER TISCHER

Title of Thesis: THE CYCLIC USE OF LINEAR MULTISTEP FORMULAS
FOR THE SOLUTION OF STIFF DIFFERENTIAL
EQUATIONS

1. I agree that this thesis may be made available for consultation within the library;
2. I agree that the thesis may be made available for photocopying; and
3. I note that, in any case, my consent is required only for the three years following acceptance of my thesis.

Signed.....
Date.....
Peter Tischer
12/1/84

ADDRESS: 27 GLENCAIRN AVENUE, RINGWOOD, 3134



THE CYCLIC USE OF LINEAR MULTISTEP FORMULAS
FOR THE SOLUTION
OF STIFF DIFFERENTIAL EQUATIONS

PETER E. TISCHER B.Sc. (hons)
of
THE DEPARTMENT OF COMPUTER SCIENCE
MONASH UNIVERSITY

This thesis contains no material which has been accepted for the award of any degree or diploma in any University. To the best of my knowledge and belief, this thesis contains no material previously published or written by any person, except when due reference is made in the text of the thesis.

Signed:

P. Fischer

PETER E. TISCHER,
Department of Computer Science,
Monash University,
August, 1983.

presented for examination for the degree of
Doctor of Philosophy
Monash University, August, 1983.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Doctor Gopal K. Gupta who suggested this topic to me and then supervised this research work and to Dr. Ron Sacks-Davis who supervised my work while Dr. Gupta was on secondment at the Asian Institute of Technology, Bangkok.

I am also grateful to the technical staff in the Department of Computer Science for making the necessary software modifications that allowed me to use the DIABLO 630 printer to prepare this thesis.

Without the support of my family my work would not have been possible. Words cannot convey the extent of my indebtedness to them.

ABSTRACT

This thesis considers the use of cyclic linear multistep formulas (CLMFs) for the solution of stiff ordinary differential equations. We discuss the absolute stability properties of CLMFs applied to the linear initial value problem, $y' = \lambda y$, and we also discuss the accuracy properties of CLMFs, especially the propagation of the local discretization error.

A class of two-stage CLMFs involving a free parameter is then considered. The formulas in this class are derived to have zero roots at infinity, zero extraneous roots at the origin and to have the same leading coefficients for both constituent linear multistep formulas. By appropriate choice of the free parameter a number of families of CLMFs are derived. For one family the free parameter is chosen to maximize the $A(\alpha)$ -stability angle while for another family the free parameter is chosen to make the resultant CLMF higher order convergent than its order of consistency in the sense of Albrecht(1978).

All the new formulas are tested on the STIFF DETEST problem set (Enright, 1979) and the codes using these new formulas are compared to LSODE (Hindmarsh, 1980) which uses the backward differentiation formulas.

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
CHAPTER 2	NOTATION AND BASIC PROPERTIES	4
2.1	ONE-STEP METHODS	4
2.2	LINEAR MULTISTEP FORMULAS(LMFs)	6
2.3	CYCLIC LINEAR MULTISTEP FORMULAS(CLMFs)	12
CHAPTER 3	A SURVEY OF FORMULAS AND CODES	15
3.1	METHODS FOR SOLVING NON-STIFF EQUATIONS	16
3.2	METHODS FOR SOLVING NON-STIFF EQUATIONS	25
CHAPTER 4	STABILITY AND ACCURACY PROPERTIES OF CYCLIC METHODS	53
4.1	STABILITY OF CYCLIC LINEAR MULTISTEP METHODS	53
4.2	MAXIMUM ATTAINABLE ORDER OF A-STABLE METHODS	56
4.3	CONVERGENT ORDER	59
4.4	ERROR COEFFICIENT	68
CHAPTER 5	DERIVATION OF A NEW CLASS OF CYCLIC MULTISTEP METHODS	71
5.1	TWO-STAGE CYCLIC LINEAR MULTISTEP FORMULAS	73
5.2	A CLASS OF SECOND DERIVATIVE FORMULAS	78
5.3	NEW FORMULAS	80
5.4	THE CLASS OF CLMFs CONSIDERED BY TENDLER, BICKART AND PICEL	86
CHAPTER 6	AN IMPLEMENTATION OF CYCLIC LINEAR MULTISTEP FORMULAS	89
6.1	IMPLEMENTATION DETAILS	89
6.2	A COMPARISON OF ODIUS AND LSODE	117

TABLE OF CONTENTS

CHAPTER 7	ADDITIONAL FAMILIES OF FORMULAS	132
7.1	NEW FAMILIES OF FORMULAS	132
7.2	MODIFICATIONS TO ODIUS	137
7.3	RESULTS OF NUMERICAL TESTING	139
CHAPTER 8	CONCLUSIONS	143
	REFERENCES	145
	APPENDIX A	
	APPENDIX B	

ERRATA

Errata-?

Notation: ln5 means line 5 from the top, counting all typed information including equations but excluding page numbers. ln-3 means 3 lines from the bottom.

p4 ln-10 Def. 2.1.1 is faulty in that ϕ depends on f the function which defines the differential equation (1.1). The following is a revised definition for the order of a one-step method.

Definition 2.1.1: Order

The method (2.1.1) is said to have order p if p is the largest integer for which

$$y(x+h) - y(x) - h\phi(x, y(x), h) = O(h^{p+1})$$

holds where $y(x)$ is the true solution of the differential equation (1.1) and y and f are both $p+1$ times continuously differentiable.

p5 l-14 Alexander assumes α_{11} is constant for diagonal implicitness (as on page 36 ln18). Mörsettt actually uses the term "semi-explicitness". The term semi-implicit already occurs in reference 16, (Butcher, 1964a).

p8 ln3 Theorem 2.2.1 is actually a negative result on the existence of LMFs. Here Chapter 2 should refer to the following theorem (Dahlquist, 1956, p49) which is a positive result on the existence of LMFs with arbitrarily high orders:

Theorem 3 - It is possible to construct an operator L of degree $p \geq k+1$ starting from any polynomial $\rho(r)$ which vanishes for $r = 1$. For a given $\rho(r)$ the constant ρ is unique.

p8 ln-5 "approximate" might in this context be a word than "approach".

p10 ln-2,-3 "show that for all α " should be "show that all orders and all α ".

p28 ln-12 $O(5/6N^3)$ should be $O(5N^3/6)$ or $O(5/6 N^3)$. The same error occurs on the next line.

p61 ln-11 This sentence, as well as page 66, ln-11, should refer to "for all initial value problems defined by sufficiently smooth functions" Instead of "all initial value problems with suitably smooth true solutions".

p24,23 The variable d_j is used on page 63 in a different way to the way it is used on page 62. This is potentially confusing.

p92 ln1,2 These lines should read

$$\left\| \left(\text{safety factor} \cdot r \right)^{p+1} K_{p+1} h^p \right\|_n \leq \text{tolerance}.$$

The safety factor is given by $(0.9)(0.88)^{**} (1/(p+1))$ and ranges in value from 0.8443 for the first order formula to 0.8873 for the eighth order formula.

p101 -1 The formula is correct for $q = 2, 3, 4$ but is wrong for $q = 5$ and is wrong in general.

p108 ln11 2nd eq., $\lambda^{-\lambda^*}$ should be $\left| \frac{\lambda - \lambda^*}{\lambda} \right|$.

CHAPTER 1

INTRODUCTION

A user interested in solving a (set of) ordinary differential equations (ODEs) is usually interested in computing the solution, to a specified accuracy, of

$$\frac{dy}{dx} = f(x, y) \quad y(x_0) = y_0 \quad (y \text{ a vector}) \quad (1.1)$$

from an initial x value (x_0) to some final value (x_f). The user may be interested in the solution only at the final point x_f or he may be interested in the solution at several intermediate points. In some cases a user may be interested in finding the x value at which a component of y achieves a particular value.

When solving (1.1) numerically the problem is generally classified to be either stiff or non-stiff. Stiff problems are generally characterized by the Jacobian $\partial f(x, y)/\partial y$ of the ODEs having widely separated eigenvalues and having some eigenvalues with negative real part and large modulus. A problem which is not stiff is said to be non-stiff.

The problems involved in solving stiff equations were probably first identified by Curtiss and Hirschfelder(1952). The importance of stiff equations is discussed by Shampine and Gear(1979) and Björkel, Dahlquist, Lindberg at al.(1970) who present a comprehensive survey of application areas in which stiff equations arise. A more recent survey of application areas is presented by Aiken(1982).

Non-stiff problems are easier to solve than stiff problems and codes based on classical methods such as Adams formulas and Runge-Kutta formulas may be used. Although the classical methods are more efficient than stiff methods in those regions where accuracy is the dominant constraint on the stepsize, if the classical methods are used for solving stiff equations they become very inefficient and for some problems may take many orders of magnitude more time than special methods designed for stiff equations.

Most software available for solving differential equations may be divided into two classes - that for solving stiff equations and that for solving non-stiff equations. Some codes are capable of solving both stiff and non-stiff equations efficiently but the user must still choose the option appropriate to his problem. Progress has recently been made in developing techniques which detect stiffness and this should enable the next generation of codes to automatically choose the method appropriate to the problem. Such codes are called type-insensitive.

Testing of stiff ODE solvers by Enright, Hull and Lindberg(1975) and Enright and Hull(1976b) show that codes using the Backward Differentiation Formulas (BDFs) are the most efficient codes available. Such codes do not, however, perform efficiently for stiff problems where the eigenvalues of the Jacobian lie in the left half of the complex plane and close to the imaginary axis. Consequently, the last decade has seen intensive research on the problem of finding formulas with better stability properties than the BDFs. Many classes of formulas have been found which have better stability properties than the BDFs but codes which use these formulas are less efficient than BDF-codes. This is generally because they require more work per step than BDF-codes.

This thesis is concerned with the cyclic use of linear multistep formulas (LMFs) for the solution of stiff ODEs. A set of LMFs which is used in a prescribed order is termed a cyclic linear multistep formula (CLMF). Since the BDFs are also LMFs, CLMFs involve the same implementation cost per step as the use of the BDFs. CLMFs, however, can have better stability properties than the BDFs. It is largely for these reasons that the cyclic use of LMFs for solving stiff systems of equations is a promising line of research.

In Chapter 2 we present some background material on the stability and existence of LMFs. In Chapter 3 we discuss some of the methods for solving stiff and non-stiff equations. Although this thesis is primarily concerned with the cyclic use of LMFs for the solution of stiff equations the approach we have used may also be applied to derive CLMFs for the solutions of non-stiff equations.

In Chapter 4 we discuss the stability properties of cyclic formulas with respect to the linear initial value problem $y' = \lambda y$ where λ is a complex constant (or matrix). By applying the Daniel-Moore theorem (Wanner, Hairer and Nørsett, 1978) we are able to bound the maximum order of an A-stable cyclic method. In Chapter 4 we also consider the accuracy properties of a wide range of methods by using an approach due to Albrecht (1978). By extending this approach we discuss the propagated error behaviour of CLMFs with the aim of comparing the accuracy properties of CLMFs with the same consistent order.

In Chapter 5 we focus our attention on two-stage CLMFs. In particular, we study a class of k -step/ $(k+1)$ -step, k th order CLMFs which have constant leading coefficients, zero roots at infinity and zero extraneous roots at the origin. This class of formulas has a degree of freedom which we use to maximize the stability properties of the CLMFs. The fourth order formula of this family is A-stable and therefore attains the maximum order possible for a two-stage, A-stable CLMF.

In Chapter 6 we consider implementation issues and present details of a code, ODI0US, which implements the first family of CLMFs derived in Chapter 5. The performance of ODI0US is then compared to the performance of a well-known BDF-code, LSODE, due to Hindmarsh (1980), by using the test package STIFF DETEST, described by Enright (1979).

In Chapter 7 we derive some additional families of two-stage CLMFs by suitable use of the free parameter in the class of two-stage CLMFs derived in Chapter 5. We then go on to consider the performance of all the new formulas in a revised version of the code ODI0US. In the final chapter we present our conclusions and suggest areas for further work.

In this chapter we present details of the notation we use and some basic definitions for Runge-Kutta Formulas (RKFs), Linear Multistep Formulas (LMFs) and Cyclic Linear Multistep Formulas (CLMFs). We also present some known results about the stability and existence of LMFs.

2.1 ONE-STEP METHODS

Henrici (1962) defines a one-step method as

$$y_{n+1} - y_n = h\phi(x_n, y_n, h) \quad (2.1.1)$$

where y_{n+1} is the numerical solution at x_{n+1} and x_n and y_n are known. The stepsize is $x_{n+1} - x_n = h$. ϕ is generally called the 'increment function' and is a weighted average of values of the function f of (1.1) taken at different points.

Definition 2.1.1: Order

The method (2.1.1) is said to have order p if p is the largest integer for which

$$y(x+h) - y(x) - h\phi(x, y(x), h) = O(h^{p+1})$$

holds where $y(x)$ is a function which has continuous derivatives $y^{(1)}(x)$, $y^{(2)}(x), \dots, y^{(p+1)}(x)$ defined on the interval $[x, x+h]$.

The most widely used one-step methods are the Runge-Kutta formulas (RKFs). A RKF computes y_{n+1} , the numerical approximation at $x_n + h$, using the formula

$$y_{n+1} = y_n + h \sum_{i=1}^M b_i f(x_n + c_i h, y_i)$$

where the internal approximations $Y_1, Y_2, \dots, Y_M$ are given by

$$Y_i = Y_n + h \sum_{j=1}^M a_{ij} f(x_n + c_j h, Y_j), \quad i = 1, 2, \dots, M.$$

Such a formula is also called an M-stage RKF. For each internal approximation Y_i at least one function evaluation is required. An M-stage RKF may be represented in the following tabular form, known as a Butcher tableau.

c_1	a_{11}	a_{12}	$\dots$	a_{1M}
c_2	a_{21}	a_{22}	$\dots$	a_{2M}
$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$
c_M	a_{M1}	a_{M2}	$\dots$	a_{MM}
	b_1	b_2	$\dots$	b_M

The RKF is explicit if $a_{ij} = 0$ for $j \geq i$. The RKF is termed semi-implicit (Wörseth, 1974) or diagonally implicit (Alexander, 1977) if $a_{ij} = 0$ for $j > i$ and some diagonal element a_{ii} is non-zero. For diagonally implicit RKFs we can calculate the internal approximations in the order $Y_1, Y_2, \dots, Y_M$. In calculating Y_i we have to solve the implicit equation

$$Y_i = Y_n + h \sum_{j=1}^{i-1} a_{ij} f(x_n + c_j h, Y_j) + h \sum_{j=i}^M a_{ij} f(x_n + c_j h, Y_j)$$

where the quantities on the right hand side have already been calculated. If in addition to being diagonally implicit, the RKF has the property that all the non-zero a_{ij} are equal, then the method is termed singly diagonally implicit.

Since this thesis is primarily concerned with the cyclic use of LMFs we will not be discussing the properties of RKFs in any great detail. For a discussion of the basic properties of RKFs the reader is referred to Lambert(1973).

2.2 LINEAR MULTISTEP FORMULAS (LMFs)

Following the notation of Henrici(1962) we represent a k-step LMF for solving (1.1) as

$$\alpha_k Y_{n+1} + \alpha_{k-1} Y_n + \dots + \alpha_0 Y_{n-k+1} = h(\beta_k f_{n+1} + \beta_{k-1} f_n + \dots + \beta_0 f_{n-k+1}) \quad (2.2.1)$$

where k is a fixed integer, h is the stepsize assumed constant, $x_m = a + mh$, $f_m = f(x_m, y_m)$ and α_i and β_i are real constants which do not depend on n . Since (2.2.1) is a k-step formula $|\alpha_k| + |\beta_k| = 0$, $|\alpha_0| + |\beta_0| = 0$. Where $\beta_k = 0$ the LMF is termed explicit whereas for $\beta_k \neq 0$ it is termed implicit.

In keeping with Henrici's approach, we define the following polynomials associated with the difference equation (2.2.1).

$$\rho(r) = \sum_{j=0}^k \alpha_j r^j \quad \text{and} \quad \sigma(r) = \sum_{j=0}^k \beta_j r^j. \quad (2.2.2)$$

We also define the operator

$$L = \rho(E) - h\sigma(E)$$

where $D = d/dx$ and E is the displacement operator

$$Ey(x) = y(x+h).$$

It is assumed that $\rho(r)$ and $\sigma(r)$ have no common divisors.

Definition 2.2.1: Order (Consistent Order, Local Truncation Order)

The order of a method (2.2.1) is the largest integer p such that for all polynomials $y(x)$ of degree p

$$L y(x) = 0.$$

Definition 2.2.2: Convergence

A method defined by (2.2.1) is convergent if for all initial value problems (1.1) where $f(x, y)$ satisfies the Lipschitz condition, and for all y_0 the relation

$$\lim_{h \rightarrow 0} y_n = y(x)$$

holds for all $x \in [a, b]$.

Definition 2.2.3: Consistency

The method is said to be consistent if the order p is at least equal to 1. For LMFs this condition can be shown to be equivalent to the requirement that

$$\rho(1) = 0 \quad \text{and} \quad \rho'(1) = o(1).$$

By the definition of order, a LMF of order p is exact for polynomials of degree p or less. If $y'(x)$ has continuous derivatives $y^{(1)}(x)$, $y^{(2)}(x), \dots, y^{(p+2)}(x)$ on the interval $[x_{n-k+1}, x_{n+1}]$ we can show (Dahlquist, 1956)

$$L y(x) = c_{p+1} h^{p+1} y^{(p+1)}(x_{n+1}) + O(h^{p+2})$$

where c_{p+1} is a non-zero constant which is independent of $y(x)$.

If $y_{n-1} = y(x_{n-1})$, $f_{n-1} = y^{(1)}(x_{n-1})$, $f_{n+1} = f(x_{n+1})$ and y_{n+1} satisfies (2.2.1), then we have

$$y_{n+1} = y(x_{n+1}) + c_{p+1} h^{p+1} y^{(p+1)}(x_{n+1}) + O(h^{p+2}).$$

c_{p+1} is termed the local truncation error constant.

A sufficient condition for one-step methods to converge is that the order be at least one (Henrici, 1962, pp 123 - 124). For LMFs we require an extra condition.

Definition 2.2.4: Stability (D-stability, Zero-stability, Stability at the Origin)

The LMF is stable if the modulus of no root of $\rho(r)$ exceeds 1 and if the roots of modulus 1 are simple. The LMF is strongly stable (or strongly D-stable) if $\rho(r)$ has only one root which equals 1 and all other roots of $\rho(r)$ are less than 1 in magnitude.

The conditions of consistency and stability are together necessary and sufficient conditions for convergence (Henrici, 1962).

The following theorem of Dahlquist(1956) assures us of the existence of stable LMFs.

Theorem 2.2.1:

Although for any positive integer k there exists a consistent LMF of order $p = 2k$, the order of a stable k -step LMF cannot exceed $k+2$. If k is odd the order cannot even exceed $k+1$. An order of $k+2$ can only occur if all roots of $\rho(r)$ are on the unit circle.

To further investigate the behaviour of the numerical solution, we apply the multistep method (2.2.1) to solve the differential equation

$$y' = \lambda y, \quad y(0) = 1 \quad (2.2.3)$$

where λ is a complex constant number (or matrix). The difference equation (2.2.1) then reads

$$\sum_{i=0}^k (\alpha_i - h\lambda\beta_i) y_{n-k+i+1} = 0 \quad (2.2.4)$$

and the corresponding characteristic polynomial is

$$\rho(r) - h\lambda\sigma(r) = 0. \quad (2.2.5)$$

If the roots of (2.2.5) are $r_1, r_2, \dots, r_k$ then, on the assumption that the roots are distinct, we may rewrite the numerical solution of (2.2.3) as

$$y_n = \sum_{i=1}^k d_i r_i^n$$

where d_i are suitable constants.

Since the rate of growth of the true solution to the differential equation is $e^{\lambda h}$, we would expect one root of (2.2.5) to approach $e^{\lambda h}$. Let this root be r_1 and call it the principal root. Other roots are called the extraneous roots. The roots of $\rho(r)$ are known as the roots at the origin while the roots of $\sigma(r)$ are known as the roots at infinity. We now define

Definition 2.2.5: Relative Stability

A LMF is relatively stable for those values of $h\lambda$ where the extraneous roots are less than or equal to the principal root in absolute value.

Definition 2.2.6: Absolute Stability

A LMF is absolutely stable for those values of $h\lambda$ where all the roots of (2.2.5) are less than or equal to 1 in absolute value.

For values of $h\lambda$ where the LMF is relatively stable, the principal root will dominate the extraneous roots and the long term numerical solution will largely reflect the influence of the principal root, i.e.

$$\text{as } n \rightarrow \infty, y_n \rightarrow r_1^n.$$

For solving most ordinary differential equations we only require relative stability for small values of $h\lambda$. Note that the condition of strong D-stability guarantees that the principal root dominates the extraneous roots for $h = 0$, and thus for sufficiently small stepsizes.

For stiff equations, relative stability for small values of $h\lambda$ is not enough. Consider what happens for $\text{Re}(h\lambda) < 0$. In this case the true solution is decreasing in magnitude and for $\text{Re}(h\lambda) < 0$, $|\lambda| \gg 1$ the true solution becomes very small in magnitude very quickly. Stiff equations always have some components which behave in this way. To track the stiff components $|h\lambda|$ must be made small to ensure accuracy in the relative sense. After a while, these stiff components may have decreased in magnitude so much that they are no longer of any significance in the numerical solution and we could increase the stepsize, providing that the stiff components remain insignificant at the new stepsize. We are now interested in tracking the stiff components accurately in an absolute, rather than a relative, sense. Therefore, for a method to be suitable for stiff equations we generally require that the absolute stability region method include a large region of the left half plane $\text{Re}(h\lambda) < 0$ by satisfying one of the following requirements.

Definition 2.2.7: A-stability (Dahlquist, 1963)

A method is said to be A-stable if all numerical approximations tend

to zero as $n \rightarrow \infty$ when it is applied to the differential equation $y' = \lambda y$ with a fixed positive h and a (complex) constant λ with negative real part.

The following theorems due to Dahlquist(1963) show that the condition of A-stability is only satisfied by a subset of LMFs which are low order and implicit.

Theorem 2.2.2:

An explicit, k -step LMF cannot be A-stable.

Theorem 2.2.3:

The order p of an A-stable LMF cannot exceed 2. The smallest truncation error for such formulas is obtained from the trapezoidal rule for which $p = 2$, $k = 1$.

Since the maximum order of an A-stable method is restricted to 2 for LMFs, the following weaker absolute stability property has been proposed.

Definition 2.2.8: A(α)-stability (Widlund, 1967)

A method is said to be A(α)-stable, $\alpha \in [0, \pi/2]$ if all numerical approximations to $y' = \lambda y$ converge to zero as $n \rightarrow \infty$ with h fixed for all $|\arg(-\lambda)| < \alpha$, $|\lambda| = 0$.

Concerning A(α)-stability we have the following results.

Theorem 2.2.4:

No explicit LMF can be A(0)-stable.

Theorem 2.2.5:

There is only one A(0)-stable LMF with order $p \geq k+1$, the trapezoidal rule.

Theorem 2.2.6:

For all $\alpha \in [0, \pi/2]$ there exist A(α)-stable methods for which $k = p = 3$ and $k = p = 4$.

Theorems (2.2.4), (2.2.5) and (2.2.6) are due to Widlund(1967).

Kong(1977) and Grigorieff and Schroll(1978) show that for all $\alpha \in [0, \pi/2]$, A(α)-stable LMFs exist. They also show that high order LMFs with large α are characterized by very large error coefficients and

very small accuracy regions. As an example, Grigorieff and Schroll(1976) present a tenth order formula which is $A(89.9999^\circ)$ -stable but which has an error constant of 1.2×10^{24} .

For the true solution of (2.2.3) we have

$$y^{(p)}(x_{n+1}) = \lambda^p y(x_{n+1}).$$

If $|\lambda| \gg 1$, the p^{th} derivative of a stiff component can be very much larger in magnitude than the stiff component itself. Since most codes attempt to keep

$$\left| \left| c_{p+1} h^{p+1} y^{(p+1)}(x_{n+1}) \right| \right|$$

below some user specified error tolerance on each step, we would like stiff components with $|h\lambda| \gg 1$ to be very quickly damped out. This leads to the following stability requirement.

Definition 2.2.9: L-stability (Ehle, 1973) or Stiff A-stability (Axelson, 1969)

A method is said to be L-stable if in addition to being A-stable, the numerical solution y_n to $y' = \lambda y$ approaches zero as $|\lambda|$ approaches ∞ .

For the initial value problem (2.2.3), we can see that as $|\lambda| \rightarrow \infty$, the roots of the characteristic polynomial tend to the roots of $\phi(r)$. Thus for maximal damping as $|\lambda| \rightarrow \infty$ we require that the roots of $\phi(r)$ be zero. This motivates the use of the Backward Differentiation Formulae (BDFs) for stiff equations. The k^{th} order BDF is determined by requiring that $\phi(r) = \beta_k r^k$ and that the resulting formula be k^{th} order. The first and second order BDFs are L-stable whereas the BDFs for $k \geq 7$ are no longer A(0)-stable. In Chapter 3 we will consider the BDFs in greater detail.

2.3 CYCLIC LINEAR MULTISTEP FORMULAS (CLMFs)

In this section we present details of an alternative notation to Henrici's(1962) which we will use in this thesis since it facilitates the discussion of the stability and accuracy properties of a wide range of formulas. This notation was first introduced by Butcher(1966) and has since been used by Pierce(1974), Skeel(1976) and was applied to CLMFs by Albrecht(1978).

Let $y_j, j = 0, 1, \dots, m$ denote the approximations of $y(x_j)$, $x_j = a + jh$, $x_j \in [a, b]$ defined by (2.2.1) with $f_j = f(x_j, y_j)$ and the starting values

$$y_j = \eta_j(h), j = 0, 1, \dots, k-1,$$

$$z_j = [y_j, y_{j+1}, \dots, y_{j+k-1}]^T,$$

$$z(x_j) = [y(x_j), y(x_{j+1}), \dots, y(x_{j+k-1})]^T,$$

$$\zeta_0 = [\eta_0(h), \eta_1(h), \dots, \eta_{k-1}(h)]^T,$$

$$\phi(x_j, z_j, z_{j+1}; h) = [0, 0, \dots, \beta_k f_{j+k} + \dots + \beta_0 f_j]^T,$$

$$A = \begin{bmatrix} 0 & 1 & 0 & : & 0 \\ 0 & 0 & 1 & : & 0 \\ 0 & 0 & 0 & : & 0 \\ : & : & : & : & : \\ 0 & 0 & 0 & : & 1 \\ -\alpha_0 & -\alpha_1 & -\alpha_2 & : & -\alpha_{k-1} \end{bmatrix},$$

where $y(x_j)$ is the exact solution of the differential equation at x_j and α_k has been set to 1 as a scaling factor. We can therefore represent the LMF (2.2.1) as

$$z_{j+1} = Az_j + h\phi(x_j, z_j, z_{j+1}; h), z_0 = \zeta_0(h). \quad (2.3.1)$$

The matrix A is in Frobenius form. It is the companion matrix to the $p(r)$ polynomial defined by (2.2.2), where the polynomial has been normalized by choosing $a_k = 1$.

A very wide class of methods can be written in the form

$$z_{j+1} = \tilde{A}z_j + h\tilde{\phi}(x_j, z_j, z_{j+1}; h) \quad (2.3.2)$$

with a certain matrix $\tilde{A}$ and vector function $\tilde{\phi}$.

Definition 2.3.1:

An ordered set of M , k -step formulas is called an M -cyclic, k -step method, if these formulas ("stages") are used cyclically in the given order for the calculation of subsequent approximations y_j , $j = k, k+1, \dots, M$.

An M -stage, k -step CLMF may be conveniently represented in the form (2.3.2). For example, consider a two-stage, k -step CLMF with stages

$$z_{2j+1} = A^{(1)} z_{2j} + h\phi^{(1)}(x_{2j}, z_{2j}, z_{2j+1}; h),$$

$$z_{2j+2} = A^{(2)} z_{2j+1} + h\phi^{(2)}(x_{2j+1}, z_{2j+1}, z_{2j+2}; h) \quad j = 0, 1, \dots$$

By substitution it can be reduced to (2.3.2) with $\tilde{A} = A^{(2)} A^{(1)}$

$$w_j = z_{2j},$$

$$w_{j+1} = A^{(2)} A^{(1)} z_{2j} + h\tilde{\phi}(x_j, w_j, w_{j+1}; h),$$

$$w_{j+1} = \tilde{A}w_j + h\tilde{\phi}(x_j, w_j, w_{j+1}; h).$$

If we have an M -stage, k -step CLMF where the stages are represented by

$$z_{Mj+1} = A^{(1)} z_{Mj+1-1} + h\phi^{(1)}(x_{Mj+1-1}, z_{Mj+1-1}, z_{Mj+1}; h), \quad i = 1, 2, \dots, M$$

then we can represent the CLMF by

$$w_{j+1} = \tilde{A}w_j + h\tilde{\phi}(x_j, w_j, w_{j+1}; h) \quad (2.3.3)$$

$$\text{where } w_j = z_{Mj}, \tilde{A} = \prod_{i=1}^M A^{(i)}.$$

CHAPTER 3

A SURVEY OF EXISTING FORMULAS AND CODES

Most codes available today for solving non-stiff ordinary differential equations (ODEs) use formulas which were derived around the turn of the century. Codes in use today are very much better than codes written only about fifteen years ago. Although great improvements in efficiency have been made, it is probably fair to say that the most significant achievements have been in reliability, convenience and diagnostic capabilities. These improvements in codes have usually been accomplished by improving implementations of well known, "classical" formulas.

In the case of solving stiff equations, a great many methods have been proposed but few have been put into practice. There is basically just one family of formulas - the Backward Differentiation Formulas (BDFs) - which have been implemented in widely available, quality software. New algorithms have to compete with very sophisticated, well tuned codes based on the BDFs. Consequently, there is a gap between existing algorithms and available software as a fair comparison between new and existing algorithms would require investing the same effort in the design and coding of the new algorithms as has been put into the sophisticated BDF-codes.

In this survey we will discuss only those formulas which have been demonstrated to be useful in solving ODEs or have potential to be useful. This survey does not attempt to cover all the formulas suggested in the literature. We also discuss some of the codes being used at present and some of the techniques used in implementation. Since the main concern of this thesis is the use of CLMFs for solving stiff equations, we discuss the formulas and codes for solving non-stiff equations only briefly.

3.1. METHODS FOR SOLVING NON-STIFF EQUATIONS

Most widely used codes for solving non-stiff ODE's are based on the Runge-Kutta, Adams or extrapolation methods. We discuss the three classes of methods and the codes based on them briefly.

3.1.1 Runge-Kutta Methods

Explicit RKFs have been extensively used for solving non-stiff equations because they are easy to program. Since RKFs are one-step methods, there is no need for either a special starting procedure or to generate past values when the stepsize is changed and since the formulas are explicit, there are none of the complications involved in solving implicit equations at each step. The main drawback with explicit RKFs is that they require at least k function evaluations per step for a formula of order k . This is in contrast to Adams methods which usually need no more than two function evaluations per step irrespective of the order. Thus, RKFs are not expected to be competitive with Adams methods when function evaluations are relatively expensive.

Most recent codes use a pair of RKFs for error estimation as suggested by Fehlberg(1969). Two numerical solutions are computed by using two formulas of different orders and the error in the solution computed by the lower order formula is estimated by the difference in the two numerical solutions. This approach to error estimation is known as using a RK-Fehlberg pair. The higher order formula is constructed so that all the internal approximations which are required for the first formulas are also used by the second formula. The higher order formula generally requires at least one additional internal approximation and therefore at least one more function evaluation. Several sets of RK-Fehlberg pairs have been derived by Verner(1978).

Since we assume that the higher order solution computed for error estimation is more accurate than the lower order solution, we may then argue that the higher order solution is the one that should be used as the approximation in the current step. The process of correcting the

numerical solution by adding in an estimate of the error in that solution is known as local extrapolation. Accepting the higher order solution is effectively local extrapolation and this device is frequently employed when using RK-Fehlberg pairs, as a more accurate solution is obtained and as the region of absolute stability increases slightly for explicit RKFs as the order increases.

For a code to be efficient over a wide range of accuracy requirements, it is important that it be able to vary the order of the formulas being used. Such variable order codes must estimate the error that would have resulted in using formulas of different orders. Most codes which implement RKFs are fixed order codes. It is possible to design variable order RK-Fehlberg codes since embedded RKFs have been derived by Bettis(1978) and Verner(1979). Bettis derives a set of embedded formulas from order 1 to 6 while Verner derives five sets of embedded formulas which include two sets of formulas up to order 5, one set up to order 6, another up to order 7 and the last one has embedded formulas up to order 8.

The difficulty with using such embedded formulas is illustrated by the last set in Verner(1979). It is a set of formulas up to order 8 and the pair of orders 7 and 8 requires 13 function evaluations as do other similar pairs. However the embedded 6,7 pair requires 12 function evaluations, the 5,6 pair requires 12 evaluations and the 4,5 pair requires 9. RK-Fehlberg pairs for these orders requiring only 10, 8 and 6 evaluations respectively have been derived by Verner(1978). A variable order RK-Fehlberg code using an order 7,8 pair which has an order 3,4 pair formula embedded in it and an additional order 3,4 pair formula using 4 function evaluations has been designed by Shampine, Gordon and Wisniewski(1980).

Enright and Hull(1976a) tested a number of fixed order codes based on RK-Fehlberg pairs and concluded that these methods were best when derivative evaluations were relatively inexpensive. Shampine, Watts and Davenport(1976) considered only two RKF codes, both of which used the same RK-Fehlberg pair. The code RKF4, by Hull and Enright(1974), accepted the fourth order solution while the code RKF45, by Shampine and

Watts(1977, 1979), in effect performed local extrapolation by accepting the fifth order solution. Numerical testing showed that the code which performed local extrapolation was more efficient.

Gupta(1980) has shown that high order RK-Fehlberg pairs are quite efficient in many application environments. He concludes that if the cost of a function evaluation is less than the equivalent of 5 square-root or exponential evaluations per equation then a suitable RK-Fehlberg code is likely to be more efficient than an Adams code. It is not until the cost of function evaluations is more than the equivalent of about 50 square-root or exponential evaluations per equation that Adams codes are clearly more efficient. The cost of many problems lies in between these extremes, and for these problems the relative efficiency of a RK-Fehlberg code to an Adams code is difficult to predict. As well as the cost of each equation, issues such as the number of equations being solved, the accuracy requirements, the compiler and the machine being used will affect the performance. We note that other considerations like the frequency of output required must also be considered before one can decide which code will be more efficient.

Currently two of the most widely used RK-Fehlberg codes are RKF45 of Shampine and Watts(1977, 1979) and DVERK of Hull, Enright and Jackson(1976). DVERK is based on a pair of formulas of order 5 and 6 derived by Verner(1978) and is available in the INSL library. RKF45 is available in the book of Forsythe, Malcolm and Moler(1977).

3.1.2 Linear Multistep Methods

Adams formulas are the most commonly used LMFs for solving non-stiff equations and there are a number of codes currently available which are based on these formulas. Some of these codes are listed in Table 3.1. In this section we will briefly discuss the basis of the Adams formulas before describing some of the strategies used by these codes to implement them.

Code	Reference
DVDQ	Krogh(1969)
DIFSUB	Gear(1971b,c)
GEAR	Hindmarsh(1974)
VOAS	Sedgwick(1973)
EPISODE	Byrne and Hindmarsh(1975)
STEP	Shampine and Gordon(1975)
LSODE	Hindmarsh(1980)

Table 3.1: Codes based on Adams formulas

The Adams formulas are derived as follows. We present the derivation here because it facilitates the discussion of several techniques used in ODE codes. Suppose we have past approximations y_{n+1-j} to the true solution $y(x_{n+1-j})$, $j = 1, 2, \dots$ and that we have approximations $f_{n+1-j} = f(x_{n+1-j}, y_{n+1-j})$ to the derivatives at these points. It is required to advance a step from x_n to x_{n+1} with stepsize $h_{n+1} = x_{n+1} - x_n$. Using the identity

$$y(x) = y(x_n) + \int_{x_n}^x f(t, y(t)) dt$$

the k th order Adams-Bashforth formulas may be regarded as approximating $f(t, y(t))$ over the interval $[x_n, x_{n+1}]$ by a polynomial $Q_{k,n}(t)$ of degree $k-1$ which interpolates the previously computed values f_{n+1-j} , $j = 1, 2, \dots, k$. Thus the k th order Adams-Bashforth formula is

$$y_{n+1}^p = y_n + \int_{x_n}^{x_{n+1}} Q_{k,n}(t) dt. \quad (3.1.2.1)$$

Similarly, if we define $P_{k,n}(t)$ as the polynomial of degree $k-1$ which interpolates f_{n+1-j} , $j = 0, 1, \dots, k-1$ we derive the k th order Adams-Moulton formula

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} P_{k,n}(t) dt. \quad (3.1.2.2)$$

The Adams-Moulton formulas are implicit and consequently a set of non-linear equations must be solved for y_{n+1} . Rather than writing y_{n+1} in terms of y_n as in (3.1.2.2) we may express y_{n+1} in terms of y_{n+1}^p as follows. Subtracting (3.1.2.1) from (3.1.2.2) gives

$$y_{n+1} = y_{n+1}^p + \int_{x_n}^{x_{n+1}} (P_{k,n}(t) - Q_{k,n}(t)) dt.$$

The polynomials $P_{k,n}(t)$ and $Q_{k,n}(t)$ are of degree $k-1$ and are equal at the $k-1$ points x_{n+1-j} , $j = 1, 2, \dots, k-1$. Let $f_{n+1}^p = Q_{k,n}(x_{n+1})$. Then we may write

$$y_{n+1} = y_{n+1}^p + g_{k,n+1} h_{n+1} (f_{n+1} - f_{n+1}^p) \quad (3.1.2.3)$$

$$\text{where } g_{k,n+1} = \frac{1}{h_{n+1}} \int_{x_n}^{x_{n+1}} \frac{(t - x_n) \dots (t - x_{n-k+1})}{(x_{n+1} - x_n) \dots (x_{n+1} - x_{n-k+1})} dt.$$

The non-linear equation (3.1.2.3) is solved at each step of an Adams code. When solving non-stiff equations functional (or 'simple') iteration is used to solve equation (3.1.2.3). Thus we form the approximations

$$y_{n+1}^{(m+1)} = y_{n+1}^p + g_{k,n+1} h_{n+1} (f(x_{n+1}, y_{n+1}^{(m)}) - f_{n+1}^p), \quad m = 0, 1, \dots$$

where $y_{n+1}^{(0)} = y_{n+1}^p$.

The codes DIFSUB, GEAR and LSODE iterate to convergence (that is, till the difference between two successive approximations is sufficiently small) and allow a maximum of three iterations before a step is rejected. We may therefore refer to these codes as Adams-Moulton codes. However, the codes VOAS and STEP iterate only once and are referred to as PECE codes.

