DUBLIN CITY UNIVERSITY, DUBLIN
School of Mathematical Sciences
M. Sc. THESIS

THE NUMERICAL SOLUTION OF ORDINARY AND ALGEBRAIC DIFFERENTIAL EQUATIONS USING ONE STEP METHODS

by

Gerard Keogh B. Sc.

Supervisor: Dr. John Carroll, School of Mathematical Sciences

This Thesis is based on the candidates own work
September 1990

Acknowledgements.

I would like to acknowledge and thank the following for their help and assistance while doing this work
My supervisor Dr John Carroll
The staff at D C U , especially Pauline O' Gorman
The postgraduate students of mathematics at D C U
The staff of the Mathematics Dept at Cork R T C
My parents and family, especially my sister Maureen
Finally, Siobhan, for her kindness, love and care over many months, a very special thank you my darling

Abstract

This thesis addresses the problem of finding numerical solutions to ordinary and algebraic differential equation systems Our primary focus is the application of one-step numerical schemes to these problem classes

Firstly we concentrate on the narrower class of explicit Ordinary Differential Equation (ODE) systems We analyse the theory necessary develop efficient algorithms based on our chosen one-step numerical schemes These algorithms are then applied to the solution of a standard test set of ODE systems The results are then compared with those obtained using standard software packages on the same problem test set

Our theory is then extended to include the wider class of Algebraic Differential Equation (more commonly called Differential Algebraic Equation (DAE)) systems Based on this theory, we are able to adapt our one-step schemes to solve this harder class of problem Once again the resulting algorithms are tested on a selection of problems and results are compared with those obtained from standard software packages On all problems considered, we demonstrate that our techniques can often provide efficient alternatives to the more complex methods adopted in the standard software packages designed for these problem classes

Contents

Chapter 1

Introduction.

1.1 Systems of differential equations.

The time behaviour of many natural and technical processes, can be described by systems of Ordinary Differential Equations (ODEs) In general, two types of systems arise. The explicit first order system

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \qquad t \in [a, b] \tag{1 1}$$

with $\mathbf{y}(a)$ given For this system, $\mathbf{y}(t)$, $\mathbf{y}'(t)$ and $\mathbf{f}(\ )$ are $n - dimensional$ vectors The second system is the general implicit ODE given by

$$\mathbf{F}(t, \mathbf{y}(t), \mathbf{y}'(t)) = 0 \qquad t \in [a, b] \tag{1 2}$$

with both $\mathbf{y}(a)$ and $\mathbf{y}'(a)$ given. Once again $\mathbf{y}(t), \mathbf{y}'(t)$ and $\mathbf{F}(\ )$ are $n - dimensional$ vectors

Typically explicit systems (1 1) and implicit systems (1.2) arise in similar areas For example, electronic circuits can be modelled by systems of ODEs Dynamic elements such as capacitors and inductors generate differential equations while the inclusion of static elements in the circuit give rise to algebraic equations The algebraic equations are coupled to the differential equations forming a Differential Algebraic Equation (DAE) system, see Campbell [12]

Control problems, solved by variational techniques, provide us with another example of ordinary differential systems In some cases, the Euler-Lagrange equations lead to explicit systems However, the best known control problem, is the linear quadratic regulator [1]

$$\mathbf{x}' = A\mathbf{x} - B\mathbf{u} \qquad \mathbf{x}(a) = \mathbf{x}_a$$

with the associated cost functional

$$J(u) = \int_a^b \left\{ \mathbf{x}^t H \mathbf{x} + \mathbf{u}^t Q \mathbf{u} \right\} dt$$

The matrices H and Q are symmetric and positive definite Using the theory of Lagrange multipliers, the necessary conditions for a minimum, see Campbell & Meyer

[1] We drop the dependence of $\mathbf{x}$ and $\mathbf{u}$ on t for clarity

[11], are

$$\begin{aligned} \mathbf{x}' &= A\mathbf{x} + B\mathbf{u} & \mathbf{x}(a) &= \mathbf{x}_a \\ \lambda' &= -A^t\lambda - H\mathbf{x} & \lambda(b) &= 0 \\ 0 &= B^t\lambda + Q\mathbf{u} \end{aligned}$$

Once again, we obtain a system of differential algebraic eqautions

When a system of time dependent Partial Differential Equations (PDEs), are solved using an approximate technique, such as finite differences or finite elements [59], a system of differential equations arises If the system is a coupled system of elliptic and parabolic PDEs, then the resulting equations generated by the approximate technique are differential algebraic, see Petzold & Lotstedt [59]

The final example we introduce is the singularly perturbed scalar differential system

$$\begin{aligned} y' &= g(t, y, z, \epsilon) \\ \epsilon z' &= h(t, y, z, \epsilon) \qquad t \in [a, b] \end{aligned} \tag{1 3}$$

with both $y(a)$ and $z(a)$ given Usually, ϵ is a small parameter with $|\epsilon| \ll 1$ Systems of this form prove to be unsuitable for numerical solution The reason for this is that the perturbation parameter, ϵ may take a value smaller than the smallest machine representable number Then the qualitative assessment of the solution becomes important The limiting case $\epsilon = 0$ must be understood and the resulting DAE system solved numerically

1.2 Objectives and review.

In this thesis, we concern ourselves with the solution of explicit ODE systems [2] (1 1) and DAE systems, which can be written in the form

$$E\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)) \qquad t \in [a, b] \tag{1 4}$$

with $\mathbf{y}(a)$ and $\mathbf{y}'(a)$ given In general the matrix E is singular To this end, we will draw on theoretical results for the analytic solution of both (1 1) and (1.4), where necessary Our main objective is to use the theory given in this work to develop numerical methods for the solution of ODE and DAE systems We evaluate the performance of the techniques we propose against some standard algorithms available for the numerical solution of these problems

In Chapter 2, we study explicit ODEs and their numerical solution We concentrate on 'stiff' ODE systems These problems are similar to the singularly perturbed systems, but they are suitable for numerical solution Concepts of convergence, order of accuracy and stability will be discussed for numerical methods applied to ODEs We show how numerical methods can be implemented to solve explicit ODE systems

Chapter 3 develops the one step numerical methods that form the core of this thesis Again, we analyze the accuracy and stability of these schemes We give two algorithms based on the one-step formula proposed Finally, we test them on some well known problems that have appeared in the literature

[2] We simply call these ODEs for the remainder of this thesis

The emphasis changes in Chapter 4, where we consider theoretical aspects of DAEs Two important topics are addressed in Chapter 4. the index, or degree of complexity of a DAE and the difficulties associated with initial conditions We outline a selection of techniques that have appeared in the literature for dealing with these problems

Once again in Chapter 5 we return to numerical methods. We explain why some DAEs are solvable by numerical methods suitable for explicit ODEs and others are not We show that the index or degree of complexity of a DAE, determines both the accuracy and stability of a particular numerical scheme.

Chapter 6 parallels Chapter 3 We extend our one step schemes to the solution of systems of the form (1.4). Again, we study the accuracy of these schemes, using the theory developed in Chapter 5. We outline how we intend to change our one step schemes so that they can be used to solve DAEs We then consider two well known algorithms, which are available as Fortran routines for the numerical solution of DAE systems These algorithms are called DASSL (Differential Algebraic System Solver) and LSODI (Livermore Solver for Implicit ODEs) Finally, a wide selection of test problems are proposed and solved using all algorithms outlined in this thesis

The last Chapter, discusses how successful we feel our software has been in solving the problems considered We discuss where we feel progress can be made in the future in solving DAE systems and close with a discussion of possible extensions of DAE type problems which to our knowledge have not appeared in the literature

Chapter 2

The numerical solution of Ordinary Differential Equations (ODEs).

2.1 The theory of ODEs.

In this Chapter it is our intention to examine the theory of ODEs along with some numerical methods for their solution In particular we are concerned with solving the general first order nonlinear vector ODE of the form,

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \tag{2 1}$$

where

$$\mathbf{y}(t) \quad \mathbf{R} \rightarrow \mathbf{R}^n \quad \text{and} \quad \mathbf{f}(t, \mathbf{y}(t)) \quad \mathbf{R} \times \mathbf{R}^n \rightarrow \mathbf{R}^n,$$

subject to the conditions

$$\mathbf{y}(a) = \mathbf{y}_a \quad \text{and} \quad t \in [a, b].$$

The first question we ask ourselves regarding (2 1) is, does a solution $\mathbf{y}(t)$ exist and, if so, is it unique In section 2 we outline the conditions that we need to impose on (2 1) for a unique well-posed stable solution to exist We also examine the concept of stiffness which is very important for the numerical solution of ODEs Section 3 introduces discrete variable (numerical) methods for the solution of (2 1) In particular we introduce two well known classes of methods, the Runge Kutta (RK) methods and the Linear Multistep Methods (LMM) Section 4 discusses the error, order and convergence of numerical methods when applied to (2 1) Stability of numerical methods is introduced in section 5. We give several definitions of stability and demonstrate their usefulness through relevant examples Section 6 deals with the implementation of numerical methods. Finally in Section 7, we look at some well documented techniques for estimating the error in the numerical integration of (2 1) and the associated problem of stepsize adjustment.