After computing y_{n+1} the code will form an estimate of the local error. This estimate may be obtained by comparing the results of

formulas of different orders. Let y_{n+1}^* be the solution obtained from the Adams-Moulton formula of order $k+1$, thus

$$y_{n+1}^* = y_n + \int_{x_n}^{x_{n+1}} P_{k+1,n}(t) dt. \quad (3.1.2.4)$$

We may use

$$e_{n+1} = y_{n+1}^* - y_{n+1}$$

as an estimate of the local error. e_{n+1} may be computed as a difference between predicted and computed values as follows. From (3.1.2.4) and (3.1.2.1)

$$y_{n+1}^* = y_{n+1}^p + \int_{x_n}^{x_{n+1}} (P_{k+1,n}(t) - Q_{k,n}(t)) dt.$$

Thus

$$y_{n+1}^* = y_{n+1}^p + g_{k+1,n+1} h_{n+1} (f_{n+1}^p - f_{n+1}^p). \quad (3.1.2.5)$$

From (3.1.2.3) and (3.1.2.5) we have

$$e_{n+1} = (g_{k+1,n+1} - g_{k,n+1}) h_{n+1} (f_{n+1}^p - f_{n+1}^p).$$

The codes DVDQ and STEP actually accept the higher order value, y_{n+1}^* , as the approximation to $y(x_{n+1})$. Thus these PECE codes predict using an Adams-Bashforth formula of order k and correct using an Adams-Moulton formula of order $k+1$. Alternatively, they can be viewed as correcting the approximation, y_{n+1} , of order k by adding in the estimated error to form

$$y_{n+1} + e_{n+1} = y_{n+1}^*$$

this is again local extrapolation.

After computing the estimate of the local error, the codes will choose an appropriate stepsize and order for the next step. All modern Adams codes are variable order. Since at the start of the integration,

values from only one point are available, variable order codes usually start by using a one-step formula and then let the order changing technique take care of the order being used.

Codes vary the stepsize to take as large a step as possible consistent with the requirement that the local errors be less than a user-supplied tolerance. Two techniques are commonly used to implement variable stepsize Adams methods. One technique assumes that the points x_{n+1-i} , $i = 1, 2, \dots, k-1$ in (3.1.2.1) and (3.1.2.4) are equally spaced. We can then express the Adams formulas in the conventional form

$$y_{n+1} = y_n + h \sum_{j=0}^k \beta_j f_{n+1-j} \quad (3.1.2.6)$$

where the coefficients β_j , $j = 0, 1, \dots, k$ are constant. In this approach, called a fixed-coefficient implementation, interpolation is used to generate approximations to the solution at evenly spaced points when a stepsize change is required. The other technique, called a variable-coefficient implementation, does not require past values to be equally spaced. In this case we may still express the Adams formulas in the form (3.1.2.6) but the coefficients, β_j , now depend on the stepsizes used in the past steps.

The amount of work that each implementation requires at each step differs. For a variable-coefficient, k -step Adams formula the coefficients, β_j , must be recalculated if the stepsize has changed during the past k steps. On the other hand, for fixed-coefficient formulas, the coefficients can be precomputed, but interpolation must be performed when the stepsize is changed.

Both theoretical and empirical results for the techniques indicate that the variable-coefficient implementations have better stability properties. Gear and Tu(1974) show that codes based on fixed-coefficient implementations must restrict the frequency with which the stepsize is changed in order to remain stable. No such restriction is required for a variable-coefficient implementation as was shown by Piotrowski(1969).

Another feature which distinguishes the codes of Table 3.1 is the way they represent the polynomials $Q_{k,n}$ and $P_{k,n}$ of (3.1.2.1) and

(3.1.2.2). Rather than representing these polynomials in ordinate form by storing the past values f_{n+1-j} , $j = 1, 2, \dots, k-1$, two popular representations are based on the use of either a vector of divided differences or a Nordsieck vector. If the divided differences $[f_n, f_{n-1}, \dots, f_{n+1-j}]$, $j = 1, 2, \dots, k$ are used then it is easy to update the representation of $Q_{k,n}(t)$ from one step to the next even after a change of stepsize or order. Furthermore, the divided differences are useful for estimating the local errors at different orders. This is the approach taken in VOAS. The codes DVDQ and STEP use a divided difference representation but store the scaled quantities $h_{n+1}(h_{n+1}^{-k}h_n) \dots (h_{n+1}^{-k}h_n^{k-2-j})[f_n, f_{n-1}, \dots, f_{n-j+1}]$ rather than the unscaled values. Since large values of the derivatives usually correspond to small stepsizes, the scaled divided differences are less likely to vary in magnitude as much as the unscaled quantities. Another popular representation of $Q_{k,n}(t)$ is based on the scaled derivatives $h^j Q_{k,n}^{(j)}(t)/j!$. This approach was first proposed by Nordsieck(1962) and was popularized by Gear(1971a). It is used in all the other Adams codes in Table 3.1. Jackson and Sedgwick(1977) show that the ordinate, divided difference and Nordsieck representations are all related by co-ordinate transformations.

If it is required to output the solution at user specified points then interpolation of the computed solution values is required. If the solution y_{out} is required at $x = x_{out}$ where $x_n \leq x_{out} \leq x_{n+1}$ then y_{out} may be calculated from the formula

$$y_{out} = y_n + \int_{x_n}^{x_{out}} P_{k,n}(t) dt.$$

Since a representation of $P_{k,n}(t)$ has already been computed by the code, the computation of y_{out} can be achieved very cheaply. Thus Adams codes are well suited to applications for which the output is required at a large number of points. Stetter(1979) discusses various strategies for interpolating computed values in Adams codes.

A summary of the different properties of the Adams codes appears in Table 3.2.

Code	Type	Local Extrap.	Stepsize Strategy	Representation
DVDQ, STEP	PECE	Yes	Var. Coeff.	Scaled Div. Diff.
DIFSUB, GEAR and LSODE	AM	No	Fixed Coeff.	Nordsieck
EPISODE	AM	No	Var. Coeff.	Nordsieck
VOAS	PECE	No	Var. Coeff.	Divided Difference

Table 3.2: Properties of Adams Codes

The codes GEAR and LSODE are highly tuned versions of the original code DIFSUB developed by Gear(1971b,c). Variations of these codes are widely available, appearing in both the IMSL and NAG libraries. As will be discussed later, these codes contain options for solving stiff problems as well. Excellent documentation for the code STEP appears in the book by Shampine and Gordon(1975). All the strategies used by STEP are fully described and a listing of the code is given. The code also forms part of the DEPAC package developed at Sandia Laboratories.

3.1.3 Extrapolation Methods

These methods are based on the Richardson extrapolation process. The technique is applicable if the computed solution $y(t, h)$ can be expressed as

$$y(t, h) = y(t) + \sum_{i=1}^m \tau_i(t) h^i + O(h^{m+1}).$$

If such a series exists, we may approximate it by some function $R_m(t, h)$ with $m+1$ unknowns determined by the following requirement

$$R_m(t, h_j) = y(t, h_j), \quad j = 0, 1, \dots, m.$$

$y(t)$ can now be approximated by $R_m(t, 0)$ which is an approximation of order m . $R_m(t, h)$ may be a polynomial or a rational function.

Gragg(1965) and Bulirsch and Stoer(1966) used this idea for solving non-stiff equations. The method they suggested involves using a modified midpoint rule which has been shown to have an asymptotic expansion having only the even powers of h . A modification of the original code of Bulirsch and Stoer is presented by Fox(1971). This code has been tested by Hull, Enright, Fellen et al.(1972), Enright and Hull(1976a) and Shampine, Watts and Davenport(1976). The extrapolation code DIFSYI due to Hussels(1973) improves the performance of the original code of Bulirsch and Stoer with respect to the starting stepsize. Deuflhard(1980) has modified DIFSYI to produce a code, DIFEXI, which is slightly more efficient. In DIFEXI polynomial extrapolation is used and Deuflhard claims this is more efficient than rational extrapolation.

Extrapolation methods must calculate a series of approximate solutions $y(t, h_j)$, $h_0 > h_1 > \dots > h_m > 0$. For the extrapolation process to be stable, the stepsizes h_j must decrease rapidly, and more work needs to be done in computing $y(t, h_{j+1})$ than $y(t, h_j)$. Unlike LMFs, to achieve higher orders the amount of work per step for extrapolation methods must increase substantially. To justify this large amount of work per step, higher order extrapolation methods require that large stepsizes be used. The testing of Shampine, Watts and Davenport(1976) shows that the efficiency of extrapolation codes is impaired by factors which constrain the stepsize, such as stiffness or mild stiffness, lack of smoothness in the true solution, e.g. by the presence of discontinuities in the derivatives of the true solution, or by the need to output the solution at a number of closely spaced, user specified points. For problems where high accuracy is required and the solution is smooth, extrapolation codes can use very high order formulas and can be very efficient when function evaluations are relatively inexpensive.

3.2. METHODS FOR SOLVING STIFF EQUATIONS

It seems that formulas which are suitable for stiff equations are generally implicit and their implementation requires the iterative solution of a set of non-linear equations at each step. As discussed

earlier, when solving non-stiff equations using implicit formulas, the solution may be obtained using simple iteration. As we will see, simple iteration is not suitable for the solution of stiff equations. Recently Shampine(1979, 1980c) has looked at the problem of solving these non-linear equations. We will follow his notation and discuss the problem in some detail because of its importance.

Consider the use of an implicit LMF. At each step we must solve the set of implicit equations

$$y_{n+1} - h\delta_k f(x_{n+1}, y_{n+1}) = \psi \quad (3.2.0.1)$$

where ψ is constant and groups the contribution from terms at past points. Using simple iteration we obtain the next approximate solution $y_{n+1}^{(s)}$ by

$$y_{n+1}^{(s)} = h\delta_k f(x_{n+1}, y_{n+1}^{(s-1)}) + \psi.$$

For simple iteration to converge, we must have $h\delta_k \|\partial f / \partial y\| < 1$. Since for stiff equations $\|\partial f / \partial y\|$ is usually large, the stepsize h must be severely constrained to obtain convergence. Therefore, codes for solving stiff equations must use some other technique to solve equations (3.2.0.1). The techniques which stiff codes use are all based on the Newton-Raphson method. These techniques, as we will see, are computationally more expensive than simple iteration.

The modified Newton (MN) iteration for solving the non-linear equations (3.2.0.1) associated with implicit LMFs is as follows

$$(I - h'y'y')(y_{n+1}^{(s)} - y_{n+1}^{(s-1)}) = -y_{n+1}^{(s-1)} + h\delta_k f(x_{n+1}, y_{n+1}^{(s-1)}) + \psi. \quad (3.2.0.2)$$

Here $h'y'y'$ is an approximation to $h\delta_k$ and the matrix J is an approximation to the Jacobian evaluated at the current point. If $h'y'y' = h\delta_k$ the method is known as the simplified Newton (SN) iteration. This technique requires the solution of a set of linear equations at each iteration. The coefficient matrix $I - h'y'y'$ is known as the Newton iteration matrix (or more simply, iteration matrix). Most implementations assume that the Jacobian is only varying slowly and because the Newton matrix is

expensive to form, the same iteration matrix is often used for a number of steps even though the stepsize h and the leading coefficient β_k may vary. The iteration matrix is often stored in its LU decomposition form. Klopfenstein(1971) and Krogh and Stewart(1982) discuss the effect of having an inaccurate Jacobian on the rate of convergence of the iterations.

In order to solve a system of non-linear equations using the Newton technique (3.2.0.2) we need $O(N^2)$ cells of storage for the iteration matrix where N is the number of equations in the differential system. If the Jacobian is small, the user may be willing to supply the explicit Jacobian but even for moderately large systems, most users find supplying an explicit Jacobian a tedious and error-prone task. Therefore, the Jacobian is often approximated by numerical differencing. This requires N or $N+1$ function evaluations. Forming the LU factors of the iteration matrix requires $O(N^3/3)$ operations, while using the LU factorization to solve a linear system at each iteration requires $O(N^2)$ operations.

When solving very large systems of differential equations the cost of the linear algebra can become a major part of the total computation cost and very large problems can only be handled efficiently by methods which take advantage of any special structural properties of the Jacobian. For instance, a device due to Curtis, Powell and Reid(1974) may achieve substantial reductions in the cost of approximating the Jacobian by numerical differencing by taking advantage of the known structure of the Jacobian. For example, banded Jacobians of half bandwidth m can be formed by differencing by using only $2m + 1$ extra function evaluations regardless of the size of the system of equations. Carver and MacEwen(1981) have investigated the use of sparse matrix techniques in handling large stiff differential systems arising from using the method of lines to solve partial differential equations and show that some problems can be more efficiently solved if the Jacobian is approximated by a sparse matrix. Recently attempts have been made to solve the equations (3.2.0.2) by iterative methods rather than by direct LU-decomposition. One such technique is discussed by Gear and Saad(1981). As will be seen in the following section, the cost of solving the non-linear equations has had a great influence on the development of new

formulas for stiff equations.

For LMF, the iteration matrix is of the form $I - h\tilde{g}_k J_k$. If we choose to retain the scaled iteration matrix $J - (1/h\tilde{g}_k)I$ then if either h or $\tilde{g}_k$ change the new scaled iteration matrix differs from the current scaled iteration matrix by only a constant diagonal matrix. If the scaled iteration matrix is kept in its LU decomposed form then the most efficient and stable way to obtain the LU decomposition of the new scaled iteration matrix is by forming the new scaled iteration matrix and subsequently its LU factors. Unfortunately, in this situation we cannot exploit the fact that the two matrices differ only by a constant diagonal matrix. Enright(1978) has proposed that the scaled iteration matrix be decomposed into LHL^{-1} where L is a lower triangular matrix and H is a Hessenberg matrix. Then if

$$J - c_1 I = LHL^{-1}$$

$$\begin{aligned} \text{we have } J - c_2 I &= (J - c_1 I) + (c_1 - c_2)I \\ &= L(H + (c_1 - c_2)I)L^{-1} \end{aligned}$$

and in the event that either the stepsize or the leading coefficient change, we can very efficiently update the LHL^{-1} factorized form of the scaled iteration matrix. For a system of N differential equations the LHL^{-1} factorization costs $O(5/6N^3)$ operations as compared to the LU factorization cost of $O(1/3N^3)$ operations. Solving a linear system with the iteration matrix in LHL^{-1} form costs $O(2N^2)$ operations compared to $O(N^2)$ operations when the matrix is in LU form. On the other hand, updating the LHL^{-1} factorization can be done in $O(N)$ operations while in the LU form a new factorization costing operations which are $O(N^3)$ is necessary. In order to be able to solve linear systems more efficiently with the iteration matrix in LHL^{-1} form, Enright suggests that the Hessenberg matrix itself be decomposed into LU factors L_1 and H_1 where L_1 is bidiagonal. This reduces the cost of solving the linear system to $O(3/2 N^2)$ operations while increasing the cost of updating the factorization to $O(N^2)$ operations.

The LHL^{-1} factorization can achieve considerable savings if the problem has a constant or slowly changing Jacobian and the number of equations is medium to large. It is not clear how this technique will perform when the Jacobian is not varying slowly and in any case many large systems of equations are either banded or sparse and the LHL^{-1} factorization is not able to exploit these special structural properties of the Jacobian. Partly to overcome this problem, Enright and Kamei(1979) have proposed an iteration scheme which is based on the LHL^{-1} factorization of the iteration matrix and tries to partition the Hessenberg matrix automatically into two parts, one corresponding to stiff components and the other corresponding to non-stiff components. A reduction of the computational expense may then be achieved by approximating that part of the Hessenberg matrix which corresponds to the non-stiff components by either a zero matrix or a diagonal matrix.

As we have noted before, most formulas suitable for solving stiff equations are implicit and therefore involve the solution of non-linear equations. In an effort to reduce the cost involved in solving the non-linear equations associated with implicit formulas, Lambert and Sigurdsson(1972) suggested a class of formulas which are not implicit in the usual sense. These formulas are called linearly-implicit. Whereas implicit methods use an approximation to the Jacobian as an aid to solving the non-linear equations at each step, linearly-implicit formulas incorporate an approximation to the Jacobian into the actual formula and require the solution of at least one set of linear equations at each step.

As an example the following formula is an A-stable, linearly-implicit first order formula

$$y_{n+1} = y_n + hf_n + hJ(y_{n+1} - y_n)$$

where J is an approximation to the Jacobian.

The main drawback with linearly-implicit formulas is that the nature of the approximation to the Jacobian can affect the absolute stability properties of the method. In the example given above, the order of the formula is unaffected by the accuracy of J but the absolute

stability properties are. For instance, if we replace J by the zero matrix then the formula reduces to the forward Euler method.

Most stiff problems have transient regions where small stepsizes are needed to track all components accurately in a relative sense. In these regions the stepsize is so restricted by the accuracy requirement that the problem is effectively non-stiff and simple iterations may be used. Therefore the stiff codes could be made more efficient if they could automatically change to simple iterations whenever possible. Recently, progress has been made in designing type-insensitive codes which attempt to recognize whether the problem is locally stiff or non-stiff and attempt to choose a suitable formula and iterative method automatically. The first efforts by Shampine(1977, 1980b) developed techniques so that Adams and Runge-Kutta codes could decide whether stiffness is the reason they are performing inefficiently. More recently Shampine(1981, 1982a) has designed codes based on implicit A-stable and $A(\alpha)$ -stable formulas which can automatically recognize stiffness. Shampine also presents results of an experimental code where the convergence rate of the iterative method is monitored and at each step the code tries to select an iterative method which will converge sufficiently quickly. Petzold(1983) discusses stiffness detection techniques which have been used in a version of LSODE.

Stiff differential problems arising in practice often have additional features which ODE solvers should be able to deal with. As discussed by Gear(1982) and Thomson and Tuttle(1982), one or more of the following occur frequently in problems being solved by practitioners:

- (a) The equations may be implicit. Systems arising from the Method of Lines using collocation methods for solving PDEs lead to linearly-implicit equations which may be written as

$$A(x, y)y' = g(x, y)$$

where A is a square matrix, usually banded. A version of LSODE, called LSODI, is available to solve such equations.

- (b) The system of equations may consist of both algebraic and differential equations and therefore the number of variables is more than

the number of differential equations. Petzold(1982a) discusses the problems involved in solving such systems and a code for solving such equations is described by Petzold(1982b).

(c) Discontinuities in the derivatives of the solutions also need to be dealt with and a code should be able to locate a discontinuity and take appropriate action. Carver(1977), Gear and Osterby(1981) and Gear(1982) discuss the difficulties and how these might be overcome. A version of LSODE, called LSODAR, has facilities for root finding to help locate a discontinuity.

(d) The solution may have high frequency components which do not damp out. Petzold(1978) has investigated a technique which attempts to predict the rate of change of the behaviour of the starting point of each cycle and attempts to step over many cycles. Gear and Gollivan(1982) discuss the automatic detection of oscillatory behaviour and how a code might automatically select a method appropriate for the problem.

3.2.1 One-Step Methods

A considerable amount of research in recent years has been directed to the problem of solving stiff systems by using implicit one-step methods. Recently Cash(1982b) has presented an survey of implicit Runge-Kutta methods. In this section we look at some of the more promising developments in regard to using implicit one-step methods for stiff systems.

Butcher(1964a) investigated r-stage implicit RKFs and showed that such formulas can be of order $2r$. Ehle(1968) proved that an r-stage implicit RKF of order $2r$ is A-stable. The fact that implicit RKFs could be of high order and still be A-stable has attracted a great deal of interest and several sets of A-stable and L-stable implicit RKFs have been proposed by Butcher(1964b), Ehle(1968) and Chipman(1971). The main problem with all these formulas is that the solution of the resulting non-linear equations is quite expensive.

Consider the following r-stage implicit RKF:

$$y_{n+1} = y_n + h \sum_{j=1}^r b_j f(x_n + c_j, y_j)$$

where the internal approximations, y_i , are given by

$$y_i = y_n + h \sum_{j=1}^r a_{ij} f(x_n + c_j h, y_j), \quad i = 1, 2, \dots, r. \quad (3.2.1.1)$$

In order to advance a step it is necessary to solve the non-linear equations (3.2.1.1). If there are N differential equations in the system (1.1) then for an r-stage RKF, the non-linear system (3.2.1.1) consists of Nr equations. The solution of (3.2.1.1) can be achieved using a modified Newton scheme.

At the k^{th} iteration of the modified Newton scheme we compute the approximation $y_i^{(k)}$ to y_i , $i = 1, 2, \dots, r$. Let the residual, $z_i^{(k)}$, associated with $y_i^{(k-1)}$ be defined by

$$z_i^{(k)} = -y_i^{(k-1)} + y_n + h \sum_{j=1}^r a_{ij} f(x_n + c_j h, y_j^{(k-1)}).$$

Then in the k^{th} iteration we obtain $y_i^{(k)}$ by

$$y_i^{(k)} = y_i^{(k-1)} + w_i^{(k)}$$

where $w_i^{(k)}$, $i = 1, 2, \dots, r$ are the solution of the following set of linear equations

$$\begin{bmatrix} I - ha_{11}J & -ha_{12}J & \dots & -ha_{1r}J \\ -ha_{21}J & I - ha_{22}J & \dots & -ha_{2r}J \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{r1}J & -ha_{r2}J & \dots & I - ha_{rr}J \end{bmatrix} \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \\ \vdots \\ w_r^{(k)} \end{bmatrix} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_r^{(k)} \end{bmatrix}. \quad (3.2.1.2)$$

Here I is the $N \times N$ unit matrix and J is an approximation to the Jacobian

matrices $J(x_n + c, h, y_j^{(k-1)})$. Thus in the system (3.2.1.2) we solve for the r vectors $w_i^{(k)}$, $i = 1, 2, \dots, r$ simultaneously. Each vector $w_i^{(k)}$ has N components.

Even for small systems of differential equations, implicit RKFs implemented in this manner may not be competitive with LMFs. Several attempts have been made to reduce the linear algebra costs associated with implicit RKFs. The earliest use of a successful approach was by Bickart and Picel(1973). (Although the approach was applied to composite multistep formulas it is equally applicable to implicit RKFs). The clearest description of a variation on the theme is that of Butcher(1976) who suggested the following scheme.

Define the $r \times r$ matrix $A = (a_{ij})$ and let the Jordan canonical form of A^{-1} be

$$T^{-1}A^{-1}T = \begin{bmatrix} \lambda_1^{-1} & 0 & 0 & \dots & 0 \\ \mu_1 & \lambda_2^{-1} & 0 & \dots & 0 \\ 0 & \mu_2 & \lambda_3^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \lambda_r^{-1} \end{bmatrix},$$

μ_i is zero if $\lambda_i = \lambda_{i+1}$, $i = 1, 2, \dots, r-1$, zero or an arbitrary non-zero number if $\lambda_i = \lambda_{i+1}$. Let D be the $r \times r$ diagonal matrix with diagonal entries $\lambda_1, \lambda_2, \dots, \lambda_r$.

Define the $r \times r$ transformation matrices $P = (p_{ij})$ and $Q = (q_{ij})$ by

$$P = DT^{-1}A^{-1},$$

$$Q = T^{-1}$$

and consider the following transformations

$$\tilde{w}_i^{(k)} = \sum_{j=1}^r q_{ij} w_j^{(k)}, \quad i=1, 2, \dots, r,$$

$$\tilde{z}_i^{(k)} = \sum_{j=1}^r p_{ij} z_j^{(k)}, \quad i=1, 2, \dots, r.$$

Butcher(1976) shows that when expressed in terms of the transformed variables $\tilde{w}_i^{(k)}$ and $\tilde{z}_i^{(k)}$, the system (3.2.1.2) reduces to

$$\begin{bmatrix} I - h\lambda_1 J & 0 & \dots & 0 \\ -h\mu_1 J & I - h\lambda_2 J & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I - h\lambda_r J \end{bmatrix} \begin{bmatrix} \tilde{w}_1^{(k)} \\ \tilde{w}_2^{(k)} \\ \vdots \\ \tilde{w}_r^{(k)} \end{bmatrix} = \begin{bmatrix} \tilde{z}_1^{(k)} \\ \tilde{z}_2^{(k)} \\ \vdots \\ \tilde{z}_r^{(k)} \end{bmatrix} \quad (3.2.1.3)$$

When using Butcher's technique, we form the LU factors of each of the distinct diagonal blocks. To solve the linear system (3.2.1.2) at each iteration we transform $\tilde{z}_i^{(k)}$ to obtain $\tilde{w}_i^{(k)}$, solve the linear system (3.2.1.3) and transform $\tilde{w}_i^{(k)}$ to obtain $w_i^{(k)}$. In solving (3.2.1.2) directly we have to solve an $N \times N$ linear system: by using Butcher's technique we can solve (3.2.1.3) by solving r , $N \times N$ linear systems sequentially. Note, where A has complex eigenvalues, we may be forced to use complex arithmetic in order to implement Butcher's technique.

For large differential systems, the cost involved in forming and factorizing the iteration matrix is the dominant linear algebra cost since this is an $O(N^3)$ operation for a system of N equations. For implicit LMFs, this cost is $O(N^3/3)$. When using Butcher's technique, the cost of forming and factorizing the iteration matrices is minimized when the matrix A has only one r -fold eigenvalue, in which case only one iteration matrix need be stored. This iteration matrix is of order N

and its LU factorization requires $O(N^3/3)$ operations.

Implicit RKFs which are r -stage and whose coefficient matrix (a_{ij}) has only a single r -fold eigenvalue are known as singly-implicit RKFs and have been proposed by Burrage(1978) as being suitable for the solution of stiff equations. The formulas have a maximum attainable order of $r+1$ for an r -stage method. Singly-implicit RKFs of orders up to fifteen have been derived by Burrage(1978). The formulas are $A(\alpha)$ -stable with $\alpha > 83^\circ$.

At each iteration, a singly-implicit RKF requires two transformations, $z_i^{(k)}$ from $z_i^{(k)}$ and $w_i^{(k)}$ from $\phi_i^{(k)}$. For an r -stage formula this costs $O(2Nr^2)$ operations. Forming the residuals $z_i^{(k)}$ will require r function evaluations per iteration. For an implicit LMF, solving the linear system at each iteration will require $O(N^2)$ operations with one function evaluation necessary per iteration. For large differential systems a singly-implicit RKF will require roughly r times as much work per step as an implicit LMF since it will require roughly r times as much linear algebra per iteration and r times as many function evaluations as an implicit LMF.

For small to medium large differential systems, the cost of transforming the quantities $z^{(k)}$ and $\phi^{(k)}$ becomes comparable to the cost of solving the linear system (3.2.1.3). If, for example, there are twenty equations in the differential system and we are using a six-stage, singly-implicit formula then at each iteration we are doing about half as many operations in transforming variables as in actually solving the linear system (3.2.1.3). For $N = 20$, $r = 6$, transforming variables costs $2Nr^2 = 1440$, solving linear systems $rN^2 = 2400$ while the $1/3 N^3$ cost of an LU factorization is $1/3 N^3 = 2667$.

Burrage, Butcher and Chipman(1980) have implemented a family of singly-implicit RKFs in a code called STRIDE using a transformation which is discussed by Butcher(1979). Error estimation in the code is handled by embedding an r -stage, r th order formula within an $(r+1)$ -stage, $(r+1)$ st order formula. Numerical testing by Gaffney(1982) indicate that STRIDE is expensive with respect to function evaluations and

is only likely to be competitive with BDF based codes for problems where the Jacobian of the system has eigenvalues close to the imaginary axis. Gaffney has also considered the cost of transforming variables in STRIDE and found that when a stiff system of 6 equations was solved, approximately 48% of the total CPU time was spent in transforming variables as against 34% in solving linear equations.

Nørsett(1974) has proposed a class of RKFs with reduced linear algebra costs called semi-implicit RKFs. For semi-implicit RKFs the coefficient matrix A is lower triangular. Thus for these formulas the internal approximations Y_i can be computed in the order $i = 1, 2, \dots, r$ by solving r systems of N non-linear equations sequentially. The maximum attainable order of an r -stage, semi-implicit RKF is $r+1$, although for r even and less than ten, the maximum attainable order is r . Semi-implicit RKFs may have high order and may be A -stable. For example, sixth order, six-stage A -stable formulas are known (Cooper and Sayfy, 1979).

Semi-implicit RKFs where all the non-zero diagonal elements of the coefficient matrix A are equal to some number λ are termed diagonally implicit RK, DIRK (Alexander, 1977). For these formulas λ a Newton iteration scheme is used to solve the non-linear equations for the Y_i then only one iteration matrix is necessary. DIRK formulas are actually a subset of singly-implicit RKFs and may be implemented using Butcher's scheme. However, most codes which implement DIRK formulas solve the sets of non-linear equations sequentially. DIRK formulas require less computational effort per step than singly-implicit RKFs implemented using Butcher's technique but singly-implicit RKFs may achieve higher order for a given number of stages than DIRK formulas.

Mono-implicit RKFs (Cash, 1975, 1982b), call Implicit Endpoint Quadrature Rules by Van Bokhoven(1980), are a generalization of implicit RKFs. If we consider r -stage formulas of the type

$$Y_{n+1} - Y_n = h \sum_{j=1}^r b_j f(x_n + c_j h, Y_{n+j}).$$

$$Y_{n+j} = \delta_j Y_{n+1} + (1 - \delta_j) Y_n + \sum_{i=1}^r a_{ij} f(x_n + c_j h, Y_{n+1})$$

where b_j, δ_j, c_j and $a_{ij}, i = 1, 2, \dots, r$ and $j = 1, 2, \dots, r$ are constants, then if $\delta_j = 0$ for all j the formula is an implicit RKF. If, however, we have $a_{j1} = 0, 1 \leq j$, then the resulting formula is a mono-implicit RKF which is only implicit in a single unknown, Y_{n+1} . The non-linear system to be solved at each step is of size N , N being the number of equations in the differential system, but the Newton iteration matrix is now a polynomial of degree r in J , an approximation to the Jacobian.

Singhal(1980) considered singly-implicit, mono-implicit RKFs, formulas where the Newton iteration matrix factorizes exactly as the power of a single matrix, but found that such formulas were not in general competitive with LMFs. Cash(1982b) therefore considers an approach used by Skeel and Kong(1976) in conjunction with blended LMFs and derives low order formulas for which the coefficient matrix of the Newton scheme in a sense nearly factorizes as a power of a single matrix. Cash also presents some numerical results which suggest that such formulas may be competitive with LMFs at low error tolerances.

For problems where the explicit Jacobian is easy to obtain and is not much more expensive than the evaluation of the function f , the following modified Rosenbrock formulas (also called Generalized RKFs and ROW methods) may be suitable:

$$Y_{n+1} = Y_n + h \sum_{i=1}^r c_i k_i$$

$$Ek_i = f(x_n + A_i h, Y_n + \sum_{j=1}^{i-1} a_{ij} k_j) + B_i h f(x_n, Y_n)$$

$$+ \sum_{j=1}^{i-1} c_{ij} k_j, \quad i = 1, 2, \dots, r,$$

$$E = I - h Y f(x_n, Y_n).$$

ROW methods are a generalization due to Wolfbrandt(1977) of a class of formulas proposed by Rosenbrock(1963). A more computationally efficient formulation of ROW methods was found by Kaps and Wanner(1981). An r -stage ROW method can have a maximum order of $r+1$ and requires the evaluation of the Jacobian and an LU decomposition at each step. Each stage of a ROW method requires only the solution of linear systems of equations. It is, of course, possible to use an approximation to the Jacobian, but since J appears in the formulas itself the use of an approximate Jacobian may affect the order of the method.

Kaps and Rentrop(1979) have implemented two, three-stage fourth order ROW methods. One of the codes is based on formula which is $A(89.3^\circ)$ -stable while the other is based on an A -stable formula. Because the formulas are low order, Kaps and Rentrop are able to approximate the Jacobian by numerical differencing at each step. Results of using the two codes to solve the 25 test problems in Enright, Hull and Lindberg(1975) are presented and compared with the performance of a BDF-code. Kaps and Rentrop conclude that the codes were reliable and efficient in solving problems whose Jacobians have eigenvalues near the imaginary axis and on other problems as long as the number of equations is small and low accuracy was required.

Shampine(1980a) discusses various aspects of designing a code based on ROW formulas. He presents details of a code, DEGRK, based on a pair of A -stable formulas of orders 3 and 4 with damping at infinity. Shampine's code does not require the differential system in autonomous form as the codes of Kaps and Rentrop(1979) do but the user must supply not only the Jacobian but also the derivative $\partial f / \partial x$ explicitly. Results of numerical testing are presented to show that for the restricted class of small problems, DEGRK is efficient. The code is type-insensitive and uses a RK-Fehlberg pair to solve the equations whenever the equations are locally non-stiff.

Since an LU decomposition is required at each step, ROW methods are unlikely to be suitable for medium to large systems. Steihaug and Wolfbrandt(1979) have considered Rosenbrock-like formulas where the Jacobian matrix in a Rosenbrock matrix can be replaced by a general

matrix A without affecting the order properties of the formula. Such methods are known as W-methods. As the matrix A is no longer assumed to be the exact Jacobian, such Rosenbrock-like formulas must satisfy a large number of order requirements. Currently, only low order examples are known.

Day and Murthy(1982) derive a second order, L-stable W-method which requires two function evaluations and the solution of five linear systems per step and a third order, L-stable formula which requires three function evaluations and seven linear systems per step. Both formulas have an embedded lower order formula which can be used as an error estimator. Day and Murthy also derive a third order, L-stable formula with a lower order embedded error estimator which requires two function evaluations and six linear systems per step and which is like a ROW method in that it requires the exact evaluation of the Jacobian at some point but is also like a W-method in that it can use this Jacobian approximation in subsequent steps.

Since the stability properties of the formulas depend on how the matrix A approximates the Jacobian, any implementation of these W-methods will have to monitor the outdatedness of the Jacobian approximation and this may prove an obstacle to an efficient implementation. For instance, the non-linear stability of linearly-implicit one-step methods such as ROW and W methods over a large class of non-linear problems was investigated by Hairer, Bader and Lubich(1982) who showed that even if the methods were A-stable extra conditions for the parameters were needed in order that the methods produce contractive numerical solutions for sufficiently small h. This restriction on stepsize, however, did not depend upon the stiffness of the differential equation but instead depended upon the second derivatives of the true solution of the differential equation.

3.2.2 Linear Multistep Methods

The most commonly used multistep methods for stiff equations are the backward differentiation formulas (BDFs) as discussed by Henrici(1962, Section 5.1-4). Indeed the BDFs were being used for solving stiff equations by Curtis and Hirschfelder(1952). Gear(1971b) implemented the BDFs in a variable-order, variable-stepsize code DIFSUB which has probably the first such code for solving stiff equations. DIFSUB has proven to be one of the most efficient codes for the solution of stiff equations and is still in widespread use.

The basis of BDFs is discussed by Henrici(1962). The formulas are based on a polynomial $P_{n+1}(x)$ interpolating through $y_{n-k+1}, \dots, y_n$ and y_{n+1} . To obtain y_{n+1} , we require that $P_{n+1}(x)$ satisfy the differential equation

$$y' = f(x, y)$$

at $x = x_{n+1}$. That is the equation

$$P'_{n+1}(x_{n+1}) = f(x_{n+1}, y_{n+1})$$

defines the BDFs. The first and second order BDFs are A-stable and BDFs of orders up to 6 are A(α)-stable with the following α :

Order	3	4	5	6
Stability Angle α	86.0	73.4	51.8	17.9

Because of the small stability angle of the 6th order BDF, some of the codes use only orders up to 5. An important property of the BDFs is that the characteristic polynomials of the formulas have zero roots at $h\lambda = \infty$.

In spite of the relatively poor absolute stability properties of the BDFs, many widely used codes for solving stiff equations are based on the BDFs. As noted in Section 3.1.2, GEAR and LSODE are improved

versions of DIFSUB produced by Hindmarsh(1974, 1980). All the three codes can handle either stiff or non-stiff equations efficiently as the user, by setting parameters in the calling sequence, can select either the BDFs or the Adams-Moulton formulas as well as an appropriate iterative method. GEAR and LSODE allow several types of iterative schemes. Several versions of GEAR and LSODE exist which have been specifically written to solve implicit differential equations or large systems of differential equations where the Jacobian is either banded or sparse. Various versions of LSODE are discussed by Hindmarsh(1982).

The codes DIFSUB, GEAR, LSODE use the Nordsieck representation and implement the BDFs as fixed-coefficient formulas. As discussed in Section 3.1.2, when fixed-coefficient formulas are used, frequent stepsize changes can cause instability and therefore the above codes keep the stepsize and order constant for $q+1$ steps after the stepsize is changed using formula of order q . Recently Gear(1980) has suggested how Adams methods and BDFs may be implemented using modified divided differences.

A variable-coefficient implementation of BDFs using Lagrangian form representation is discussed by Brayton, Gustavson and Hachtel(1972) who report that the variable-coefficient implementation was found to be much more stable than the fixed-coefficient implementation. However the variable-coefficient implementation could also become unstable for some sequences of stepsize changes. Byrne and Hindmarsh(1975) discuss another implementation of variable-coefficient BDFs; this one using the Nordsieck representation. Extensive testing of this latter code, EPISODE, (Byrne, Hindmarsh, Jackson et al, 1977) has shown that sometimes it may solve problems which could not be solved by the fixed-coefficient implementation. Curtis(1980) discusses the reasons for this and cites the performance of his code FACSIMILE which uses a fixed-coefficient implementation of BDFs and does not fail on these problems. He argues that the failure of the fixed-coefficient codes on these problems may actually be due to the heuristics used in the codes and may not necessarily be due to the fixed-coefficient implementation.

The advantages of a variable-coefficient implementation of better stability and therefore allowing the possibility of stepsize changes at

each step do not overcome the two main problems of such codes. Firstly, additional computing is required at the current step unless the last q steps have been taken with the same order and stepsize. Secondly, and more importantly, if the formula's coefficients are changing at each step, then the Newton iteration matrix $W_{n+1} = I - h_{n+1}\beta_k J$ also varies. This means that either the iteration matrix must be updated more frequently or the iterations may converge more slowly.

Jackson and Sacks-Davis(1980) have proposed a scheme for changing stepsize. They suggest a technique in which the leading coefficient of the formula is unaffected by a change in the stepsize and therefore the Newton iteration matrix changes as in a fixed-coefficient implementation. On the other hand the other coefficients of the formula do depend on past stepsizes and the stability of implementations based on this stepsize changing technique approaches that of a variable-coefficient implementation.

Testing of stiff ODE solvers by Enright, Hull and Lindberg(1975) and Enright and Hull(1976b) has shown that the BDF-codes are the most efficient codes available except for problems which have Jacobians having eigenvalues with large imaginary parts. The BDF-codes become very inefficient for such problems and a lot of research work has been directed towards finding formulas which will perform better than BDF-codes for such problems.

One approach to finding new LMFs has been to consider k -step LMFs which have all roots zero at infinity but which have order less than k , and thus have extra parameters which can be used to improve the $A(\alpha)$ -stability angle. This approach has been adopted by Klopfenstein(1971) and more recently Varah(1978) and Skelboe and Christiansen(1981). For example, Skelboe and Christiansen derive a sixth order, eight-step formula which has all roots zero at infinity and is $A(55.9^\circ)$ stable. For a given order, better $A(\alpha)$ -stability angles are obtained but always with the added overhead costs of larger local truncation error coefficients.

The second approach has been to consider k^{th} order, k -step LMFs which do not have all roots zero at infinity. Nordsieck(1962) suggested that all LMFs are ultimately equivalent to finding a polynomial of some

Given degree approximating the solution of the differential equation. Wallace and Gupta(1973) develop a polynomial formulation of LMFs. They show that a polynomial approximation to the solution is updated at each step by the addition of a multiple of a constant polynomial. This constant polynomial which they call a modifier polynomial uniquely determines the LMF. Wallace and Gupta(1973), Gupta(1976) and Gupta and Wallace(1975) have derived a number of families of LMFs. These families consist of k th order, k -step formulas and can contain $A(\alpha)$ -stable formulas with order as high as twelve. They also derive $A(\alpha)$ -stable formulas of order less than or equal to six which exhibit larger stability angles than the BDFs of the same order.

Gupta(1982) has implemented in a code called DSTIFF a family of formulas which use a modifier polynomial based on a least squares approximation of previously computed values. The code implements formulas which are $A(\alpha)$ -stable, $\alpha > 62.78^\circ$ and which are stable up to tenth order. The local truncation error coefficients of the formulas increase in magnitude with increasing order and the modulus of the largest root at infinity also increases with increasing order. For the formulas of sixth order and higher, the magnitude of the largest root at infinity is greater than or equal to 0.79. Recently Gupta(1982) has presented results of testing DSTIFF and has compared its performance with that of LSODE. The test results indicate that DSTIFF is comparable in efficiency to LSODE. For stiff problems where the eigenvalues of the Jacobian are close to the imaginary axis, DSTIFF is much more efficient. Over a range of problems LSODE generally takes fewer function evaluations and DSTIFF makes fewer Jacobian evaluations. DSTIFF also keeps the local error in the computed solution below the user specified tolerance much more successfully than LSODE. In spite of the use of high order formulas in DSTIFF, LSODE is more efficient at stringent tolerances. This seems to be partly due to the large roots at infinity of the high order formulas used in DSTIFF.

3.2.3 Extrapolation Methods

Dahlquist(1963) discussed the use of extrapolation in solving stiff equations. He indicated that, though extrapolation could be applied using the trapezoidal rule, if these extrapolated values are used in the subsequent computation, the resultant method is no longer A -stable.

Lindberg(1971, 1972, 1973) has studied various questions connected with extrapolation using the trapezoidal rule and the following implicit midpoint rule:

$$y_{n+1} = y_n + hf(x_{n+1/2}, (y_n + y_{n+1})/2).$$

Both these methods have global error expansions in even powers of h only and thus seem well suited for repeated extrapolation. However numerical oscillations are obtained when solving stiff equations using extrapolation and the amplitude of these oscillations may sometimes be large. Lindberg has suggested the use of smoothing and extrapolation together to get rid of these oscillations. He has developed a code IMPEX2 whose details are given in Lindberg(1973). The numerical testing of Enright, Hull and Lindberg(1975) and Enright and Hull(1976b) indicates that a version of IMPEX2 was efficient at large tolerances but had difficulty solving problems which have strong non-linear coupling.

Liniger and Odeh(1972) discuss methods based on "averaging" of multistep methods. The technique is similar to extrapolation. While in extrapolation one solves an ODE using one formula and various step sizes, in "averaging" solutions are obtained using several low order multistep formulas. These solutions are then combined to obtain higher order solutions. For certain sets of formulas used, the "averages" are A -stable. Algorithms of up to order six which are A -stable are presented (Liniger, 1976).

Bader and Deufhard(1981) have developed an extrapolation method based on a linearly-implicit analogue of the explicit midpoint rule. For the method to achieve order $2m$, m subintegrations must be performed. Each subintegration starts with a step taken using a linearly-implicit analogue of the backward Euler formula and then m steps are taken using

the linearly-implicit midpoint rule. Each subintegration requires an LU-factorization of the iteration matrix. To achieve fourth order requires two subintegrations, two LU-decompositions, five function evaluations and the solution of eight linear systems involving the iteration matrix. Achieving sixth order requires three subintegrations, three LU-decompositions, ten function evaluations and the solution of fifteen linear systems.

It is interesting to note that the linearly-implicit analogue of the midpoint rule can also be considered to be a two-stage cyclic formula where the first stage is the forward Euler rule and the second stage is a linearly-implicit analogue of the backward Euler rule. Thus, a linearly-implicit analogue of the midpoint rule is given by

$$(I - hJ)y_{n+2} = (I + hJ)y_n + 2hf_n + 2hJy_{n+1}$$

where J is an approximation to the Jacobian. This is equivalent to the following cyclic formula where the first stage is the forward Euler formula given by

$$y_{n+1} = y_n + hf_n$$

and the second stage is a linearly-implicit analogue of the backward Euler formula and is given by

$$(I - hJ)y_{n+2} = y_{n+1} + 2hf_{n+1} - hf_n + hJ(-2y_{n+1} + y_n).$$

If the expression for the first stage is used to substitute for y_{n+1} in the expression for the second stage then we have the expression for the linearly-implicit analogue of the midpoint rule.

Bader and Deufhard(1981) compared the performance of their extrapolation code METAN1 with the BDF based code GEAR on twenty five problems at three tolerances. They found that both codes took approximately the same amount of time to solve most problems but that GEAR took fewer function evaluations and required far fewer LU-decompositions. (At the tolerance of 10^{-6} METAN1 required more than twice as many LU-decompositions than GEAR). Thus the fact that extrapolation codes require a number of LU-decompositions per step means that such codes are

not well suited to large problems unless advantage can be taken of the structure of the Jacobian of the problem. Deufhard, Bader and Nowak(1980) have also incorporated sparse matrix techniques into METAN1 to produce a code METAS1 which is used in a package for the simulation of large differential systems arising in chemical kinetics.

Shampine(1982b) has modified METAN1 to produce a type-insensitive code. If the code diagnoses the problem to be locally non-stiff, a trivial Jacobian approximation of $J = 0$ is used. The code therefore avoids always having to solve a number of linear systems at each step.

3.2.4 Second Derivative Methods

Since no LMF of order greater than two can be A-stable, considerable interest has been shown in classes of formulas which do not suffer from this limitation. One such class is a class of k-step second derivative multistep formulas(SDFs) due to Enright(1972). These formulas are more accurate than the BDFs of the corresponding order and A(α)-stable formulas of up to order 9 exist. However, in order to apply these formulas an approximation to the second derivative must be computed at each step. If the differential system is written in the autonomous form

$$y'(x) = f(y(x))$$

then we have the relation $y''(x) = J(y(x))y'(x)$. All current codes based on Enright's formulas compute the second derivative terms in this manner and require that the equations be in autonomous form and that the user supply a routine for evaluating the Jacobian as well as one for evaluating the function f .

Enright's formulas are k-step SDFs of the form

$$y_{n+1} = y_n + h \sum_{j=0}^k \beta_j y_{n+1-j}' + h^2 \gamma_0 y_{n+1}''. \quad (3.2.4.1)$$

Like the Adams formulas, these formulas have the property that all

extraneous roots at the origin are zero and like the BDFs, all roots at infinity are zero. Enright chose the parameters β_j , $j = 0, 1, \dots, k$ and γ_0 to achieve maximum order. Thus the formulas (3.2.4.1) which are k -step are of order $k+2$. The formulas up to order 4 are A-stable.

In order to solve stiff systems using Enright's formulas, the resulting systems of non-linear equations have a Newton iteration matrix of the form

$$W_{n+1} = I - h\delta_0 J - h^2 \gamma_0 J^2 \quad (3.2.4.2)$$

where J is some approximation to the Jacobian at the current point. In order to form W_{n+1} a matrix multiplication is required. Thus forming the Newton iteration matrix in its LU factored form requires $O(4N^3/3)$ operations compared to $O(N^3/3)$ for LMFs.

Implementations of Enright's formulas perform a lot more linear algebra per step than implementations of the BDFs. As well as the matrix multiplications discussed above there are other costs that do not appear in codes based on the BDFs such as the matrix-vector multiplications required to form the second derivative terms. On the other hand, because Enright's formulas are of up to order nine and have much smaller truncation error coefficients than the BDFs, they tend to require far fewer steps than codes based on the BDFs. Also, because of their better stability properties, SDF-codes do not experience the difficulties of BDF-codes in solving stiff problems that contain eigenvalues with large imaginary parts. Test results indicate that for small to medium problems, SDF-codes are competitive with BDF-codes. However, for large problems the cost of forming W_{n+1} as in (3.2.4.2) makes them relatively much more expensive. Also the requirement that the user supply an exact Jacobian makes SDF-codes less suitable for consideration as general purpose codes.

In his thesis, Enright(1972) presented a code SDBASIC for solving stiff systems. This code is a variable-step, variable-order implementation. It uses a one-step, two-half-step error estimate and restricts stepsize changes to halving and doubling. Since then more efficient implementations which use a predictor-corrector approach and estimate

the error as a difference between predicted and corrected values have been developed. These include the codes SECIDER (Addison, 1977) and SDSTEP (Sacks-Davis, 1980).

Variations of Enright's original formulas have been proposed. Since matrix squaring when forming W_{n+1} involves $O(N^3)$ operations and may reduce sparsity and increase bandwidth, Enright(1974) considered ways of avoiding the matrix squaring. He proposed a class of k -step SDF of the form (3.2.4.1) which are of order $k+1$ and for which $\beta_0^2 + 4\gamma_0 = 0$. For these formulas, the iteration matrix W_{n+1} of (3.2.4.2) can be expressed in the form

$$W_{n+1} = (I - h\delta_0 J/2)^2.$$

Thus the need to form a matrix multiplication is avoided although the solution of a system of equations $W_{n+1} x = b$ will now require two forward and backward substitutions rather than one. The resulting set of SDF have poorer $A(\alpha)$ -stability properties than Enright's original formulas.

In another approach, Sacks-Davis and Shampine(1981) show that SDFs can be used to solve both stiff and non-stiff problems in a type-insensitive code. Since Enright's formulas have small truncation error coefficients they are suitable for solving non-stiff problems as well as stiff systems. The availability of the exact Jacobian leads to a natural test for stiffness based on the norm of $h_{n+1}J$.

3.2.5 Blended Linear Multistep Methods

Lambert and Sigurdsson(1972) present multistep formulas whose coefficients depend on the stepsize and the Jacobian of the equations being solved. The accuracy of the Jacobian does not affect the order of the method. Skeel and Kong(1977) have proposed a subclass of these formulas as suitable for a general purpose ODE solver. One of the aims of Skeel and Kong was to design a code efficient for non-stiff as well as stiff problems.

Skeel and Kong's approach is to take a multistep formula involving higher derivatives and convert it into a variable matrix coefficient multistep formula so that the order of accuracy and the region of absolute stability are unaffected. For example, for linear problems, the (k+2)-order SDF of Enright(1972) may be expressed as a blend of the (k+2)-order Adams-Moulton formula and the (k+1)-order BDF.

$$\{ \text{Enright}^{k+2} \} \rightarrow \{ \text{AMF}^{k+2} \} + \gamma_k^* h^2 \{ \text{BDF}^{k+1} \}$$

where J is an approximation to the Jacobian and γ_k^* is a constant. In the case of Enright's third order formula

$$y_{n+1} - y_n - 2/3 hf_{n+1} - 1/3 hf_n + 1/6 h^2 y_{n+1}'' = 0$$

the third order Adams-Moulton formula is

$$y_{n+1} - y_n - 5/12 hf_{n+1} - 2/3 hf_n + 1/12 hf_{n-1} = 0$$

while the second order BDF is

$$3/2 y_{n+1} - 2 y_n + 1/2 y_{n-1} - hf_{n+1} = 0$$

and the blended formulas which has the same absolute stability properties as Enright's third order formula is

$$y_{n+1} - y_n - 5/12 hf_{n+1} - 2/3 hf_n + 1/12 hf_{n-1} - 1/6 h^2 (3/2 y_{n+1}'' - 2 y_n'' + 1/2 y_{n-1}'' - hf_{n+1}') = 0.$$

By adjusting the coefficient γ_k^* it is possible to improve the absolute stability region without sacrificing accuracy. In this case the corresponding SDF is (k+1)st order. As remarked earlier, the SDFs of Enright(1972) have Newton iteration matrices which are quadratic polynomials in the Jacobian and cannot be reduced to a product of real linear factors. Skeel and Kong have implemented their formulas in a code called BLENDED-BDFSUB and avoid this difficulty by choosing to use an approximate Newton iteration matrix which can be expressed as a

perfect square in a real linear factor. It is not clear what effect this approximation has on the rate of convergence of the iterations.

Compared to LMFs, the blended formulas require an extra matrix by a vector multiplication per iteration and when stiff equations are being solved, the exact Newton iteration matrix is quadratic in the Jacobian. However, the family of formulas presented by Skeel and Kong(1977) are A-stable up to order four and A(α)-stable up to order twelve. Skeel and Kong compared the performances of the BDF-code DIFSUB and a modified form of DIFSUB which implements blended LMFs and found that the blended code was better than the BDF-code in solving non-stiff equations and much better in solving stiff oscillatory problems. For other problems, it seems the performance of blended code is similar to that of DIFSUB. However, as we have noted, the blended LMFs incur extra computational expense at each iteration. These costs are not really reflected in the test results because the test problems are small systems and for these the linear algebra costs are small.

3.2.6 Composite and Cyclic Multistep Formulas

One way of overcoming the limitations of LMFs with respect to the achievable order of A-stable formulas is to use a number of LMFs together to simultaneously compute the solution at a number of points. Such a scheme is known as a composite LMF.

Rosser(1967) and Shampine and Watts(1969) propose the following type of formulas which compute a block of r new values of y at each stage by using r LMFs of the following type

$$\sum_{j=1}^r a_j y_{n+j} = e_1 y_n + h d_1 f_n + \sum_{j=1}^r b_j f_{n+j}, \quad j = 1, 2, \dots, r. \quad (3.2.6.1)$$

The r LMFs are used together to find $y_{n+1}, y_{n+2}, \dots, y_{n+r}$ at one stage. The main advantages of such methods are the possibility of high order, A-stable methods. In the composite scheme (3.2.6.1), the computation of a new block of values requires past values from only one point, and

which have order greater than their step number.

Tendler, Bickart and Picel(1978) derive $A(\alpha)$ -stable cyclic LMFs which are k th order, k -step for $k = 3, 4, \dots, 7$ and which display better $A(\alpha)$ -stability properties, order for order, than the BDFs. The third and fourth order cyclic formulas use three LMFs cyclically while the higher order cyclic formulas use four. By using the first and second order BDFs, Tendler, Bickart and Picel(1978) have implemented their cyclic formulas in a code called STINT. STINT is efficient for stiff problems where the Jacobian of the system of equations has some eigenvalues with negative real part and close to the imaginary axis but otherwise it is less efficient than a BDF based code such as GEAR or LSODE.

Of all the alternatives to implicit LMFs for solving stiff equations which we have considered, only cyclic LMFs can be implemented in a way which requires the same amount of linear algebra per step as implicit LMFs. Since cyclic LMFs exist which have better $A(\alpha)$ -stability properties, order by order, than the BDFs it seems that the use of cyclic LMFs for solving stiff systems of equations is a particularly promising line of research at this moment.

therefore changes in the stepsize can be easily made after each block has been computed. Schemes like (3.2.6.1) are also referred to as block-implicit one-step methods and have been discussed by Watts and Shampine(1971), Bickart and Picel(1973) and Williams and DeHoog(1974). Sloate and Bickart(1973) present a fourth order A -stable method using two, three-step LMFs and they also present details of an implementation. Their formulas require values at two past points.

In general, these schemes have one main drawback in common with implicit Runge-Kutta schemes. If a composite LMF calculates r new values at r points, it generally requires the solution of a system of rN non-linear equations, N being the number of equations in the differential system. The methods proposed for reducing the linear algebra costs associated with the solution of non-linear equations for implicit RKF can also be applied to composite LMFs. The technique of Bickart(1977), for instance, was originally applied to composite multistep formulas.

Semi-implicit RKFs can be implemented in such a way that r sets of N implicit equations need to be solved sequentially in each step. Semi-implicit RKF have composite LMF analogues in the cyclic composite LMFs, also known as cyclic LMFs (CLMFs), which use r LMFs in a set order to calculate the solution at r points. In this way, r systems of N non-linear equations are solved successively to generate the solution at r points. If all the formulas in the cyclic formula have the same Newton iteration matrix, the cost per step of a cyclic formula is the same as that of using a single LMF for computing the solution at all r points.

Donelson and Hansen(1971) derived stable cyclic LMFs from k -step LMFs of order $2k-1$ for $k = 2, 3$ and 4 and therefore showed that a cyclic LMF could be stable even though its constituent LMFs were not. Furthermore these methods were shown to be globally convergent to order $2k$. Thus, they also demonstrated that a cyclic method could have higher order than the order of its constituent formulas. Albrecht(1978) has given general conditions for determining the order of a wide class of methods and shows how his results apply to composite LMFs and in particular to cyclic methods. Mihelcic(1977) derives $A(\alpha)$ -stable cyclic LMFs

CHAPTER 4

STABILITY AND ACCURACY PROPERTIES OF CYCLIC METHODS

In this chapter we discuss the absolute stability and accuracy properties of CLMFs, with the possibility of extending our approach to cyclic methods in general.

4.1 STABILITY OF CYCLIC LINEAR MULTISTEP METHODS

We now present definitions of stability and strong D-stability for methods represented in the form (2.3.2).

Definition 4.1.1: Stability (D-stability, Zero-stability, Stability at the Origin)

A method presented in the form (2.3.2) is stable iff

- (a) the eigenvalues of A do not exceed one in absolute value,
- (b) for eigenvalues of modulus 1 there exist only linear, elementary divisors.

Definition 4.1.2: Strong D-stability (Strong-stability, Strong Zero-stability)

A method represented in the form (2.3.2) is strongly D-stable if the method is stable and only one eigenvalue exists which is of modulus 1 and that eigenvalue is equal to 1.

For CLMFs represented in the form (2.3.3) we have

$$A = \prod_{i=1}^M A(i).$$

Since the eigenvalues of a product of matrices are unaffected by cyclic permutations in the order of the product, we can immediately see that cyclic permutations in the order of the stages of the CLMF will not affect the stability properties at the origin of the CLMF.

Donelson and Hansen(1971) showed that it was possible for a CLMF to be stable although none of its constituent LMFs were stable. This approach to the stability at the origin of CLMFs has also been used by Stetter(1973).

If we apply a LMF (2.3.1) to the linear initial value problem

$$y' = \lambda y, \quad y(0) = 1 \quad (4.1.1)$$

where λ is a complex, constant number (or matrix) then, using the notation (2.3.1), we can simplify

$$z_{j+1} = Az_j + h\phi(x_j, z_j, z_{j+1}; h)$$

to obtain $z_{j+1} = Lz_j$ (4.1.2)

$$\text{where } L = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ \frac{-a_0 + h\lambda\beta_0}{a_k - h\lambda\beta_k} & \frac{-a_1 + h\lambda\beta_1}{a_k - h\lambda\beta_k} & \frac{-a_2 + h\lambda\beta_2}{a_k - h\lambda\beta_k} & \dots & \frac{-a_{k-1} + h\lambda\beta_{k-1}}{a_k - h\lambda\beta_k} \end{bmatrix}.$$

L is the companion matrix to the stability polynomial (2.2.5). Since the eigenvalues of L are the roots of the stability polynomial, all the stability concepts which we have previously defined in terms of the roots of the stability polynomial can be equivalently defined in terms of the eigenvalues of L .

Let us now consider the application of an M -stage, k -step CLMF (2.3.3) to the linear initial value problem (4.1.1).

For the i th LMF of the cycle we have

$$z_{j+1} = L^{(i)} z_{j+1-1}, \quad i = 1, 2, \dots, M.$$

Thus we can simplify (2.3.1) to obtain

$$z_{j+M} = C z_j \quad \text{where } C = \prod_{i=1}^M L(i).$$

The characteristic polynomial of C , $P(h\lambda, r)$, for the CLMF (2.3.1) is its stability polynomial. This polynomial is of maximum degree M in $h\lambda$ and the polynomial coefficient of r^k in $P(h\lambda, r)$ is

$$\prod_{i=1}^M (\alpha_k^{(i)} - h\lambda \beta_k^{(i)}).$$

The CLMF will be absolutely stable for the linear initial value problem for those values of $h\lambda$ where all the eigenvalues of C are less than or equal to 1 in absolute value. Note that LMFs can be regarded as CLMFs with all stages the same and will have consistent stability properties if viewed as either LMFs or CLMFs.

It is possible to extend this approach to consider the absolute stability properties of cyclic methods whose constituent formulas are not necessarily LMFs. For example, consider the second derivative method

$$y_{n+1}^{(k)} + \alpha_{k-1} y_n^{(k)} + \dots + \alpha_0 y_{n-k+1}^{(k)} - h(\beta_k y_{n+1}^{(k)} + \beta_{k-1} y_n^{(k)} + \dots + \beta_0 y_{n-k+1}^{(k)})$$

$$+ h^2 (\gamma_k y_{n+1}^{(k)} + \gamma_{k-1} y_n^{(k)} + \dots + \gamma_0 y_{n-k+1}^{(k)}) = 0.$$

When the above formula is used to solve (4.1.1), we may write

$$z_{j+1} = S z_j \quad \text{with } S = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ s_0 & s_1 & s_2 & \dots & s_{k-1} \end{bmatrix},$$

$$s_1 = \frac{-\alpha_1 + h\lambda \beta_1 - (h^2 \lambda^2 / 2) \gamma_1}{\alpha_k - h\lambda \beta_k + (h^2 \lambda^2 / 2) \gamma_k}, \quad i=0, 1, \dots, k-1.$$

Definition 4.1.2: Stability Curve

The stability curve of a method with stability polynomial $P(h\lambda, r)$ is the locus of

$$P(h\lambda, e^{i\theta}) = 0, \quad 0 \leq \theta \leq 2\pi. \quad (4.1.3)$$

The stability curve of a method is the locus of those points where the stability polynomial has at least one root whose magnitude is equal to 1. Based on the continuity argument, we can say that if the method is absolutely stable at any point z in the complex $h\lambda$ plane, then the method is also absolutely stable at any point v that can be reached from z without crossing the stability curve. By repeatedly solving (4.1.3) for various values of θ , $0 \leq \theta \leq 2\pi$, we can determine the stability curve and region of absolute stability for any method. This is known as the root locus method. For methods whose stability polynomials have a maximum degree in $h\lambda$ which is less than or equal to four, (4.1.3) can be solved analytically, otherwise a numerical method must be used.

4.2 MAXIMUM ATTAINABLE ORDER OF A-STABLE METHODS

Dahlquist(1956) proved that if a LMF were A-stable, its order could not exceed two. In this section we present a theorem which is in some ways a generalization of Dahlquist's results and which is applicable to a wide range of methods, including CLMFs. This theorem uses the concept of a stability polynomial and relates the maximum attainable order of an A-stable method from a particular class of methods to the properties of the class of stability polynomials associated with that class of methods. For LMFs, the stability polynomial is $\rho(r) - h\lambda\sigma(r)$. Its degree in r gives the number of past points used by the formula.

Where different kinds of methods give rise to the same general type of stability polynomial, the methods display the same absolute stability behaviour for linear problems. For example, second derivative methods and the blend of two LMFs have associated stability polynomials in two variables, $h\lambda$ and r . These polynomials are quadratic in $h\lambda$ while their

degree in r depends on the number of past points used by the formula.

Let the general stability polynomial be a polynomial $P(h\lambda, r)$. The roots at the origin are given by the roots of $P(0, r)$ while the roots at infinity are given by the roots of $P(\infty, r)$. For convenience, we will refer to $P_k(h\lambda)$ in

$$P(h\lambda, r) = P_k(h\lambda)r^k + P_{k-1}(h\lambda)r^{k-1} + \dots + P_0(h\lambda) \quad (4.2.1)$$

as the leading polynomial coefficient. If $P_k(h\lambda)$ is of strict degree i in $h\lambda$, we will refer to i as the degree of implicitness of the stability polynomial and if the leading polynomial coefficient is expressible in the form $(1 - ch\lambda)^i$ where c is real then we will refer to the stability polynomial as being singly-implicit.

We now define an order property of the stability polynomial. Let the most recent past point be x_n and assume that the solution is calculated simultaneously at M points, $x_{n+1}, x_{n+2}, \dots, x_{n+M}$. The rate of growth of the true solution of (4.1.1) from x_n to x_{n+M} is $e^{Mh\lambda}$. We would therefore expect one of the roots of (4.2.1) to approximate $e^{Mh\lambda}$. If we have one root, R_1 , such that $R_1 = e^{Mh\lambda} + O((h\lambda)^{p+1})$, then we have

$$P(h\lambda, e^{Mh\lambda}) = P(h\lambda, R_1 + O((h\lambda)^{p+1})) = O((h\lambda)^{p+1}).$$

Definition 4.2.1: Linear Order

If a method has stability polynomial (4.2.1) which is irreducible, $P_k(0) \neq 0$, and we have

$$P(h\lambda, e^{Mh\lambda}) = c(h\lambda)^{p+1} + O((h\lambda)^{p+2}), \quad c \neq 0, \quad h\lambda \rightarrow 0, \quad p \geq 1, \quad (4.2.2)$$

then the stability polynomial is said to have linear order p .

By considering a Taylor series expansion of $P(h\lambda, e^{Mh\lambda})$ for $h \rightarrow 0$ it can be shown that if the stability polynomial has linear order p and if $\partial P / \partial (0,1) \neq 0$, then the stability polynomial has a root which is a $(p+1)^{\text{st}}$ order approximation of $e^{Mh\lambda}$.

For LMFs, a formula has local truncation order p if and only if

$$p(h\lambda) - h\lambda o(e^{h\lambda}) = c_{p+1}(h\lambda)^{p+1} + O((h\lambda)^{p+2}), \quad c_{p+1} \neq 0,$$

and thus for LMFs, linear order is equivalent to local truncation order. This result also generalizes to higher derivative methods. If the method is stable and the stability polynomial has linear order p , then the method will converge with order p for the linear initial value problem (4.1.1) given p^{th} order consistent initial conditions (Reimer, 1968).

A stable method with consistent order q and given q^{th} order consistent initial conditions will converge with order q for all initial value problems which possess a suitably differentiable true solution. In particular, it will converge with order q for the linear problem (4.1.1). Some methods, e.g. some explicit RKFs, will converge for linear problems with higher order than their order of consistency. So we have, in general, that the consistent order of a method is less than or equal to its linear order.

The following theorem links the property of A-stability to the degree of implicitness and the linear order of the stability polynomial associated with the numerical method. This theorem was conjectured by Daniel and Moore(1970) and was proven by Wanner, Hairer and Nørsett(1978).

Theorem 4.2.1: Daniel-Moore Theorem

Let a numerical method for solving an ordinary differential equation have a stability polynomial $P(h\lambda, r)$ which is irreducible, $P_k(0) \neq 0$, $\deg(P_k(h\lambda)) = i$, $\partial P / \partial r$ at $(0,1) \neq 0$ and which has linear order p . Then for the numerical method to be A-stable, $p \leq 2i$.

The Daniel-Moore theorem indicates, that if we want to construct methods which have high order and are A-stable we must construct methods which have a high degree of implicitness, i. The degree of implicitness, however, is related to the amount of linear algebra involved in the application of the numerical method for solving stiff equations.

For example, let us consider implicit RKFs. For an implicit RKF, the leading polynomial coefficient is also the characteristic polynomial of the coefficient matrix (a_{ij}) and the RKF will be singly-implicit if

its stability polynomial is simply-implicit. An r -stage, implicit RKF has $1 = r$ and Butcher's r -stage, implicit RKFs of order $2r$ reach the upper bound on the attainable order of an A -stable, r -stage implicit RKF.

For LMFs, the Newton iteration matrix is $\alpha_k I - h\lambda_k^j$ while the leading polynomial coefficient is $\alpha_k I - h\lambda_k^k$. As discussed in Chapter 3, composite multistep formulas require the same amount of linear algebra per application as implicit RKFs. In the special case where the composite multistep formula is a CLMF the leading polynomial coefficient of the cyclic method is the product of the leading polynomial coefficients of the constituent LMFs.

According to the Daniel-Moore theorem the maximum attainable linear order of an A -stable, M -stage, CLMF is $2M$. If the M -stage, CLMF has N explicit stages, $0 \leq N \leq M$, then the degree of implicitness is $M-N$, and the maximum attainable linear order of an A -stable, M -stage CLMF with N explicit stages is $2(M-N)$.

In Chapter 5 we derive a fourth order, A -stable two-stage, CLMF which reaches the maximum attainable order for methods whose stability polynomial is quadratic in $h\lambda$. Thus it is possible to construct M -stage CLMFs where all constituent LMFs share the same Newton iteration matrix and the CLMF is A -stable of order $2M$. These cyclic formulas require the same amount of computation per step as implicit LMFs, and should be particularly well suited for solving stiff systems of equations.

4.3 CONVERGENT ORDER

In the discussion on the Daniel-Moore theorem we introduced the concept of linear order and considered the convergence properties of methods applied to the linear initial value problem with constant Jacobian. By using the approach presented by Albrecht(1978) we now discuss the convergence properties of more general kinds of formulas to linear initial value problems whose solutions are suitably smooth.

Consider a k -step method, (2.3.2),

$$z_{j+1} = Az_j + h\phi(x_j, z_j, z_{j+1}; h), \quad z_0 = z_0(h),$$

with a certain matrix A and function ϕ .

Definition 4.3.1: Local Discretization Error

The local discretization error, $\tau(z, h, x_j)$, of the method (2.3.2) is given by

$$\tau(z, h, x_j) = h^{-1} (z(x_{j+1}) - Az(x_j) - h\phi(x_j, z(x_j), z(x_{j+1}); h)).$$

Definition 4.3.2: Order of Consistency

Let a method (2.3.2) have a local discretization error of the form

$$\tau(z, h, x_j) = h^q g(x_j) \underline{t} + O(h^{q+1})$$

where $q \in \mathbb{N}$, the initial value problem has $x = a$ as an initial condition and endpoint $x = b$, $\underline{t}$ is a constant vector, $\underline{t} \in \mathbb{R}^k$ and g is a bounded function $g: [a, b] \rightarrow \mathbb{R}$. $\underline{t}$ is termed the local discretization error vector, $||\underline{t}||$ is called the local discretization coefficient while q is called the order of consistency of (2.3.2).

For LMFs, the order of consistency is equal to the local truncation order $s-1$ for stable LMFs, it is equal to the order of convergence. $\underline{t}$, the local discretization error vector, is of the form

$$[0, 0, \dots, c_{q+1}]^T$$

where c_{q+1} is a non-zero error constant, the local truncation error constant.

Let us now consider the form of the local discretization error for CLMFs. In particular, we consider the two-stage, k -step CLMFs

$$z_{2j+1} = A^{(1)} z_{2j} + h\phi^{(1)}(x_{2j}, z_{2j}, z_{2j+1}; h),$$

$$z_{2j+2} = A^{(2)} z_{2j+1} + h\phi^{(2)}(x_{2j+1}, z_{2j+1}, z_{2j+2}; h).$$

Let the individual stages have orders of consistency p and q , then for a

suitable differentiable exact solution y we can say

$$\begin{aligned} z(x_{2j+1}) &= A^{(1)} z(x_{2j}) + h\phi^{(1)}(x_{2j}, z(x_{2j}), z(x_{2j+1}); h) \\ &\quad + \bar{t}^{(1)} h^{r+1} y^{(r+1)}(x_{2j}) + O(h^{r+2}), \end{aligned}$$

$$\begin{aligned} z(x_{2j+2}) &= A^{(2)} z(x_{2j+1}) + h\phi^{(2)}(x_{2j}, z(x_{2j+1}), z(x_{2j+2}); h) \\ &\quad + \bar{t}^{(2)} h^{r+1} y^{(r+1)}(x_{2j+1}) + O(h^{r+2}), \end{aligned}$$

where $r = \min(p, q)$ and $c_{r+1}^{(i)}$ is zero if the order of consistency of the i th stage is greater than r and $\bar{t}^{(i)} = [0, 0, \dots, c_{r+1}^{(i)}]^T$, $i = 1, 2$. By substitution we can get

$$\begin{aligned} w(x_{j+1}) &= A^{(2)} A^{(1)} w(x_j) + h\phi(x_{2j}, w(x_j), w(x_{j+1}); h) \\ &\quad + \bar{t}^{(2)} h^{r+1} y^{(r+1)}(x_{2j+1}) + A^{(2)} \bar{t}^{(1)} h^{r+1} y^{(r+1)}(x_{2j}). \end{aligned}$$

We can see that $\bar{t} = \bar{t}^{(2)} + A^{(2)} \bar{t}^{(1)}$.

Albrecht(1978) has shown that method (2.3.2) converges with order q for initial value problems with sufficiently differentiable true solutions if the method and the starting vector have order of consistency q and the method is strongly D -stable. He also gives additional conditions to ensure convergence which is of order $q+1$. Since cyclic methods are expressible in the form (2.3.2) and are also capable of fulfilling the additional conditions for higher order convergence, we now present details of Albrecht's proof.

For a method (2.3.2), if A is D -stable and if ϕ satisfies a uniform Lipschitz condition for $x \in [a, b]$ and $h \in [0, h_0]$

$$\|\phi(x, y_1, z_1; h) - \phi(x, y_2, z_2; h)\| \leq K_1 \|y_1 - y_2\| + K_2 \|z_1 - z_2\|. \quad (4.3.1)$$

then there is a constant C , such that for $j = 0, 1, \dots, m-k+1$

$$\|z(x_j) - z_j\| \leq C(\|z(x_0) - z_0(h)\| + \max_{0 \leq l \leq m-1} \|\tau(z, h, x_l)\|). \quad (4.3.2)$$

Thus z_j converges with order at least q , if the method and the starting vector have order of consistency q . We have

$$z(x_{j+1}) = Az(x_j) + h\phi(x_j, z(x_j), z(x_{j+1}); h) + \tau(z, h, x_j),$$

$$z_{j+1} = Az_j + h\phi(x_j, z_j, z_{j+1}; h),$$

from which we obtain

$$q_{j+1} = Aq_j + h\tau(z, h, x_j)$$

where $q_j = z(x_j) - z_j$, $d_j = \phi(x_j, z(x_j), z(x_{j+1}); h) - \phi(x_j, z_j, z_{j+1}; h)$ and by induction

$$q_{j+1} = A^{j+1} q_0 + h \sum_{l=0}^j A^{j-l} d_l + h \sum_{l=0}^j A^{j-l} \tau(z, h, x_l).$$

With $S_j = h \sum_{l=0}^j A^{j-l} \tau(z, h, x_l)$

and (4.3.1) one obtains by induction the following expression for the global error in z_j :

$$\|q_j\| = \|z(x_j) - z_j\| \leq K(\|z_0 - z_0(h)\| + \|S_j\|) \quad (4.3.3)$$

for all $h \in (0, h_0]$, $h_0 K_2 D < 1$, $D = \sup_{i \in N} \|A^i\|$.

Usually $\|S_j\|$ is bounded by

$$h(j+1)D \max_{0 \leq l \leq m-k} \|\tau(z, h, x_l)\| = (x_{j+1} - a)D \max_{0 \leq l \leq m-k} \|\tau(z, h, x_l)\|$$

which leads to a bound (4.3.2) of order q . This is the well known result that a q th order consistent method will converge with order q , given that the initial conditions are q th order consistent and that the

method is stable. Albrecht argues that in certain conditions we can get a sharper bound. His argument goes as follows.

By examining the expression for S_j we see that the local discretization error introduced in calculating z_1 contributes a term $A^{j-1} \tau(z, h, x_1)$ to our bound on the global error (4.3.3). Let us, therefore, express the local discretization error in terms of the eigenvectors of the matrix A .

Let $\mu_i = 1$, $\mu_i \neq 1$, $i = 2, \dots, k$ be the eigenvalues of the $k \times k$ matrix A and let $\{e_i, i = 1, 2, \dots, k\}$ be a normal eigenbasis in R^k , then with

$$\tau(z, h, x_1) = h^q g_{1-} \cdot c_{1-}^{1+1},$$

$$\underline{t} = \sum_{i=1}^k d_i e_i,$$

we can express S_j as follows:

$$S_j = h^{q+1} \sum_{l=0}^j g_l A^{j-l} \underline{t} + O(h^{q+1}),$$

$$S_j = h^{q+1} \left\{ \sum_{l=0}^j d_l \sum_{i=1}^k g_l e_i \right\} + \sum_{l=0}^j \sum_{i=2}^k d_l \mu_i^{j-l} g_l e_i + O(h^{q+1}).$$

S_j will be bounded for all j and all $h \in (0, h_0]$ if and only if

$$1) d_1 = 0 \text{ or}$$

$$ii) \left| \sum_{l=0}^j \sum_{i=1}^k g_l \right| \leq c \text{ for all } j \in N, c > 0 \text{ independent of } h \text{ and } j$$

and in either case we have $\|S_j\| = O(h^{q+1})$.

Lemma 4.3.1:

Let $\mu_i = 1$, $\mu_i \neq 1$, $i = 2, \dots, k$, be the eigenvalues of a $k \times k$ matrix A and let $\{e_i, i = 1, 2, \dots, k\}$ be a normal eigenbasis in R^k , then the following relations are equivalent for $\underline{t} \in R^k$:

$$\underline{t} = \sum_{i=1}^k d_i e_i \text{ with } d_1 = 0,$$

$$\underline{p}^T \underline{t} = 0 \text{ with } A^T \underline{p} = \underline{p}, \underline{p} \in R^k \setminus \underline{0}.$$

This leads us to conditions for the method to converge with higher order than the order of consistency. The following theorem is due to Albrecht (1978) and gives conditions so that a method will converge with higher order than its order of consistency.

Theorem 4.3.1:

Let the method (2.3.2) have order of consistency q and

(1) be strongly D -stable, with ϕ satisfying the Lipschitz condition (4.3.1),

(2) with a local discretization error

$$\tau(z, h, x_1) = h^q g_{1-} + O(h^{q+1})$$

for a suitably differentiable true solution $z(x)$,

(3) and a starting vector $\zeta_0(h)$ of order $q+1$, i.e.

$$\|z_0 - \zeta_0(h)\| = O(h^{q+1}).$$

Then the method converges with order $q+1$ if and only if, either

$$\underline{p}^T \underline{t} = \underline{0} \text{ where } \underline{p}^T A = \underline{p}, \underline{p} \neq \underline{0} \quad (4.3.4a)$$

or

$$\left| \sum_{l=0}^j \sum_{i=1}^k g_l \right| \leq c \text{ for all } j \in N, c > 0 \text{ independent of } h \text{ and } j. \quad (4.3.4b)$$

Since condition (4.3.4b) is problem dependent, it is expected that it will be of limited value.

We present an example of a CLMF with first order stages where the CLMF converges with second order.