2.2 Existence and uniqueness.

We assume that $\mathbf{f}(t, \mathbf{y}(t))$ is Lipshitz continuous on [a, b], that is there exists a constant L such that

$$\|\mathbf{f}(t,\mathbf{y}(t)) - \mathbf{f}(t,\mathbf{z}(t))\|_\infty \leq L \|\mathbf{y}(t) - \mathbf{z}(t)\|_\infty \tag{2 2}$$

for all $t \in [a,b]$ and all $\mathbf{y}(t)$, $\mathbf{z}(t) \in \mathbf{R}^n$[1].

More specifically if we require the first partial derivatives of $\mathbf{f}(\cdot)$ be bounded by a constant K that is,

$$\left\| \frac{\partial \mathbf{f}_i}{\partial \mathbf{y}_j} \right\| \leq K \quad 1 \leq i,j \leq n \tag{2 3}$$

for all $t \in [a,b]$ and all $\mathbf{y}(t) \in \mathbf{R}^n$, then (2 2) and (2 3) guarantee a unique, well posed (in the sense that the solution can be made as accurate as possible by keeping perturbations small) solution to problem (2 1), see Gear [31]

The most important attribute of (2 1) we are concerned with is *stiffness* When solving ODEs numerically, stiffness will dictate how well a numerical method will perform on the ODE To determine whether or not (2 1) is stiff, we need to know something about the nature of its solutions in the neighbourhood of a particular solution $\mathbf{y}(t)$ Hall & Watt [38] consider such a neighbourhood where (2 1) can be closely approximated by the linearized variational equations

$$\mathbf{y}'(t) - J(t)[\mathbf{y}(t) - \mathbf{y}] - \mathbf{f}(t,\mathbf{y}) = 0 \tag{2 4}$$

where $J(t)$ is the Jacobian matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$, evaluated at $(t,\mathbf{y})$

Remark 2 1 We deal only with stiff problems in this thesis Non-stiff ODEs are better solved by numerical methods such as Adams formulae, see [31]

If the variation of $J(t)$ in an interval of t is sufficiently small, the localized eigensolutions of (2 4) are approximately exponentials $e^{\lambda_i t}$, where the $\lambda_i's$ are the localized eigenvalues of the Jacobian matrix, assumed without loss of generality to be distinct Thus the solution $\mathbf{y}$ of (2 1) in a neighbourhood of the exact solution $\mathbf{y}(t)$ at t are of the form

$$\mathbf{y} = \mathbf{y}(t) + \sum_{i=1}^{n} c_i e^{\lambda_i t} \mathbf{v}_i$$

where the c_i are constants and the $\mathbf{v}_i$ are the eigenvectors of $J(t)$ If we assume that $Re(\lambda_i), < 0 \ \forall\, i = 1(1)n$, then clearly the components of the solution $\mathbf{y}$ will decay at different rates, given by $|\, 1/Re(\lambda_i)\,|$, these are called the *time constants* of the system The ODE (2 1) is *stiff*, if we have widely differing local *time constants* It is the range in the local values of the "time constants" of a problem that provides a measure of stiffness

Definition 2 1 (Lambert [46]) The ODE (2.1) is said to be stiff in an interval I of $[a,b]$ if, for $t \in I$, we have

$$Re(\lambda_i) < 0 \quad i = 1(1)n$$

[1]The Maximum norm is sufficient for the type of functions we consider, however the Supremum norm may be more appropriate in certain situations

and

$$S(t) = \frac{\max_{i=1,n} Re(\lambda_i)}{\min_{i=1,n} Re(\lambda_i)} \gg 0$$

where the λ_i's are the eigenvalues of the Jacobian matrix of $\mathbf{f}(t, \mathbf{y}(t))$, evaluated on the solution $\mathbf{y}(t)$ at t. The ratio $S(t)$ is called the local *stiffness ratio* of the problem, see Lambert [46]. Problems may be marginally stiff if $S(t)$ is $O(10)$ while *stiffness ratios* of $O(10^6)$ are not uncommon in practical problems. Sometimes a problem which is stiff is referred to as a problem with "widely differing time constants", or as a system with a "large Lipshitz constant", since

$$\rho\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right) < \left\|\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right\| = L$$

where $\rho(\ )$ is the spectral radius of the Jacobian of $\mathbf{f}(\cdot)$.

2.3 Discrete variable methods.

Without loss of generality we consider the scalar version of (2.1)

$$y' = f(t, y(t)) \qquad t \in [a, b] \tag{2.5}$$

with $y(a) = y_a$. The exact solution of (2.5) is approximated on set of discrete points

$$a = t_0, t_1, t_2, \cdots, t_f = b$$

If the discrete variable method [2] approximates the true solution $y(t_n)$ at the point t_n by y_n, then we shall consider the class of discrete variable methods given by

$$y_i = s_i(h), \quad 0 \le i \le k-1, \quad \text{starting values}$$

and

$$\sum_{i=0}^{k} \alpha_i y_{n+i} = h\, \Phi_f(t_n, y_{n+k}, \cdots, y_n, h) \tag{2.6}$$

$$0 \le n \le N - k,$$

where $h = t_{n+1} - t_n$ and $N h = t_n - t_0$. If y_{n+k} does not appear in $\Phi_f(\ )$[3] then (2.6) is said to be explicit otherwise it is implicit. The above class of methods contains a reasonably wide selection of the most popular discrete variable methods, (see Hall & Watt [38]). For example the general implicit one-step method, commonly known as the Backward Euler (BE) method, is defined by

$$y_{n+1} = y_n + h\, f(t_{n+1}, y_{n+1}) \tag{2.7}$$

We consider two important subclasses of (2.6). The first of these is the Runge Kutta (RK) methods. The idea behind the RK methods is to integrate from t_n to $t_{n+1} = t_n + h$, by approximating the integral in

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))\, d\tau \tag{2.8}$$

[2] Discrete variable methods are commonly called numerical methods, or numerical schemes when discussing ODEs. We adopt this convention throughout the thesis.

[3] The function $\Phi_f(\ )$ is often referred to as the increment function.

by a quadrature rule The *classical* RK formulae used well known quadrature rules such as Simpson's rule and were all explicit

To approximate (2 8) we choose quadrature points

$$c_1, c_2, \cdot \quad , c_q$$

and weights

$$b_1, b_2, \quad , b_q.$$

We then use the quadrature formula

$$y(t_{n+1}) = y(t_n) + \sum_{i=1}^{q} b_i k_i + Error \tag{2 9}$$

with the derivatives approximated by

$$k_i = h f \left(t_n + c_i h, y_n + \sum_{i=1}^{q} a_{ij} k_j \right)$$

For RK methods therefore, we have that

$$\Phi = \sum_{i=1}^{q} b_i k_i$$

In general this is a set of implicit equations which we solve for the k_i's and use a discrete version of (2 9) for our next value of $y(t_{n+1})$ thus

$$y_{n+1} = y_n + \sum_{i=1}^{q} b_i k_i$$

These implicit Runge Kutta (IRK) methods were first introduced by Butcher [8] It has become standard to follow Butcher and display the coefficients as an array thus

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12}, & , & a_{1q} \\ c_2 & a_{21} & a_{22}, \cdot & , & a_{2q} \\ c_3 & a_{31} & a_{32}, & , & a_{3q} \\ & \cdot & \vdots & & \cdot \\ c_q & a_{q1} & a_{q2}, & , & a_{qq} \\ \hline & b_1 & b_2, \cdot \cdot & , & b_q \end{array} \tag{2 10}$$

It is common to adopt the following shorthand notation

$$\begin{array}{c|c} \mathbf{c} & \mathrm{A} \\ \hline & \mathbf{b} \end{array} \tag{2 11}$$

where A represents the matrix of coefficients a_{ij}, b is is the vector of weights and c is the vector of quadrature points The quadrature points are usually called abscissae, while in the literature they are sometimes called the integration stages We point out that this representation includes the classical explicit formulae if $a_{ij} = 0$, whenever $i < j$ Then each k_i is given explicitly in terms of the previous ones