Example 4.3.1:

Consider the cyclic use of the forward Euler and backward Euler one-step LMFs. The resultant CLMF has a linear order of two.

$$\begin{aligned}
y_{n+1} &= y_n + hf_n, & y(x_{n+1}) - y_{n+1} &= 1/2 h^2 y^{(2)}(x_{n+1}) + O(h^3) && \text{forward euler,} \\
y_{n+2} &= y_{n+1} + hf_{n+1}, & y(x_{n+2}) - y_{n+2} &= -1/2 h^2 y^{(2)}(x_{n+2}) + O(h^3) && \text{backward euler.}
\end{aligned}$$

By treating both formulas as though they were two-step formulas we have

$$A^{(1)} = A^{(2)} = A^{(2)} A^{(1)} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad \underline{p} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad (A^{(2)} A^{(1)})^T \underline{p} = \underline{p}.$$

$$\underline{t} = \begin{bmatrix} 0 \\ -1/2 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} \quad \text{so } \underline{p}^T \underline{t} = 0.$$

$\underline{p}^T \underline{t} = 0$ so we have the higher order convergence condition (4.3.4a). The method is strongly D-stable since the eigenvalues of $\tilde{A}$ are 1 and 0 and the local discretization error is of the form required by Albrecht's theorem. Therefore the cyclic use of the forward Euler and the backward Euler formulas results in a CLMF which, given second order consistent initial conditions, converges with second order although each constituent method is only first order consistent and the resultant cyclic method is only first order consistent.

In the following example we present a CLMF whose consistent order, linear order and convergent order are all different.

Example 4.3.2:

Consider the following two-stage, two-step CLMF

$$\begin{aligned}
y_{n+1} &= y_n - hf_n - h(f_{n+1} - 1/2 f_n + 1/2 f_{n-1}) = 0 && \text{stage 1,} \\
y_{n+2} &+ 7/6 y_{n+1} - 13/6 y_n - h(11/12 f_{n+2} + 9/12 f_{n+1} + 9/6 f_n) = 0 && \text{stage 2.}
\end{aligned}$$

Both stages are second order with error constants $-7/12$ and $-49/72$ respectively. The characteristic polynomial of the CLMF is

$$\begin{aligned}
P(h\lambda, r) &= r^2(11/12(h\lambda)^2 - 23 h\lambda/12 + 1) + r(28/12(h\lambda)^2 - 7h\lambda/6 - 1) \\
&\quad + (9/12(h\lambda)^2 + 13 h\lambda/12).
\end{aligned}$$

The CLMF has roots 0 and 1 at the origin and is therefore strongly D-stable. The CLMF has linear order 4. Written in the form (2.3.2), the CLMF has

$$\begin{aligned}
\tilde{A} &= \begin{bmatrix} 0 & 1 \\ 13/6 & -7/6 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \\
\underline{t} &= \begin{bmatrix} 7/12 \\ 0 \end{bmatrix}, \quad \tilde{A}^T \underline{p} = \underline{p} \text{ with } \underline{p} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \underline{p}^T \underline{t} = 0.
\end{aligned}$$

Therefore, by Albrecht's criterion, the CLMF is third order convergent. Each stage is second order consistent, the CLMF is second order consistent, converges with third order and in the special case of linear initial value problems with constant Jacobians, converges with fourth order.

The exact relationship between linear order and convergent order for CLMFs has not yet been determined. It is not known whether it is possible to construct methods which have a consistent order q and which converge with order greater than $q+1$ for all initial value problems with suitably smooth solutions. Since in computing a block of new values we introduce a local discretization error of $h\tau(z, h, x_1)$, it is clear that we cannot expect all the points in the block to be converging with order greater than $q+1$. It may, however, be possible that some new values will converge with higher orders than others in the block. For example, we may be able to construct methods with consistent order q where all values converge with order $q+1$ but where the endpoint in each block converges with order $q+2$.

Albrecht(1978) has also proven the following lemma for M-stage CLMFs.

- (b) For each k there exist k cyclic, linear k -step methods which are convergent to order $2k$.

4.4 ERROR COEFFICIENT

The local discretization error of a method (definition 4.3.1) gives us an indication of the short term accuracy properties of the method since it is the error that results in making one application of the method given that all past information is known exactly. In this section we will consider how the local discretization error introduced in computing the solution at the current point persists to effect the errors in the solution at subsequent points.

In computing the solution at x_{n+1} we introduce a local discretization error of $h\tau(z, h, x_{n+1})$. By (4.3.3) we can bound the amount of this error that effects the global error at the j th subsequent step by $h\lambda^j\tau(z, h, x_{n+1})$. Let us express the local discretization error in terms of the eigenvectors of λ as we did in the derivation of the higher order convergence conditions (4.3.4a,b). For strongly D-stable methods the matrix λ has only one eigenvalue which is equal to 1 and all other eigenvalues are less than 1 in magnitude. As the integration proceeds the local discretization error introduced at step x_{n+1} will be damped if it does not lie in the direction of the eigenvector of λ corresponding to the eigenvalue 1. As j gets large

$$h\lambda^j\tau(z, h, x_{n+1}) \rightarrow d_1 e_1 h^{q+1} g(x_{n+1}) + O(h^{q+2}).$$

Thus, $d_1 e_1 h^{q+1} g(x_{n+1})$ is the amount of the dominant term in the local discretization error that persists to effect the errors in the solution at subsequent steps.

It is trivial to show that $\bar{P}_1^T e_1 = 0$ for $1 = 2, 3, \dots, k$ so we can find d_1 by the equation $\bar{P}_1^T e_1 = d_1 \bar{P}_1^T e_1$. This leads to the following definition of the propagated error coefficient.

Lemma 4.3.2:

For a CLMF expressed in the form (2.3.3), the higher order convergence condition is invariant with respect to the cyclic permutations of the $A(i)$ in $A^{(M)}(M-1), \dots, A(1)$.

We have seen how Albrecht's approach to the convergence properties of a general class of multistep methods provides a means of studying the convergence properties of cyclic methods, particularly CLMFs. Donelson and Hansen (1971) used a different approach to the convergence properties of CLMFs and were also able to show that CLMFs could converge with higher order than their order of consistency. Since we feel Albrecht's approach is far more convenient and easy to generalize we will only touch on the Donelson-Hansen approach very briefly.

Donelson and Hansen consider the use for M constant sized steps of a M -stage, k -step CLMF. From this we can find M, k th order difference equations which can then be combined to form a single (kM) th order difference equation which only uses y values from M equally spaced points. M such (kM) th order difference equations can be obtained by considering all the cyclic permutations in the order of the stages. Donelson and Hansen call this set of $M, (kM)$ th order difference equations the auxiliary composite method and show that if the cyclic method is stable with order of consistency q and the auxiliary method has order $q+1$, then the cyclic method converged with order $q+1$.

Donelson and Hansen were especially interested in finding cyclic methods which had high convergent order with respect to the number of steps involved. k -stage, k -step CLMFs which were of consistent order $2k-1$ but converged with orders $2k-1$ or $2k$ were found for $k = 2, 3, 4$. Donelson and Hansen did not attempt to find such formulas for $k \geq 5$. In their own words, "The arithmetic for the case $M = k = 4$ is already a time consuming task. For $k > 4$, one must use a computer to accomplish this job in a reasonable amount of time".

Using the Donelson-Hansen approach Mishak (1972) has proven the following results.

- (a) For each k there exist $(k-1)$ cyclic, linear k -step methods which are convergent to order $2k-1$.

Definition 4.4.1: Propagated error coefficient.

Given a strongly D-stable, k-step method (2.3.2) that has a local discretization error of order q of the form

$$\tau(z, h, x_j) = h^q g(x_j) \underline{t} + O(h^{q+1}),$$

$q \in \mathbb{N}$, $\underline{t}$ a constant, non-zero vector, $\underline{t} \in \mathbb{R}^k$ and g a bounded function $g: [a, b] \rightarrow \mathbb{R}$, and where $\underline{e}_1$ and $\underline{p}$ are given by $\underline{A}\underline{e}_1 = \underline{e}_1$ and $\underline{A}\underline{p} = \underline{p}$, the propagated error coefficient D_{q+1} is given by

$$D_{q+1} = \frac{\underline{p}^T \underline{t}}{\underline{p}^T \underline{e}_1}.$$

Henrici(1962) has suggested that for LMFs the local truncation error coefficient scaled by the inverse of the sum of the beta coefficients is a better indicator of the long term accuracy properties than just the local truncation error coefficient itself. For the Adams formulas there is no difference in the error coefficients as the scaling factor turns out to be 1. For the k-step BDF the scaling factor is the inverse of the leading coefficient β_k and thus the scaling factor is always greater than or equal to 1. Many codes which implement the BDF, such as the DIFSUB - GEAR - LSODE family, use the scaled local truncation error coefficient in their stepsize and order selection and error estimation strategies.

As we show in Appendix B, the propagated error coefficient for LMFs is equal to the local truncation error coefficient scaled by the sum of the beta coefficients. Thus, the use of the propagated error coefficient in implementing CLMFs can be regarded as a generalization of the use of the scaled local truncation error coefficient for LMFs. Since the CLMF calculates the solution at a number of new points, we could use the propagated error coefficient weighted by the number of stages in controlling the error in an implementation of CLMFs.

Codes which implement LMFs and use the local truncation error coefficient attempt to control the amount of error introduced at any particular step on the assumption that errors in past points are negligible.

This approach can be generalized to cyclic methods by using the local discretization error coefficient. Codes which implement LMFs using the scaled local truncation error coefficient can be viewed as trying to control the amount of error introduced in a step which will propagate to effect the solution at later steps. For multistep formulas this error may be greater than the error introduced in the step, as is the case for the BDFs with stepnumber greater than 1, because a k-step multistep formula may directly use solution values from a given point at up to k future points. We can generalize this approach to error control for cyclic methods by using the propagated error coefficient divided by the number of stages in the cyclic method.

CHAPTER 5

DERIVATION OF A NEW CLASS OF CYCLIC MULTISTEP FORMULAS

A class of $A(\alpha)$ -stable, cyclic composite formulas that has been implemented in a variable stepsize, variable order code called STINT is described by Tendler, Bickart and Piceel(1978). The formulas used in STINT exhibit improved $A(\alpha)$ -stability properties, order by order over the BDFs and include formulas of orders up to seven. Unfortunately this code is not very efficient. For example, on the test problems of the test set of Enright, Hull and Lindberg(1975), using the relative error criterion, the BDF based code CEAR took 24.576 minutes to solve all the problems as compared to 39.701 minutes for STINT. Using the absolute error criterion the times were 34.694 minutes for CEAR and 62.033 minutes for STINT.

In this chapter we are interested in deriving a new class of cyclic formulas. Some desirable properties of cyclic formulas suitable for solving both stiff and non-stiff equations are:

- (i) the extraneous roots at the origin should be equal to zero (as with the Adams formulas);
- (ii) the roots at infinity should be equal to zero (as with the BDFs);
- (iii) the formulas should exhibit better $A(\alpha)$ -stability properties, order by order, than the BDFs.

When solving stiff equations, it is common to use a Newton iteration to solve the corrector equation. The iteration matrix used is of the form $\alpha_k I - h\beta_k \partial f / \partial y$ where α_k and β_k are the leading coefficients of the corrector formula. This matrix is expensive to re-evaluate in its factored form. In order to reduce the number of re-evaluations necessary to ensure fast convergence of the Newton scheme over a cycle, it is desirable that

- (iv) the leading coefficients of the multistep formulas not vary within a cycle.

It should also be noted that if in the M th stage of a cycle, a step is rejected and a stepsize reduction required, then the work in calculating the solution during the first $M-1$ stages is wasted.

For this reason

- (v) the number of stages within the cycle is limited to two.

Finally it is required that

- (vi) the leading truncation error coefficients of the multistep formulas used in the cycle should not be 'large'.

We note that the formulas used in the code STINT fail to satisfy (i), (iv) and (v). In particular, the third and fourth order STINT formulas use three stages while the fifth, sixth and seventh order formulas use four stages. STINT uses the average of the leading coefficients of the stages in forming the Newton iteration matrix and thus the coefficient used in the iteration matrix has a relative error in the worst case of 11.11%, 4.667%, 11.65%, 16.31% and 5.898% for the formulas of orders three to seven respectively. The first and second order formulas used in STINT are three-stage cyclic formulas where all the stages are identical and are the BDFs of the appropriate order.

We have previously observed that an M -stage CLMF has a stability polynomial $P(h\lambda, t)$ which is of degree M in $h\lambda$. Thus, the stability properties of two-stage cyclic formulas closely resemble those of second derivative formulas. By matching the characteristic polynomial of the two-stage cyclic formulas with that of a certain class of second derivative formulas we are able to find a class of cyclic formulas satisfying conditions (i) to (v). The new formulas exhibit improved $A(\alpha)$ -stability properties over both the BDFs and STINT formulas and include formulas of orders up to 8. These new methods are based on two-stage CLMFs which require values from k past points and are globally convergent to order k . The constituent stages are k th order consistent. We also derive methods with stages that are k th order consistent which are globally convergent to order $k+1$ in the sense of Albrecht(1978).

5.1 TWO-STAGE CYCLIC LINEAR MULTISTEP FORMULAS

Let us consider a two-stage CLMF where each stage is a k-step LMF. We may write the first stage as

$$\begin{aligned} \alpha_k^{(1)} y_{2n+k} + \alpha_{k-1}^{(1)} y_{2n+k-1} + \dots + \alpha_0^{(1)} y_{2n} \\ = h(\beta_k^{(1)} f_{2n+k} + \dots + \beta_0^{(1)} f_{2n}), \alpha_k^{(1)} = 1, \end{aligned}$$

and similarly the second stage as

$$\begin{aligned} \alpha_k^{(2)} y_{2n+k+1} + \alpha_{k-1}^{(2)} y_{2n+k} + \dots + \alpha_0^{(2)} y_{2n+1} \\ = h(\beta_k^{(2)} f_{2n+k+1} + \dots + \beta_0^{(2)} f_{2n+1}), \alpha_k^{(2)} = 1, \end{aligned}$$

We have already discussed representing the two-stage, k-step CLMFs in Albrecht's notation (2.3.3). We have also discussed the form of the local discretization error.

Since the component stages are LMFs, when the stages are represented by

$$z_{mj+1} = \Lambda^{(1)} z_{mj+1-1} + h\phi^{(1)}(x_{mj+1-1}, z_{mj+1-1}, z_{mj+1}, h), \quad i=1,2,$$

we may use the fact that the matrices $\Lambda^{(1)}$ and $\Lambda^{(2)}$ are Frobenius matrices to simplify Albrecht's higher order condition

$$P^T \bar{L} = \bar{0} \quad \text{where } \bar{L}^T P = P, \quad P = 0. \quad (4.3.4a)$$

In fact, for the two-stage CLMFs that we are considering the condition for higher order convergence reduces to a simple relation involving the coefficients of the component LMFs.

Lemma 5.1.1:

Let the two-stage, k-step CLMFs be represented by

$$w_{j+1} = \bar{A} w_j + h\bar{\phi}(x_j, w_j, w_{j+1}, h) \quad (2.3.3)$$

where the stages are q^{th} order with error coefficients $c_{q+1}^{(1)}$ and $c_{q+1}^{(2)}$.

Let

$$\alpha_{\text{even}}^{(1)} = \alpha_0^{(1)} + \alpha_2^{(1)} + \alpha_4^{(1)} + \dots + \alpha_{\frac{k}{2}}^{(1)} \quad i=1,2.$$

Then, if $\bar{A}$ is strongly D-stable, condition (4.3.4a) is satisfied for a two-stage CLMF if either

$$(1) \quad \alpha_{\text{even}}^{(1)} = 0, \alpha_{\text{even}}^{(2)} = 0 \text{ and } c_{q+1}^{(1)} = 0$$

or

$$(11) \quad \alpha_{\text{even}}^{(1)} = 0, \text{ and } c_{q+1}^{(2)} + (\alpha_{\text{even}}^{(2)} / \alpha_{q+1}^{(1)}) c_{q+1}^{(1)} = 0.$$

where the local discretization error vector $\underline{e}$ is given by

$$\begin{bmatrix} 0, 0, \dots, c_{q+1}^{(1)}, c_{q+1}^{(2)} - \alpha_{k-1}^{(2)} c_{q+1}^{(1)} \end{bmatrix}^T.$$

Proof:

Let $\underline{p} = [p_1, p_2, \dots, p_k]^T$. Then we may write the equations $(\bar{A}^T - I)\underline{p} = \underline{0}$ as

$$\begin{bmatrix} -1 & 0 & 0 & \dots & -\alpha_0^{(1)} & \gamma_0 \\ 0 & -1 & 0 & \dots & -\alpha_1^{(1)} & \gamma_1 \\ 1 & 0 & -1 & \dots & -\alpha_2^{(1)} & \gamma_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -\alpha_{k-1}^{(1)} & \gamma_{k-1} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \dots \\ p_k \end{bmatrix} = 0.$$

where $\gamma_0 = \alpha_{k-1}^{(2)} \alpha_0^{(1)}$ and $\gamma_i = \alpha_{k-1}^{(2)} \alpha_i^{(1)} - \alpha_i^{(2)}$, $i = 1, 2, \dots, k-1$. The last column of $(\bar{A}^T - I)$ is therefore

$$\begin{bmatrix} \alpha_{k-1}^{(2)} \alpha_0^{(1)} \\ \alpha_{k-1}^{(2)} \alpha_1^{(1)} - \alpha_0^{(2)} \\ \alpha_{k-1}^{(2)} \alpha_2^{(1)} - \alpha_1^{(2)} \\ \dots \\ \alpha_{k-1}^{(2)} \alpha_{k-1}^{(1)} - \alpha_{k-2}^{(2)} - 1 \end{bmatrix}.$$

If we label each of the above equations from 1 to k and sum either the even numbered equations or the odd numbered equations, then using the fact that if

$$\alpha_{\text{odd}}^{(1)} = \alpha_1^{(1)} + \alpha_3^{(1)} + \dots + \alpha_{\frac{k+1}{2}-1}^{(1)}$$

then

$$\alpha_{\text{even}}^{(1)} + \alpha_{\text{odd}}^{(1)} = 0 \quad i = 1, 2 \quad (\text{by considering } \sum_{j=0}^k \alpha_j^{(1)} = 0 \text{ from consistency})$$

and

$$\alpha_k^{(1)} = 1, \quad i = 1, 2,$$

we obtain the single equation

$$\alpha_{\text{even}}^{(1)} p_{k-1} - (\alpha_{k-1}^{(2)} \alpha_{\text{even}}^{(1)} + \alpha_{\text{even}}^{(2)}) p_k = 0.$$

Now, if $\alpha_{\text{even}}^{(1)} = \alpha_{\text{even}}^{(2)} = 0$ then the coefficient matrix is of rank at most $k-2$ and this contradicts the requirement of strong D-stability. There are two cases to consider.

Case 1

If $\alpha_{\text{even}}^{(1)} = 0$, then $\alpha_{\text{even}}^{(2)} = 0$ and the eigenvector $\underline{p}$ takes the form

$$[p_1, p_2, \dots, p_{k-1}, 0]^T$$

with $p_{k-1} \neq 0$. Then the condition $\underline{p}^T \underline{e} = 0$ requires $\alpha_{q+1}^{(1)} = 0$ and we have condition (i) of the lemma.

Case 2

If $\alpha_{\text{even}}^{(1)} \neq 0$ then $\underline{p}$ takes the form

$$\begin{bmatrix} p_1, p_2, \dots, p_{k-2}, \frac{\alpha_{k-1}^{(2)} \alpha_{\text{even}}^{(1)} + \alpha_{\text{even}}^{(2)}}{\alpha_{\text{even}}^{(1)}}, p_k \end{bmatrix}^T, \quad p_k \neq 0.$$

Then the condition $\underline{p}^T \underline{e} = 0$ becomes

$$\frac{\alpha_{k-1}^{(2)} \alpha_{\text{even}}^{(1)} + \alpha_{\text{even}}^{(2)}}{\alpha_{\text{even}}^{(1)}} \alpha_{q+1}^{(1)} + \alpha_{q+1}^{(2)} - \alpha_{k-1}^{(2)} \alpha_{q+1}^{(1)} = 0$$

and we have condition (ii) of the lemma.

We note that by condition (i) of the lemma we have the interesting result, reported by Donelson and Hansen (1972), that a strongly D-stable, 2-stage CLMF composed of Simpson's rule (which is fourth order) and any other third order LMF will be globally convergent to order four. Condition (ii) of the lemma may be used to find two-stage cyclic methods which are globally convergent to order $q+1$ but whose constituent formulas are each of order q .

We now investigate the $A(\alpha)$ -stability properties of two-stage cyclic LMFs. We have already shown that for the linear problem, $y' = \lambda y$, (2.3.3) takes the form

$$\underline{z}'_M = C \underline{z}_j, \quad C = \sum_{i=1}^M L^{(i)} \text{ with } L^{(i)} \text{ given by (2.1.2)}.$$

The $A(\alpha)$ -stability properties of the two-stage CLMF will be characterized by the eigenvalues of C (which depends on λ). For $M=2$, $k=3$, the characteristic polynomial $P_3(\lambda, r) = \det(C - rI)$ is given by

$$\begin{aligned} P_3(\lambda, r) = & \lambda^2 n^2 [-\beta_3^{(1)} \beta_3^{(2)} r^3 \\ & + (-\beta_3^{(1)} \beta_1^{(2)} + \beta_2^{(1)} \beta_2^{(2)} - \beta_1^{(1)} \beta_3^{(2)}) r^2 \\ & + (\beta_2^{(1)} \beta_0^{(2)} - \beta_1^{(1)} \beta_1^{(2)} + \beta_0^{(1)} \beta_2^{(2)}) r \end{aligned}$$

$$\begin{aligned}
& + \beta_0^{(1)} \beta_0^{(2)}] \\
& + h\lambda [(\alpha_3^{(1)} \beta_3^{(2)} + \beta_3^{(1)} \alpha_3^{(2)}) r^3 \\
& + (\alpha_3^{(1)} \beta_1^{(2)} - \alpha_2^{(1)} \beta_2^{(2)} + \alpha_1^{(1)} \beta_3^{(2)} \\
& + \beta_1^{(1)} \alpha_3^{(2)} - \beta_2^{(1)} \alpha_2^{(2)} + \beta_3^{(1)} \alpha_1^{(2)}) r^2 \\
& + (-\alpha_2^{(1)} \beta_0^{(2)} + \alpha_1^{(1)} \beta_1^{(2)} - \alpha_0^{(1)} \beta_2^{(2)} \\
& - \beta_2^{(1)} \alpha_0^{(2)} + \beta_1^{(1)} \alpha_1^{(2)} - \beta_0^{(1)} \alpha_2^{(2)}) r \\
& + (-\alpha_0^{(1)} \beta_0^{(2)} - \beta_0^{(1)} \alpha_0^{(2)})] \\
& + [-\alpha_3^{(1)} \alpha_3^{(2)} r^3 \\
& + (-\alpha_3^{(1)} \alpha_1^{(2)} + \alpha_2^{(1)} \alpha_2^{(2)} - \alpha_1^{(1)} \alpha_3^{(2)}) r^2 \\
& + (\alpha_2^{(1)} \alpha_0^{(2)} - \alpha_1^{(1)} \alpha_1^{(2)} + \alpha_0^{(1)} \alpha_2^{(2)}) r \\
& + \alpha_0^{(1)} \alpha_0^{(2)}] \quad (5.1.1)
\end{aligned}$$

In general, when the component LMFs are k -step, $P_k(h\lambda, r)$ for two-stage CLMFs may be expressed as

$$P_k(h\lambda, r) = \sum_{j=0}^k (p_j - h\lambda q_j + h^2 \lambda^2 m_j) r^j \quad (5.1.2)$$

where

$$p_j = \sum (-1)^t (\alpha_t^{(1)} \alpha_s^{(2)}),$$

$$q_j = \sum (-1)^t (\alpha_t^{(1)} \beta_s^{(2)} + \beta_t^{(1)} \alpha_s^{(2)}),$$

$$m_j = \sum (-1)^t (\beta_t^{(1)} \beta_s^{(2)}),$$

and the summations are taken over all s, t such that $s+t = 2j$ and $s, t \in [0, \min(k, 2j)]$. (Stetter, 1973, also gives expressions for the characteristic polynomials).

5.2 A CLASS OF SECOND DERIVATIVE FORMULAS

Let us consider the following class of k -step, k th order second derivative formulas:

$$\sum_{j=0}^k \alpha_j y_{n-j+1} = h \sum_{j=0}^k \beta_j y_{n-j+1}' + h^2 \sum_{j=0}^k \gamma_j y_{n-j+1}''.$$

When applied to the linear problem, $y' = \lambda y$, the associated characteristic polynomial is

$$\psi_k(h\lambda, r) = \sum_{j=0}^k (\alpha_j - h\lambda \beta_j - h^2 \lambda^2 \gamma_j) r^j.$$

The similarity between $P_k(h\lambda, r)$ and $\psi_k(h\lambda, r)$ suggests that we may be able to find two-stage, cyclic formulas with the same $A(\alpha)$ -stability properties as second derivative formulas. Because we desire that the cyclic method exhibit properties (i), (ii) and (iv), we restrict our attention to the class of second derivative formulas of the form

$$y_{n+1} = y_n - h \sum_{j=0}^k \beta_{k-j} y_{n-j+1}' + h^2 \gamma_k y_{n+1}' \quad (5.2.1)$$

where $\gamma_k = -(\beta_k/2)^2$. This last condition is necessary because the leading polynomial coefficient of the stability polynomial associated with the second derivative formula must have real roots if we are going to find a CLMF with matching stability polynomial and which has real, as opposed to complex, coefficients.

The requirement that the formula be of order k imposes k conditions on the β_j , $j = 0, \dots, k$ so that the formula (5.2.1) has one degree of freedom. The class of formulas (5.2.1) was considered by Enright(1974)

Order	BDF	STINT	New Formulas
2	90.00	90.00	90.00 ($\beta_2 = 0.575$)
3	86.03	89.43	90.00 ($\beta_3 = 1.72$)
4	73.35	80.86	90.00 ($\beta_4 = 1.72$)
5	51.84	77.48	86.64 ($\beta_5 = 1.59$)
6	17.84	63.25	76.32 ($\beta_6 = 1.40$)
7		33.53	57.66 ($\beta_7 = 1.275$)
8			22.15 ($\beta_8 = 1.18$)

Table 5.1: A comparison of α values

who was motivated by the fact that the Newton iteration matrix used to solve (5.2.1) could be expressed as a perfect square. Enright chose to use the one degree of freedom to raise the order of the formula to $k+1$. Alternatively it is possible to use this degree of freedom to improve the $A(\alpha)$ -stability properties of the formulas and we determined the formula of order k which maximized the $A(\alpha)$ -stability angle α . The angle α for these formulas appears in Table 5.1 together with the values of the leading coefficient, β_k (calculated to three significant figures). In cases where there is more than one k^{th} order formula with the stability angle listed in Table 5.1, the smallest value of β_k was determined. For a comparison we give the values of α for the BDFs and STINT formulas in Table 5.1. It may be observed that the new formulas have far better $A(\alpha)$ -stability properties than either the BDFs or STINT formulas.

In the following section we will derive two-stage, CLMFs with the same $A(\alpha)$ -stability properties as the new second derivative formulas. Unlike a second derivative method, a code based on CLMFs will not require a user supplied analytic Jacobian.

5.3 NEW FORMULAS

A two-stage CLMF for which the component LMFs are k -step contains $4k+4$ free parameters. Two of these are scaling factors. The requirement that each formula be of order k accounts for $2k+2$ parameters. The requirement that $k-1$ roots at the origin be zero, that k roots at infinity be zero and that the two LMFs have the same leading coefficients account for the remaining $2k$ free parameters. The conditions (i), (ii) and (iv) together with the requirement that the component LMFs be of order k uniquely define a class of CLMFs. Unfortunately, it turns out that these cyclic formulas have inferior $A(\alpha)$ -stability angles than the BDFs of the corresponding order for orders greater than two.

To derive CLMFs with the same $A(\alpha)$ -stability properties as the k -step, k^{th} order second derivative formulas described in the previous section, it is necessary to introduce another free parameter.

Consider the cyclic use of a k -step, k^{th} order LMF with a $(k+1)$ -step, k^{th} order LMF as illustrated in Figure 5.1. Although the $(k+1)$ -step formula requires $k+1$ past values, one of these values is calculated within the cycle, so the cycle as a whole only requires k past values. The characteristic polynomial of the $(k+1)$ -step, k -step cycle may be obtained from (5.1.2) by setting $\alpha_0^{(1)} = \beta_0^{(1)} = 0$.

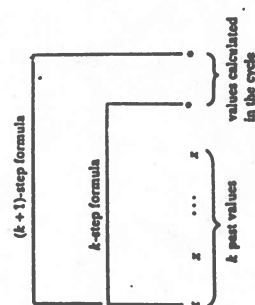


Figure 5.1: Two-stage cyclic formulas

To illustrate the derivation of the new class of cyclic formulas, let us consider the 2-step/3-step CLMF. The characteristic polynomial

for this method is given by $P_3(h\lambda, r)/r$ where $\alpha_0^{(1)} = \beta_0^{(1)} = 0$ and $P_3(h\lambda, r)$ is defined by (5.1.1). There are 14 coefficients to be determined: $\alpha_1^{(1)}, \beta_1^{(1)}, \alpha_1^{(2)}, \beta_1^{(2)}, \alpha_2^{(1)}, \beta_2^{(1)}, \alpha_2^{(2)}, \beta_2^{(2)}, \alpha_3^{(1)}, \beta_3^{(1)}, \alpha_3^{(2)}, \beta_3^{(2)}$. We have

$$\alpha_3^{(1)} = 1 \quad (5.3.1a)$$

$$\alpha_3^{(2)} = 1 \quad (5.3.1b)$$

$$\beta_3^{(1)} = \beta_3^{(2)} \quad (5.3.1c)$$

$$\alpha_1^{(1)} + \alpha_2^{(1)} + \alpha_3^{(1)} = 0 \quad (5.3.1d)$$

$$\alpha_1^{(1)} + 2\alpha_2^{(1)} + 3\alpha_3^{(1)} - \beta_1^{(1)} - \beta_2^{(1)} - \beta_3^{(1)} = 0 \quad (5.3.1e)$$

$$1/2 (\alpha_1^{(1)} + 4\alpha_2^{(1)} + 9\alpha_3^{(1)}) - \beta_1^{(1)} - 2\beta_2^{(1)} - 3\beta_3^{(1)} = 0 \quad (5.3.1f)$$

$$\alpha_0^{(2)} + \alpha_1^{(2)} + \alpha_2^{(2)} + \alpha_3^{(2)} = 0 \quad (5.3.1g)$$

$$\alpha_1^{(2)} + 2\alpha_2^{(2)} + 3\alpha_3^{(2)} - \beta_1^{(2)} - \beta_2^{(2)} - \beta_3^{(2)} = 0 \quad (5.3.1h)$$

$$1/2 (\alpha_1^{(2)} + 4\alpha_2^{(2)} + 9\alpha_3^{(2)}) - \beta_1^{(2)} - 2\beta_2^{(2)} - 3\beta_3^{(2)} = 0 \quad (5.3.1i)$$

$$\alpha_2^{(1)}\alpha_0^{(2)} - \alpha_1^{(1)}\alpha_1^{(2)} = 0 \quad (5.3.1j)$$

$$-\beta_3^{(1)}\beta_1^{(2)} + \beta_2^{(1)}\beta_2^{(2)} - \beta_1^{(1)}\beta_3^{(2)} = 0 \quad (5.3.1k)$$

$$\beta_2^{(1)}\beta_0^{(2)} - \beta_1^{(1)}\beta_1^{(2)} = 0 \quad (5.3.1l)$$

$$\beta_3^{(1)} = \beta_3^* \quad (5.3.1m)$$

Equations (5.3.1a) and (5.3.1b) result from our choice of scaling factor, (5.3.1c) is the requirement that the leading coefficients within the cycle do not vary, (5.3.1d)-(5.3.1i) are the conditions that the component LMFs are second order, (5.3.1j)-(5.3.1l) are the conditions

that the extraneous root at the origin as well as the roots at infinity are zero and (5.3.1m) is the condition that the cyclic method have the same characteristic polynomial as the second order second derivative formula (5.2.1) with the stability angle given in Table 5.1. (The second derivative method has characteristic polynomial $\psi_k(h\lambda, r)$ and takes one step of length h . The two-stage CLMF takes two steps of length h and is derived to have characteristic polynomial $\psi_k(2h\lambda, r)$). The value of $\beta^* = \beta_2$ is also given in Table 5.1.

There are 14 coefficients to be determined and (5.3.1a-m) consists of 13 equations. It is possible to parameterize the solutions to the non-linear equations (5.3.1a-m) in the following form.

Stage 1 coefficients

$$\alpha_3^{(1)} = 1 \quad \beta_3^{(1)} = \beta^*$$

$$\alpha_2^{(1)} = \alpha_2^{(1)} \quad \beta_2^{(1)} = \beta_2^{(1)}$$

$$\alpha_1^{(1)} = \alpha_1^{(1)} \quad \beta_1^{(1)} = \beta_1^{(1)}$$

Stage 2 coefficients

$$\alpha_3^{(2)} = 1 \quad \beta_3^{(2)} = \beta^*$$

$$\alpha_2^{(2)} = \alpha_2^{(2)} + s \quad \beta_2^{(2)} = \beta_2^{(2)} + s\beta^*$$

$$\alpha_1^{(2)} = s\alpha_2^{(1)} \quad \beta_1^{(2)} = \beta_1^{(2)} + s\beta_2^{(1)}$$

$$\alpha_0^{(2)} = s\alpha_1^{(1)} \quad \beta_0^{(2)} = \beta_0^{(2)} + s\beta_1^{(1)}$$

Here the $\alpha_j^{(i)}$ and the $\beta_j^{(i)}$ are constants and s is a free parameter. Note that the free parameter affects only the coefficients of the second stage. Now, the first stage is

$$\begin{aligned}
y_{2n+3} + \bar{\alpha}_2^{(1)} y_{2n+2} + \bar{\alpha}_1^{(1)} y_{2n+1} \\
= h(\beta_2^* f_{2n+3} + \beta_2^{(1)} f_{2n+2} + \beta_1^{(1)} f_{2n+1}) \\
- h(\beta_3^{(2)} f_{2n+4} + \beta_2^{(2)} f_{2n+3} + \beta_1^{(2)} f_{2n+2} + \beta_0^{(2)} f_{2n+1})
\end{aligned}$$

while the second stage is

$$\begin{aligned}
y_{2n+4} + \alpha_2^{(2)} y_{2n+3} + \alpha_1^{(2)} y_{2n+2} + \alpha_0^{(2)} y_{2n+1} \\
= h(\beta_3^{(2)} f_{2n+4} + \beta_2^{(2)} f_{2n+3} + \beta_1^{(2)} f_{2n+2} + \beta_0^{(2)} f_{2n+1})
\end{aligned}$$

or

$$\begin{aligned}
y_{2n+4} + \bar{\alpha}_2^{(2)} y_{2n+3} \\
= h(\beta_3^* f_{2n+4} + \beta_2^{(2)} f_{2n+3} + \beta_1^{(2)} f_{2n+2} + \beta_0^{(2)} f_{2n+1}) \\
- s [y_{2n+3} + \bar{\alpha}_2^{(1)} y_{2n+2} + \bar{\alpha}_1^{(1)} y_{2n+1} \\
- h(\beta_3^* f_{2n+3} + \beta_2^{(1)} f_{2n+2} + \beta_1^{(1)} f_{2n+1})].
\end{aligned}$$

But if the first stage is solved exactly then the term in square brackets is zero and the solution y_{2n+4} calculated from the second stage is invariant to the free parameter, s . Similarly, although the parameter s affects the error constant, $c_q^{(2)}$, of the second stage, the term $c_{q+1}^{(2)} - \alpha_{k-1}^{(2)} c_{q+1}^{(1)}$ is invariant to s so that the vector

$$\bar{c} = [0, 0, \dots, c_{q+1}^{(1)}, c_{q+1}^{(2)} - \alpha_{k-1}^{(2)} c_{q+1}^{(1)}]^T$$

appearing in the expression for the local discretization error does not depend on s . Of course, the characteristic polynomial is also invariant with respect to s .

The higher order cases can be handled similarly and for orders $q = 2, 3, \dots, 8$ the coefficients of the q -step/($q+1$)-step two-stage CLMFs were found by solving sets of non-linear equations such as (5.3.1a-m). The non-linear equations can generally be reduced to a system of $k-1$ simultaneous equations for the k -step/($k+1$)-step case. Thus in the 2-step/3-step case we are able to find the following analytic solution for the non-linear equations.

Given β^* , choose $\beta_2^{(1)}$ and $\beta_0^{(2)}$ to satisfy

$$\begin{aligned}
(\beta_2^{(1)})^2 (-8\beta^* + 4) + \beta_2^{(1)} (-25(\beta^*)^2 + 24\beta^* - 5) \\
+ (-18(\beta^*)^3 + 24(\beta^*)^2 - 8\beta^*) = 0
\end{aligned} \quad (5.3.2)$$

$$\text{and } \beta_0^{(2)} = \frac{(3\beta^* + \beta_2^{(1)} - 2)(\beta^* - 1/2)}{3\beta_2^{(1)} + 6\beta^* - 4}.$$

The first stage coefficients are then given by

$$\begin{aligned}
\alpha_0^{(1)} &= 0 & \beta_0^{(1)} &= 0 \\
\alpha_1^{(1)} &= 3 - 4\beta^* - 2\beta_2^{(1)} & \beta_1^{(1)} &= 3\beta^* + \beta_2^{(1)} - 2 \\
\alpha_2^{(1)} &= 4\beta^* + 2\beta_2^{(1)} - 4 & \beta_2^{(1)} &= \beta_2^{(2)} \\
\alpha_3^{(1)} &= 1 & \beta_3^{(1)} &= \beta^*
\end{aligned}$$

while the second stage coefficients are given by

$$\begin{aligned}
\alpha_0^{(2)} &= \alpha_1^{(1)} & \beta_0^{(2)} &= \beta_0^{(2)} + s\beta_1^{(1)} \\
\alpha_1^{(2)} &= \alpha_2^{(1)} & \beta_1^{(2)} &= -1/2 + \beta^* - 2\beta_0^{(2)} + s\beta_2^{(1)} \\
\alpha_2^{(2)} &= -1 + s & \beta_2^{(2)} &= 3/2 - 2\beta^* + \beta_0^{(2)} + s\beta_3^{(1)} \\
\alpha_3^{(2)} &= 1 & \beta_3^{(2)} &= \beta^*
\end{aligned}$$

(5.3.2) will always be solvable and if

$$49(\beta^*)^4 - 144(\beta^*)^3 + 186(\beta^*)^2 - 112\beta^* + 25 \geq 0$$

the system will have real solutions. There are no real solutions to the non-linear system (5.3.1) for $\beta^* \in (0.603\ 417\ 396, 0.660\ 120\ 093)$ while for β^* not in this range (including the endpoints) there are always two distinct real solutions.

We were unable to find analytic solutions for the non-linear systems for the k -step/ $(k+1)$ -step case where $k \geq 2$, so in these cases the non-linear equations were solved numerically. For some values of the free parameter (other than those given in Table 5.1), we were unable to solve the non-linear system, despite having good starting conditions. We conjecture that for these values of the free parameter, the non-linear systems may not have real solutions. Where the non-linear systems had more than one real solution, the solution with the smaller local discretization error coefficients was chosen.

The resulting coefficients, to 16 significant figures, are listed in Appendix A. All computations were done on a Burroughs 6700 in Double Precision. The non-linear equations were solved using the IMSL routine ZSYSTM which is an implementation of Brown's algorithm.

As previously mentioned, Enright(1974) derived a different set of second derivative formulas of form (5.2.1) for which β_k was chosen to raise the order of the formula to $k+1$ rather than to maximize the $A(\alpha)$ -stability angle. It is interesting to note that it is possible to derive k -step/ $(k+1)$ -step CLMFs of convergent order $k+1$ with the analogous characteristic polynomial as Enright's formulas if condition (5.3.1m) is replaced by condition (ii) of the lemma (the condition for higher order convergence). In particular, there is a 3-step/4-step method whose constituent formulas are third order, but which is globally convergent to fourth order and has an $A(\alpha)$ -stability angle of 89.99° . We will discuss these formulas further in Chapter 7.

There are other approaches to deriving two-stage CLMFs with stability polynomials analogous to the second derivative formulas. A different approach to the one we have presented is based on cyclic methods whose constituent formulas are each k -step LMFs of order $k-1$. Condition

(ii) of the lemma is then used to ensure that the overall method is k th order.

5.4 THE CLASS OF CLMFs CONSIDERED BY TENDLER, BICKART AND PICEL

In this chapter we have managed to find CLMFs with better $A(\alpha)$ -stability angles than the CLMFs found by Tendler, Bickart and Picel(1978) known as the STINT formulas. The CLMFs we derived require only two stages to satisfy conditions (i) to (iv) while the STINT formulas require at least three stages. We now show why Tendler, Bickart and Picel were forced to consider three-stage CLMFs.

Tendler, Bickart and Picel(1978) in deriving the STINT formulas required that the i th stage should be a k -step LMF with $\beta_j^{(i)} = 0$ for $j = 0, 1, \dots, k-i$. Thus the CLMF formula would only use derivative values from points computed in the current cycle. This property can be seen as generalizing a property of the BDF formulas and is sufficient to guarantee that all extraneous roots at infinity are zero. For convenience we term CLMFs which do not use derivative values from past points as TBP CLMFs.

We may use the third order STINT formula as an example. The third order STINT formula consists of the following LMFs

$$11\ y_{n+1} - 18\ y_n + 9\ y_{n-1} - 2\ y_{n-2} = 6\ h f_{n+1}, \quad (5.4.1a)$$

$$11\ y_{n+2} - 18\ y_{n+1} + 9\ y_n - 2\ y_{n-1} = 6\ h f_{n+2}, \quad (5.4.1b)$$

$$3\ y_{n+3} - 12\ y_{n+2} + 9\ y_{n+1} = 2\ h f_{n+3} - 4\ h f_{n+2} - 4\ h f_{n+1}. \quad (5.4.1c)$$

Thus the CLMF does not use the derivative values from any of the past points x_{n-2} , x_{n-1} and x_n .

For TBP cyclic formulas the first stage must be a BDF-type LMF, (all beta coefficients except that at the new point x_{n+1} are zero). The second stage can only use the derivative values from x_{n+1} and x_{n+2} .

However, we have already shown that the properties of the CLMF are unchanged if we add a multiple of the first stage to the second stage. Therefore, by adding a suitable multiple of the first stage to the second stage we can find a CLMF with equivalent properties to the first cyclic formula and where both the first and second stages are BDF-type formulas. This process can then be extended to the third and later stages. Thus, the class of CLMFs considered by Tendler, Bickart and Picel(1978) is essentially the class of k^{th} order, M -stage CLMFs where the first stage is the k^{th} order, k -step BDF, the second stage is a k^{th} order, $(k+1)$ -step BDF-type formula and the i^{th} stage, $1 \leq i \leq M$ is a k^{th} order, $(k+i-1)$ -step BDF-type formula.

A k^{th} order, $(k+1)$ -step BDF-type formula involves 1 degrees of freedom and can be expressed as a linear combination of the k^{th} order BDF and the k^{th} order extrapolation involving only y values in the following way,

$$\{ k^{\text{th}} \text{ order BDF} \} + \sum_{j=1}^1 \gamma_j \nabla^{k+1} y_{n+1-j} = 0,$$

where $\nabla^{k+1} y_{n+1-j}$ is the $(k+1)^{\text{st}}$ order backward difference of the y values based at y_{n+1-j} and the $\gamma_j, j=1, \dots, 1$ are constants.

Thus if we consider the third order TINT formula (5.4.1a-c) we find that this is equivalent to the CLMF with stages (5.4.1a), (5.4.1b) and whose third stage is $(5.4.1c) + 2/3 (5.4.1b) + 2/3 (5.4.1a)$. This third stage is a third order, five-step BDF-type formula of the form

$$3 y_{n+3} - 14/3 y_{n+2} + 13/3 y_{n+1} - 6 y_n + 14/3 y_{n-1} - 4/3 y_{n-2} = 2 h f_{n+3}$$

which we can also express as

$$11 y_{n+3} - 18 y_{n+2} + 9 y_{n+1} - 2 y_n - 6 h f_{n+3} - 2 \nabla^4 y_{n+3} - 4 \nabla^4 y_{n+2} = 0.$$

If we consider two-stage, TBP CLMFs which are effectively k^{th} order and k -step then we really only have one degree of freedom at our disposal. If we use this free parameter to ensure that the second stage has the same leading coefficient as the first stage, then the second stage

will be the k^{th} order, k -step BDF and is therefore identical to the first stage. Therefore, to find TBP cyclic formulas which improve on the BDFs with respect to stability properties and yet retain the computationally desirable property that all stages share the same leading coefficient we are forced to consider CLMFs with at least three stages.

CHAPTER 6

AN IMPLEMENTATION OF CYCLIC LINEAR MULTISTEP FORMULAS

In the previous chapter we derived a class of two-stage CLMFs with a free parameter and derived a family of formulas by choosing the free parameter to maximize the $A(\alpha)$ -stability angle. This family contains formulas of order two to eight which for order greater than three have significantly larger $A(\alpha)$ -stability angles than the BDF of the same order. In this chapter we discuss an implementation in a code called ODIUS of the family of formulas derived in the previous chapter. Since this family of formulas does not include a first order formula the first order BDF has been used in ODIUS. For brevity, the formulas used in ODIUS will be referred to as the ODIUS formulas. The performance of ODIUS has been evaluated using the stiff ODE solver testing program STIFF DETEST due to Bedet, Enright and Hull(1975) and described by Enright(1979). Results of this performance evaluation are presented and compared with the performance of a BDF-code LSODE (Hindmarsh, 1980).

6.1 IMPLEMENTATION DETAILS

ODIUS implements a set of CLMFs where all formulas have two stages and the two constituent LMFs for a given CLMF have the same leading coefficient. The code assumes that the two-stage, k^{th} order formula will have stages that are k -step and $(k+1)$ -step.

CLMFs are a subset of the class of composite multistep formulas. For a composite multistep formula a set of M LMFs is used to compute the approximate solution values simultaneously at a number of points. With an M -stage CLMF, a set of M LMFs are applied in a given order to compute the approximate solution sequentially at $x_{n+1}, x_{n+2}, \dots, x_{n+M}$. The overall implementation approach adopted by ODIUS is to treat CLMFs more in the manner of composite linear multistep methods which calculate a

number of solution values at a time rather than to implement CLMFs in a LMF based code which, for a given order, happens to use a number of possibly different LMFs in a prescribed sequence.

ODIUS attempts to calculate the solution values at two new points. If values at one of these new points fail to meet the accuracy requirements or if at one of these points a convergence failure necessitates a stepsize reduction, then the current cycle is rejected and a new cycle is attempted. This approach was adopted because most of the theory concerning CLMFs has been based on the assumption that complete cycles of constant sized steps have been taken. It is possible, however, to follow the second approach and allow stepsize changing within the cycle in the event of a rejected stage within a cycle.

ODIUS consists of four subroutines written in FORTRAN 77. SHOFER is a driver routine which calls a routine CYCLE which actually tries to calculate the next cycle of values. CYCLE is called each time a new cycle is to be attempted and calls two LINPACK subroutines DGEFA and DGEEL to handle linear systems of equations and calls user provided subroutines F and JACOBI. F calculates the derivatives of the solution at a given point and JACOBI calculates the Jacobian of the differential system. Further documentation on ODIUS is available in Fischer(1983).

6.1.1 Representation

The code uses a fixed-coefficient implementation of the formulas. A backward difference representation is used and backward differences of both the computed solution (y -values) and the derivative (f -values) are maintained. If the order of the current formula is denoted by p there are at least $p+2$ backward differences available at the time when the order of formula is selected. As each stage in the cycle is accepted the number of differences maintained is increased by one so that if the cycle is rejected the differences as they were at the start of the cycle can be recovered. Thus for the two-stage CLMF the number of differences can be as high as $p+4$ for a p^{th} order formula.

The choice of a backward difference representation was prompted by the fact that all the formulas used in ODIOUS could be conveniently represented in terms of linear combinations of backward differences. For example, ODIOUS stores no coefficients for predictor formulas as it constructs predictor formula coefficients from the corrector formula coefficients. A Nordsieck representation could have been chosen but this possibility was not pursued as a conventional Nordsieck representation assumes that the formula has an order which is at least as great as its stepnumber, i.e. a k -step formula is at least k^{th} order. In the case of the ODIOUS formulas, the second stage is generally a $(k+1)$ -step, k^{th} order formula and its representation in Nordsieck form would require a generalized Nordsieck representation.

6.1.2 Order/Stepsize Selection

In selecting order and stepsize it is assumed that the local discretization error introduced in computing the solution at a point within the cycle by using a p^{th} order formula will be

$$c_{p+1} h^{p+1} y^{(p+1)}(x_{n+2}), \quad c_{p+1} \text{ a constant.}$$

For the ODIOUS formulas, the error constant used is the max norm of the local discretization error vector since this is similar to the approach used in defining the error constant for one-stage formulas such as LMFs.

After a successful cycle using a p^{th} order formula ODIOUS considers using the $(p-1)^{\text{st}}$ order formula, if p is not already equal to one, the p^{th} order formula and the $(p+1)^{\text{st}}$ order formula, if p is not already the maximum order. The p^{th} order backward difference of the f -values is used to estimate the dominant higher derivative term in the local discretization error expansion and for each order $p-1$, p and $p+1$ where this is appropriate a candidate stepsize rh is chosen subject to the condition that r is the largest positive value which satisfies the expression

$$\left| \left| r^{p+1} K_{p+1} h \nabla_f^p \right| \right| \leq (\text{safety factor}) * \text{tolerance.}$$

The safety factor used is $(0.9)(3/4) ** (1/(p+1))$ and ranges in value from 0.7794 for the first order formula to 0.8717 for the eighth order formula.

The following heuristics are also used in order and step size selection:

- (1) The current order is the order of the last successfully completed cycle. If the current step number is k , stepsize and order selection is considered only after at least $k+1$ steps at constant stepsize have been taken with the current formula.
- (2) If a stepsize increase using the current order is possible and the last stepsize change was a reduction due to an error failure, the current order is chosen and the stepsize is not increased.
- (3) If the stepsize can be increased by no more than 10% the order and stepsize are not changed. Otherwise, the order which corresponds to the largest candidate stepsize is chosen and the corresponding stepsize is used. If the stepsize and order are not changed then at the completion of the next cycle the code will again consider changing the stepsize and order.
- (4) The candidate stepsizes are constrained to be no greater than a user supplied value h_{MAX} and stepsize increases are restrained in a manner which depends on the number of steps, N , taken using the current formula at a constant stepsize, the number of backward differences available, L , and the number of past points, P , used by the formula. This expression for the greatest increase in the stepsize at a given order P is $\min(N, L-1)/(P-1)$. This restraint ensures that interpolated past points lie in the interval spanned by the past points calculated by the current formula. An arbitrary factor of 10 is used if we are considering a one-step formula. Since the limit on the size of stepsize increases depends on the number of constant sized steps taken using the last successful formula, this limit varies and ranges for this limit on the maximum

The stepsize and order selection strategy guarantees that when the last cycle to compute a block of at least $k+1$ constant steps is attempted all the past values will have been computed by the current CMF (with the exception of values at the point after which the stepsize changed) and this should ensure that errors introduced by the stepsize change get damped out. Many fixed-coefficient implementations of multistep formulas adopt this approach. In the case of the Adams formulas, Gear and Tu(1974) have proven that the interpolation technique with the k -step Adams formula is stable if the step selection scheme guarantees that the step is held constant for k steps after a stepsize change. Gear and Tu(1974) also proved that the interpolation technique is stable if the underlying fixed-step formula is stable and if the rate of change of the stepsize is small.

ODIOUS gets started using a first order formula. To compute an initial stepsize it uses the following estimate of the second derivative at x_0 :

$$y^{(2)}(x_0) = \frac{f(x_0 + \delta, y_0 + \delta f_0) - f(x_0, y_0)}{\delta}, \quad \delta = 10^{-4}.$$

6.1.3 Stage Error Test

At each stage in the cycle the k th order backward difference of the f -values is computed from the backward differences at the last point and the current value of f_{n+1} without updating the backward difference representation. The following identity is used

$$\nabla_{f_{n+1}}^i = f_{n+1} - \sum_{r=0}^{i-1} \nabla_{f_n}^r,$$

$\nabla_{f_{n+1}}^i$ being the i th order backward difference of f_{n+1} . This identity is also used in evaluating various predictor formulas. The product of this with the local discretization error coefficient for that stage is then computed and this product is required to be below the user error tolerance in magnitude.

stepsize increase are given in the Table 6.1. Since the increase in stepsize allowed at lower orders is larger the code is biased towards using the lower order formulas. Most stiff problems have regions where the problem is effectively non-stiff and in solving such problems, the stepsize can be increased significantly when the integration passes from a non-stiff region to a stiff region. The bias toward the lower order formulas is intended to cause the order to be lowered in such a situation. Using the lower order formula we can still increase the stepsize considerably and more frequently while at the same time we are less likely to encounter stability difficulties as the lower order formulas are known to be more stable.

The stepsize is changed using the interpolation technique. An interpolatory polynomial of degree $p+1$ is used, p being the order of the cycle just completed. After a stepsize change caused by an error failure for the p th order formula, an interpolatory polynomial of degree p is used.

Present Order p	Order at the next stage		
	down $p-1$	along p	up $p+1$
1	*	10.00	2.00 - 4.00
2	10.00	4.00 - 5.00	2.00 - 2.50
3	4.00 - 6.00	2.00 - 3.00	1.33 - 2.00
4	3.00 - 3.50	2.00 - 2.35	1.50 - 1.75
5	2.00 - 2.67	1.50 - 2.00	1.20 - 1.60
6	2.00 - 2.25	1.60 - 1.80	1.33 - 1.50
7	1.60 - 2.00	1.33 - 1.67	1.14 - 1.43
8	1.67 - 1.83	1.43 - 1.57	1.25 - 1.38

Table 6.1: Restraint on Stepsize Increases

Sacks-Davis(1977) has shown that the above test can overestimate the actual error when stiff problems are being solved and suggested a method of estimating the error which does not suffer from this drawback. Since this test is more expensive than the first test in that it requires the solution of a set of linear equations, the second error test, based on the Sacks-Davis error estimator, is used only when a stage has failed the first test. This second test uses

$$c_{i,p} = \left\| \frac{1}{h_k} W^{-1} \nabla^k f_{n+1} \right\|.$$

as its estimate of the local discretization error. W is the scaled Newton iteration matrix and $c_{i,p}$ is the element of the local discretization error vector corresponding to x_{n+1} . This test is expected to be a more accurate estimator with regard to stiff components. The test, however, involves the solution of a set of N linear equations, N being the number of equations in the differential system, and its accuracy is dependent on the accuracy of the iteration matrix. If a stage fails both these error tests, the cycle is rejected.

On passing the stage error test, the backward difference representation is updated to include the values from the current point.

6.1.4 Recovery from Error Failure

On an error failure, candidate stepsizes are chosen for the current order and the next order down, if the current order is not already equal to one, using the same approach as outlined in the stepsize selection section. More conservative safety factors are used and the appropriate backward differences from the stage which failed the error test are used.

The safety factor used is $(0.88)(0.7)^{**} (1/(p+1))$ and ranges in value from 0.7363 for the first order formula to 0.8458 for the eighth order formula.

At the start of the cycle the backward difference representation is based at the current point x_n . If in attempting a two-stage cycle the first stage is accepted and the second stage rejected, then the recovery from the error failure is based on the backward difference representation at the point x_{n+1} . In this case, after the stepsize and order for the new cycle have been selected the cycle will be attempted again with a new stepsize and a new solution will be calculated at a different x_{n+1} . For this reason the backward difference representation is backed out to the last point of the last accepted cycle, i.e. x_n .

6.1.5 Corrector Iterations

To advance the solution using an implicit LMF, it is necessary to solve a set of equations of the form (6.1.5.1) at each stage.

$$y_{n+1} - h_k f(x_{n+1}, y_{n+1}) = \psi, \quad \psi_k = 0, \quad (6.1.5.1)$$

ψ is a constant term which groups together the contributions from past values of y_i and f_i . (6.1.5.1) is generally a set of non-linear equations which must be solved iteratively by generating a sequence of approximate solutions $y_{n+1}^{(0)}, y_{n+1}^{(1)}, \dots, y_{n+1}^{(s)}$. The solution of (6.1.5.1) involves the following steps:

- Prediction i.e. generating a starting value $y_{n+1}^{(0)}$ for the sequence of approximate solutions;
- Corrector iterations i.e. improving the current approximate solution;
- Convergence testing i.e. deciding when an approximate solution is accurate enough.

Generally, an iteration of the form (6.1.5.2) is used to solve the implicit equations (6.1.5.1)

$$(I - h_k J') (y_{n+1}^{(i)} - y_{n+1}^{(i-1)}) = -y_{n+1}^{(i-1)} + h_k f(x_{n+1}, y_{n+1}^{(i-1)}) + \psi \quad (6.1.5.2)$$

where J is some approximation to the Jacobian of the differential system. Most stiff codes assume that the Jacobian is only varying slowly and that the same Jacobian approximation can be used not only for a number of iterations at one step out also for a number of steps. This is called Simplified Newton (SN) Iteration. If the iteration matrix has a leading coefficient $h'g_k$ which is different from that in (6.1.5.2), this is known as Modified Newton (MN) Iteration.

6.1.5.1 Choice of Predictor

In solving the non-linear equations associated with implicit formulas we need to use a predictor formula to generate starting values. The objective of using a predictor is to compute a good approximation to the solution of the non-linear equations (6.1.5.1), the corrected value. When talking about the accuracy of a predictor, we are really interested in how closely the predicted value agrees with the corrected value and we are less concerned with how closely the predicted value agrees with the true solution of the differential equation. Thus, in assessing the merits of various predictor formulas, we will consider the prediction error i.e. the amount by which the corrected and predicted values differ. However, in deciding what we mean by a small prediction error, we will need to consider the prediction error in both stiff and non-stiff components and also consider the nature of the iterative method. For instance, if we are using an iterative method which highly damps errors in non-stiff components but not in stiff components, then we will be interested in using a predictor formula which accurately predicts the stiff components.

Let us therefore consider the difference in the predicted and corrected values where the corrector is a k -step, q th order LMF with coefficients $\alpha_i, \beta_i, i = 0, 1, \dots, k, \alpha_k = 1, \beta_k = 0$ and the predictor is a k -step, r th order explicit LMF with coefficients $\alpha_i^*, \beta_i^*, i = 0, 1, \dots, k, \alpha_k^* = 1, \beta_k^* = 0$. With the predicted value denoted by P_{n+1} and the corrected value by Y_{n+1} we have

$$Y_{n+1} - P_{n+1} = \sum_{i=0}^k [(\alpha_i - \alpha_i^*)y_{n+1-k+1} - h(\beta_i - \beta_i^*)f_{n+1-k+1}^*].$$

If we consider the LMF with coefficients $\bar{\alpha}_i = \alpha_i - \alpha_i^*, i = 0, 1, \dots, k$ and $\bar{\beta}_i = \beta_i - \beta_i^*$, it will have order $s \geq \min(r, q)$ since the order conditions for LMFs are linear with respect to the alpha and beta coefficients. For convenience, we term this formula the difference LMF.

Let us denote the global errors at past points by

$$e_{n-1} = y(x_{n-1}) - Y_{n-1}, \quad i = -1, 0, 1, \dots, k-1.$$

Now we have

$$f(x_{n-1}, y(x_{n-1})) - f_{n-1} = J(x_{n-1}, y_{n-1}^*)e_{n-1}, \quad i = -1, 0, 1, \dots, k-1,$$

where $J(x_{n-1}, y_{n-1}^*)$ is the Jacobian of the differential system evaluated somewhere on the line between y_{n-1} and $y(x_{n-1})$ and $f_{n-1} = f(x_{n-1}, y_{n-1})$. The prediction error is then

$$\begin{aligned} Y_{n+1} - P_{n+1} &= \sum_{i=0}^k [\bar{\alpha}_i(y(x_{n+1-k+1}) - e_{n+1-k+1}) \\ &\quad - h\bar{\beta}_i(y'(x_{n+1-k+1}) - J(x_{n+1-k+1}, y_{n+1-k+1}^*)e_{n+1-k+1})] \\ Y_{n+1} - P_{n+1} &= \sum_{i=0}^k [\bar{\alpha}_i y(x_{n+1-k+1}) - h\bar{\beta}_i y'(x_{n+1-k+1})] \\ &\quad - \sum_{i=0}^k [\bar{\alpha}_i - h\bar{\beta}_i J(x_{n+1-k+1}, y_{n+1-k+1}^*)]e_{n+1-k+1}. \end{aligned} \quad (6.1.5.3)$$

For a sufficiently differentiable $y(x)$, the first summation term of (6.1.5.3) is $O(h^{s+1})$ because the difference LMF is of order s while the second summation term depends on the global error in past points. If the numerical solution calculated by the corrector is converging with order q then we have $e_{n-1} = O(h^q)$ and the second summation term is generally $O(h^q)$. However, if all the $\bar{\alpha}_i$ coefficients, $i = 0, 1, \dots, k$, are

zero then the second summation term is $O(h^{q+1})$. For the difference LMF to be q th order, the predictor must have the same order as the corrector i.e. $r = q$. Thus, the prediction error is minimized by requiring that the predictor and corrector have the same alpha coefficients and the same order.

Therefore, the explicit LMF with the same alpha coefficients and the same order as a given implicit LMF is often chosen as the predictor for that LMF. For the $k+1$ st order, k -step Adams-Moulton formula this results in the choice of the $k+1$ st order, $(k+1)$ -step Adams-Bashforth formula as the predictor. Codes which implement LMF using the Nordsieck representation, e.g. LSODE, generate starting values for the implicit equations by extrapolating the polynomial representation at the last computed point x_n . Descloix(1963) showed that when the Adams-Moulton formulas are being implemented using the Nordsieck representation, the predicted value satisfies the difference equation of an Adams-Bashforth formula.

Deriving an explicit LMF predictor that has the same alpha coefficients and the same order as a given implicit LMF corrector is quite straightforward. Since the predictor and the corrector have the same order, the difference LMF must have at least the same order as well. The requirements that the difference LMF have all alpha coefficients zero and that it be a given order does not define a unique difference LMF. If in addition to our requirements we want the predictor to be of minimal stepnumber, then we find that, if the corrector is of order q , the difference LMF must be $h\beta_k \nabla_k^q$, i.e. the q th order backward difference of the f values (the scaling factor is the leading coefficient β_k , $\bar{\beta}_k = \beta_k - \beta_k^* = \beta_k$ by the definitions of the predictor and the corrector).

Thus if we write the implicit LMF as

$$y_{n+1} = \psi + h\beta_k f_{n+1}^q$$

where ψ is a constant which groups together the terms from past points, we can conveniently rewrite the corrector as

$$y_{n+1} = \psi + h\beta_k (f_{n+1}^q - \nabla_k^q f_{n+1}^q) + h\beta_k \nabla_k^q f_{n+1}^q$$

and so the explicit LMF predictor with the same alpha coefficients and order is given by

$$y_{n+1} = \psi + h\beta_k (f_{n+1}^q - \nabla_k^q f_{n+1}^q).$$

We illustrate this by using the 2nd order BDF as an example. The corrector is

$$y_{n+1} = 4/3 y_n - 1/3 y_{n-1} + 2/3 h f_{n+1}^q.$$

We may rewrite this as

$$\begin{aligned} y_{n+1} &= (4/3 y_n - 1/3 y_{n-1}) + 2/3 h(2f_n - f_{n-1}) \\ &\quad + 2/3 h(f_{n+1} - 2f_n + f_{n-1}). \end{aligned}$$

Thus we obtain

$$p_{n+1} = 4/3 y_n - 1/3 y_{n-1} + 2/3 h(2f_n - f_{n-1})$$

as the second order explicit LMF predictor for the second order BDF that has the same alpha coefficients.

In (6.1.5.3) there are two components to the prediction error. If we make the difference LMF of order $q+1$ then the predictor and the corrector will agree in the first term of the local discretization error expansion and the first term in (6.1.5.3) will be $O(h^{q+2})$. The second summation term, which indicates the effect of errors in the past points, will still be $O(h^{q+1})$. For example, the second order predictor which has the same alpha coefficients and the same local discretization error coefficient as the second order BDF is given by

$$y_{n+1} = 4/3 y_n - 1/3 y_{n-1} + 2/3 h(f_{n+1}^q - \nabla_k^q f_{n+1}^q).$$

In discussing the prediction error expression (6.1.5.3) we assumed that $\|h_j(x_{n+1-k+1}, y_{n+1-k+1})e_{n+1-k+1}\|$ was small. This is generally true for non-stiff components but may not be for stiff components. Because the Jacobians of stiff differential systems have some eigenvalues with large modulus, we might expect that the derivative values contain large errors in the stiff components. Robertson and Williams(1975) have suggested that extrapolation of past solution values should be used for predicting stiff components accurately and they show how this can lead to a significant improvement in performance. Thus, for a c^{th} order LMF we would use the $(q-1)$ -step, q^{th} order extrapolation of the past y values as a predictor. That is we use a formula of the type

$$y_{n+1}^{q+1} = 0.$$

We expect the stiff components in the numerical solution to decay and to eventually be approximated by zero. Let us assume that the magnitude of errors in the stiff components have been kept below some accuracy requirement ϵ . Once the stiff components in the true solution have decayed to be effectively zero the prediction error in the stiff components using a $(q-1)$ -step extrapolation will be bounded in magnitude by $|(2^{q-1}-1)\epsilon|$. Assuming that errors in the non-stiff components at past points are $O(h^q)$ we obtain that the prediction error in the non-stiff components is $O(h^q)$.

If we use extrapolation from more than $k+1$ past points we have a free parameter which we can use to improve the prediction of either stiff or non-stiff components. To improve non-stiff prediction we can make the predictor have the same local discretization error as the corrector. To improve stiff component prediction we might consider extrapolation formulas of the type

$$y_{n+1}^{q+1} + \nabla y_n^{q+1} = 0.$$

For such formulas, the sum of the moduli of the coefficients at past points is $(q+1)q-1$, $q > 1$, which leads to a bound on the predicted error

in stiff components of $|(q+1)q-1|\epsilon$.

If a code chooses to perform Jacobian evaluation and iteration matrix factorizations frequently in order to have a quickly convergent iterative method then the choice of predictor is likely to have little effect on the overall cost of solving the non-linear equations. For a code that performs Jacobian evaluations and iteration matrix factorizations less frequently, the choice of predictor is likely to have more impact on the cost of solving the non-linear equations as will be discussed in the next section on corrector iterations.

The main advantage of using explicit LMFs as predictors is that explicit LMFs are inexpensive to evaluate. More expensive formulas exist which may be used as predictor formulas. In particular, linearly-implicit (LI) formulas require the solution of systems of linear equations but do not require the solution of non-linear equations and are therefore more expensive than explicit LMFs but less expensive than implicit LMFs. The formulas we will consider are a subset of the set of multistep formulas with variable matrix coefficients considered by Lambert and Sigurdsson(1972).

A class of LI predictors may be derived as follows. We have discussed earlier how we can find an explicit predictor which has the same order and alpha coefficients as a given implicit LMF. The prediction error for the explicit LMF is $h\beta_k \nabla_{f,n+1}^q$. Since in solving stiff ODEs we use some form of Newton iteration, an approximation j to the Jacobian matrix is available. It is therefore possible to improve the explicit LMF predictor by adding to the predicted value an amount $h\beta_k \nabla_{f,n+1}^q$ which approximates the prediction error $h\beta_k \nabla_{f,n+1}^q$ of the explicit LMF predictor. We thus obtain an LI predictor as follows. The corrector is

$$y_{n+1} = \psi + h\beta_k f_{n+1}.$$

The explicit LMF predictor is

$$p_{n+1} = \psi + h\beta_k (f_{n+1} - \nabla_{f,n+1}^q).$$

The LI predictor is therefore given by

$$y_{n+1}^p = \psi + h g_k(f_{n+1} - \nabla_f^q) + h g_k J \nabla_f^q y_{n+1}^p \quad (6.1.5.4)$$

$$\text{where } \nabla_f^q y_{n+1}^p = y_{n+1}^p + \nabla_f^q y_{n+1} - y_{n+1}.$$

Note that for linear problems of the form $y' = Jy$, $J \nabla_f^q y_{n+1} = \nabla_f^q y_{n+1}$ so that given an exact Jacobian approximation, the LI predictor predicts the LMF corrector exactly, and has the same $A(\alpha)$ -stability and A -stability properties as the implicit LMF. The prediction error, divided by $h g_k$, for this LI predictor is given by

$$\begin{aligned} \frac{1}{h g_k} (y_{n+1} - y_{n+1}^p) &= (\nabla_f^q y_{n+1}^p - J \nabla_f^q y_{n+1}^p) \\ &= \nabla_f^q y'(x_{n+1}) - J \nabla_f^q y(x_{n+1}) \\ &\quad + \sum_{i=0}^{q-1} (J(x_{n-1}, y_{n-1}^*) - J) \left\{ \frac{q}{i+1} \right\} (-1)^{i+1} e_{n-1} \\ &\quad + J(x_{n+1}, y_{n+1}^*) e_{n+1} - J e_{n+1}^p \end{aligned} \quad (6.1.5.5)$$

where e_{n+1}^p is $y_{n+1} - y_{n+1}^p$. We can show that in certain circumstances the LI predictor is less effected by global error in the past points than an explicit LMF predictor. Since the LI predictor has the same alpha coefficients and the same order as the corrector, e_{n+1}^p will be of the same order as e_{n+1} . The backward difference terms in $y(x_{n+1})$ and $y'(x_{n+1})$ in (6.1.5.5) will be $O(h^q)$ and if $J = J(x_{n+1-k+1}, y_{n+1-k+1}^*) + O(h)$, $i = -1, 0, 1, \dots, q-1$, then the global error terms on the right hand side of (6.1.5.5) will be $O(h^{q+1})$ and their effect on the prediction error will be $O(h^{q+2})$. For explicit predictors, the effect on the prediction error of the global errors in the past points will be

$O(h^{q+1})$.

In particular, consider the LI predictor of the form

$$y_{n+1}^p = \psi + h g_k(f_{n+1} - \nabla_f^{q+1}) + h g_k J \nabla_f^{q+1} y_{n+1}^p \quad (6.1.5.6)$$

Such an LI predictor will have a prediction error which is of higher order than that for the explicit LMF predictor.

In implementing LI predictors, we may write these formulas in a computationally more efficient form. Thus we would rewrite (6.1.5.6) as

$$y_{n+1}^p = y_{n+1} - \nabla_f^{q+1} - (I - h g_k J)^{-1} (y_{n+1} - \nabla_f^{q+1} - \psi - h g_k(f_{n+1} - \nabla_f^{q+1})).$$

Therefore, the major cost of using the LI predictor is the solution of one set of linear equations involving the Newton iteration matrix.

ODIOUS uses LI predictors because these predictors are accurate for non-stiff components and have the same $A(\alpha)$ -stability properties as the LMF correctors. They are also likely to predict stiff components accurately. However, LI predictors incorporate the Newton iteration matrix into the formula and their properties are effected by the matrix used. ODIOUS, therefore, only uses LI predictors if the product of the step-size and the formula's leading coefficient has not changed by more than 5%. If the product of the stepsize and the formula's leading coefficient has changed by more than 5%, ODIOUS uses the $(k+1)$ -step, k th order extrapolation of past computed solution values, since in such a situation ODIOUS uses the MN iteration. As will be discussed in the next section, the MN iteration damps out errors more slowly in the stiff components than in the non-stiff components and therefore extrapolation is chosen in an attempt to predict stiff components accurately.

When ODIOUS uses LI predictors the $(k+1)$ -step LI predictor of the form (6.1.5.6) is used if enough past points are available otherwise the k -step LI predictor of the form (6.1.5.4) is used.

6.1.5.2 Behaviour of Corrector Iterations

We now discuss the iterative solution of the implicit equation (6.1.5.1) using the method defined by (6.1.5.2). This discussion is similar to that given by Shampine(1980c) and we follow Shampine's approach in considering effects on both stiff and non-stiff components. We will consider the following iteration which is more general than (6.1.5.2),

$$(I - h'\beta_K'J)(y_{n+1}^{(i)} - y_{n+1}^{(i-1)}) = R(-y_{n+1}^{(i-1)} + h\beta_K'f(x_{n+1}, y_{n+1}^{(i-1)}) + \psi) \quad (6.1.5.7)$$

where $y_{n+1}^{(i)}$ denotes the i th approximation to the solution of (6.1.5.1), $y_{n+1}^{(0)}$ denotes the predicted value, R is a relaxation factor and J is a non-trivial approximation to the Jacobian. We also define the difference between y_{n+1}^* , the exact solution of (6.1.5.1) and the i th approximate solution as

$$e_{n+1}^{(i)} = y_{n+1}^* - y_{n+1}^{(i)}.$$

Note that for $R = 1$ and $h'\beta_K' = h\beta_K'$ (6.1.5.7) reduces to the SN iteration and the choice of $R = 1$ and $h'\beta_K' = h\beta_K'$ leads to the MN iteration. ODIUS only uses the SN iteration and the MN iteration. The latter is used when the product of the stepsize and the formula's leading coefficient has changed from the values used when the iteration matrix was decomposed. We will also briefly discuss other strategies which we considered implementing in ODIUS.

We first derive a simple expression for the error in the iteration by using the approach of Shampine(1980c) since the resulting expression gives us some insight into the error damping properties of the iteration (6.1.5.7). Using (6.1.5.1) to substitute for ψ in (6.1.5.7), we obtain

$$(I - h'\beta_K'J)(y_{n+1}^{(i)} - y_{n+1}^{(i-1)}) = R \{-y_{n+1}^{(i-1)} + y_{n+1}^* + h\beta_K'f(x_{n+1}, y_{n+1}^{(i-1)}) - h\beta_K'f(x_{n+1}, y_{n+1}^*)\},$$

$$= R(I - h\beta_K'J(x_{n+1}, y^*))(y_{n+1}^* - y_{n+1}^{(i-1)}).$$

We have used the mean value theorem and $J(x_{n+1}, y^*)$ is the Jacobian matrix with its entries evaluated at some point along the line between $y_{n+1}^{(i-1)}$ and y_{n+1}^* . We may rewrite this as

$$(I - h'\beta_K'J)(y_{n+1}^{(i)} - y_{n+1}^* + y_{n+1}^* - y_{n+1}^{(i-1)}) = R(I - h\beta_K'J(x_{n+1}, y^*))(y_{n+1}^* - y_{n+1}^{(i-1)}).$$

Therefore

$$e_{n+1}^{(i)} = (I - h'\beta_K'J)^{-1} \{ R h\beta_K'(J(x_{n+1}, y^*) - J) + R(h\beta_K' - h'\beta_K')J + (1 - R)(I - h'\beta_K'J) \} e_{n+1}^{(i-1)}. \quad (6.1.5.8)$$

For the SN iteration (that is, $R = 1$, $h\beta_K' = h'\beta_K'$) (6.1.5.8) can be written as

$$e_{n+1}^{(i)} = (I - h'\beta_K'J)^{-1} h\beta_K'(J(x_{n+1}, y^*) - J) e_{n+1}^{(i-1)}. \quad (6.1.5.9)$$

This indicates that the rate of convergence is affected by the size of the formula's leading coefficient. Formulas with smaller leading coefficients should lead to quicker convergence when the SN iteration is being used. The critical task is to make $\|J(x_{n+1}, y^*) - J\|$ of acceptable size, 'small' elements of $J(x_{n+1}, y^*)$ need not be approximated at all well and advantage may be gained by setting small elements of J to zero. (see also Shampine 1980, Carver :981).

Let us use (6.1.5.9) to consider how errors in both the stiff and non-stiff components are damped when the SN iteration is used. The part of the error matrix which depends on the stepsize and the formula is $(I - h'\beta_k'J)^{-1}h\beta_k'$. We term this matrix the error matrix for the SN iteration. If λ is an eigenvalue of J then there is a corresponding eigenvalue $r(\lambda)$ of the error matrix which is given by

$$r(\lambda) = \frac{h\beta_k}{1 - h\beta_k'\lambda}.$$

If we consider eigenvalues with $|h\lambda| \ll 1$ to be non-stiff eigenvalues then we have

$$r(\lambda) \approx h\beta_k$$

so that errors corresponding to non-stiff eigenvalues are strongly damped if the errors in approximating the small eigenvalues of $J(x_{n+1}, y)$ are small.

Similarly, if we consider eigenvalues with $\text{Real}(h\lambda) < 0$, $|h\lambda| \gg 1$ to be stiff eigenvalues, then we have

$$r(\lambda) \approx -\frac{1}{\lambda}.$$

Hence errors in stiff eigenvalues are very heavily damped. This indicates that using the SN iteration with an accurate Jacobian approximation requires only one iteration to determine the very stiff components accurately and this is more or less independent of the predictor used.

Now consider MN iteration with relaxation factor $R = 1$ and $h\beta_k = h'\beta_k'$. For such iterations the error behaviour may be written as (cf. 6.1.5.8)

$$e_{n+1}^{(i)} = (I - h'\beta_k'J)^{-1} [h\beta_k'(J(x_{n+1}, y) - J) + (h\beta_k - h'\beta_k')J] e_{n+1}^{(i-1)} \quad (6.1.5.10)$$

We now have two components inside the parentheses on the right hand side of (6.1.5.10). One depends on the nature of the Jacobian approximation and the other depends on the difference between $h\beta_k$ and $h'\beta_k'$. This complicates analysis of the error damping behaviour as, in general, it is

not clear how these two terms will interact. We therefore treat both terms separately and bound the worst case behaviour by assuming their contributions add. If λ is an eigenvalue of J corresponding to an eigenvalue λ^* of $J(x_{n+1}, y)$ then we can bound the corresponding eigenvalue of the error matrix by

$$\left| \frac{h\beta_k(\lambda - \lambda^*)}{1 - h'\beta_k'\lambda} \right| + \left| \frac{(h\beta_k - h'\beta_k')\lambda}{1 - h'\beta_k'\lambda} \right|.$$

For non-stiff components we have $|h\lambda| \ll 1$ and errors are strongly damped provided the small eigenvalues of $J(x_{n+1}, y)$ are reasonably well approximated. For stiff components, as the magnitude of $h\lambda$ increases the rate of growth in the error tends to

$$\left| \frac{h\beta_k - h'\beta_k'}{h\beta_k} \right| \quad \text{providing } \lambda - \lambda^* \text{ is small.}$$

For the SN iteration used with an accurate Jacobian approximation, the stiffness of the problem does not affect the convergence rate. The presence of very stiff components does, however, affect the convergence rate of the MN iteration.