It turns out however that the nonlinear equations which arise from the application of the IRK method to (2 5) are very expensive to solve One way to circumvent this difficulty is to use a lower triangular array of coefficients a_{ij} in (2.10), such methods have been called *semi-explicit* RK methods in the literature, see Alexander [1] If, in addition, all the a_{ii} are equal, we have a Diagonally Implicit RK (DIRK) formula These formula have been extensively studied by Norsett [52], Alexander [1] and Crouziex [23] and have the general form

$$\begin{array}{c|cccc} c_1 & \alpha & & & \\ c_2 & a_{21} & \alpha & & \\ c_3 & a_{31} & a_{32} & \alpha & \\ \cdot & \vdots & \vdots & & \\ c_q & a_{q1} & a_{q2} & \cdots & \cdots \alpha \\ \hline & b_1 & b_2 & \cdots & \cdots b_q \end{array} \tag{2 12}$$

Once again, we adopt the shorthand notation

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array} \tag{2.13}$$

but in this case, A is a diagonal matrix, with equal diagonal elements

The second class of methods we consider are called *Linear Multistep Methods* (LMM), (usually called multistep methods) These methods use previously calculated information to generate an approximation to $y(t_{n+k})$ by y_{n+k} The coefficients for these methods are generated by fitting an interpolating polynomial through the points

$$y_n, y_{n+1}, \quad , y_{n+k}$$

An alternative formulation is to fit a Taylor series to the linear combination

$$\sum_{j=0}^{k} \alpha_j y_{n+j} - h \sum_{j=0}^{k} \beta_j f_{n+j} = 0 \tag{2 14}$$

$$j = 0,1,2, \quad ,k$$

up to a certain order of accuracy, by undetermined coefficients, using the previously calculated values.

Several well known sets of LMMs have been derived based on this formula, such as the Adams/Bashforth, Adams/Moulton and Backward Differentiation Formulae (BDF) due to Gear [31] These formulae form the basis of the most highly successful algorithm for the numerical solution of ODEs implemented to date, the LSODE package of Hindmarsh [43]

2.4 Order and convergence.

Consider the ODE (2 5) and assume that the approximate solution y_n is obtained by (2 6) then,

Definition 2.2· The global error at t_n, is defined as

$$e_n = y_n - y(t_n). \tag{2 15}$$

A natural requirement for any numerical method of the form (2.6) is that e_n can be made as small as possible by making h sufficiently small, this is the concept of convergence.

Definition 2.3: A method of the class (2.6) is said to be convergent if, when applied to (2.5) we have,

$$\lim_{h \to 0} \; y_n = y(t_n)$$

where $n\,h \;=\; t_n - a$ for any $t_n \in [a, b]$.

As an attempt at accessing the global error we introduce the following concept.

Definition 2.4: The local truncation error (lte) of (2.6) at t_{n+k} is given by,

$$\tau_{n+k} = \sum_{i=1}^{k} \alpha_i \, y(t_{n+k}) - h \, \Phi_f(t_n; y(t_{n+k}), \cdot, y(t_n); h). \tag{2.16}$$

The quantity τ_{n+k} is the amount by which the true solution of (2.1) fails to satisfy (2.6) and may be regarded as the first measure of accuracy. If we consider differential equations whose solutions are sufficiently differentiable, then it is possible to obtain an expression for the lte in terms of higher derivatives of $y(t)$. In this situation we may write

$$\tau_n = C(t_n, y(t_n))\, h^{p+1} + O(h^{p+2})$$

where $C(\cdot)$ is called the principal error function.

Definition 2.5: The method (2.6) is said to of order p, if p is the largest integer such that

$$\tau_n = O(h^{p+1}), \;\; \text{as} \;\; h \to 0.$$

The appropriate minimal level of local accuracy can now be defined.

Definition 2.6: A method of the class (2.6) is said to be consistent if,

$$\max \|\tau_n\| \to 0 \;\; \text{as} \;\; h \to 0.$$

It is consistent of order p, if

$$\max \|\tau_n\| \;=\; O(h^p)$$

Remark 2.2: Hall & Watt [38] deal with a normalised version of the local truncation error called the local discretization error. Under the assumption that $h \to 0$, they show that by controlling the local discretization error we also control the global error.

Definition 2.7: The local error (le) of a numerical method is given by

$$le = u(t_n) - y_n$$

where the local solution $u(t)$ is the solution to the ODE,

$$u'(t) = f(t, u(t)), \quad u(t_{n-1}) = y_{n-1}.$$

Thus in contrast to the lte where *exact back values* are assumed, the local error takes the solution through the last computed point (t_{n-1}, y_{n-1}). The local error is a very useful concept for test purposes. Usually we think of the the solution u_n as being a very accurate numerical approximation to the true solution obtained with a small step size. Using the concept of local error we are able to include in our test set problems which do not have closed form analytic solutions. We then demand that our estimate of the error behaves like the local error on all problems in our test set to ensure reliability of the integration method.

2.5 Stability of numerical methods.

The stability of a numerical method is related to the propagation of perturbations throughout the solution trajectory These perturbations arise from several sources including the local truncation error of the method, errors in initial values, round-off errors in the computed solution values and the presence of extraneous eigenvalues in the solution The numerical method used must be capable of controlling these errors throughout the solution trajectory A numerical method is stable if these perturbations remain bounded

Definition 2 8 (A-stability) A numerical method is *absolutely stable* for a given fixed steplength and for a given ODE, if the global error remains bounded as $n \to \infty$, (see Hall & Watt [38] page 34)

The problem with this condition is that it depends on the particular initial value problem. This has led most workers to consider specific test equations Then the general procedure for examining the stability characteristics of a particular numerical method is to apply the method to the test equation and determine the region of stability which results.

If the procedure is applied to multistep methods we get a stability polynomial, while a stability function results when we apply it to one step methods, see Lambert [46] We mention that in this thesis we only implement one step methods and hence the stability analysis with which we are concerned is mainly that associated with this type of method

2.5.1 Stability properties of the linear test equation.

The linear test equation is given by

$$y' = \lambda y \qquad y(a) = y_a \tag{2 17}$$

where λ is a complex constant with $Re(\lambda) < 0$ Let us denote the stability function that results when we apply a particular one step method to the solution of (2 17) by $R(z)$, where $z = \lambda h$ and h is the stepsize We can then formulate several stability concepts

Definition 2 9 A numerical method is A-stable if

$$|R(z)| < 1$$

for all z with $Re(z) < 0$.

Definition 2 10 If further we have that

$$R(z) \to 0 \quad \text{as} \quad z \to -\infty$$

the method is said to be *L-stable*

Remark 2 3 The importance of the A-stability concept lies in the fact that methods which are A-stable do not restrict the step size on stiff problems The *Euler* method, for example, fails to be A-stable A simple calculation yields

$$R(z) = 1 + z.$$

Thus for A-stability we would require $h < 1/|\lambda|$ This is a severe restriction on the step length if $\lambda \ll 0$ In fact all explicit methods fail to be A-stable, see Hall & Watt [38]

Remark 2 4 L-stability ensures that the numerical approximations to rapidly decaying solutions with very small time constants will decay quickly These concepts were introduced by Dahlquist [24] and Ehle [26] respectively.

Remark 2 5· A-stability proves too restrictive for multistep methods, (the stability of these methods is dealt with in Gear [31] and Lambert [46]) Dahlquist [24] proved that a multistep method fails to be A-stable if the order is greater than two Gear [31] introduced a weaker form of stability called stiff stability, which ensures that a multistep method is A-stable in a region D and accurate in a region A of the complex plane The following diagram illustrates the concept

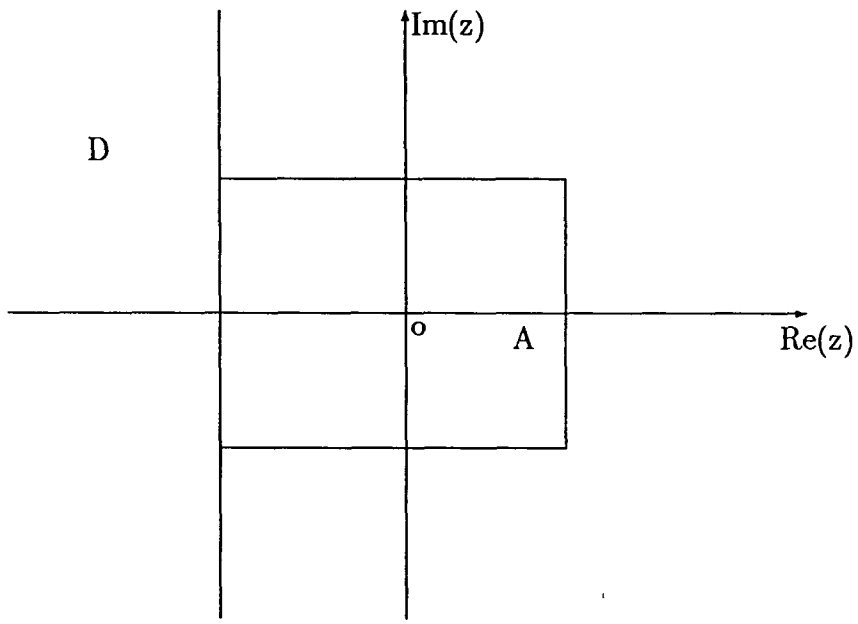

fig 2 1

Gear [31] showed that k step methods of order k are stiffly stable for $k \leq 6$