If the Jacobian is constant and the Jacobian approximation is exact, SN iteration will converge to the exact solution in one iteration. If λ is an eigenvalue of the Jacobian then the corresponding eigenvalue $r(\lambda)$ of the error matrix for MN iteration is

$$r(\lambda) = \frac{(h\beta_k - h'\beta_k')\lambda}{1 - h'\beta_k'\lambda}.$$

The $\sup_{\text{real}(\lambda) \leq 0} |r(\lambda)|$ is $\frac{|h'\beta_k' - h\beta_k|}{h'\beta_k'}$ and this limit is approached as eigenvalues become arbitrarily large.

Consider the following example. Assume that the required error tolerance is ϵ and that a $(k+1)$ -step extrapolation formula is being used as a predictor. From the previous section, the prediction error in the stiff components, once the stiff components have decayed to be effectively zero, is bounded by $|(2^{k+1} - 1)\epsilon|$. If we require that the non-

linear equations be solved to a tolerance of $\tau\epsilon$ in no more than three MN iterations, we have

$$\left[\frac{|h'_{g'_k} - h_{g_k}|}{|h'_{g'_k}|} \right]^{1/3} (2^{k+1} - 1)\epsilon \leq \tau\epsilon,$$

that is

$$\left[\frac{|h'_{g'_k} - h_{g_k}|}{|h'_{g'_k}|} \right] \leq \left[\frac{\tau}{2^{k+1} - 1} \right]^{1/3}.$$

Let τ be 0.1, which is a common choice, and let k be 5. The convergence of the MN iteration cannot be guaranteed if the product of the stepsize and the formula's leading coefficient, h_{g_k} , has changed by more than 11.7%.

The example indicates that the use of the MN iteration is of limited value in stiff regions. Most problems described as stiff have regions of sharp change (boundary layers) in which the integration is relatively expensive but not stiff. Since the error damping properties of the MN iteration for non-stiff components are similar to those of the SN iteration, the MN iteration could be successfully used in such regions.

The iteration schemes we have considered to date have had the relaxation factor, R , set to 1. We now generalize the MN iteration by considering schemes like (6.1.5.7) where $h'_{g'_k} = h_{g_k}$ and $R = 1$ is a scalar relaxation factor.

Burrage, Butcher and Chipman(1980) in their singly-implicit Runge-Kutta code STRIDE choose

$$R = \frac{2h'_{g'_k}}{h'_{g'_k} + h_{g_k}}.$$

This choice of the relaxation factor is based on the assumption that errors in the Jacobian matrix approximation are not significant and thus the term involving $J(x_{n+1}, y) - J$ in (6.1.5.8) may be ignored. This reduces (6.1.5.10) to

$$e_{n+1}^{(i)} = (I - h'_{g'_k J})^{-1} (R(h_{g_k} - h'_{g'_k J}) e_{n+1}^{(i-1)} + (1 - R)(I - h'_{g'_k J}) e_{n+1}^{(i-1)}). \quad (6.1.5.11)$$

The error matrix of (6.1.5.11) is $I - R(I - h_{g_k J})(I - h'_{g'_k J})^{-1}$ and if λ is an eigenvalue of J then the corresponding eigenvalue of the error propagation matrix is $1 - R(1 - h_{g_k \lambda})(1 - h'_{g'_k \lambda})^{-1}$.

The STRIDE choice of relaxation factor minimizes

$$\sup_{\text{Real}(\lambda) \leq 0} \left| 1 - \frac{R(1 - h_{g_k \lambda})}{1 - h'_{g'_k \lambda}} \right|$$

and this supremum is in fact equal to

$$\left| \frac{h_{g_k} - h'_{g'_k}}{h_{g_k} + h'_{g'_k}} \right|.$$

This is always less than 1 if h_{g_k} and $h'_{g'_k}$ have the same sign. The supremum is attained for $h\lambda = 0$ and approached as $h\lambda \rightarrow -\infty$ so that this choice leads to similar rates of convergence for both stiff and non-stiff components. Chipman(1977) presents results of a stiff, non-linear chemical kinetics problem which was solved by STRIDE at two tolerances, 10^{-5} and 10^{-8} , and solved using either the STRIDE relaxation factor or MN iteration. For both tolerances substantially more work was required when using MN iteration. The amount of extra work was 8% to 22% more function evaluations and 18% to 22% more Jacobian evaluations.

Experiments were carried out with ODIOUS using either MN iteration or the STRIDE relaxation factor on the problems in the STIFF DETEST test set. MN iteration was more efficient overall, resulting in 2.6% fewer function evaluations. The STRIDE relaxation factor was sometimes more efficient at the larger tolerance of 10^{-2} and for problems with large stiffness ratios. For instance, over all problems at 10^{-2} ODIOUS used 1.5% fewer function evaluations when using the STRIDE relaxation factor. The differences in performances for individual problems ranged from 56% more function evaluations for problem B5 when using the STRIDE relaxation factor to 28% fewer function evaluations for C5 at 10^{-6} .

The use of a scalar relaxation factor in (6.1.5.7) can be viewed as an attempt to approximate the SN iteration matrix $I - h\delta_k J$ by an iteration matrix $R^{-1}(I - h'\delta'_k J)$. The choice $R=1$ approximates the small eigenvalues of $(I - h\delta_k J)$ well and preserves the quick convergence of the SN iteration for non-stiff components. The STRIDE relaxation factor represents a compromise in that we attempt to approximate very non-stiff ($h\lambda = 0$) and very stiff ($h\lambda \rightarrow -\infty$) eigenvalues of the SN iteration matrix equally well. Thus the STRIDE relaxation factor should provide quicker convergence for stiff components than the MN iteration at the expense of slower convergence for non-stiff components.

This approach suggests that we define another relaxation factor by requiring that

$$R^{-1}(I - h'\delta'_k J) - (I - h\delta_k J) = (R^{-1} - 1)I$$

so that we are approximating stiff eigenvalues of the SN iteration matrix well. We may term this choice for R the stiff relaxation factor and this is given by $R = (h'\delta'_k)/(h\delta_k)$. For stiff components, as $h\lambda \rightarrow -\infty$ the errors using the stiff relaxation factor are very heavily damped in the same manner as when the SN iteration is used. For non-stiff components the rate of decay tends to $1-R$, i.e. $(h\delta_k - h'\delta'_k)/(h\delta_k)$. Thus the rate of convergence for non-stiff components is slower than for the MN iteration or the STRIDE relaxation factor.

The relaxation factor approach can be used whenever the iteration matrix is used. For instance, we can use relaxation factors in conjunction with linearly implicit formulas and in conjunction with the Sacks-Davis error test (Sacks-Davis, 1977) in order to have a better approximation to the SN iteration matrix.

The use of a scalar relaxation factor involves only a comparatively small number of extra operations - $O(N)$ operations, N being the number of equations in the differential system - but can lead to a substantial reduction in the cost of solving some differential systems. However, as the results of the tests indicate, the use of a fixed scalar relaxation factor may actually significantly degrade the performance of the ODE

solver on particular problems. It is possible to choose the relaxation factor adaptively by making use of information like the relative change in $h\delta_k$, the norm of the Jacobian approximation and the observed rate of convergence. More research is needed to investigate the effectiveness of such schemes. In deriving (6.1.5.8) from (6.1.5.7) no assumptions were made on the properties of R that would preclude our considering the case where R is a matrix. Again, more research would be needed to determine the cost-effectiveness of using matrix relaxation factors.

In discussing various implementation strategies for prediction and for the corrector iterations we have seen that a particular scheme will have different effects on stiff and non-stiff components. A scheme that may be suitable for non-stiff components may be far less effective for stiff components and vice versa. This suggests that a stiff code should attempt to recognize whether the problem is locally stiff or non-stiff and then choose the appropriate predictor and form of corrector iterations. Such codes are known as type-insensitive and a type-insensitive code would be able to take advantage of the various forms of predictors and the various relaxation factors that have been mentioned in the last two sections.

6.1.5.3 Convergence Test

Shampine(1979, 1980c) discusses the behaviour of the SN and MN iterations and the nature of convergence tests based on rate estimates. We need a method of deciding when to terminate the corrector iterations when solving the implicit equations at each step. A simple approach is to assume that the iterative method is converging and therefore assume that $y_{n+1}^{(m)}$ is more accurate than $y_{n+1}^{(m-1)}$. The amount of correction $y_{n+1}^{(m)} - y_{n+1}^{(m-1)}$ may therefore be used to estimate the amount of error in $y_{n+1}^{(m-1)}$. If the correction amount is smaller than some threshold, $y_{n+1}^{(m-1)}$ may be considered accurate enough and $y_{n+1}^{(m)}$ may be accepted as the best approximation to y_{n+1}^* , the exact solution of the implicit equations.

This convergence test may be improved by estimating a rate of convergence. The iterations are assumed to be converging only if the norm of the correction amounts is found to be decreasing. If convergence is assumed to be linear then, by estimating the rate of convergence, we can predict the norm of the next correction amount. This predicted value of the norm may be used in the convergence test without actually calculating the next correction.

It should be noted that we have only considered whether $y_{n+1}^{(m)}$ is close to y_{n+1}^* and have not considered how close $f(x_{n+1}, y_{n+1}^{(m)})$ is to $f(x_{n+1}, y_{n+1}^*)$. By applying a mean value theorem we have

$$f(x_{n+1}, y_{n+1}^{(m)}) - f(x_{n+1}, y_{n+1}^*) = J(x_{n+1}, y^*)(y_{n+1}^{(m)} - y_{n+1}^*)$$

where $J(x_{n+1}, y^*)$ is the Jacobian of the differential system evaluated on a line between $y_{n+1}^{(m)}$ and y_{n+1}^* . Since for a stiff ODE we expect some eigenvalues of the Jacobian to have very large modulus, a small $\|y_{n+1}^{(m)} - y_{n+1}^*\|$ does not imply that the quantity $\|f(x_{n+1}, y_{n+1}^{(m)}) - f(x_{n+1}, y_{n+1}^*)\|$ is small. This question of the accuracy in the accepted function value has been ignored in most BDF codes, possibly because the BDF have the property that all beta coefficients of past f values are zero and thus errors in the f values do not affect the stability of the computed solution at subsequent points.

Shampine(1979) makes the following criticisms of the rate convergence test.

(1) We need at least two iterates to form a convergence rate estimate. This means we cannot accept the predicted value and we can only accept the first corrected value if we use a rate estimate from the previous step. This reduces the reliability of the convergence test.

(2) The rate convergence test assumes linear convergence but this assumption is invalid if round-off errors are dominating the calculations. Thus the rate estimate may be unreliable if we have a particularly accurate predicted or first corrected value.

(3) The reliability of the test is dependent on the nature of the iterative method. Shampine cites the example that the choice of the iterative method in the code GEAR could lead to large differences in the accuracy of the computed solution. He argues that if the convergence test were reliable and did not depend on the iterative method then the iterative method used would have an effect on the efficiency of the code but not on the reliability of the code's results.

Shampine also notes that the non-linear equations (6.1.5.1) may have several solutions and suggests that we should accept the solution which is close to the predicted value in the sense that the iterations contract from the predicted value. The verification of this last requirement requires a rate estimate and therefore at least two iterations would be needed.

Using results for the convergent behaviour of formulas for ODEs, Shampine(1980c) argues that (6.1.5.1) should be solved in terms of requiring that the norm of the residual, $r_{n+1}^{(m)} = y_{n+1}^{(m)} - h\delta_K f(x_{n+1}, y_{n+1}^{(m)}) - \psi$, be small and notes that a convergence test based on the residual has certain advantages over the rate convergence test. In particular,

- (1) the residual test can be applied to any iterate including the predicted value;
- (2) the residual test makes no assumptions on the nature of the iterative method.

We can illustrate Shampine's ideas as follows. If we let $e_{n+1}^{(m)} = y_{n+1}^{(m)} - y_{n+1}^*$, then we have

$$r_{n+1}^{(m)} = y_{n+1}^{(m)} - h\delta_K f(x_{n+1}, y_{n+1}^{(m)}) - \psi = -(I - h\delta_K J(x_{n+1}, y^*))e_{n+1}^{(m)}.$$

Since for stiff equations $J(x_{n+1}, y^*)$ has some eigenvalues with negative real part and large modulus, we expect

$$\|(I - h\delta_K J(x_{n+1}, y^*))\| \geq 1.$$

Therefore, we expect that

$$\| r_{n+1}^{(m)} \| = \| (I - h\delta_K J(x_{n+1}, y^*)) e_{n+1}^{(m)} \| \geq \| e_{n+1}^{(m)} \|.$$

When solving stiff equations we would expect the norm of the residual to be larger than the norm of the error vector; just how much larger will depend on how much of the error is in the direction of the stiff eigenvalues of the Jacobian. If the norm of the residual is small then this guarantees that the error in the current iterate is small. However, if the norm of the correction vector is small this does not guarantee that we are solving the implicit equations accurately in terms of a small residual.

Current codes which use a rate convergence test do not consider accepting the predicted value. Since the codes often use the SN iteration, one SN iteration will greatly damp the errors in the stiff components. Thus, for all iterates other than the predicted value, the norm of the residual should be comparable to the norm of the correction amount and tests on the correction amount should produce reliable results in a situation where the iterative method strongly damps errors in the stiff components.

The residual test requires that the function be evaluated at the iterate which is being tested. Since both the current iterate and the function value at the current iterate are used in the residual test, the residual test acts as a convergence test on both the current iterate and its function value and we can accept both quantities if the norm of the residual is small enough.

It is still possible with the residual test to monitor the rate at which the norm of the residual is decreasing and to attempt to predict the norm of the residual that would result at the next iteration. Thus, the final function evaluation may be avoided.

In solving the implicit equations (6.1.5.1) we need to compute approximations to both y_{n+1}^* and $f(x_{n+1}, y_{n+1}^*)$. Using the rate convergence test we compute $y_{n+1}^{(m)}$ to test $y_{n+1}^{(m-1)}$. If $y_{n+1}^{(m-1)}$ is accurate enough we accept $y_{n+1}^{(m)}$. At this point we haven't evaluated the function at $y_{n+1}^{(m)}$ and we have a number of choices about how we calculate a final $f_{n+1}^{(m)}$ value. Most codes for stiff ODEs choose the final f -value to satisfy

the LMF so they have

$$f_{n+1}^{(m)} = (y_{n+1}^{(m)} - \psi) / h\delta_K = f(x_{n+1}, y_{n+1}^*) - e_{n+1}^{(m)} / h\delta_K.$$

If we evaluate the function at $y_{n+1}^{(m)}$ we obtain

$$f_{n+1}^{(m)} = f(x_{n+1}, y_{n+1}^*) - J(x_{n+1}, y^*) e_{n+1}^{(m)}.$$

For non-stiff components it is better to evaluate the function for the final approximation to f_{n+1}^* as the corresponding eigenvalues of the Jacobian will be small. For example, most codes for solving non-stiff ODEs using the Adams formulas choose to perform a final function evaluation. Codes for solving stiff equations usually choose not to perform the final function evaluation. For stiff components it is probably preferable to choose the final approximation to f_{n+1}^* by satisfying the formula (Robertson and Williams 1975, Shampine 1980).

ODIOUS uses both the residual test and the correction amount test. Using the residual test enables ODIOUS to accept the predicted value, as often happens when LI predictors are used for linear problems. If the correction amount test is applicable, it is used before the residual test and if it leads to the iterate's acceptance, then the final f value is chosen to satisfy the LMF. If an iterate is accepted via the residual test then the final f value used is the function evaluated at the accepted iterate. The threshold used in both tests is $1/100^{\text{th}}$ the user error requirement.

6.1.5.4 Recovery from Convergence Failure

ODIOUS uses the following heuristics on convergence failure. If the Jacobian had been evaluated on the last accepted step and adequate convergence had not been achieved then the current cycle is rejected and the backward difference representation is restored to that current at the last point where the solution was calculated by the last accepted cycle. For the next cycle, the stepsize is halved and if the lower order formula would be accurate enough at the reduced stepsize, then the

order is lowered. The Jacobian, J , is then evaluated at the last point calculated by the last accepted cycle and the scaled iteration matrix $(J - 1/h_k I)$ is LU-factorized.

If the Jacobian had not been evaluated on the last accepted step, then it is assumed that an out of date Jacobian was responsible for the convergence difficulties. The Jacobian is therefore re-evaluated using the numerical solution value from the last accepted step and the scaled iteration matrix is LU factorized. If the non-linear equations had been converging but converging too slowly, then the iteration count is reset and the non-linear equations re-attempted with the next correction being calculated using the current iterate and the new iteration matrix. If the convergence failure had not been due to slow convergence, then the non-linear equations are re-attempted using a linearly-implicit predicted value calculated using the new iteration matrix.

6.2 A COMPARISON OF ODIUS AND LSODE

Testing of stiff ODE solvers by Enright, Hull and Lindberg(1975) and Enright and Hull(1976b) has shown that BDF-codes are the most efficient codes available. They do not, however, perform efficiently for problems which have a Jacobian with eigenvalues with negative real part and large imaginary part. The widely available code LSODE, due to Hindmarsh(1980), is a fixed coefficient implementation of the BDFs of orders 1 to 5. We have therefore chosen to compare our code against LSODE as LSODE is well known and widely regarded as a highly efficient BDF-code which is truly representative of the state of the art in BDF-codes for stiff problems.

6.2.1 Comparison of Formulas

The ODIUS formulas were derived to have high order with large stability angles and therefore to be more suitable than the BDFs for the efficient solution of problems where the Jacobian of the differential system has some eigenvalues with large modulus, negative real part and which are close to the imaginary axis. The comparison of formulas is complicated by the fact that although both codes use LMFs, ODIUS uses two-stage CLMFs. Nevertheless it is useful to compare the the BDFs and ODIUS formulas with respect to the following properties:

- (a) Roots at zero and infinity;
- (b) Stability;
- (c) Leading coefficients and beta coefficients;
- (d) Error coefficients.

We will look at each of these briefly.

6.2.1.1 Roots at zero and infinity

Both sets of formulas have all roots at infinity equal to zero. This is a desirable property since the stiff components decay rapidly if these roots are zero. At the origin, the BDFs of order greater than one have all extraneous roots non-zero. The most commonly used LMFs for solving non-stiff equations are the Adams-Moulton formulas which have all extraneous roots zero at the origin. The ODIUS formulas were derived to have all extraneous roots zero at the origin by analogy with the Adams-Moulton formulas as it was felt that this would prove beneficial in the solution of non-stiff problems.

6.2.1.2 Stability

The ODIUS formulas were derived to have better $A(\alpha)$ -stability angles than the BDFs. Thus a code using the ODIUS formulas would be able to use higher order formulas more effectively than a BDF-code for those problems where eigenvalues of the Jacobian lie in the left complex

half-plane close to the imaginary axis. A comparison of the stability angles for the BDFs and the ODIIOUS formulas is given in Table 5.1 in the previous chapter. From this table it can be seen that the ODIIOUS formulas can achieve a given stability angle with a formula whose order is two greater than the corresponding BDF.

6.2.1.3 Leading Coefficients and Beta Coefficients

In section 6.1.5.2 we discussed the behaviour of corrector iterations. In particular we showed that the convergence rate of the SN iteration was directly proportional to the size of the formula's leading coefficient (equation 6.1.5.9). For the ODIIOUS formulas of order greater than two the leading coefficients vary in size from 1.72 to 1.18 while the largest leading coefficient for the BDFs of order greater than two is 0.545. We would therefore expect that the use of a BDF will lead to quicker corrector convergence and may result in a BDF-code using fewer function evaluations per step than a code using the ODIIOUS formulas.

The BDFs have all beta coefficients at past points zero and therefore the stability properties of an implementation of the BDFs are less likely to be affected by the way that the derivative is calculated. For example, after a stepsize change using the interpolation technique interpolation errors in past derivative values do not directly affect the calculation of the solution at the new point. Similarly, a BDF-code having calculated the solution at a point x_{n+1} does not use the derivative at x_{n+1} in calculating the solution at subsequent points. The ODIIOUS formulas, on the other hand, do not have this property of having all the beta coefficients at past points zero.

6.2.1.4 Error Coefficients

A direct comparison of error coefficients is difficult since ODIIOUS implements a pair of LMFs as a CLMF and therefore treats two steps together as a cycle. The error control attempts to control the local discretization error over the cycle in contrast to the step by step error control in LMF codes.

The error coefficient used in ODIIOUS is the max-norm of the first two local discretization error coefficients. The form of error coefficient used in ODIIOUS is based on the fact that we can relate the local discretization error vector for quite general methods to a bound on the endpoint global error. In chapter 4, equation (4.3.2), we give such a bound by following the approach of Albrecht(1978). Notice that the first order BDF when used in ODIIOUS is implemented using a larger error coefficient than that normally used for the first order BDF implemented as a LMF. This is because the error constant in ODIIOUS is taken over two steps.

The effect of the error coefficients is detailed in the Table 6.2. BDF1 refers to the first local discretization error coefficient for the BDFs while BDF2 refers to BDF1 scaled using the inverse of the sum of the beta coefficients as suggested by Henrici(1962). (In the case of the BDFs this is simply the leading coefficient). The ratios BDF1/ODIIOUS and BDF2/ODIIOUS in the last two columns indicate the ratio in stepsizes that result from using the BDF error coefficients compared to the ODIIOUS error coefficients.

The ODIIOUS second order formula is the only ODIIOUS formula, (the first order formula is a BDF), that is comparable in terms of the magnitude of the error coefficient with the BDF of the same order. LSODE chooses to use the first scaled local discretization error coefficient, that is the coefficients in column BDF2, and it can take steps which are at least 40% larger when using the 3rd to 5th order BDFs than ODIIOUS can using the same order formula.

The error coefficient used in ODIIOUS is based on controlling the local discretization error introduced in a cycle. In chapter 4, section

Order	ODIOUS	BDF1	BDF2	Ratio of Stepsizes	
		BDF1 / ODIOUS	BDF2 / ODIOUS		
1	1.	.5	.5	1.4142	1.4142
2	.2451	.2222	.3333	1.0332	.9026
3	1.2441	.1363	.25	1.7380	1.4936
4	1.2558	.096	.2	1.6724	1.4440
5	1.5440	.0730	.1667	1.6630	1.4002

Table 6.2: Error Coefficient Comparison (magnitudes)

4, we showed that the long term error behaviour of CLMFs can be very different from the short term error behaviour as indicated by the size of the local discretization error. In chapter 4, section 4, we defined the propagated error coefficient which can be used to compare the longer term accuracy properties of formulas. For LMFs the propagated error coefficient is equal to the local truncation error coefficient scaled by the sum of the beta coefficients and the propagated error coefficients for the BDFs are the values referred to in Table 6.2 as BDF2.

We want to compare the ODIOUS formulas with the BDFs with respect to the short and long term error behaviour. In ODIOUS the stepsize for a k -step formula is usually held constant for at least m steps after a stepsize change, where $m = k+1$ for k odd and $m = k+2$ for k even. For a small number of steps we can work out the local discretization error introduced in computing the solution at evenly spaced points x_{n+1} , $i = 1, 2, \dots$, given that past values at x_{n-j} , $j = 0, 1, 2, \dots, k-1$ are exact. We assume we can express the error in the solution at x_{n+1} in the form $c_{i,p} h^{p+1} (p+1) (x_{n+1}) + O(h^{p+2})$ for a p th order formula and we refer to $c_{i,p}$ as the i th local discretization error coefficient. Let us therefore model the short term behaviour of the build up in error for a k -step formula by introducing a short term error coefficient, the local discretization error coefficient at the m th step weighted by $1/m$. The error propagation coefficient weighted by the number of steps in the cycle can be used as an indication of the longer term error behaviour.

In Table 6.3 the ODIOUS formulas are compared with the BDFs on the basis of short and long term error behaviour. Using these coefficients we can see that the second order ODIOUS formula is more accurate than the corresponding BDF. The third order ODIOUS formula can be more efficiently implemented than it is currently as both its short and long term coefficients are smaller than the error coefficient used in ODIOUS. However, the higher order formulas have large error coefficients with the long term error coefficient being even larger than the error coefficient used in ODIOUS. The fourth to eighth order ODIOUS formulas can not therefore be expected to be competitive with the corresponding BDF unless high accuracy is required for problems which need formulas with very good stability angles.

To demonstrate that the third order ODIOUS formula is more accurate than its local discretization error vector would indicate we present the first six local discretization error coefficients for the third order ODIOUS formula in Table 6.4. For this formula the local discretization error coefficients actually decrease in magnitude as a result of cancellation between the local discretization error introduced in the current step and the local discretization error which propagates from past

Order	ODIOUS		BDFs	
	short term	long term	short term	long term
1	-0.5	-0.5	-0.5	-0.5
2	-0.1300	-0.1374	-0.2922	-0.3333
3	-0.2560	0.08747	-0.2176	-0.25
4	1.4840	2.7111	-0.1835	-0.2
5	2.9623	5.8556	-0.1517	-0.1667
6	4.1356	7.8429	-	-
7	5.1849	10.6212	-	-
8	8.3430	15.6042	-	-

Table 6.3: Comparison of Error Coefficients ODIOUS - BDFs

	Local Discretization Error Coefficient	Change in Global Error *
Cycle 1	Step 1 Step 2	-1.24 4108 -1.19 8783
Cycle 2	Step 1 Step 2	-1.47 5195 -1.02 3850
Cycle 3	Step 1 Step 2	-1.30 0262 -0.84 89167
Error Propagation Coefficient		0.174 9333

* This change in global error is the change in global error between the same numbered steps in consecutive cycles.

Table 6.4: Short Term Error Behaviour of the Third Order ODIIOUS Formula points.

The first six local discretization error coefficients for the fourth order ODIIOUS formula are presented in Table 6.5. In this case we also see some error cancellation but the local discretization error coefficients grow strongly in magnitude due to the propagation of error from past points. In both the third and fourth order ODIIOUS formulas we can see that once at least k constant sized steps have been taken with a k -step formula, the change in the local discretization error coefficients between the same numbered stages in successive cycles is equal to the propagated error coefficient. For the ODIIOUS formulas this occurs because the extraneous roots at the origin are zero and thus the local discretization error associated with the extraneous roots gets completely damped out after k steps.

A code using the ODIIOUS formulas could take larger steps than a BDF-code if it can use a higher order formula for a given accuracy requirement. Many factors, however, influence the choice of order in a variable order code. These factors include, for example, the nature of

	Local Discretization Error Coefficient	Change in Global Error *
Cycle 1	Step 1 Step 2	-1.25 5794 -1.14 2429
Cycle 2	Step 1 Step 2	0.57 29524 3.48 1571
Cycle 3	Step 1 Step 2	5.99 5175 8.90 3793
Error Propagation Coefficient		5.222 2222

* This change in global error is the change in global error between the same numbered steps in consecutive cycles.

Table 6.5: Short Term Error Behaviour of the Fourth Order ODIIOUS Formula

the problem, the accuracy requirement, the error coefficients used in the code, the error propagating behaviour of the formulas and the nature of the stepsize changing technique. We therefore have to look at the results of numerical testing to see how the higher order of the ODIIOUS formulas offsets the effect of the larger error coefficients.

The code ODIIOUS was tested on the problems in the stiff test set STIFF DETEST (Bedet, Enright and Hull, 1975) for the tolerances 10^{-2} , 10^{-4} and 10^{-6} . In Table 6.6 we give an order usage profile. The code BDF-ODIIOUS was also run on the same problems at the same tolerances. BDF-ODIIOUS is ODIIOUS modified to implement the first to sixth order BDFs as two-stage CLMFs with both stages identical. The only changes in ODIIOUS necessary to accomplish this were changes to constants. In Table 6.7 we give an order usage profile for BDF-ODIIOUS.

Order	Cycles Attempted	% of total no. of cycles attempted	No. of cycles which failed stage 1	No. of cycles which failed stage 2	% of cycles attempted which were successful
1	270	2.47%	9	9	93.3%
2	1596	14.60%	38	49	94.5%
3	672	6.15%	53	13	90.2%
4	1943	17.78%	57	23	95.9%
5	2239	20.49%	9	41	97.8%
6	2237	20.47%	5	103	95.2%
7	1310	11.99%	1	97	92.5%
8	661	6.05%	0	97	85.3%
Total	10928		172	432	
			1.57%	3.95%	

Table 6.6: Order Usage Profile for ODIIOUS

Because the error coefficients for the BDFs decrease with increasing order the BDF-codes tend to take a large proportion of steps using the higher order formulas. The error coefficients for the ODIIOUS formulas decrease for the first two orders and then increase with increasing order. Thus ODIIOUS tends to use its lower order formulas, particularly the second order formula, for a greater proportion of steps than BDF-ODIIOUS. At the tolerances used in the numerical testing ODIIOUS used formulas of order greater than five on only 38.5% of the steps.

Order	Cycles Attempted	% of total no. of cycles attempted	No. of cycles which failed stage 1	No. of cycles which failed stage 2	% of cycles attempted which were successful
1	258	2.64%	6	12	93.0%
2	580	5.93%	17	17	94.1%
3	780	7.98%	7	25	95.9%
4	1335	13.65%	7	37	96.7%
5	3227	33.01%	9	75	97.4%
6	3597	36.79%	1	112	96.9%
Total	9777		47	278	
			.48%	2.84%	

Table 6.7: Order Usage Profile for BDF-ODIIOUS
(B5 not solved to completion)

6.2.2 Comparison of Performance

The stiff ODE solver testing program, STIFF DETEST, due to Bedet, Enright and Hull(1975) and described by Enright(1978), was used to compare the performances of ODIIOUS and LSOE and the results are presented in Tables 6.8 and 6.9. All testing was carried out on a VAX 11/780. The results for problem B5 at all tolerances and F4 at 10^{-4} and 10^{-6} have been omitted as they distort the comparison since LSOE does particularly poorly on B5. The results for the problem F4 at 10^{-4} and 10^{-6} have also been omitted as ODIIOUS failed to solve this problem with these tolerances within the limit of 3000 function evaluations.

Overall, ODIIOUS takes about 35% more steps and 85% more function evaluations than LSOE. Therefore at the three tolerances ODIIOUS is never more efficient than LSOE unless the problem requires special stability properties.

Tolerance	Number of Functions	Number of Jacobians	Number of Steps	CPU time (secs)
10^{-2}	5393/3290 (1.639:1)	605/574 (1.054:1)	2620/2285 (1.147:1)	45.02/31.00 (1.452:1)
10^{-4}	8092/4364 (1.854:1)	778/624 (1.247:1)	4624/3282 (1.409:1)	89.30/61.78 (1.445:1)
10^{-6}	15380/7968 (1.930:1)	1060/869 (1.220:1)	8730/6305 (1.385:1)	182.55/118.00 (1.547:1)
Total	28865/15622 (1.848:1)	2443/2067 (1.182:1)	15974/11872 (1.346:1)	316.87/210.78 (1.503:1)

Table 6.8: ODIOUS/LSODE Overall Performance Results

Tolerance	Number of Functions per Step	Number of Steps per Jacobian
10^{-2}	2.0584/1.4398	4.3306/3.9808
10^{-4}	1.7500/1.3297	5.9434/5.2596
10^{-6}	1.7617/1.2638	8.2358/7.2555
Overall	1.8070/1.3159	6.5387/5.7436

Table 6.9: ODIOUS/LSODE Efficiency of Solving Implicit Equations

For problem B5 ODIOUS is very efficient compared to LSODE, as is shown in Table 6.10. Problem B5 is characterized by the Jacobian of the differential system having a complex conjugate pair of eigenvalues with negative real part and large imaginary part ($-10 \pm 100i$). The last column in Table 6.10 refers to the global error at the endpoint in units of the error tolerance.

As far as efficiency per step is concerned, LSODE averages only about 1.3 function evaluations per successful step. While LSODE only takes a small number of function evaluations per step it does evaluate

Tolerance	Number of Functions	Number of Jacobians	Number of Steps	CPU time (secs)	Global Error at Endpoint
10^{-2}	227/2545	18/120	146/2351	2.63/ 41.68	.02/15.39
10^{-4}	811/2768	30/147	466/2397	12.42/ 49.43	.60/ 3.98
10^{-6}	1899/2948	49/150	1008/2585	28.95/ 51.65	.46/ 2.85
Total	2937/8261	97/417	1620/7333	44.00/142.76	

Table 6.10: ODIOUS/LSODE Performance on Problem B5

the Jacobian quite frequently. For all problems excluding B5, and F4 at 10^{-4} and 10^{-6} LSODE averages a Jacobian evaluation every 5.74 steps. LSODE's performance on problem B5 is atypical in that B5 has a constant Jacobian and when solving B5 LSODE only evaluates the Jacobian every 20 steps as the stepsize and order are held constant for long intervals.

Although the major cause for the difference in performance between ODIOUS and LSODE is the difference in formulas implemented, we still need to assess the effectiveness of the implementation strategies of the two codes. This task is made easier by the fact that we have a code, BDF-ODIOUS, which uses the ODIOUS implementation strategies and uses the BDFs as two-stage CLMFs.

A major difference in the performance of LSODE and ODIOUS is the fact that LSODE averages fewer function evaluations per successful step than ODIOUS (1.3 as compared to 1.8). BDF-ODIOUS uses the same code as ODIOUS and the same formulas as LSODE. For the problems in the STIFF DETEST test set, excluding B5 at all tolerances and F4 at 10^{-4} and 10^{-6} , BDF-ODIOUS averages 1.8 function evaluations per successful step and evaluates the Jacobian, on average, once every 5.67 successful steps. The differences in the formulas seem to have little impact on the cost of solving the non-linear equations at each step. This may indicate that LSODE solves the non-linear equations at each stage more efficiently than ODIOUS. LSODE's efficiency in solving the non-linear equations at each step may be partly due to the Jacobian re-evaluation

strategy and partly due to the convergence criteria used. LSOE updates the Newton iteration matrix if

- i) the leading coefficient of the iteration matrix has changed by more than 30%;
- ii) more than 20 steps have elapsed since the last update;
- iii) a convergence failure has occurred.

ODIOUS on the other hand updates the iteration matrix if the leading coefficient of the iteration matrix has changed by more than 20% or if there has been a convergence failure. Thus, the Jacobian re-evaluation strategies of both codes are quite similar. The fact that ODIOUS evaluates the Jacobian every 6.54 successful steps as opposed to LSOE's evaluating the Jacobian every 5.74 successful steps is therefore more likely due to the fact that when the higher order, higher stepnumber formulas in ODIOUS are being used the stepsize is held constant for more steps than in LSOE.

A difference between LSOE and ODIOUS is that LSOE evaluates the Jacobian at the predicted value for the next step, on the assumption that the Jacobian is only varying slowly, whereas ODIOUS evaluates the Jacobian using the last accepted step. This may lead to slower convergence for ODIOUS for non-linear problems, especially the non-linear problems in classes C to F inclusive. The ODIOUS strategy is a conservative one.

LSOE estimates the rate of corrector convergence and uses this to predict the size of the correction amount that would be applied to the next iterate. If the predicted correction amount size is less than some small fraction of the error tolerance (the threshold varies from 0.0556 at order 1 to 0.00348 at order 5) then the current iterate is accepted. When we consider that LSOE requires an average of 1.32 function evaluations per successful step then we can see that LSOE operates largely in a Predict Evaluate Correct (PEC) mode.

ODIOUS does not use the predicted convergence amount for convergence testing but instead uses two convergence criteria: one based on the residual test and the other on the magnitude of the current

correction amount. ODIOUS is also capable of accepting the predicted value, which LSOE cannot. The residual test is more conservative than the correction amount test when both tests use the same threshold (see section 6.2.5.3). It is therefore expected that when more than one iteration are required for convergence, the final iterate is being accepted via the correction amount test. ODIOUS requires about 1.8 function evaluations per successful step, and so must operate a significant amount of the time in a PECEC mode. The correction amount test needs to calculate the next iterate in order to test the current iterate, and having accepted the current iterate, this next iterate is actually used as the final value.

ODIOUS at the low tolerance of 10^{-2} only takes about 14.66% more steps than LSOE whereas at the 10^{-4} and 10^{-6} tolerances it takes around 40% more steps than LSOE. Thus ODIOUS is more competitive at the lowest tolerance. This can partly be attributed to the fact that ODIOUS is biased toward the lower order formulas when very large stepsize increases are possible. For example, when using first and second order formulas ODIOUS allows the stepsize to increase by factors up to 10. The fact that ODIOUS is more competitive with LSOE at the lowest tolerance is also in part due to the small error coefficient of the second order ODIOUS formula.

Whereas LSOE accepts or rejects attempted steps on a step-by-step basis, ODIOUS accepts or rejects cycles on a cycle basis. This means that if the first stage of a cycle passes the error requirement while the second has not, then the first stage is rejected as well. For ODIOUS, 4.6% of the cycles attempted are rejected because they fail the error test. As shown in Table 6.8 most of the failed cycles occur on the second stage.

The ODIOUS formulas were derived to have very good stability properties so that an ODE solver incorporating these formulas would be efficient for problems where the Jacobian of the differential system has eigenvalues with negative real part, large imaginary part and large modulus. The results of the testing indicate that the formulas are

successful in this respect. ODIUS, however, is less efficient than LSODE on problems which don't require that the formulas have good stability angles. We wish to point out that there is only one problem in the test set used which requires that the formulas have good stability angles. If used as a general purpose stiff equation solver ODIUS may take as much as twice as many function evaluations as ODIUS but is more reliable in the sense that it can handle a wider range of problems efficiently.

CHAPTER 7

ADDITIONAL FAMILIES OF FORMULAS

In Chapter 5 we derived a class of two-stage CLMFs where the formulas had constant leading coefficients, all roots zero at infinity and all extraneous roots zero at the origin. After satisfying these requirements one free parameter remained. The ODIUS formulas were derived by choosing this free parameter to maximize the $A(\alpha)$ -stability angles. An implementation of the ODIUS formulas was described in the last chapter. When the performance of the code, ODIUS, was compared to the performance of LSODE it was found that ODIUS was very efficient for problems where formulas with large $A(\alpha)$ -stability angles were required. LSODE, however, was more efficient for stiff problems in general because of the smaller error coefficients of the BDFs. The performance comparison presented in the last chapter also suggested several areas where ODIUS might be improved.

In this chapter we consider alternative choices for the free parameter in an attempt to find formulas with better accuracy properties than the ODIUS formulas but which still possess better $A(\alpha)$ -stability angles than the BDFs. We then evaluate the performance of a revised version of ODIUS which uses the new families of formulas.

7.1 NEW FAMILIES OF FORMULAS

We noted in Chapter 5 that we could choose the free parameter in deriving the two-stage CLMFs to make the resulting CLMF higher order convergent in the sense of Albrecht(1978). In fact, for a given order there are two values of the free parameter that result in higher order convergence with the larger value giving the formula with the greater $A(\alpha)$ -stability angle. Since the first and second order formulas of the ODIUS family already have small error coefficients we only considered

formulas with $k \geq 3$. We derived a family of two-stage, k -step/ $(k+1)$ -step CLMFs for $k = 3$ to 7 by requiring that the formulas be higher order convergent and by choosing the larger value of the free parameter. Coefficients for all new formulas are given in Appendix A. Since the formula for $k = 7$ is no longer $A(0)$ -stable, we will only consider the higher order convergent formulas for $k = 3$ to 6 and we call this family the SDRROOT formulas.