It is useful to consider RK methods applied to the linear test equation (2 17) It is easy to verify the relation (*c f* Hall & Watt [38])

$$y_{n+1} = R(z)\, y_n$$

with

$$R(z) = 1 + z\, \mathbf{b}^t (I - z\, A)^{-1} \mathbf{e} \tag{2.18}$$

where $\mathbf{e} = (1, 1, \cdots, 1)^t$, $\mathbf{b}$ and A have meanings given in (2 10) and I is a q-dimensional identity matrix We then have A-stability if

$$|1 + z\, \mathbf{b}^t (I - z\, A)^{-1} \mathbf{e}| < 1$$

Writing $R(z)$ as $1 + \mathbf{b}^t [1/z(I - z\, A)]^{-1}\, \mathbf{e}$ and taking the limit as $z \to -\infty$ we get L-stability if

$$\mathbf{b}^t A^{-1} \mathbf{e} = 1 \tag{2 19}$$

Solving the model problem (2 17) exactly over one step of length h , we have

$$y(t_{n+1}) = exp(z)\, y(t_n)$$

while the numerical solution is

$$y_{n+1} = R(z)\, y(t_n)$$

thus

$$exp(z) = R(z) + O(z^{p+1})$$

The stability function R(z) is thus a rational approximation to the exponential of order p.

Example 2 1. Consider the θ-method given in RK form by

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1-\theta & \theta \\ \hline & 1-\theta & \theta \end{array}$$

The stability function R(z), that results from the application of the θ-method to the model problem (2.17), is from (2.18)

$$R(z) = \frac{1 + z - z\,\theta}{1 - z\,\theta}$$

For A-stability we require $|R(z)| < 1$, this gives the following inequality for z with $Re(z) < 0$

$$|z|^2\,(1 - 2\,\theta) + 2\,Re(z) < 0$$

We can ensure this inequality holds with $Re(z) < 0$, if we choose $\theta \geq 0\ 5$ Since the coefficient matrix A for the method is not invertible, the method cannot be L-stable by (2 19) However if $\theta = 1$, no function evaluation is required at t_n and the method reduces to

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

which is the Backward Euler method in RK formulation.

Several integration routines used the θ-method as the core integrator These include the STINT integrator of Hall & Watt [38], Prothero & Robinson [60] used it for the solution of stiff chemical kinetics problems and Chua & Dew [22] used this scheme in gas dynamics simulations Berzins *et al* [3] provide a θ-scheme in their SPRINT package and Carroll [18] has also used it in his Composite Integration Scheme

2.5.2 S-stability.

In their work on large nonlinear systems, Prothero & Robinson [60] found that

1 Some A-stable methods could give highly unstable solutions.

2 The accuracy of the solution obtained is sometimes unrelated to the order of the method used

To overcome these difficulties they introduced the concept of S-stability, which is concerned with both stability and accuracy of numerical approximations to the solution of the stiff equation

$$y' = \lambda \{y - g(t)\} + g'(t) \tag{2 20}$$

which has solution

$$y(t) = \exp(\lambda t)\{y(0) - g(0)\} + g(t)$$

Note. $g(t)$ and $g'(t)$ are bounded functions over a suitable interval of interest [0,T] In the limit as $t \to \infty$ we have

$$y(t) \to g(t)$$

regardless of the initial conditions

Let us assume that the one step method IRK method (2 10) is applied to (2 20), we then have the following definition of S-stability due to Prothero & Robinson [60]

Definition 2 11· A one step method is S-stable if when applied to the test equation (2 20) over one step from t_n to t_{n+1}, there exist real positive constants λ_0 and h_0 such that

$$\frac{|y_{n+1} - g_{n+1}|}{|y_n - g_n|} < 1$$

provided that $y_n \neq g_n$, for all $0 \leq h \leq h_0$ and for all complex λ with $Re(-\lambda) > \lambda_0$, with t_n and $t_{n+1} \in [0, T]$ If we also have

$$\lim_{Re(\lambda) \to -\infty} \frac{y_{n+1} - g_{n+1}}{y_n - g_n} \to 0$$

for all positive h the method is *Strongly S-stable* . Note S-stability $\Rightarrow$ A-stability and Strong S-stability $\Rightarrow$ L-stability (trivially take $g \equiv 0$) The converse however is not true

Roughly speaking, S-stability means that for a given λ with $Re(\lambda) < 0$, the upper bound on admissible stepsizes to ensure $y_n \to g(t_n)$, does not tend to zero as $Re(-\lambda) \to \infty$ The following example illustrates the concept of S-stability in practice It is similar to an example given in Carroll [19] but uses a different function $g(t)$

Example 2 2 The application of the Backward Euler method to (2 20) yields

$$y_{n+1} - g_{n+1} = \frac{y_n - g_n + (g_n - g_{n+1}) + h\, g'_{n+1}}{1 - z}$$

We consider the specific case $g(x) = x^2 + 1$, with $x_n = n\, h$ and denoting $1 - z$ by q , we have

$$y_{n+1} - g_{n+1} = \frac{y_n - g_n}{q} + \frac{1 + h^2}{q}$$

Using this relation we can an expression for $y_n - g_n$ in terms of $y_0 - g_0$, we then have

$$\frac{y_{n+1} - g_{n+1}}{y_n - g_n} = 1/q + q^{n-1} \left\{ \frac{y_0 - g_0}{1 + h^2} + \left(1 + q + \cdot \quad + q^{n-1}\right) \right\}^{-1}$$

After some manipulation and noting that $q > 1$ we get the following inequality which must be satisfied for the method to be S-stable on this problem

$$1 + h^2 + (y_0 - g_0) < q\ (y_0 - g_0)$$

since $q = 1 - z = 1 - \lambda h$ we can rewrite the above inequality as

$$h^2 + \lambda h (y_0 - g_0) + 1 < 0$$

If $y_0 - g_0 = 0$ then the above inequality will never be satisfied as the discriminant of the quadratic is imaginary While if $y_0 - g_0 \gg 0$, the the roots are 0 and $-\lambda (y_0 - g_0)$ With $\lambda \ll 0$ we therefore have S-stability for nearly all positive h.

Remark 2 6 Carroll [19] points out that S-stability is only meaningful when a transient component is present in the true solution Thus S-stable methods are only appropriate for very stiff systems, that is when $y_0 - g_0$ is very different from zero

Remark 2 7 Writing $e_n = y_n - g_n$ Verwer [70] considers the condition for S-stability in the following equivalent form

$$|e_{n+1}| < |e_n|$$

and states that the requirement is unnatural since $e_n \to 0$ as $n \to \infty$ unless the local truncation error does He proposed an alternative version of S-stability called S^0-stability, which ensures that e_n is uniformly bounded with n, for all λ with $Re(\lambda) < 0$

2.6 Implementation of numerical methods.

In order for a classical Linear Multistep Method or one-step method to be A-stable it is necessary for the method to be implicit, Hall & Watt [38] Recall the general Implicit LMM given in section 3, reproduced here for completeness

$$\sum_{j=0}^{k} \alpha_j y_{n+j} - h \sum_{j=0}^{k} \beta_j f_{n+j} = 0, \quad \alpha_k = 1, \ \beta_k > 0 \tag{2 21}$$