In the previous chapter we considered both the short and long term accuracy properties of formulas by defining two error coefficients: a short term and a long term error coefficient. In our revised implementation of CLMFs we use the larger absolute-valued error coefficient in stepsize and order selection.

In Table 7.1 we give the short term and long term error coefficients for both the ODIUS and SDRROOT families of formulas.

Order	ODIOUS		SDRROOT	
	short term	long term	short term	long term
1	-0.5	-0.5		
2	-0.1300	-0.1374		
3	-0.2560	0.08747	-0.3078	0.
4	1.4840	2.7111	-0.3034	0.
5	2.9623	5.8556	-0.4567	0.
6	4.1356	7.4299	-0.5335	0.
7	5.1849	10.5621	*	
8	8.3430	15.6042	*	

* no longer $A(0)$ -stable

Table 7.1: Comparison of Error Coefficients ODIUS - SDRROOT

It can be seen from Table 7.1 that the SDRROOT formulas have smaller error coefficients than the ODIUS formulas. The SDRROOT family does not include seventh or eighth order formulas but our evaluation of

LSODE and ODIUS in the previous chapter indicated that a code which does not have higher order formulas can still be efficient. The SDRROOT formulas are higher order convergent in the sense of Albrecht(1978) so the k -step/ $(k+1)$ -step formula is actually $(k+1)$ st order convergent. In Table 7.2 we present the first six local discretization error coefficients for the third order SDRROOT formula as the trend in the error coefficients demonstrates the higher order convergence of the SDRROOT formulas.

It is difficult to see how this higher order convergence property could be exploited in an implementation. We have seen in the previous chapter that both LSODE and ODIUS change the stepsize frequently. This is evidenced by the fact that both codes evaluate the Jacobian every five or six steps, even for linear problems where the Jacobian is constant. The higher order convergence property, however, is based on the assumption that constant sized steps are taken.

In the case of the SDRROOT formulas we are not in a position to take advantage of the good long term error properties so we consider two

	Local Discretization Error Coefficient		Change in Global Error *
	Step 1	Step 2	
Cycle 1			
	-0.99 51745	-1.60 1708	
Cycle 2			
	-1.97 5266	-1.82 0158	-0.980 0915
			-0.218 4500
Cycle 3			
	-1.97 5266	-1.82 0158	0.
			0.
Error Propagation Coefficient			0.

* This change in global error is the change in global error between the same numbered steps in consecutive cycles.

Table 7.2: Short Term Error Behaviour of the Third Order SDRROOT formula

Order	BDFERR		SDOPT	
	short term	long term	short term	long term
3				
4	-0.1731	0.2000	-0.1931	0.1931
5	-0.3610	0.1667	-0.1837	0.1837
6	-0.4445	0.1429	-0.2900	0.2900
			-0.3287	0.3287

Table 7.3: Comparison of Error Coefficients BDFERR - SDOPT

The $A(\alpha)$ -stability angles of the BDFs, ODIUS, SDRROOT, BDFERR and SDOPT formulas are given in Table 7.4 while the error coefficients are given in Table 7.5. For the SDOPT and BDFERR formulas the seventh order formula is no longer $A(0)$ -stable.

Order	BDF	ODIOUS	SDRROOT	BDFERR	SDOPT
1	90.00				
2	90.00	90.00			90.00
3	86.03	90.00	89.99		
4	73.35	90.00	84.88	85.93	85.85
5	51.84	86.64	70.63	71.90	72.82
6	17.84	76.32	44.70	46.08	47.83
7	*	57.66	*	*	*
8	*	22.15	*	*	*

* these formulas are no longer $A(0)$ -stable.

Table 7.4: Comparison of $A(\alpha)$ -stability angles

additional families of formulas where we have traded off very good long term error properties against good short term error properties. Since the first to third order ODIUS formulas are comparable in terms of the error coefficient and $A(\alpha)$ -stability angles with the BDF we will only consider finding new third and higher order formulas. Where we had a choice between two values for the free parameter, as in the case of the SDRROOT formulas, we chose the larger value as it leads to larger $A(\alpha)$ -stability angles.

We noted in Section 6.3 that when the propagated error coefficient and the local discretization error differ in sign we can get favourable cancellation between the local discretization error introduced at the current point and the local discretization error propagated from past points. We therefore derived fourth, fifth and sixth order formulas which have a positive propagated error coefficient which is the same magnitude as that for the BDF of the corresponding order. This family of formulas, along with the first to third order ODIUS formulas, we call the BDFERR family of formulas.

Since we will use the larger absolute-valued error coefficient to control the stepsize and order selection in the code, it is natural to choose formulas where both the short and long term error coefficients are equal in magnitude and different in sign. We refer to this family of formulas, along with the first and second order ODIUS formulas, as the SDOPT formulas.

The short and long term error coefficients for the BDFERR and SDOPT formulas are given in Table 7.3 while the coefficients of the formulas to 16 significant digits are given in Appendix A.

Order	BDF	ODIOUS	SDRROOT	BDFERR	SDOPT
1	-0.5				
2	-0.3333	-0.1374			
3	-0.25	-0.2560	-0.3078		0.1931
4	-0.2	2.7111	-0.3034	0.2	0.1837
5	-0.1667	5.8556	-0.4567	-0.3610	0.2900
6	-0.1429	7.4299	-0.5335	-0.4445	0.3287
7	-	10.5621	-	-	-
8	-	15.6042	-	-	-

Table 7.5: Comparison of Error Coefficient

As the last two tables show, the SDRROOT, BDFERR and SDOPT families of formulas exhibit better $A(\alpha)$ -stability angles than the BDFs, although the improvement is not as marked as for the ODIOUS formulas. The SDRROOT, BDFERR and SDOPT formulas, however, have far smaller error coefficients than the ODIOUS formulas. For fourth order and lower, the SDOPT formulas actually have smaller error coefficients than the BDFs. In all the families we have derived the error coefficients increase in magnitude for the formulas with order greater than four but this is not necessarily a disadvantage as the $A(\alpha)$ -stability angles also fall away markedly.

7.2 MODIFICATIONS TO ODIOUS

In comparing ODIOUS to LSODE it became obvious that LSODE is a more efficient implementation than ODIOUS. This is especially so when BDF-ODIOUS is compared to LSODE as in this case both codes use the same formulas. After some experimentation we decided to make some changes to ODIOUS and we call the revised code MODIOUS and the version which uses the BDFs, BDF-MODIOUS. The versions which implement the SDRROOT, BDFERR and SDOPT families of formulas are called SDRROOT, BDFERR and SDOPT respectively. The new codes only differ with respect to the formulas they use and no attempt has been made to tune the code to any particular

set of formulas.

MODIOUS differs from ODIOUS with respect to recovery from error failure, Jacobian evaluation policy and convergence test.

Whereas ODIOUS only evaluates the Jacobian at the last accepted point, MODIOUS evaluates the Jacobian at the point where the solution is to be calculated by using the k th order extrapolated y -value, k being the order of the current formula. After a convergence failure the Jacobian is re-evaluated at the current iterate, in the case of slow convergence, and at the predicted value, in the case of a non-convergent iteration. These changes lead to an improvement in performance for the problems in classes E and F, especially F4.

We have attempted to implement a rate convergence test and the convergence test was modified as follows. Whenever at least two corrections have been made a rate estimate, RATE, is made using

$$\text{RATE} = \frac{\|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}\|_{\infty}}{\|y_{n+1}^{(i)} - y_{n+1}^{(i-1)}\|_{\infty}}, \quad i=1,2,$$

and this is used to update an overall rate estimate, RATE_N, using RATE_N = max(RATE, 0.7 * RATE_N). RATE_N is re-initialized whenever the product of the stepsize and the leading coefficient, h_k , changes.

The correction amount test now accepts the current iterate, $y_{n+1}^{(i+1)}$, if

$$\|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}\|_{\infty} \leq \tau * \text{eps}$$

where τ is 0.001 for the first iterate and 0.01 for subsequent iterates. The threshold is less stringent for iterates other than the first because for iterates other than the first we know the iteration is converging. Where a rate estimate is available, the current iterate, $y_{n+1}^{(i+1)}$, is accepted if

$$\frac{\text{RATE}_N}{1 - \text{RATE}_N} \|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}\|_{\infty} \leq 0.005 * \text{eps},$$

where eps is the user accuracy requirement. Since the residual test is more stringent than the correction amount test, the threshold used has

been changed to $0.05 * \text{eps}$.

LSODE uses a heuristic whereby a Jacobian re-evaluation is forced if no such evaluation has occurred in the last twenty steps. This heuristic may have been included to improve the performance of LSODE on problems like B5 in the STIFF DETEST test set. The effect of this heuristic is that LSODE solves B5 very efficiently in terms of function evaluations per step because B5 is a linear problem with a constant Jacobian and LSODE when solving B5 often uses the same formula and step-size for very large numbers of steps. We have a similar heuristic in MODIOUS, mainly to make BDF-MODIOUS more comparable with LSODE and the Jacobian is re-evaluated if at least twenty steps have been taken with the old iteration matrix.

The change in the error coefficient used has not caused any changes in the strategies in our codes.

7.3 RESULTS OF NUMERICAL TESTING

The new codes were run on the STIFF DETEST problem set at the absolute error tolerances of 10^{-2} , 10^{-4} and 10^{-6} . In Table 7.6 we present results for all problems other than B5 while in Table 7.7 we present performance figures for problem B5.

The SDOPT and BDFERR families have the smallest error coefficients and for problems other than B5 the codes SDOPT and BDFERR take only 6.6% and 9.1% more steps than LSODE. The codes SDRROOT and MODIOUS are far less competitive with LSODE on problems other than B5 due to the larger error coefficients of the formulas they use.

LSODE, however, is still more efficient with respect to function evaluations than all of the new codes although the new codes are more efficient than the basic MODIOUS code. For problems other than B5 BDF-MODIOUS averages 1.6 function evaluations per step as compared to BDF-MODIOUS which averages 1.8. The codes using the new formulas average 1.7 function evaluations per step with the exception of SDRROOT which averages 1.5. The new codes still accept or reject steps on a cycle, as

	LSODE	BDF-MODIOUS	MODIOUS	SDRROOT	BDFERR	SDOPT
10^{-2} functions	3290	3936	5320	4769	4260	4483
Jacobians	574	572	650	601	566	562
steps	2285	2134	2976	2926	2284	2296
time (in secs.)	41.12	40.33	49.52	52.13	43.22	43.6
10^{-4} functions	5883	6991	11190	10510	8146	8261
Jacobians	776	772	1059	1004	803	777
steps	4416	4552	6400	6954	4976	4746
time (in secs.)	80.67	81.77	118.1	132.4	98.26	94.10
10^{-6} functions	10664	12935	24769	21226	14819	14715
Jacobians	1094	1088	1725	1512	1137	1161
steps	8410	8122	15078	14074	9228	8962
time (in secs.)	155.0	159.2	276.4	279.3	189.8	182.7
Overall functions	19837	23862	41279	36505	27225	27459
Jacobians	2444	2432	3434	3117	2506	2500
steps	15111	14808	24454	23954	16488	16004
time (in secs.)	276.8	281.3	444.0	463.83	331.3	320.4

Table 7.6: Overall Performance Results (excluding problem B5).

opposed to a step-by-step, basis and this may lead to a greater percentage of steps' being rejected and may, therefore, increase the number of function evaluations necessary per successful step.

For problem B5, the codes using the BDFs are generally outperformed by the codes using the new formulas, as is shown by Table 7.7. The most efficient of the new codes on problem B5 was the code using the BDFERR formulas. Although the SDOPT formulas have $A(\alpha)$ -stability angles which are comparable to those of the BDFERR formulas, the code BDFERR is far more efficient on problem B5, especially at the largest tolerance of 10^{-2} . It is interesting to note that LSODE averages 1.1 function evaluations per step for problem B5 while BDF-MODIOUS averages 1.7 although both codes take very nearly the same number of steps.

	LSODE	BDF-MODIUS	MODIUS	SDRROOT	BDFERR	SDOPT
10 ⁻² functions	2545	3966	225	247	190	4019
Jacobians	120	122	16	17	16	131
steps	2351	2270	126	144	124	2116
time (in secs.)	41.68	57.65	2.550	2.717	2.317	57.05
10 ⁻⁴ functions	2768	3839	667	947	802	685
Jacobians	147	129	37	50	39	36
steps	2397	2352	408	626	530	456
time (in secs.)	49.43	58.52	9.733	16.78	12.77	10.88
10 ⁻⁶ functions	2949	4325	1537	1539	1181	1696
Jacobians	150	136	71	71	51	68
steps	2585	2544	920	1016	836	1050
time (in secs.)	53.65	64.82	28.73	27.38	20.62	26.25
Overall functions	8262	12130	2429	2733	2173	6400
Jacobians	417	387	124	138	106	235
steps	7333	7166	1454	1786	1490	3622
time (in secs.)	144.8	181.0	41.01	46.88	35.71	94.18

Table 7.7: Performance Results for Problem B5

The code using the BDFERR formulas was the most efficient of the new codes. A comparison of the performance of BDFERR to that of LSODE for all problems is given in Table 7.8.

Tolerance	Number of Functions	Number of Jacobians	Number of Steps	CPU time (secs.)
10 ⁻²	4450/5835 (.763:1)	582/694 (.839:1)	2408/4636 (.529:1)	45.53/82.80 (.550:1)
10 ⁻⁴	8958/8651 (1.035:1)	842/923 (.893:1)	5506/6813 (.808:1)	111.3/130.1 (.855:1)
10 ⁻⁶	16000/13613 (1.175:1)	1188/1244 (.959:1)	10064/10995 (.915:1)	210.4/208.8 (1.008:1)
Total	29408/28099 (1.047:1)	2612/2851 (.913:1)	17978/22444 (.801:1)	367.2/421.6 (.871:1)

Table 7.8: BDFERR/LSODE Overall Performance Results

BDFERR takes only 6.6% more steps than LSODE for problems other than B5 and is more efficient than LSODE for problems like B5 which require formulas with high A(α)-stability angles. Therefore, the BDFERR formulas might be suitable for inclusion in a general purpose stiff ODE solver.

CHAPTER 8

CONCLUSIONS

In this thesis several sets of CLMFs with better $A(\alpha)$ -stability angles, order by order, than the BDFs have been derived. These formulas have been derived to have the computationally desirable properties that the number of formulas in the cycle is restricted to two and that each formula in the cycle has the same leading coefficient.

In Chapter 6 the implementation of these formulas was considered. An investigation using the problem set and testing package STIFF DETEST(Enright, 1979) showed that three of the new families of formulas performed efficiently for a problem where BDF-codes generally perform poorly. Of the new families of formulas the BDFERR formulas were probably the most efficient and the overall performance of the code using the BDFERR formulas was comparable to that of the BDF-based code LODE.

The cyclic code was used to implement the BDFs as two-stage CLMFs and a comparison of the performance of this code with LODE indicates that it should still be possible to improve the cyclic implementation. One possibility would be to adopt a different approach to handling error failures. Currently, the code accepts or rejects steps on a cycle by cycle basis where it might be more efficient to accept or reject steps on a step by step basis. It may also be possible to improve the efficiency with which the cyclic code solves implicit equations.

The CLMFs derived are superior in a number of ways to the STINT CLMFs derived by Tendler, Bickart and Pice (1978). In particular, the new formulas use only two stages and have the same leading coefficient for both stages. Although direct comparison of our results with those of Tendler, Bickart and Pice are difficult because the results for STINT are in terms of cycles, not steps, and STINT trades off storage space against execution speed in storing an extra copy of the Jacobian, we believe that the best performed of our families of formulas, the BDFERR formulas, are a significant improvement on the STINT formulas.

On the STIFF DETEST problem set BDFERR takes 13% less time overall than the BDF-code, LODE. This is because LODE performs poorly on problem B5. On problems other than B5 LODE is more efficient than BDFERR. By trading off some of the properties of the new formulas against either improved error behaviour or better $A(\alpha)$ -stability angles it may be possible to find new families of CLMFs that lead to more efficient implementations. For instance, the value of having zero extraneous roots at the origin when solving stiff equations is not clear and further work could also be done in investigating the relative importance of restricting the number of stages to two.

The approach we have adopted to convergence and to stability with respect to the linear initial value problem, $y' = \lambda y$, can readily be applied to cyclic formulas whose constituent formulas are not LMFs. For example, the approach could be applied to cyclic formulas which include blended LMFs, hybrid formulas or singly-implicit composite multistep formulas as stages.

REFERENCES

1. Addison, C. (1977), "Implementing a Stiff Method Based on Second Derivative Formulas", M. Sc. thesis, Department of Computer Science, University of Toronto.
2. Aiken, R. C. (1982), "Stiff Review 1974-1982 : 1. Applications", in Proceedings of the International Conference on Stiff Computation, Park City, Utah.
3. Albrecht (1978), "On the Order of Composite Multistep Methods for Ordinary Differential Equations", Numer. Math., 29, pp 381-396.
4. Alexander, R. (1977), "Diagonally Implicit RK Methods for Stiff Ordinary Differential Equations", SINUM, 14, pp 1006-1021.
5. Axelsson, O. (1969), "A Class of A-stable Methods", BIT, 9, pp 185-199.
6. Bader, G. and Deurlhard, P. (1981), "A Semi-implicit Midpoint Rule for Stiff Differential Systems of Ordinary Differential Equations", SFB123, Technical Report 114, University of Heidelberg.
7. Bedet, R. A., Enright, W. H. and Hull, T. E. (1975), "STIFF DETEST : A Program for Comparing Numerical Methods for Stiff Ordinary Differential Equations", Technical Report No. 81, Department of Computer Science, University of Toronto.
8. Bettis, D. G. (1978), "Efficient Embedded Runge-Kutta Methods", in Numerical Treatment of Differential Equations, Lecture Notes in Mathematics, 631, Springer-Verlag, pp 9-18.
9. Bickart, T. A. (1977), "An Efficient Solution Process for Implicit Runge-Kutta Methods", SINUM, 14, pp 1022-1027.
10. Bickart, T. A. and Picel, Z. (1973), "High Order Stiffly Stable Composite Multistep Methods for Numerical Integration of Stiff Differential Equations", BIT, 13, pp 272-286.
11. Bjurel, G., Dahlquist, G. G., Lindberg, B., Linde, S. and Oden, L. (1970), "Survey of Stiff Ordinary Differential Equations", Report No. NA 70.11, Department of Information Processing, Royal Institute of Technology, Stockholm.
12. Brayton, R. K., Gustavson, F. G. and Hatchel, G. D. (1972), "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward Differentiation Formulas", Proc. IEEE, 60, pp 98-108.
13. Bulirsch, R. and Stoer, J. (1966), "Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods", Numer. Math., 8, pp 1-13.
14. Burrage, K. (1978), "A Special Family of Runge-Kutta Methods for Solving Stiff Differential Equations", BIT, 18, pp 22-41.
15. Burrage, K., Butcher, J. C. and Chipman, F. (1980), "STRIDE: Stable Runge-Kutta Integrator for Differential for Differential Equations", BIT, 20, pp 326-340.
16. Butcher, J. C. (1964a), "Implicit Runge-Kutta Processes", Math. Comp., 18, pp 50-64.
17. Butcher, J. C. (1964b), "Integration Processes Based on Radau Quadrature Formulas", Math. Comp., 18, 233-244.
18. Butcher, J. C. (1966), "On the Convergence of Numerical Solutions to Ordinary Differential Equations", Math. Comp., 20, pp 1-10.
19. Butcher, J. C. (1976), "On the Implementation of Implicit Runge-Kutta Methods", BIT, 16, pp 237-240.
20. Butcher, J. C. (1979), "A Transformed Implicit Runge-Kutta Method", JACM, 26, pp 731-738.
21. Byrne, G. D. and Hindmarsh, A. C. (1975), "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations", ACM-TOMS, 1, pp 71-96.
22. Byrne, G. D., Hindmarsh, A. C., Jackson, K. R. and Brown, H. G. (1977), "A Comparison of Two ODE Codes : GEAR and EPISODE", Comp. and Chem. Engg., 1, pp 133-147.
23. Carver, M. B. (1977), "Efficient Handling of Discontinuities and Time Delays in Ordinary Differential Equation Systems", Proceedings of the International Conference SIMULATION 77, Montreaux, Switzerland, pp 153-158.

24. Carver, M. B. and MacEver, S. R. (1981), "On the use of Sparse Matrix Approximation to the Jacobian in Integrating Large Sets of Ordinary Differential Equations", SIAM J. Sci. Stat. Comp., 2, pp 51-64.
25. Cash, J. R. (1975), "A Class of Implicit Runge-Kutta Methods for the Numerical Integration of Stiff Ordinary Differential Equations", JACM, 22, pp 504-511.
26. Cash, J. R. (1982a), "A Survey of Runge-Kutta Methods for the Numerical Integration of Stiff Differential Systems", in Proceedings of the International Conference on Stiff Computation, Park City, Utah.
27. Cash, J. R. (1982b), "Mono-implicit Runge-Kutta Formulae for the Numerical Solution of Stiff Differential Systems", IMA J. Numer. Anal., 2, pp 289-301.
28. Chipman, F. H. (1971), "A-Stable Runge-Kutta Processes", BIT, 11, pp 384-388.
29. Chipman, F. (1977), "Some Experiments with STRIDE", in "Working Papers for the 1979 SIGNUM Meeting on Numerical Ordinary Differential Equations", R. D. Skeel (Ed.), Technical Report No. UIUCDCS-R-79-763, Department of Computer Science, University of Illinois at Urbana-Champaign.
30. Cooper, J. F. and Sayfy, A. (1979), "Semi-explicit, A-stable Runge-Kutta Methods", Math. Comp., 33, pp 541-556.
31. Curtis, A. R. (1980), "The FACSIMILE Numerical Integrator for Stiff Ordinary Differential Problems", in Computational Techniques for Ordinary Differential Equations, ed. I. Gladwell and D. K. Sayers, Academic Press.
32. Curtis, A. R., Powell, M. J. D. and Reid, J. K. (1974), "On the Estimation of Sparse Jacobian Matrices", JIMA, 13, pp 117-119.
33. Curtis, C. E. and Hirschfelder, J. O. (1952), "Integration of Stiff Equations", Proc. Nat. Acad. Science U. S., 38, pp 235-243.

34. Dahlquist, G. G. (1956), "Convergence and Stability in the Numerical Solution of Ordinary Differential Equations", Math. Scandinavica, 4, pp 33-53.
35. Dahlquist, G. G. (1963), "A Special Stability Property for Linear Multistep Methods", BIT, 3, pp 27-43.
36. Daniel, J. W. and Moore, R. E. (1970), "Computation and Theory in Ordinary Differential Equations", Freeman and Co., p80.
37. Day, J. D. and Murthy, D. N. (1982), "Two Classes of Internally Stable Generalized Runge-Kutta Processes Which Remain Consistent With an Inaccurate Jacobian", Math. Comp., 39, pp 491-509.
38. Descoux, J. (1963), "A Note on a Paper by A. Nordsieck", Report No. 131., Dept. of Computer Science, University of Illinois at Urbana-Champaign.
39. Deuflhard, P. (1980), "Order and Stepsize Control in Extrapolation Methods", SFB 123, Technical Rept. 93, University of Heidelberg.
40. Deuflhard, P., Bader, G. and Nowak U. (1980), "LARKIN - a software package for the numerical simulation of LARGE systems arising in chemical reaction KINETICS", SFB 123, Technical Rept. 100, University of Heidelberg.
41. Donelson, J. III and Hansen, E. (1971), "Cyclic Composite Multistep Predictor-Corrector Methods", SINUM, 8, pp 137-147.
42. Ehle, B. L. (1968), "High Order, A-stable Methods for the Numerical Solution of Systems of Ordinary Differential Equations", BIT, 8, pp 276-278.
43. Ehle, B. L. (1973), "On Padé Approximation to the Exponential Function and A-stable Methods for the Numerical Solution of Initial Value Problems", SIAM Journal of Mathematical Analysis, 4, pp 671-679.
44. Enright, W. H. (1972), "Studies in the Numerical Solution of Stiff Ordinary Differential Equations", Technical Report No. 46, Department of Computer Science, University of Toronto.

45. Enright, W. H. (1974), "Optimal Second Derivative Methods for Stiff Systems", in *Stiff Differential Systems*, R. Willoughby, ed., Plenum Press, pp 95-109.
46. Enright, W. H. (1978), "Improving the Efficiency of Matrix Operations in the Numerical Solution of Stiff Ordinary Differential Equations", ACM-TOMS, 4, pp 127-136.
47. Enright, W. H. (1979), "Using a Test Package for the Automatic Assessment of Methods for ODE's", in "Performance Evaluation of Numerical Software", L. D. Fosdick (Ed.), North Holland, pp 199-213.
48. Enright, W. H., Hull, T. E. (1976a), "Test Results on Initial Value Methods for Non-stiff Ordinary Differential Equations", SINUM, 13, pp 944-961.
49. Enright, W. H. and Hull, T. E. (1976b), "Comparing Numerical Methods for the Solution of Stiff Systems of ODEs Arising in Chemistry", in *Numerical Methods for Differential Systems*, L. Lapidus and W.E.Schlessler, Eds., Academic Press, pp 45-66.
50. Enright, W. H., Hull, T. E. and Lindberg, B. (1975), "Comparing Numerical Methods for Stiff Systems of ODEs", BIT, 15, pp 10-48.
51. Enright, W. H. and Kamei, M. S. (1979), "Partitioning of Stiff Systems and Exploiting the Resulting Structure", ACM-TOMS, 5, pp 374-385.
52. Fehlberg, E. (1969), "Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittwelten-Kontrolle", Computing, 4, pp 93-106.
53. Forsythe, G. E., Malcolm, M. A. and Moler, C. B. (1977), "Computer Methods for Mathematical Computation", Prentice-Hall.
54. Fox, P. A. (1971), "Integration of First Order Systems of Differential Equations", in *Mathematical Software*, J. Rice, Ed., Academic Press, New York, Chapter 9, pp 477-507.
55. Gaffney, P. W. (1982), "A Survey of FORTRAN Subroutines Suitable for Solving Stiff Oscillatory Differential Equations", Technical

- Memorandum ORNL/CSD/TM-134, Oak Ridge National Laboratory, Oak Ridge, Tennessee.
56. Gear, C. W. (1971a), "The Automatic Integration of Ordinary Differential Equations", Comm. ACM, 14, pp 176-190.
 57. Gear, C. W. (1971b), "Algorithm 407 - DIFSUB for solution of Ordinary Differential Equations", Comm. ACM, 14, pp 185-190.
 58. Gear, C. W. (1971c), "Numerical Initial Value Problems in Ordinary Differential Equations", Prentice-Hall Inc.
 59. Gear, C. W. (1980), "Unified Modified Divided Differences, Implementation of Adams and BDF Formulas", Technical Report UIUCDCS-R-80-1014, Department of Computer Science, University of Illinois, Urbana.
 60. Gear, C. W. (1982), "Stiff Software : What do we have and What do we need?", in *Proceedings of the International Conference on Stiff Computation*, Park City, Utah.
 61. Gear, C. W. and Osterby, O. (1981), "Solving Ordinary Differential Equations with Discontinuities", Technical Report UIUCDCS-R-81-1064, Department of Computer Science, University of Illinois, Urbana.
 62. Gear, C. W. and Gallivan, K. A. (1982), "Automatic Methods for Highly Oscillatory Ordinary Differential Equations" *Proceedings of the Biennial Dundee Conference on Numerical Analysis*, Dundee. Lecture Notes in Mathematics, 912, Springer Verlag, Berlin, pp 115-124.
 63. Gear, C. W. and Saad Y. (1982), "Iterative Solution of Linear Equations in ODE Codes", Technical Report UIUCDCS-R-81-1054, Department of Computer Science, University of Illinois, Urbana.
 64. Gear, C. W. and Tu, K. W. (1974), "The Effect of Variable Mesh Size on the Stability of Multistep Methods", SINUM, 11, pp 1025-1043.
 65. Gragg, W. B. (1965), "On Extrapolation Algorithms for Ordinary Initial Value Problems", SINUM, 2, pp 384-403.

- Equations", SINUM, 9, pp 603-637.
78. Hull, T. E., Enright, W. H. and Jackson, K. R. (1976), "User's Guide for DVERK - A Subroutine for Solving Non-Stiff ODE's" Technical Report No. 100, Department of Computer Science, University of Toronto.
 79. Hussels, H. G. (1973), "Schrittenweisensteuerung bei der Integration gewöhnlicher Differentialgleichungen mit Extrapolationsverfahren", M. Sc. Thesis, University of Köln.
 80. Jackson, K. R. and Sacks-Davis, R. (1980), "An Alternative Implementation of Variable Step-Size Multistep Formulas for Stiff ODEs", ACM-TOMS, 6, pp 295-318.
 81. Jackson, L. W. and Sedgwick, A. E. (1977), "Vandermonde Matrices, Coordinate Transformations and Adams' Methods", Technical Report TR77-5, Department of Computing Science, University of Alberta.
 82. Kaps, P. and Rentrop, P. (1979), "Generalized Runge-Kutta Methods of Order Four with Stepsize Control for Stiff Ordinary Differential Equations", Numer. Math., 33, pp 55-68.
 83. Kaps, P. and Wanner, G. (1981), "A Study of Rosenbrock type Methods of High Order", Numer. Math., 38, pp 279-298.
 84. Klopfenstein, R. W. (1971), "Numerical Differentiation Formulas for Stiff Systems of Ordinary Differential Equations", RCA Review, 32, pp 447-462.
 85. Kong, A. K. (1977), "A Search for Better Linear Multistep Methods for Stiff Problems", Technical Report UIUCDCS-R-77-899, Department of Computer Science, University of Illinois at Urbana-Champaign.
 86. Krogh, F. T. (1969), "A Variable Step, Variable Order Multistep Method for the Numerical Solution of ODE's", in Information Processing 68, ed. A.J.H. Morrel, North Holland Publishing Company, pp 194-199.
 87. Krogh, F. T. and Stewart, K. (1982), "Asymptotic ($h \rightarrow \infty$) Stability for BDFs applied to Stiff Equations", Jet Propulsion Laboratory, Computing Memorandum #78, Pasadena, California.

66. Grigorieff, R. D. and Schroll, J. (1978), "Über $A(\alpha)$ -stabile Verfahren hoher Konsistenzordnung", Computing, 20, pp 207-228.
67. Gupta, G. K. (1976), "Some New High-Order Multistep Formulas for Solving Stiff Equations", Math. Comp., 30, pp 417-432.
68. Gupta, G. K. (1980), "A Note about Overhead Costs in ODE Solvers", ACM-TOMS, 6, pp 319-326.
69. Gupta, G. K. (1982), "Description and Evaluation of a Stiff ODE Code DSTIFF", Presented at Int. Conf. on Stiff Computation, Park City, Utah, (also available as Technical Report No. 23, Department of Computer Science, Monash University).
70. Gupta, G. K. and Wallace, C. S. (1975), "Some New Multistep Methods for Solving Ordinary Differential Equations", Math. Comp., 29, pp 489-500.
71. Hairer, E., Bader, G. and Lubich, Ch. (1982), "On the Stability of Semi-implicit Methods for Ordinary Differential Equations", BIT, 22, pp 211-232.
72. Henrici, P. (1962), "Discrete Variable Methods for Ordinary Differential Equations", John Wiley and Sons.
73. Hindmarsh, A. C. (1974), "GEAR : Ordinary Differential Equation System Solver", Report UICD - 30001, Rev. 3, University of California, Lawrence Livermore Laboratories, California.
74. Hindmarsh, A. C. (1980), "LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers", ACM-SIGNUM Newsletter, 15, pp 10-11.
75. Hindmarsh, A. C. (1982), "Stiff System Problems and Solutions at Lawrence Livermore National Laboratories", in Proceedings of the International Conference on Stiff Computation, Park City, Utah.
76. Hull, T.E. and Enright, W. H. (1974), "A Structure for Programs that Solve Ordinary Differential Equations", Technical Report No. 66, Department of Computer Science, University of Toronto.
77. Hull, T. E., Enright, W. H., Fellen, B. M. and Sedgwick, A. E. (1972), "Comparing Numerical Methods for Ordinary Differential

83. Lambert, J. D. (1973), "Computational Methods in Ordinary Differential Equations", John Wiley and Sons.
89. Lambert, J. D. and Sigurdsson, S. T. (1972), "Multistep Methods with Variable Matrix Coefficients", SINUM, 9, pp 715-733.
90. Lindberg, B. (1971), "On Smoothing and Extrapolation for the Trapezoidal Rule", BIT, 11, pp 29-52.
91. Lindberg, B. (1972), "Error Estimate and Step Size Strategy for the Implicit Midpoint Rule with Smoothing and Extrapolation", Report NA 72-59, Department of Information Processing, Royal Institute of Technology, Stockholm.
92. Lindberg, B. (1973), "IMPEX2, a Procedure for Solution of Systems of Stiff Differential Equations", Report NA 73-03, Department of Information Processing, Royal Institute of Technology, Stockholm.
93. Liniger, W. (1976), "High Order A-stable Averaging Algorithms for Stiff Differential Equations", in Numerical Methods for Differential Systems, L. Lapidus and W.E. Schiesser, ed., Academic Press, pp 1-23.
94. Liniger, W. and Odeh, F. (1972), "A-stable Accurate Averaging of Multistep Methods for Stiff Differential Equations", IBM J. Res. Develop., 16, pp 335-348.
95. Mihelcic, M. (1977), "Fast A-stable Donelson-Hansensche zyklische Verfahren zur numerischen Integration von 'stiff' Differentialgleichungssystemen", Angewandte Informatik, 7, pp 299-305.
96. Mishak, R. (1972), "Lineare zyklische Multischrittverfahren hoher Ordnung", Ph.d. thesis, Technical University of Vienna.
97. Nordsieck, A. (1962), "On Numerical Integration of Ordinary Differential Equations", Math. Comp., 16, pp 22-49.
98. Mørseth, S. P. (1974), "Semi-explicit Runge-Kutta Methods", Mathematics and Computation, Report No. 6/74, Mathematics Department, University of Trondheim, Norway.
99. Petzold, L. R. (1978), "An Efficient Numerical Method for Highly Oscillatory Ordinary Differential Equations", SINUM, 18, pp 455-

- 479.
100. Petzold, L. R. (1982a), "Differential/Algebraic Equations are not ODEs", SIAM J. Sci. Stat. Comp., 3, pp 367-384.
101. Petzold, L. R. (1982b), "A Description of DASSL, A Differential-Algebraic System Solver", IMACS World Congress.
102. Petzold, L. R. (1983), "Automatic Selection of Methods for Solving Stiff and Non-stiff Systems of Ordinary Differential Equations", SIAM J. Sci. and Stat. Comp., 4, pp 136-148.
103. Pierce, S. (1974), "Analysis and Optimization of Cyclic Methods for Application in Orbit Calculations", NASA report CR-132826.
104. Piotrowski, P. (1969), "Stability, Consistency and Convergence of Variable, K-step Methods for Numerical Integration of Systems of Ordinary Differential", in Lecture Notes in Mathematics, 109, Springer-Verlag, pp 221-227.
105. Reimer, M. (1968), "Finite Difference Forms Containing Derivatives of Higher Order", SIAM J. Numer. Anal., 13, pp 664-685.
106. Robertson, H.H. and Williams, J. (1975), "Some Problems of Algorithms for Stiff Differential Equations", J. Inst. Math. Appl., 16, pp 23-34.
107. Rosenbrock, H. H. (1963), "Some Implicit Processes for the Numerical Solution of Differential Equations", Computer J., 5, pp 329-330.
108. Rosser, J. B. (1967), "A Runge-Kutta for all Seasons", SIAM Review, 9, pp 417-452.
109. Sacks-Davis, R. (1977), "Error Estimates for a Stiff Ordinary Differential Equation Procedure", Math. Comp., 31, pp 939-953.
110. Sacks-Davis, R. (1980), "Fixed Leading Coefficient Implementation of SD-Formulas for Stiff ODEs", ACM-TOMS, 6, pp 540-562.
111. Sacks-Savis, R. and Shampine, L. F. (1981), "A Type-Insensitive ODE Code Based on Second Derivative Formulas", Comp. & Maths. with Appls., 7, pp 487-495.

112. Sedgwick, A. E. (1973), "An Effective Variable Order Variable Step-size Adams Method", Technical Report No. 53, Department of Computer Science, University of Toronto.
113. Shampine, L. F. (1977), "Stiffness and Non-Stiff Differential Equation Solvers II : Detecting Stiffness with Runge-Kutta Methods", ACM-TOMS, 3, pp 44-53.
114. Shampine, L. F. (1979), "Evaluation of Implicit Formulas for the Solution of ODEs", BIT, 19, pp 495-502.
115. Shampine, L. F. (1980a), "Implementation of Rosenbrock Methods", Technical Report SAND80-2367J, Sandia Laboratories, Albuquerque, New Mexico.
116. Shampine, L. F. (1980b), "Lipschitz Constants and Robust ODE Codes", in Computational Methods in Nonlinear Mechanics, J. T. Oden, Ed., North-Holland.
117. Shampine, L. F. (1980c), "Implementation of Implicit Formulas for the Solution of ODEs", SIAM J. Sci. Stat. Comput., 1, pp 119-130.
118. Shampine, L. F. (1981), "Type-Insensitive ODE Codes based on Implicit A-Stable Methods", Math. Comp., 36, pp 499-509.
119. Shampine, L. F. (1982a), "Type-Insensitive Codes Based on $k(\alpha)$ -stable Formulas", Math. Comp., 39, pp 109-123.
120. Shampine, L. F. (1982b), "Type-Insensitive Codes Based on Extrapolation Methods" Technical Report SAND82-1195, Sandia National Laboratories, Albuquerque, New Mexico.
121. Shampine, L. F. and Gear, C. W. (1979), "A User's View of Solving Stiff Ordinary Differential Equations", SIAM Review, 21, pp 1-17.
122. Shampine, L. F. and Gordon, M. K. (1975), "Computer Solution of Ordinary Differential Equations", W. H. Freeman and Co.
123. Shampine, L. F., Gordon, M. K. and Wisniewski, J. A. (1980), "Variable Order Runge-Kutta Codes", in Computational Techniques for Ordinary Differential Equations, I. Gladwell and D. K. Sayers, Eds., Academic Press.