The implicit LMM (2 21) above, applied to the non-linear system (2 1) results in the following system of non-linear equations

$$\mathbf{y}_{n+k} - h \beta_k \mathbf{f}(t_{n+k}, \mathbf{y}_{n+k}) - \mathbf{g} = 0 \tag{2 22}$$

with $\mathbf{g}$ a known vector of past information, to be solved at each integration step for $\mathbf{y}_{n+k}$ The stability properties of (2.21) may only be retained by solving (2 22) accurately using a convergent iterative method. A simple iterative method of the form

$$\mathbf{y}_{n+k}^{(i+1)} = h \beta_k \mathbf{f}(t_{n+k}, y_{n+k}^{i}) + \mathbf{g} \quad i = 1, 2, \ \cdot$$

is impractical, since for convergence we would require

$$|h \beta_k L| < 1$$

where L is the Lipshitz constant of $\mathbf{f}(\ )$, (see Definition 2 1, section 2) For stiff systems this convergence condition imposes just the type of severe restriction on the step size that we are trying to avoid

To overcome this difficulty, we use a Newton Raphson procedure, which gives the following linear system to be solved at each Newton iteration step

$$\left[I - h \beta_k J^i \right] \Delta \mathbf{y}_{n+k}^{i+1} = -\mathbf{y}_{n+k}^{i} + h \beta_k \mathbf{f}(t_{n+k}, \mathbf{y}_{n+k}^{i}) + \mathbf{g} \tag{2 23}$$

with

$$\Delta \mathbf{y}_{n+k}^{i+1} = \mathbf{y}_{n+k}^{i+1} - \mathbf{y}_{n+k}^{i} \quad i = 1, 2, \cdot \quad , imax$$

where I is the identity matrix and J^i is the Jacobian of $\mathbf{f}(t_{n+k}, \mathbf{y}_{n+k}^i)$, with an upper bound $imax$ placed on the number of Newton iterations In practice, with a good initial guess for $\mathbf{y}_{n+k}$, convergence occurs in two or three iteration steps

In general it is not necessary to re-form the iteration matrix B at each iteration step. Since we generally have a good initial guess for $\mathbf{y}_{n+k}$, J^i is usually close to $J^0 = J$ and we can replace

$$B^i = I - h\,\beta_k\,J^i$$

by

$$B = I - h\,\beta_k\,J$$

and a single *LU-decomposition* suffices for the i-iteration steps, this is called the modified Newton method It enhances the efficiency of implicit methods as frequent re-factorisations of the iteration matrix are avoided

The IRK methods are solved in exactly the same way for the unknowns $\mathbf{k}_i$ We have from (2 9) that

$$\mathbf{k}_j = h\,\mathbf{f}\left(t_n + c_j\,h, \mathbf{y}_n + \sum_{l=1}^{q} a_{jl}\,\mathbf{k}_l\right) \tag{2.24}$$

Applying Newton iteration we get the following linear system to be solved at each iteration step

$$B\Delta\,\mathbf{k}_j^{(i+1)} = h\,\mathbf{f}\left(t_n + c_j\,h, \mathbf{y}_n + \sum_{j=1}^{q} a_{jl}\,\mathbf{k}_l^{(i)}\right) - \mathbf{k}_j^{(i)} \tag{2 25}$$

with

$$\Delta\mathbf{k}_j^{(i+1)} = \mathbf{k}_j^{(i+1)} - \mathbf{k}_j^{(i)} \quad i = 1, 2, \quad , imax$$

and the iteration matrix has the form

$$B = I - h\,A\,J$$

where A is an $nq \times nq$ matrix for (2 1) and J is the Jacobian of $\mathbf{f}(\cdot)$ at each internal stage $\mathbf{k}_l$, $l = 1(1)q$

As mentioned in section 3, an enormous gain can be obtained in computational efficiency in these iterations if we use *DIRK* formula, that is an IRK scheme with $a_{ij} = 0$ for $i < j$ and all the $a_{ii} = \alpha$, (2 12) The reason for this is that the implementation of IRK formula involves the solution via an *LU-decomposition* of a system of linear equations of dimension nq for (2 1) at each time step The implementation of DIRK methods is much simpler as each stage only depends on previous ones

Consider the DIRK(2,2) version of (2.12) given by the formula (*we drop the dependence on t for clarity*),

$$\begin{aligned} 0 &= \mathbf{k}_1 - h\,\mathbf{f}(\mathbf{y}_n + \alpha\,\mathbf{k}_1) \\ 0 &= \mathbf{k}_2 - h\,\mathbf{f}(\mathbf{y}_n + (1-\alpha)\,\mathbf{k}_1 + \alpha\,\mathbf{k}_2) \end{aligned} \tag{2 26}$$

$\mathbf{k}_1$ can be generated implicitly from the first equation using Newton's method as follows

$$B\left\{\mathbf{k}_1^{i+1} - \mathbf{k}_1^{i}\right\} = h\,\mathbf{f}\left(\mathbf{y}_n + \alpha\,\mathbf{k}_1^{i}\right) - \mathbf{k}_1^{i}$$

letting $\mathbf{z} = \mathbf{y}_n(1-\alpha)\,\mathbf{k}_1$ we get a similar iteration for $\mathbf{k}_2$

$$B\left\{\mathbf{k}_2^{i+1} - \mathbf{k}_2^{i}\right\} = h\,\mathbf{f}\left(\mathbf{z} + \alpha\,\mathbf{k}_2^{i}\right) - \mathbf{k}_2^{i}$$

both iteration schemes have an identical iteration matrix [4]

$$B = I - h\,\alpha J$$

with J the Jacobian of $\mathbf{f}(\,)$ at $\mathbf{y}_n$ This procedure can be extended to DIRK(p,q) methods, in such situations we have only one *LU-decomposition* of a system of equations of dimension n We then have q successive stages where the iterative process is applied

For Parabolic PDEs, where n is typically large, due to discretization via the method of lines, the DIRK implementation just outlined provides a considerable saving in computational expense over the full IRK implementation

The implementation of the DIRK(2,2) method used in this thesis is via the Rosenbrock [69] technique Essentially this is just one iteration of the modified Newton scheme for the DIRK methods We can implement the Rosenbrock technique directly by linearizing (2 26) about $\mathbf{y}_n$ and $\mathbf{z}$ as follows

$$\begin{aligned} \mathbf{k}_1 &= h\,\mathbf{f}(\mathbf{y}_n) + \alpha h\,\mathbf{k}_1\,J \\ \mathbf{k}_2 &= h\,\mathbf{f}(\mathbf{z}) + \alpha h\,\mathbf{k}_2\,J \end{aligned} \tag{2 27}$$

and again J is the Jacobian of $\mathbf{f}(\,)$ at $\mathbf{y}_n$ This form of implementation has been widely used by engineers, see Hall & Watt [38]

Two other well known features can be included to improve overall efficiency. Stiff systems of ODEs generally change very little over long periods of the integration interval Thus it is not unreasonable to use the iteration matrix for several steps of the solution trajectory Most codes include this feature and only update the Jacobian when it is really desirable to do so, usually after a step failure or when the iteration matrix has been used for a fixed number of successive steps

The final improvement is more cosmetic in nature as it is an aid to the user Generally the Jacobian must be evaluated by hand For large systems of ODEs, this may be difficult if not impossible without the aid of a symbolic manipulator However forward differencing can be used to estimate the partial derivatives thus

$$\frac{\partial f_i}{\partial y_j} = \frac{f_i(t, y_j + e_j\,\xi) - f_i(t, y_j)}{\xi\,e_j} \tag{2 28}$$

where e_j is the normalized j^{th} coordinate vector and ξ is a scalar increment which is small compared to the magnitude of f_i The choice of ξ can cause difficulty, in general it should be chosen to prevent scaling difficulties in evaluating the partial derivatives of f_i

[4] While this is technically incorrect, it suffices, since it can be assumed that the Jacobians for each iteration scheme are virtually identical This assumption is valid because $\mathbf{f}(\,)$ is assumed to be a reasonably smooth function

2.7 Error measurement and Stepsize Strategies.

We outline three methods for estimating the lte during the integration of (2 1) It is usual to require this quantity to lie within some user defined tolerance so that the local error remains bounded on the current step Based on the result of this test, the code may decide to automatically change the steplength, increase or decrease the order or re-evaluate the Jacobian of the system.

In multistep methods the approach to error estimation is to use Milne's device [38]. Here a k^{th} order predictor and a k^{th} order corrector pair is used to estimate the error in the latter. The simplest example of this is the Euler and Backward Euler pair, for the scalar system (2 5) this is

$$\begin{aligned} y^p_{n+1} &= y_n + h\, f_n + C_1\, h^2 \\ y^c_{n+1} &= y_n + h\, f^p_{n+1} + C_2\, h^2 \end{aligned}$$

where *p and c* mean predict and correct respectively A full description of these methods may be found in Hall & Watt [38] The difference between these two solutions

$$(C_1 - C_2)\, h^2 = y^p_{n+1} - y^c_{n+1} - h\,(f_n - f^p_{n+1})$$

is a constant multiple of the principal error $C_2\, h^2$ in the Backward Euler method

The general approach for RK methods is to generate two solutions and use their difference as an estimate of the local truncation error Two techniques are usually employed to obtain the estimate The first is the embedded methods originally introduced by Fehlberg [28] The idea is to generate two solutions of the form

$$z_{n+1} = y(t_{n+1}) + C_1\, h^p$$

$$y_{n+1} = y(t_{n+1}) + C_2\, h^{p+1}$$

using the current solution y_n The difference between these $|y_{n+1} - z_{n+1}| \approx C_1\, h^p$ The p^{th} order method for z_{n+1} is embedded in a method of order $p+1$ for y_{n+1} The following example, see Cash [21] demonstrates the technique in practice He proposed an embedded version of the Strongly S-stable DIRK(3,3) method introduced by Alexander [1] for the unknown y_{n+1}

α	α		
τ	$\tau - \alpha$	α	
1	b_1	b_2	b_3
	b_1	b_2	b_3

The first two stages of the method are then used as the basis of a second order method for z_{n+1}

α	α	
τ	$\tau - \alpha$	α
	c_1	c_2

The coefficients c_1 and c_2 are then chosen so that this method is of order two

Richardson extrapolation is the second method used to estimate the error in the RK methods We integrate from t_n to t_{n+1} twice A full step of length h is taken to give

$$y_{n+1,h} = y(t_{n+1}) = h^{p+1}\, C(t_n) + O(h^{p+2})$$

and two steps of length $h/2$ to give the more accurate solution

$$y_{n+1,h/2} = y(t_{n+1}) + 2(h/2)^{p+1} C(t_n) + O(h^{p+2})$$

subtracting these we have

$$h^{p+1} C(t_n) = \frac{y_{n+1,h} - y_{n+1,h/2}}{1 - 2^{-p}}. \tag{2.29}$$

This can be added to the more accurate solution to give an extrapolated value

$$y_{n+1} = y_{n+1,h/2} + 2^{-p} \left\{ \frac{y_{n+1,h/2} - y_{n+1,h}}{1 - 2^{-p}} \right\} \tag{2.30}$$

Using a numerical method that possesses the appropriate stability properties, the stepsize h should not be restricted on account of stability or, hopefully, by any convergence requirements in the iterative solution of the nonlinear system Usually two stepsize strategies are commonly adopted.

1. When the effects on the solution of the transient components with small time-constants are not of interest, an initial step-size that is large relative to these time-constants can be used No attempt is made to approximate the short term effects accurately We rely on the stability of the method to damp out transient solutions when calculating long range behaviour

2 When an accurate representation of the rapidly varying transients in the solution is required, the initial stepsize must be comparable with the smallest time-constants.

Methods for estimating the initial step-size will be dealt with in Chapter 3 where we discuss our methods in more detail When a method leaves the transient (non-stiff) region of integration it is desirable to increase the stepsize quickly, to take full advantage of the stability properties of the method Our implementation of step adjustment and local error estimation will also be discussed in the next Chapter

Chapter 3

Variable step integrators for the solution of stiff ODEs.

3.1 Introduction.

In this chapter, we will consider two specific methods for the numerical solution of stiff ODEs The first method we consider is the DIRK(2,2) scheme introduced in Chapter 2 The second method we consider is the second order Composite Integration scheme of Carroll [18]

In sections 2 and 3, we discuss the accuracy and stability of both methods respectively Section 4 deals with error estimation, while in section 5 we develop their iteration schemes The remaining sections implement these methods as variable step algorithms and discuss respective performances on test problems taken from the literature

3.2 The Strongly S-stable DIRK(2,2) scheme.

Recall the general matrix representation of a q-stage IRK method of order p is, (*c f* equation (2 10), Chapter 2)

$$\begin{array}{c|c} \mathbf{c} & \mathrm{A} \\ \hline & \mathbf{b} \end{array} \qquad (3\ 1)$$

where A is a $q \times q$ matrix of coefficients $a_{i,j}$, **c**, a *q-dimensional* vector of stages or quadrature points and **b**, a q-dimensional vector of weights

The traditional problem of choosing $q^2 + 2q$ coefficients in a q-stage method, so as to obtain the highest possible order of accuracy, subject to stability or other constraints, leads to a nonlinear algebraic jungle, to which civilization and order was brought by J C Butcher [7], [8] and M Crouzíex [23] These methods proved very inefficient for reasons we mentioned in Chapter 2. In fact Enright, Hull and Lindberg [27] showed that their 2-stage 4^{th}-order method produced poorer results than other implicit methods. We introduced the DIRK methods in Chapter 2 to overcome these difficulties Some well known examples of DIRK methods include

1 The implicit midpoint rule, a single stage 2^{nd}-order method which is A-stable

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

2 The 2-stage 3^{rd}-order Gaussian quadrature rule, which is also A-stable

$$\begin{array}{c|cc} 1/2+\frac{1}{2\sqrt{3}} & 1/2+\frac{1}{2\sqrt{3}} & \\ 1/2-\frac{1}{2\sqrt{3}} & \frac{-1}{\sqrt{3}} & 1/2+\frac{1}{2\sqrt{3}} \\ \hline & 1/2 & 1/2 \end{array}$$

Following Alexander [1], we refer to the general presentation (3 1) and make the following conventions
C is the $q \times q$ diagonal matrix

$$diag(c_1, c_2, \cdot \quad , c_q)$$

and **e** is a *q-dimensional* vector

$$(1, 1, \quad \cdot, 1)$$

Theorem 3 1 (Alexander [1]) Let $p \leq 5$ To ensure that a DIRK method to be of order p, for every sufficiently regular function $f(t, y(t))$, it is necessary that the relations $(3, i)$ $i = 1(1)p$ be satisfied [1]
3 1. $\mathbf{b}^t \mathbf{e} = 1$
3 2 $\mathbf{b}^t C \mathbf{e} = 1/2, \quad \mathbf{b}^t A \mathbf{e} = 1/2$

Theorem 3 2 (Alexander [1]) An A-stable semi-implicit RK formula with positive diagonal elements is S-stable iff

$$|R_0| \equiv |1 - \mathbf{b}^t A^{-1} \mathbf{e}| < 1$$

An A-stable formula of this kind is Strongly S-stable if it is L-stable
We now can state the main result of this section

Theorem 3 3 The DIRK(2,2) formula given by

$$\begin{array}{c|cc} \alpha & \alpha & \\ 1 & 1-\alpha & \alpha \\ \hline & 1-\alpha & \alpha \end{array}$$

or

$$\begin{aligned} k_1 &= h f(t_n + \alpha h, y_n + \alpha k_1) \\ k_2 &= h f(t_n + h, y_n + (1-\alpha) k_1 + \alpha k_2) \qquad (3\,2) \\ y_{n+1} &= y_n + (1-\alpha) k_1 + \alpha k_2 \qquad (3\,3) \end{aligned}$$

with $\alpha = 1 \pm 1/\sqrt{2}$ is second order accurate, A-, L-, S- and Strongly S-stable
Proof
Accuracy Using Theorem 3 1 part 2, the following relations are satisfied

$$\mathbf{b}^t C \mathbf{e} = \mathbf{b}^t A \mathbf{e} = \alpha(1-\alpha) + \alpha = 1/2$$

Thus we get second order accuracy if

$$2\alpha^2 - 4\alpha + 1 = 0$$

[1] We only supply conditions for $p \leq 2$ the interested reader is referred to Alexander [1] for the full statement which we shall not require

giving $\alpha = 1 \pm 1/\sqrt{2}$

A-stability Recall the stability function for R-K methods from Chapter 2, equation (2 16),

$$R(z) = 1 + z\,\mathbf{b}^t(I - z\,A)^{-1}\mathbf{e}$$

Substituting the DIRK(2,2) formula (3 2) into this relation we get

$$R(z) = 1 + \frac{z(1-\alpha^2 z)}{(1-\alpha z)^2} \tag{3 4}$$

Therefore we require

$$|1 - 2\alpha z + z|^2 < |(1-\alpha z)^2|^2$$

to be satisfied, with $Re(z) < 0$ for A-stability

Writing $z = Re(z) + \imath\, Im(z)$ the LHS of this inequality becomes

$$LHS = 1 + 2(1-2\alpha)Re(z) + (1-2\alpha)|z|^2$$

While the RHS becomes after some manipulation,

$$\begin{aligned} RHS &= LHS - Re(z) - \{2\alpha^2 - 4\alpha + 1\}|z|^2 - 4\alpha^3 Re(z)Im^2(z) \\ &\quad +4\alpha^4 Im^4(z) + 4\alpha Im^2(z)|z|^2 + 4\alpha^4 Re^2(z)Im^2(z) + 4\alpha^2 Im^2(z) \end{aligned}$$