124. Shampine, L. F. and Watts, H. A. (1969), "Block Implicit One-Step Methods", Math. Comp., 23, pp 731-740.
125. Shampine, L. F. and Watts, H. A. (1977), "The Art of Writing a Runge-Kutta Code, Part I", in Mathematical Software III, J. R. Rice, Ed., Academic Press, pp 257-276.
126. Shampine, L. F. and Watts, H. A. (1979), "The Art of Writing a Runge-Kutta Code, Part II", Appl. Math. Computing, 5, pp 93-121.
127. Shampine, L. F., Watts, H. A. and Davenport, S. M. (1975), "Solving Non-Stiff Ordinary Differential Equations -The State of the Art", SIAM Review, 18, pp 376-411.
128. Singhal, A. (1980), "Implicit Runge-Kutta Formulae for the Numerical Integration of ODE's", Ph.d. thesis, University of London.
129. Skeel, R. D. (1976), "Analysis of Fixed Stepsize Methods", SIAM J. Numer. Anal., 13, pp 664-685.
130. Skeel, R. D. and Kong, A. K. (1977), "Blended Linear Multistep Methods", ACM-TOMS, 3, pp 326-345.
131. Skelboe, S. and Christiansen, B. (1981), "Backward Differentiation Formulas with Extended Regions of Absolute Stability", BIT, 21, pp 221-231.
132. Sloate, H. M. and Bickart, T. A. (1973), "A-Stable Composite Multistep Methods", JACM, 20, pp 7-26.
133. Steihaug, T. and Wolfbrandt, A. (1979), "An Attempt to Avoid Exact Jacobian and Non-linear Equations in the Numerical Solution of Stiff Differential Equations" Math. Comp., 33, pp 521-534.
134. Stetter, H. J. (1973), "Analysis of Discretization Methods of Ordinary Differential Equations", Springer-Verlag, Berlin.
135. Stetter, H. J. (1979), "Interpolation and Error Estimation in Adams PC-Codes", SINUM, 16, pp 311-322.
136. Tendler, J. M., Bickart, T. A. and Picel, Z. (1978), "A Stiffly Stable Integration Process", ACM-TOMS, 4, pp 339-368.

137. Thomson, S. and Tuttle, P. G. (1982), "The Solution of Several Representative Stiff Problems in an Industrial Environment : The Evolution of an ODE Solver", in Proceedings of the International Conference on Stiff Computation, Park City, Utah.
138. Tischer, P.E. (1983), "A Description of the Cyclic Method Stiff ODE Solver, ODIOUS", Technical Note, Department of Computer Science, Monash University.
139. Van Bokhoven, W. M. G. (1980), "Efficient High Order Implicit One-Step Methods for Integration of Stiff Differential Systems", BIT, 20, pp 34-43.
140. Varah, J. M. (1978), "Stiffly Stable Linear Multistep Methods of Extended Order", SINUM, 16, pp 1234-1246
141. Verner, J. H. (1978), "Explicit Runge-Kutta Methods with Estimates of the Local Truncation Error", SINUM, 15, pp 772-790.
142. Verner, J. H. (1979), "Families of Embedded Runge-Kutta Methods", SINUM, 5, pp 857-875.
143. Wallace, C. S. and Gupta, G. K. (1973), "General Linear Multistep Methods to Solve Ordinary Differential Equations", Aust. Comp. J., 5, pp 62-69.
144. Wanner, G., Hairer, E. and Nørsett, S. P. (1978), "Order Stars and Stability Theorems", BIT, 18, pp 475-489.
145. Watts, H. A. and Shampine, L. F. (1971), "A-Stable Block-implicit Methods", BIT, 12, pp 252-266.
146. Widlund, O. B. (1967), "A Note on unconditionally Stable Linear Multistep Methods", BIT, 7, pp 65-70.
147. Williams, J. and DeHoog, F. (1974), "A Class of A-Stable Advanced Multistep Methods", Math. Comp., 28, pp 163-178.
148. Wolfbrandt, A. (1977), "A Study of Rosenbrock Processes with Respect to Order Conditions and Stiff Stability", Ph.d. thesis, Chalmers University of Technology, Göteborg, Sweden.

APPENDIX A

2 step/ 3 step 2nd order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 90.00°

first stage

$\alpha_0 = 0.$ $\beta_0 = 0.$
 $\alpha_1 = 0.2713$ 5034 9946 6529 $\beta_1 = -0.06067$ 5174 9733 2695
 $\alpha_2 = -1.271$ 3503 4994 6654 $\beta_2 = 0.2143$ 2482 5026 6731
 $\alpha_3 = 1$ $\beta_3 = 0.575$

second stage

$\gamma_0 = 0.$ $\delta_0 = -0.04894$ 5326 2459 9904
 $\gamma_1 = 0.$ $\delta_1 = 0.1728$ 9005 2491 9981
 $\gamma_2 = -1.$ $\delta_2 = 0.3010$ 5497 3754 0010
 $\gamma_3 = 1$ $\delta_3 = 0.575$

first stage error coefficient -0.1357 2080 4171 1122
 second stage error coefficient -0.1093 8830 7087 3343
 second local error coefficient -0.2451 0911 1258 4465
 propagated error coefficient -0.2747 9166 6665 6667

Table A1: ODIOUS second order formula

3 step/ 4 step 3rd order cyclic formula

zero roots at infinity
 zero extraneous roots at the origin
 constant leading coefficients
 stability angle 90.00°

$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = -0.7156 \ 8093 \ 2346 \ 3300$	$\beta 1 = -0.8629 \ 8889 \ 7865 \ 1531$
$\alpha 2 = 2.421 \ 4094 \ 9623 \ 2843$	$\beta 2 = 3.166 \ 6349 \ 1230 \ 6544$
$\alpha 3 = -2.705 \ 7285 \ 6388 \ 6513$	$\beta 3 = -4.013 \ 6936 \ 4598 \ 1575$
$\alpha 4 = 1.$	$\beta 4 = 1.72$
<u>first stage</u>	
$\gamma 0 = 0.$	$\delta 0 = 1.894 \ 8388 \ 0135 \ 5351$
$\gamma 1 = 0.$	$\delta 1 = -6.952 \ 8851 \ 6040 \ 9694$
$\gamma 2 = 0.4195 \ 7492 \ 3876 \ 3092$	$\delta 2 = 9.231 \ 4664 \ 5481 \ 5180$
$\gamma 3 = -1.419 \ 5749 \ 2387 \ 6309$	$\delta 3 = -5.312 \ 9950 \ 1963 \ 7147$
$\gamma 4 = 1.$	$\delta 4 = 1.72$
<u>second stage</u>	
first stage error coefficient	-1.244 \ 1079 \ 3765 \ 6965
second stage error coefficient	0.5673 \ 2108 \ 9850 \ 1973
second local error coefficient	-1.198 \ 7833 \ 4104 \ 3101
propagated error coefficient	0.1749 \ 3333 \ 3333 \ 3333

Table A2: ODIOUS third order formula4 step/ 5 step 4th order cyclic formula

zero roots at infinity
 zero extraneous roots at the origin
 constant leading coefficients
 stability angle 90.00°

$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = 0.6103 \ 4611 \ 0027 \ 1559$	$\beta 1 = 0.9215 \ 3663 \ 8176 \ 3085$
$\alpha 2 = -2.747 \ 6858 \ 2319 \ 2436$	$\beta 2 = -4.136 \ 1595 \ 7989 \ 8865$
$\alpha 3 = 4.670 \ 0056 \ 9352 \ 4193$	$\beta 3 = 7.141 \ 5205 \ 4976 \ 9378$
$\alpha 4 = -3.532 \ 6659 \ 8035 \ 8913$	$\beta 4 = -5.652 \ 5699 \ 8526 \ 7611$
$\alpha 5 = 1.$	$\beta 5 = 1.72$
<u>first stage</u>	
$\gamma 0 = 0.$	$\delta 0 = -2.649 \ 9581 \ 7962 \ 1567$
$\gamma 1 = 0.$	$\delta 1 = 11.89 \ 3884 \ 0377 \ 1322$
$\gamma 2 = -0.2716 \ 1471 \ 3953 \ 4076$	$\delta 2 = -20.89 \ 2800 \ 7692 \ 1304$
$\gamma 3 = 1.222 \ 7683 \ 3855 \ 3090$	$\delta 3 = 17.85 \ 5627 \ 4024 \ 0295$
$\gamma 4 = -1.951 \ 1536 \ 2459 \ 9682$	$\delta 4 = -7.606 \ 2914 \ 0192 \ 7840$
$\gamma 5 = 1.$	$\delta 5 = 1.72$
<u>second stage</u>	
first stage error coefficient	-1.255 \ 7944 \ 9088 \ 5898
second stage error coefficient	1.307 \ 8188 \ 4728 \ 9458
second local error coefficient	-1.142 \ 4291 \ 2535 \ 4875
propagated error coefficient	5.422 \ 2222 \ 2222 \ 2222

Table A3: ODIOUS fourth order formula

5 step/ 6 step 5th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 86.64°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = -0.5473$	$\beta_1 = 5248$
$\alpha_2 = 3.046$	$\beta_2 = 9811$
$\alpha_3 = -6.842$	$\beta_3 = 5474$
$\alpha_4 = 7.741$	$\beta_4 = 8458$
$\alpha_5 = -4.398$	$\beta_5 = 9270$
$\alpha_6 = 1.$	$\beta_6 = 8234$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = 2.569$
$\gamma_1 = 0.$	$\delta_1 = -14.02$
$\gamma_2 = 0.2023$	$\delta_2 = 9899$
$\gamma_3 = -1.126$	$\delta_3 = 7071$
$\gamma_4 = 2.473$	$\delta_4 = 9814$
$\gamma_5 = -2.549$	$\delta_5 = 6733$
$\gamma_6 = 1.$	$\delta_6 = 9408$
first stage error coefficient	-1.133
second stage error coefficient	1.347
second local error coefficient	-1.544
propagated error coefficient	11.71

Table A4: ODIUS fifth order formula

6 step/ 7 step 6th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 76.32°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = 0.5065$	$\beta_1 = 9492$
$\alpha_2 = -3.347$	$\beta_2 = 5889$
$\alpha_3 = 9.288$	$\beta_3 = 2648$
$\alpha_4 = -13.85$	$\beta_4 = 4235$
$\alpha_5 = 11.70$	$\beta_5 = 8331$
$\alpha_6 = -5.301$	$\beta_6 = 3655$
$\alpha_7 = 1.$	$\beta_7 = 3402$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = -1.966$
$\gamma_1 = 0.$	$\delta_1 = 12.80$
$\gamma_2 = -0.1622$	$\delta_2 = 5657$
$\gamma_3 = 1.072$	$\delta_3 = 1945$
$\gamma_4 = -2.943$	$\delta_4 = 2473$
$\gamma_5 = 4.228$	$\delta_5 = 0251$
$\gamma_6 = -3.194$	$\delta_6 = 7158$
$\gamma_7 = 1.$	$\delta_7 = 3882$
first stage error coefficient	-0.9495
second stage error coefficient	0.9261
second local error coefficient	-2.107
propagated error coefficient	14.85

Table A5: ODIUS sixth order formula

7 step/ 8 step 7th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 57.66°

first stage

$\alpha 0 = 0.$	$\alpha 0 = 0.$
$\alpha 1 = -0.4730$	$\alpha 1 = -0.5592$
$\alpha 2 = 3.614$	$\alpha 2 = 4.220$
$\alpha 3 = -11.91$	$\alpha 3 = -13.79$
$\alpha 4 = 21.97$	$\alpha 4 = 25.36$
$\alpha 5 = -24.50$	$\alpha 5 = -28.42$
$\alpha 6 = 16.51$	$\alpha 6 = 19.46$
$\alpha 7 = -6.207$	$\alpha 7 = -7.551$
$\alpha 8 = 1.$	$\alpha 8 = 1.275$

second stage

$\gamma 0 = 0.$	$\gamma 0 = 1.627$
$\gamma 1 = 0.$	$\gamma 1 = -12.27$
$\gamma 2 = 0.1354$	$\gamma 2 = 40.28$
$\gamma 3 = -1.034$	$\gamma 3 = -74.96$
$\gamma 4 = 3.390$	$\gamma 4 = 86.48$
$\gamma 5 = -6.137$	$\gamma 5 = -63.47$
$\gamma 6 = 6.516$	$\gamma 6 = 29.33$
$\gamma 7 = -3.870$	$\gamma 7 = -8.238$
$\gamma 8 = 1.$	$\gamma 8 = 1.275$

first stage error coefficient	-0.8295
second stage error coefficient	0.7053
second local error coefficient	-2.505
propagated error coefficient	21.24

Table A6: ODIUS seventh order formula

8 step/ 9 step 8th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 22.15°

first stage

$\alpha 0 = 0.$	$\alpha 0 = 0.$
$\alpha 1 = 0.4453$	$\alpha 1 = 0.4822$
$\alpha 2 = -3.859$	$\alpha 2 = -4.136$
$\alpha 3 = 14.71$	$\alpha 3 = 15.65$
$\alpha 4 = -32.27$	$\alpha 4 = -34.21$
$\alpha 5 = 44.53$	$\alpha 5 = 47.30$
$\alpha 6 = -39.61$	$\alpha 6 = -42.45$
$\alpha 7 = 22.16$	$\alpha 7 = 24.20$
$\alpha 8 = -7.119$	$\alpha 8 = -8.028$
$\alpha 9 = 1.$	$\alpha 9 = 1.18$

second stage

$\gamma 0 = 0.$	$\gamma 0 = -1.384$
$\gamma 1 = 0.$	$\gamma 1 = 12.87$
$\gamma 2 = -0.1162$	$\gamma 2 = -45.07$
$\gamma 3 = 1.007$	$\gamma 3 = 99.29$
$\gamma 4 = -3.827$	$\gamma 4 = -139.7$
$\gamma 5 = 8.302$	$\gamma 5 = 130.5$
$\gamma 6 = -11.16$	$\gamma 6 = -81.18$
$\gamma 7 = 9.369$	$\gamma 7 = 33.00$
$\gamma 8 = -4.570$	$\gamma 8 = -8.408$
$\gamma 9 = 1.$	$\gamma 9 = 1.18$

first stage error coefficient	-0.7390
second stage error coefficient	0.5521
second local error coefficient	-2.825
propagated error coefficient	31.20

Table A7: ODIUS eighth order formula

3 step/ 4 step 3rd order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
fourth order convergent
stability angle 89.99°

<u>first stage</u>	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = -0.7174 \ 2607 \ 3837 \ 5706$	$\beta 1 = -0.8427 \ 5384 \ 6909 \ 6104$
$\alpha 2 = 2.426 \ 1992 \ 8627 \ 9160$	$\beta 2 = 3.108 \ 0060 \ 1040 \ 0179$
$\alpha 3 = -2.708 \ 7732 \ 1244 \ 1589$	$\beta 3 = -3.957 \ 3451 \ 1413 \ 9985$
$\alpha 4 = 1.$	$\beta 4 = 1.700 \ 7458 \ 1204 \ 5397$
<u>second stage</u>	
$\gamma 0 = 0.$	$\delta 0 = 1.815 \ 7601 \ 1730 \ 2048$
$\gamma 1 = 0.$	$\delta 1 = -6.696 \ 3721 \ 1234 \ 8223$
$\gamma 2 = 0.4198 \ 4861 \ 9239 \ 8069$	$\delta 2 = 8.936 \ 2853 \ 7521 \ 5797$
$\gamma 3 = -1.419 \ 8486 \ 1923 \ 9807$	$\delta 3 = -5.176 \ 2678 \ 1145 \ 4826$
$\gamma 4 = 1.$	$\delta 4 = 1.700 \ 7458 \ 1204 \ 5397$
first stage error coefficient	-1.224 6541 7511 7099
second stage error coefficient	0.5075 0799 7724 9758
second local error coefficient	-1.231 3155 4186 1302
propagated error coefficient	0.

Table B1: SDRROOT third order formula

4 step/ 5 step 4th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
fifth order convergent
stability angle 84.88°

<u>first stage</u>	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = 0.6310 \ 0639 \ 1454 \ 3900$	$\beta 1 = 0.6512 \ 6127 \ 0645 \ 2652$
$\alpha 2 = -2.831 \ 0625 \ 3603 \ 4188$	$\beta 2 = -3.076 \ 7097 \ 9686 \ 2437$
$\alpha 3 = 4.784 \ 9492 \ 3060 \ 0037$	$\beta 3 = 5.592 \ 0190 \ 2906 \ 1678$
$\alpha 4 = -3.584 \ 8930 \ 8602 \ 0283$	$\beta 4 = -4.644 \ 6753 \ 8778 \ 8333$
$\alpha 5 = 1.$	$\beta 5 = 1.462 \ 2615 \ 5204 \ 8996$
<u>second stage</u>	
$\gamma 0 = 0.$	$\delta 0 = -1.430 \ 3846 \ 1209 \ 4731$
$\gamma 1 = 0.$	$\delta 1 = 6.757 \ 4697 \ 7085 \ 3677$
$\gamma 2 = -0.2745 \ 7990 \ 0858 \ 3441$	$\delta 2 = -12.54 \ 1509 \ 4326 \ 0704$
$\gamma 3 = 1.231 \ 9255 \ 0978 \ 1826$	$\delta 3 = 11.42 \ 7606 \ 5999 \ 0675$
$\gamma 4 = -1.957 \ 3456 \ 0892 \ 3482$	$\delta 4 = -5.358 \ 2095 \ 8617 \ 2786$
$\gamma 5 = 1.$	$\delta 5 = 1.462 \ 2615 \ 5204 \ 8996$
first stage error coefficient	-0.9951 7445 8073 4467
second stage error coefficient	0.3461 9242 8766 5517
second local error coefficient	-1.601 7079 2685 6315
propagated error coefficient	0.

Table B2: SDRROOT fourth order formula

5 step/ 6 step 5th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
sixth order convergent
stability angle 70.63°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = -0.5688$	$\beta_1 = -0.5358$
$\alpha_2 = 3.158$	$\beta_2 = 3.080$
$\alpha_3 = -7.067$	$\beta_3 = -7.218$
$\alpha_4 = 7.948$	$\beta_4 = 8.645$
$\alpha_5 = -4.470$	$\beta_5 = -5.297$
$\alpha_6 = 1.$	$\beta_6 = 1.311$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = 1.203$
$\gamma_1 = 0.$	$\delta_1 = -6.918$
$\gamma_2 = 0.2050$	$\delta_2 = 16.39$
$\gamma_3 = -1.138$	$\delta_3 = -20.47$
$\gamma_4 = 2.492$	$\delta_4 = 14.34$
$\gamma_5 = -2.558$	$\delta_5 = -5.689$
$\gamma_6 = 1.$	$\delta_6 = 1.311$
first stage error coefficient	-0.8514
second stage error coefficient	0.2600
second local error coefficient	-1.918
propagated error coefficient	0.

Table B3: SDRROOT fifth order formula

6 step/ 7 step 6th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
seventh order convergent
stability angle 44.70°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = 0.5216$	$\beta_1 = 0.4582$
$\alpha_2 = -3.442$	$\beta_2 = -3.101$
$\alpha_3 = 9.534$	$\beta_3 = 8.869$
$\alpha_4 = -14.18$	$\beta_4 = -13.76$
$\alpha_5 = 11.93$	$\beta_5 = 12.24$
$\alpha_6 = -5.364$	$\beta_6 = -5.924$
$\alpha_7 = 1.$	$\beta_7 = 1.205$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = -1.052$
$\gamma_1 = 0.$	$\delta_1 = 7.122$
$\gamma_2 = -0.1638$	$\delta_2 = -20.51$
$\gamma_3 = 1.081$	$\delta_3 = 32.55$
$\gamma_4 = -2.964$	$\delta_4 = -30.82$
$\gamma_5 = 4.249$	$\delta_5 = 17.67$
$\gamma_6 = -3.202$	$\delta_6 = -6.085$
$\gamma_7 = 1.$	$\delta_7 = 1.205$
first stage error coefficient	-0.7518
second stage error coefficient	0.2070
second local error coefficient	-2.201
propagated error coefficient	0.

Table B4: SDRROOT sixth order formula

7 step/ 8 step 7th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
eighth order convergent
no longer A(0)-stable

<u>first stage</u>	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = -0.4842 \ 9432 \ 2347 \ 1949$	$\beta 1 = -0.4022 \ 4696 \ 6266 \ 0178$
$\alpha 2 = 3.697 \ 3339 \ 6745 \ 2743$	$\beta 2 = 3.130 \ 6094 \ 0622 \ 9509$
$\alpha 3 = -12.17 \ 5528 \ 0201 \ 0069$	$\beta 3 = -10.55 \ 8154 \ 9600 \ 7202$
$\alpha 4 = 22.42 \ 7412 \ 9328 \ 3479$	$\beta 4 = 20.04 \ 2490 \ 5774 \ 3559$
$\alpha 5 = -24.95 \ 5284 \ 8775 \ 9557$	$\beta 5 = -23.18 \ 0355 \ 2548 \ 4687$
$\alpha 6 = 16.75 \ 5682 \ 2045 \ 4987$	$\beta 6 = 16.36 \ 6337 \ 4465 \ 7214$
$\alpha 7 = -6.265 \ 3218 \ 8479 \ 3958$	$\beta 7 = -6.530 \ 6495 \ 5708 \ 6382$
$\alpha 8 = 1.$	$\beta 8 = 1.125 \ 8262 \ 3739 \ 3152$
<u>second stage</u>	
$\gamma 0 = 0.$	$\delta 0 = .9438 \ 4108 \ 2865 \ 7327$
$\gamma 1 = 0.$	$\delta 1 = -7.345 \ 7303 \ 3111 \ 4840$
$\gamma 2 = 0.1365 \ 0785 \ 2713 \ 1970$	$\delta 2 = 24.88 \ 6600 \ 8231 \ 9065$
$\gamma 3 = -1.042 \ 1660 \ 9068 \ 2980$	$\delta 3 = -47.90 \ 5378 \ 8366 \ 2375$
$\gamma 4 = 3.411 \ 7881 \ 7487 \ 7273$	$\delta 4 = 57.33 \ 3947 \ 7184 \ 1611$
$\gamma 5 = -6.167 \ 9761 \ 9454 \ 6205$	$\delta 5 = -43.89 \ 7977 \ 2191 \ 6693$
$\gamma 6 = 6.539 \ 5823 \ 0282 \ 7825$	$\delta 6 = 21.42 \ 0038 \ 2449 \ 2905$
$\gamma 7 = -3.877 \ 7360 \ 4518 \ 9110$	$\delta 7 = -6.514 \ 0370 \ 4909 \ 0453$
$\gamma 8 = 1.$	$\delta 8 = 1.125 \ 8262 \ 3739 \ 3152$

first stage error coefficient -0.6781 6377 4242 8791
second stage error coefficient 0.1713 6107 3040 7642
second local error coefficient -2.458 3790 3888 4015
propagated error coefficient 0.

Table B5: SDRROOT seventh order formula

4 step/ 5 step 4th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 85.85°

<u>first stage</u>	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = 0.6289 \ 0168 \ 3259 \ 4980$	$\beta 1 = 0.6758 \ 9082 \ 8134 \ 9419$
$\alpha 2 = -2.822 \ 5432 \ 9551 \ 8783$	$\beta 2 = -3.173 \ 0340 \ 1776 \ 2222$
$\alpha 3 = 4.773 \ 1630 \ 7085 \ 9208$	$\beta 3 = 5.732 \ 4537 \ 6541 \ 1226$
$\alpha 4 = -3.579 \ 5214 \ 6359 \ 9923$	$\beta 4 = -4.735 \ 7028 \ 7623 \ 5470$
$\alpha 5 = 1.$	$\beta 5 = 1.485 \ 6107 \ 5585 \ 1387$
<u>second stage</u>	
$\gamma 0 = 0.$	$\delta 0 = -1.525 \ 0859 \ 5496 \ 1105$
$\gamma 1 = 0.$	$\delta 1 = 7.159 \ 6616 \ 1052 \ 3927$
$\gamma 2 = -0.2742 \ 9077 \ 1251 \ 4020$	$\delta 2 = -13.20 \ 3397 \ 0035 \ 6231$
$\gamma 3 = 1.231 \ 0311 \ 6412 \ 5085$	$\delta 3 = 11.94 \ 6829 \ 7116 \ 8892$
$\gamma 4 = -1.956 \ 7403 \ 9287 \ 3682$	$\delta 4 = -5.546 \ 0687 \ 4116 \ 3093$
$\gamma 5 = 1.$	$\delta 5 = 1.485 \ 6107 \ 5585 \ 1387$
first stage error coefficient	-1.018 8189 5367 7511
second stage error coefficient	0.4175 2417 9593 5638
second local error coefficient	-1.576 0400 2009 2524
propagated error coefficient	0.4

Table C1: BDFERR fourth order formula

5 step/ 6 step 5th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 72.82°

<u>first stage</u>	
a0 = 0.	a0 = 0.
a1 = -0.5679 4885 2860 8239	a1 = -0.5467 5443 3335 7476
a2 = 3.154 0977 1759 9855	a2 = 2.134 3875 1642 1872
a3 = -7.057 5981 6845 4791	a3 = -7.323 6509 9001 7264
a4 = 7.939 3416 3583 2696	a4 = 8.746 2658 3116 8717
a5 = -4.467 8923 3211 6940	a5 = -5.348 3230 1869 3267
a6 = 1.	a6 = 1.321 4320 5437 6310
<u>second stage</u>	
y0 = 0.	y0 = 1.244 7184 6538 9449
y1 = 0.	y1 = -7.135 6166 1598 9492
y2 = 0.2049 7717 5674 0842	y2 = 16.86 1018 6620 4665
y3 = -1.138 3384 9420 9647	y3 = -20.99 5401 2893 0655
y4 = 2.491 8902 7636 8578	y4 = 14.65 6387 3435 6182
y5 = -2.558 5289 5783 3015	y5 = -5.782 6834 3501 9718
y6 = 1.	y6 = 1.321 4320 5437 6310
first stage error coefficient	-0.8619 8671 2815 6174
second stage error coefficient	0.2910 6558 1358 8138
second local error coefficient	-1.914 3523 8464 7234
propagated error coefficient	0.3333 3333 3333 3333

Table C2: BDFERR fifth order formula

6 step/ 7 step 6th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 47.83°

<u>first stage</u>	
a0 = 0.	a0 = 0.
a1 = 0.5212 5384 5518 9155	a1 = 0.4631 3381 3635 5253
a2 = -3.446 1494 0382 2430	a2 = -3.130 3818 5814 9741
a3 = 9.528 0114 1322 8203	a3 = 8.941 9129 6291 8089
a4 = -14.17 3798 1365 3174	a4 = -13.85 4587 8086 3715
a5 = 11.92 7950 2802 3431	a5 = 12.31 2135 9611 4518
a6 = -5.363 2679 9862 7259	a6 = -5.952 0218 6326 3602
a7 = 1.	a7 = 1.209 7489 3319 1404
<u>second stage</u>	
y0 = 0.	y0 = -1.070 8016 6546 4155
y1 = 0.	y1 = 7.237 6881 3365 7788
y2 = -0.1638 3695 8524 3829	y2 = -20.81 6842 6921 3597
y3 = 1.081 2843 3015 3269	y3 = 32.99 5658 3030 6778
y4 = -2.963 5232 9548 9719	y4 = -31.19 2599 0244 5306
y5 = 4.248 7272 8712 6063	y5 = 17.85 9311 2958 5580
y6 = -3.202 6513 6326 5230	y6 = -6.132 3491 3622 8489
y7 = 1.	y7 = 1.209 7489 3319 1404
first stage error coefficient	-0.7566 1181 2252 3408
second stage error coefficient	0.2208 7427 6016 1635
second local error coefficient	-2.022 8957 5956 3720
propagated error coefficient	0.2857 1428 5714 2857

Table C3: BDFERR sixth order formula

3 step/ 4 step 3rd order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 90.00°

first stage	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = -0.7136 \ 8751 \ 0275 \ 4246$	$\beta 1 = -0.8866 \ 0293 \ 4709 \ 3741$
$\alpha 2 = 2.415 \ 9414 \ 3718 \ 8289$	$\beta 2 = 3.235 \ 1040 \ 6260 \ 8995$
$\alpha 3 = -2.702 \ 2539 \ 2691 \ 2864$	$\beta 3 = -4.079 \ 5614 \ 3577 \ 0600$
$\alpha 4 = 1.$	$\beta 4 = 1.742 \ 4938 \ 9123 \ 3540$
second stage	
$\gamma 0 = 0.$	$\delta 0 = 1.989 \ 4234 \ 1262 \ 9001$
$\gamma 1 = 0.$	$\delta 1 = -7.259 \ 1591 \ 0322 \ 9482$
$\gamma 2 = 0.4192 \ 6031 \ 0692 \ 7402$	$\delta 2 = 9.582 \ 9117 \ 0445 \ 9131$
$\gamma 3 = -1.419 \ 2603 \ 1069 \ 2710$	$\delta 3 = -5.474 \ 9302 \ 1578 \ 4931$
$\gamma 4 = 1.$	$\delta 4 = 1.742 \ 4938 \ 9123 \ 3540$
first stage error coefficient	-1.266 8296 6468 4028
second stage error coefficient	0.6393 9870 1007 6552
second local error coefficient	-1.158 5623 6248 6578
propagated error coefficient	0.3861 8745 4162 1814

Table D1: SDOPT third order formula

4 step/ 5 step 4th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
stability angle 85.85°

first stage	
$\alpha 0 = 0.$	$\beta 0 = 0.$
$\alpha 1 = 0.6290 \ 6774 \ 9002 \ 7619$	$\beta 1 = 0.6739 \ 2517 \ 1307 \ 8078$
$\alpha 2 = -2.823 \ 2152 \ 3118 \ 1439$	$\beta 2 = -3.165 \ 3566 \ 0351 \ 4983$
$\alpha 3 = 4.774 \ 0922 \ 9891 \ 2432$	$\beta 3 = 5.721 \ 2572 \ 2513 \ 9306$
$\alpha 4 = -3.579 \ 9448 \ 1673 \ 3755$	$\beta 4 = -4.728 \ 4429 \ 6612 \ 8498$
$\alpha 5 = 1.$	$\beta 5 = 1.483 \ 7490 \ 8963 \ 8528$
second stage	
$\gamma 0 = 0.$	$\delta 0 = -1.517 \ 4208 \ 6668 \ 3409$
$\gamma 1 = 0.$	$\delta 1 = 7.127 \ 1366 \ 3134 \ 6429$
$\gamma 2 = -0.2743 \ 1369 \ 4186 \ 5522$	$\delta 2 = -13.14 \ 9931 \ 3987 \ 9704$
$\gamma 3 = 1.231 \ 1020 \ 5661 \ 7627$	$\delta 3 = 11.90 \ 4973 \ 8019 \ 9125$
$\gamma 4 = -1.956 \ 7883 \ 6243 \ 1075$	$\delta 4 = -5.530 \ 9759 \ 3174 \ 0281$
$\gamma 5 = 1.$	$\delta 5 = 1.483 \ 7490 \ 8963 \ 8528$
first stage error coefficient	-1.016 9340 0484 2751
second stage error coefficient	0.4117 2237 3603 5496
second local error coefficient	-1.578 2022 5243 3172
propagated error coefficient	0.3674 4070 3842 0004

Table D2: SDOPT fourth order formula

5 step/ 6 step 5th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant loading coefficients
stability angle 72.82°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = -0.5672$	$\beta_1 = -0.5546$
$\alpha_2 = 3.150$	$\beta_2 = -3.173$
$\alpha_3 = -7.050$	$\beta_3 = -7.400$
$\alpha_4 = 7.933$	$\beta_4 = 8.823$
$\alpha_5 = -4.465$	$\beta_5 = -5.384$
$\alpha_6 = 1.$	$\beta_6 = 1.328$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = 1.275$
$\gamma_1 = 0.$	$\delta_1 = -7.295$
$\gamma_2 = 0.2049$	$\delta_2 = 17.20$
$\gamma_3 = -1.137$	$\delta_3 = -21.37$
$\gamma_4 = 2.491$	$\delta_4 = 14.88$
$\gamma_5 = -2.558$	$\delta_5 = -5.850$
$\gamma_6 = 1.$	$\delta_6 = 1.328$
first stage error coefficient	-0.8696
second stage error coefficient	0.3139
second local error coefficient	-1.910
propagated error coefficient	0.5800

Table D3: SDOPT fifth order formula

6 step/ 7 step 6th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant loading coefficients
stability angle 47.83°

<u>first stage</u>	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = 0.5207$	$\beta_1 = 0.4694$
$\alpha_2 = -3.437$	$\beta_2 = -3.167$
$\alpha_3 = 9.519$	$\beta_3 = 9.034$
$\alpha_4 = -14.16$	$\beta_4 = -13.97$
$\alpha_5 = 11.92$	$\beta_5 = 12.40$
$\alpha_6 = -5.361$	$\beta_6 = -5.987$
$\alpha_7 = 1.$	$\beta_7 = 1.215$
<u>second stage</u>	
$\gamma_0 = 0.$	$\delta_0 = -1.094$
$\gamma_1 = 0.$	$\delta_1 = 7.387$
$\gamma_2 = -0.1637$	$\delta_2 = -21.21$
$\gamma_3 = 1.080$	$\delta_3 = 33.56$
$\gamma_4 = -2.962$	$\delta_4 = -31.67$
$\gamma_5 = 4.248$	$\delta_5 = 18.09$
$\gamma_6 = -3.202$	$\delta_6 = -6.192$
$\gamma_7 = 1.$	$\delta_7 = 1.215$
first stage error coefficient	-0.7626
second stage error coefficient	0.2388
second local error coefficient	-2.203
propagated error coefficient	0.6573

Table D4: SDOPT sixth order formula

7 step/ 8 step 7th order cyclic formula

zero roots at infinity
zero extraneous roots at the origin
constant leading coefficients
no longer A(0)-stable

first stage	
$\alpha_0 = 0.$	$\beta_0 = 0.$
$\alpha_1 = -0.4835$	$\beta_1 = -0.4119$
$\alpha_2 = 3.691$	$\beta_2 = 3.197$
$\alpha_3 = -12.15$	$\beta_3 = -10.75$
$\alpha_4 = 22.39$	$\beta_4 = 20.37$
$\alpha_5 = -24.92$	$\beta_5 = -23.50$
$\alpha_6 = 16.73$	$\beta_6 = 16.55$
$\alpha_7 = -6.261$	$\beta_7 = -6.593$
$\alpha_8 = 1.$	$\beta_8 = 1.135$
second stage	
$\gamma_0 = 0.$	$\delta_0 = 0.9805$
$\gamma_1 = 0.$	$\delta_1 = -7.612$
$\gamma_2 = 0.1364$	$\delta_2 = 25.72$
$\gamma_3 = -1.041$	$\delta_3 = -49.38$
$\gamma_4 = 3.410$	$\delta_4 = 58.94$
$\gamma_5 = -6.166$	$\delta_5 = -44.99$
$\gamma_6 = 6.538$	$\delta_6 = 21.87$
$\gamma_7 = -3.877$	$\delta_7 = -6.615$
$\gamma_8 = 1.$	$\delta_8 = 1.135$
first stage error coefficient	-0.6875
second stage error coefficient	0.1988
second local error coefficient	-2.466
propagated error coefficient	1.096

Table D5: SDOPT seventh order formula

APPENDIX B

In this appendix we prove that the propagated error coefficient (definition 4.4.1) for LMFs is equal to the local truncation error coefficient scaled by the sum of the beta coefficients.

For LMFs expressed in the form (2.3.2) the matrix $\mathbf{A}$ is in Frobenius form and is the companion matrix to the $\rho(r)$ polynomial defined in (2.2.5). It is trivial to prove that if $\rho(z) = 0$ then $\mathbf{A}$ has the eigenvalue z with a corresponding eigenvector of the form

$$[1, z, z^2, \dots, z^{k-1}]^T.$$

Thus for consistent LMFs $\rho(1) = 0$ and $\mathbf{e}_1$ is of the form

$$[1, 1, 1, \dots, 1]^T.$$

Theorem B.1: Propagated error of LMFs

For a strongly D-stable, k-step LMF with coefficients $\alpha_i, \beta_i, i = 0, 1, \dots, k, \alpha_k = 1$, and local truncation error coefficient c_{q+1} , the propagated error coefficient, D_{q+1} , is equivalent to the scaled local truncation error coefficient,

$$\frac{c_{q+1}}{k \sum_{i=0}^k \beta_i}.$$

(In the case where the denominator is zero the LMF has $\alpha(1) = 0$. If the LMF is consistent this implies $\rho'(1) = 0$ and this, in turn, implies that the method is not strongly D-stable as it has an extraneous root at the origin equal to 1).

Proof: We can determine $\mathbf{p}$ from $\mathbf{A}^T \mathbf{p} = \mathbf{p}$ which defines a set of equations of the following form:

$$\begin{bmatrix} 0 & 0 & 0 & \dots & -\alpha_0 \\ 1 & 0 & 0 & \dots & -\alpha_1 \\ 0 & 1 & 0 & \dots & -\alpha_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -\alpha_{k-1} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_k \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_k \end{bmatrix} \quad (B.1)$$

Thus we have

$$\begin{aligned} p_1 &= -\alpha_0 p_k \\ p_2 &= p_1 - \alpha_1 p_k = -(\alpha_0 + \alpha_1) p_k \\ &\vdots \\ p_i &= -(\sum_{j=0}^{i-1} \alpha_j) p_k \\ &\vdots \end{aligned}$$

In particular, the last equation in (B.1) gives us

$$p_k = -(\sum_{j=0}^{k-1} \alpha_j) p_k.$$

Since the formula is consistent we have $p(1) = 0$ so

$$\sum_{j=0}^k \alpha_j = 0 \quad \Leftrightarrow \quad \alpha_k = -\sum_{j=0}^{k-1} \alpha_j \quad \Leftrightarrow \quad 1 = -\sum_{j=0}^{k-1} \alpha_j. \quad (B.2)$$

Therefore the system of equations is consistent for all real p_k . Without loss of generality we set p_k to 1. We find that $\underline{p}^T \underline{t}$ is just c_{p+1} .

$$\underline{p}^T \underline{e}_1 = -\sum_{j=0}^k \sum_{i=1}^{k-1} \alpha_j = -\sum_{j=0}^k \sum_{i=j+1}^{k-1} \alpha_j = -\sum_{j=0}^{k-1} (k-j) \alpha_j$$

$$= \sum_{j=0}^{k-1} j \alpha_j - k \sum_{j=0}^{k-1} \alpha_j$$

$$= \sum_{j=0}^{k-1} j \alpha_j + k \alpha_k \quad \text{using (B.2)}$$

$$= \sum_{j=1}^k j \alpha_j.$$

By consistency $p'(1) = 0(1)$ so we have

$$\underline{p}^T \underline{e}_1 = \sum_{j=1}^k j \alpha_j = \sum_{j=0}^k j \beta_j$$

so that we have the desired result, namely

$$D_{q+1} = \frac{\underline{p}^T \underline{t}}{\underline{p}^T \underline{e}_1} = \frac{c_{q+1}}{\sum_{j=0}^k j \beta_j}$$

If we consider the M-stage CLMF where all stages are identical then we have the following result connecting the error propagation coefficients of the LMF and CLMF. Since

$$\underline{\lambda} \underline{e}_1 = \underline{\mu} \underline{e}_1 \Rightarrow (\underline{\lambda})^j \underline{e}_1 = \underline{\mu}^j \underline{e}_1$$

the M-stage CLMF will have the same $\underline{p}$ and $\underline{e}_1$ as the LMF. The local discretization error vector for the CLMF, $\underline{t}$, is given by

$$\underline{t} = \sum_{j=0}^{M-1} (\underline{\lambda})^j \underline{t}$$

where $\underline{\lambda}$ and $\underline{t}$ are the matrix and local discretization error vector characteristic of the LMF. Therefore

$$\underline{p}^T \underline{t} = \sum_{j=0}^{M-1} \underline{p}^T (\underline{\lambda})^j \underline{t} = M \underline{p}^T \underline{t},$$

and the local propagated error coefficient of the cyclic formula is M times that of the LMF. This is only to be expected since the cyclic formula is calculating the solution at M steps. This suggests that for an M-stage CLMF we should weight the propagated error coefficient by $1/M$.