From the accuracy requirements above, the term $\{2\alpha^2 - 4\alpha + 1\}$ is zero iff $\alpha = 1 \pm 1/\sqrt{2}$ and the stability requirement reduces to

$$\begin{aligned} 0 &< -Re(z) - 4\alpha^3 Re(z)Im^2(z) + 4\alpha^4 Im^4(z) + 4\alpha^4 Im^2(z)|z|^2 \\ &\quad +4\alpha^4 Re^2(z)Im^2(z) + 4\alpha^2 Im^2(z). \end{aligned}$$

With $Re(z) < 0$ this inequality is satisfied

L-stability· From Chapter 2, equation (2.17) we have L-stability if

$$\mathbf{b}^t\, A^{-1}\, \mathbf{e} = 1$$

This relation trivially holds for the DIRK(2,2) scheme (3 2)

S- & Strong S-stability: Since the DIRK scheme (3 2) has positive diagonal elements if $\alpha = 1 \pm 1/\sqrt{2}$, both S- and Strong S-stability follow from Theorem 3 2, as the method is A and L-stable, with $|R_0| = 0$

Finally Expanding $R(z)$ as a polynomial in z we get

$$R(z) = 1 + z + (2\alpha - \alpha^2)z^2 + O(z^3)$$

With $\alpha = 1 - 1/\sqrt{2}$ we have

$$|R(z) - e^z| \text{ is } O(z^3) \quad \square$$

3.3 The Composite Integration θ-BDF2 scheme.

R E Bank *et al* [2], introduced a Composite Linear Mulitistep Method as the time integration scheme in the numerical solution of coupled systems of nonlinear partial differential equations Their technique was to use a two stage process to integrate

over one step The first stage used a Trapezoidal Rule to integrate to an intermediate point The second stage comprised a second order BDF-type scheme The method had important features of second order accuracy, A- and L-stability Bank *et al* implemented the Trapezoidal Rule and the BDF stages as a fully implicit independant steps solved via *Newton* iteration. Carroll [18] generalized their scheme so that it retained the important features of second order accuracy, A- and L-stability He replaced the Trapezoidal Rule with the one-step θ-scheme

$$y_{n+\gamma} = y_n + \gamma h\left[(1-\theta)f_n + \theta f_{n+\gamma}\right] \tag{3 5}$$

applied over the interval $t_n = nh$ to $t_{n+\gamma} = (n+\gamma)h$ with $0 < \gamma < 1$

The second stage of the integration uses a 2-step backward differentiation type formula which interpolates the three points $t_n, t_{n+\gamma}, t_{n+1}$, with the formula

$$\alpha_0 y_n + \alpha_1 y_{n+\gamma} + \alpha_2 y_{n+1} = h f_{n+1} \tag{3 6}$$

To find the coefficients $\alpha_0, \alpha_1, \alpha_2$ and γ two conditions are imposed on the composite pair of formulae

1 that it retains second order local accuracy, A- and L-stability

2. that both stages have a common *Newton* iteration matrix

This requirement is for computational efficiency, as only one LU-decomposition of the iteration matrix is required

The accuracy requirements of the scheme are obtained by combining both formulae and comparing the coefficients of the resulting expression with a Taylor series The following relationships for the unknown coefficients can be easily derived for second order accuracy, see Carroll [18]

1 $\gamma\theta = 1 - 1/\sqrt{2}$
2 $\alpha_1 = \frac{1-\alpha_2}{\gamma}$
3 $\alpha_2 = \frac{2(1-\gamma\theta)}{1-2\gamma\theta}$
4 $\alpha_0 = -\alpha_1 - \alpha_2$
5 Both (3 5) and (3 6) have a common iteration matrix if $\alpha_2\gamma\theta = 1$

Remark 3 1. It is interesting to note that $\gamma\theta = 1 - 1/\sqrt{2}$, is one of the values of α in the DIRK(2,2) method of section 2. In fact it it not difficult to show that the DIRK(2,2) scheme is a special case of the Composite Integration scheme, simply set $\theta = 1$ This fact is demonstrated in Appendix 1, where we show how the Composite Integration scheme can be put into RK matrix formulation However our implementations are quite different, we therefore expect some difference in the numerical results which are presented in Appendix 2

Carroll [18] also verifies A- and L-stability for this scheme We remind the reader that on expanding the stability function $R(z)$ in powers of z and recalling that $\gamma\theta = 1 - 1/\sqrt{2}$, we have

$$|R(z) - e^z| = \frac{3\sqrt{2}-4}{6} z^3 + O(z^4)$$

where the coefficient of z^3 compares directly with the coefficient of the principal error function for the method given in the next section

3.4 Error estimation.

3.4.1 Error estimate for the DIRK(2,2) scheme.

For the DIRK(2,2) scheme, Richardson extrapolation is used to estimate the local truncation error Recall the approach outlined for RK methods in Chapter 2, section 6 We integrate from t_n to t_{n+1}, with one step of length h and then integrate it twice using two steps of length h/2 The difference in the two solutions is a constant multiple of the local error (c f equation (2.28) reproduced here for completeness)

$$h^{p+1}C(t_{n+1}) = \frac{y_{n+1,h} - y_{n+1,h/2}}{1 - 2^{-p}} \tag{3 7}$$

As in Chapter 2, we add this to the more accurate solution to give the extrapolated solution

$$y_{n+1} = y_{n+1,h/2} + 2^{-p}\left\{\frac{y_{n+1,h} - y_{n+1,h/2}}{1 - 2^{-p}}\right\} \tag{3 8}$$

3.4.2 Error estimate for the Composite Integration scheme.

Carroll [18] gives the following expression as an estimate of the principal error function in the Composite Integration scheme

$$errest = \left\{\frac{3\gamma^2\theta - 4\gamma\theta + 1}{12(1 - \gamma\theta)}\right\} h^3 y_n^{(3)}(\xi) \tag{3 9}$$

Bank *et al* [2] and Carroll [18] suggest estimating $y_n^{(3)}(\xi)$ by the following linear combination of function values

$$y_n^{(3)}(\xi) = \frac{2}{h^2}\left\{\frac{1}{\gamma}f_n - \frac{1}{\gamma(1-\gamma)}f_{n+\gamma} + \frac{1}{1-\gamma}f_{n+1}\right\} \tag{3 10}$$

3.5 Solving the nonlinear equations.

3.5.1 The nonlinear equations arising from the DIRK(2,2) scheme.

The Rosenbrock technique [69] outlined in Chapter 2, section 5, is employed for solving the nonlinear system that arises from applying the method to (2 1) The resulting equations (2 26) for the DIRK(2,2) scheme (3 2) (reproduced here for completeness) are

$$[I - \alpha hJ]\,\mathrm{k}_1 = h\mathbf{f}(t_n + \alpha h, \mathbf{y}_n) \tag{3 11}$$

and

$$[I - \alpha hJ]\,\mathrm{k}_2 = h\mathbf{f}(t_n + h, \mathbf{y}_n + (1-\alpha)\mathrm{k}_1) \tag{3 12}$$

where J is the Jacobian of $f(t_n, y_n)$

Remark 3 2 As mentioned in Chapter 2 we are only applying one step of a *Newton* method in this case We therefore have no need to worry about convergence criteria The error estimate on time integration is the only form of control required However, as we have only one iteration step of a *Newton* scheme, the technique is only accurate for linear differential equations, but it performs well in practice when used in the variable step integrator which we develop later in this Chapter

3.5.2 Solving the nonlinear equations of the Composite Integration scheme.

To retain the stability of the Composite Integration scheme, a modified Newton method is employed to iteratively solve the nonlinear equations arising in both stages of the scheme

An application of Newton's method to the first stage θ-scheme (3 5) yields the nonlinear system (3 13) below. This system is solved iteratively for a fixed number of iterations $i = 1, 2, \cdot, imax$ or until convergence is achieved, giving

$$[I - \gamma\theta hJ]\,\Delta(y^i_{n+\gamma}) = y_n - y^i_{n+\gamma} + \gamma h\,[(1-\theta)f_n + \theta f_{n+\gamma}] \qquad (3\ 13)$$

where

$$\Delta y^i_{n+\gamma} = (y^{i+1}_{n+\gamma} - y^i_{n+\gamma}), \qquad y^0_{n+\gamma} = y_n, \quad \text{and} \quad f^0_{n+\gamma} = f_n$$

Also from (2 22) the BDF scheme (3 6) can be solved iteratively, using the *Newton* scheme (3 14) for y_{n+1} Again we impose an upper bound on the number of iterations employed, as we did for (3.13) We obtain

$$[I - \gamma\theta hJ]\,\Delta(y^i_{n+1}) = \frac{1}{\alpha_2}\left\{hf^i_{n+1} - \left(\alpha_0 y_n + \alpha_1 y_{n+v} + \alpha_2 y^i_{n+1}\right)\right\} \qquad (3\ 14)$$

with

$$\Delta y^i_{n+1} = (y^{i+1}_{n+1} - y^i_{n+1}), \qquad y^0_{n+1} = y_{n+\gamma} \quad \text{and} \quad f^0_{n+1} = f_{n+\gamma}$$

The choice of $1/\alpha_2 = \gamma\theta$ gives an identical iteration matrix on each stage and we adopt this strategy in our code, while J is the Jacobian of $f(\ )$ at y_n [2].

Remark 3 3 We point out that the use of $y^0_{n+\gamma} = y_n$ and $y^0_{n+1} = y_{n+\gamma}$ is equivalent to using a $zero^{th}$ order predictor formula on each stage A simple Euler predictor formula could equally well be used on both steps, without effecting the step adjustment mechanism in the algorithm outlined in section 6 However such a mechanism might effect the stability of the scheme.

To terminate the iteration we follow Shampine [65] and measure the rate of convergence

$$\rho_i = \frac{\|y^{i+1}_l - y^i_l\|}{\|y^i_l - y^{i-1}_l\|}$$

where i is the i^{th} iteration step and l is stage $n+\gamma$ or stage $n+1$ of the Composite scheme[3]. We compare this with a tolerance τ, in the formula

$$\frac{\rho_i}{1-\rho_i}\|y^{i+1}_l - y^i_l\| < \tau$$

According to Shampine [65] this guarantees, (with an appropriate $\rho_i < 1/2$), that we are converging at an acceptable rate to the solution y_l, with the demand that y^{i+1}_l be

[2]Once again using this Jacobian is technically incorrect but it suffices in practice

[3]To measure the rate of convergence in this way requires at least 2 iteration steps Because linear problems only require one Newton step this technique is expensive To overcome this difficulty we also terminate the Newton iteration if $\|y^{i+1}_l - y^i_l\| < \tau^2$

sufficiently close to the actual solution y_l Most current codes (such as LSODI and the Variable Step Integrator of Chua & Dew [22] use the condition that $\rho < 1/2$ while DASSL [56] uses a condition similar to the one we have adopted.

Remark 3 4 Shampine [65] points out that the convergence condition $\rho < 1/2$, is correct, if $\rho_i \rightarrow \lambda$, the largest eigenvalue of the system, with λ real However if λ is complex, then ρ_i will oscillate and the convergence rate $|\rho_i|$, will assume larger values than $Re(\rho_i)$ In general, most codes allow for this and take the largest observed value of ρ_i as their estimate of the rate of convergence. We have also adopted this policy on the second and subsequent iteration steps.

3.5.3 Other aspects of solving the nonlinear systems.

Two other items we have to consider in relation to the linear systems (3.9a,b) and (3 10a,b), are the formation of the Jacobian matrix of f(t, y(t)) and the subsequent method of solution of the linear system. The Jacobian can be provided in two separate ways, the first is for the user of the method to explicitly supply it, while the second is to estimate it with finite differences, as outlined in Chapter 2, section 5 Referring to equation (2 27), the scaling of the increment can lead to significant errors in Jacobian elements, if not properly chosen Our choice has been $\xi = 10^{-4}$ and we have had no apparent problems, working in double precision Fortran77 on VAX-11/785 and VAX-6230 computers Our original choice $\xi = 10^{-9}$ worked equally well on all test problems considered We however have endeavored to make ξ as large as possible, while at the same time keeping it within the tolerance band, where we expect our methods will perform efficiently Standard software packages such as LSODI [43] and DASSL [56] use a more complicated algorithm to choose the increments

The linear system which arises from (3 9a,b) and (3 10a,b), is solved using a standard LU-decomposition of the iteration matrix and subsequent back substitution This has the advantage that the LU-decomposed matrix can be stored for several iterations and/or time integration steps, leading to a considerable improvement in the overall efficiency of the solution method

3.6 Variable step algorithms for the solution of ODEs.

3.6.1 Algorithm DIRK(2,2).

Recall the DIRK(2,2) scheme (3 2) used the full-step half-step technique to estimate the error, thus,

ALGORITHM DIRK(2,2);
BEGIN
Given a tolerance Tol
SET $\alpha = 1 - /\sqrt{2}$,
WHILE $t_n > FinalTime$
IF the Jacobian has been used for the previous 10 steps
or the stepsize has changed THEN
COMPUTE the Jacobian;

COMPUTE the full step solution y_{n+1,h_n} from y_n using a
step-size h_n, by making one CALL to the Integrator,
COMPUTE the half step solution $y_{n+1,h_n/2}$ from y_n using
two steps of size $h_n/2$, by making two successive CALLs
to the Integrator,
COMPUTE an error estimate E_{DIRK} from (3.7) using the
weighted mean square norm,
COMPUTE the extrapolated solution call it y_{n+1} from (3 8),
IF $E_{DIRK} > tol$ THEN
BEGIN *(reject the step)*
RE-COMPUTE the solution from y_n with $h_{n+1} = h_n/2$
RETAIN this step for at least 3 subsequent steps
unless there is another step failure,
END,
IF $E_{DIRK} < tol$ THEN
BEGIN *(accept the step)*
SET $y_n = y_{n+1}$,
COMPUTE the factor by which the step length is to be
multiplied on the next step (hfactor);
$hfactor = (tol/E_{DIRK})^{1/3}$,
IF $hfactor \geq 10$ $hfactor = 10$,
IF $4 \leq hfactor \leq 10$ $hfactor = 4$,
IF $2 \leq hfactor \leq 4$ $hfactor = 2$,
Otherwise $hfactor = 1$,
SET $h_{n+1} = h_n \times hfactor$;
RETAIN this step size for at least two steps,
END;
END,
END { **(DIRK(2,2).**}

INTEGRATOR *advances the solution one step of length* h_n,
BEGIN
Solve the linear systems (3 9a) and (3 9b) respectively,
Advance the solution using the formula
$y_{n+1} = y_n + (1-\alpha)k_1 + \alpha k_2$;
END { **INTEGRATOR** }

Remark 3 5 Hfactor is the amount by which h_n can be reasonably multiplied so that estimated error on the next step stays with the specified tolerance With

$$\overset{\curvearrowright}{h_{new}} = hfactor \times h$$

we attempt to keep the error on the next step, for a method of order p, within the bound

$$|C(t_{n+1})h_{new}^{p+1}| < tol$$

equivalently

$$hfactor^{p+1}|C(t_{n+1})| < tol$$

giving

$$hfactor = (tol/E_{DIRK})^{1/p+1}.$$

Remark 3.6 The reason we keep the stepsize fixed, for 2 or 3 successive steps after the stepsize has changed, is to avoid *chattering* in the step changing mechanism and to introduce greater stability into the algorithm This constraint on the algorithm tends to make it biased toward using a constant stepsize Consequently, the number of Jacobian evaluations required to integrate an ODE over the specified time interval, is considerably reduced

Remark 3 7 The finite values we have chosen for hfactor are based on the following reasoning For example when $2 < hfactor < 4$ we reduce hfactor to 2 The reason for this is twofold, firstly if we allowed hfactor to take arbitrary values our stepsize would be changing too often, leading to chattering. Secondly, by reducing hfactor to 2, we take stepsizes which are more *conservative* giving a more stable algorithm This is necessary because we lack perfect information, having only an estimate of the error to increase or decrease the stepsize, instead of the true local error The value 2 has been chosen to reflect the fact that we allow an increase in stepsize, if $E_{DIRK} < tol/8$ This choice, along with $hfactor = 10$ were suggested by Alexander [1][4] We have also included the value $hfactor = 4$ The choices $hfactor = 2$ or 10 force $E_{DIRK} < tol/8$ or $tol/1000$, respectively. Our reason for including the value $hfactor = 4$, is that the gap between 8 and 1000 is large and an intermediate value may improve overall efficiency in the algorithm

Remark 3 8 The error test in this algorithm is constructed as follows If the magnitude, in maximum norm of the solution $y_{n+1,max}$, is greater than 1, we use the relative comparison

$$E_{DIRK} < y_{n+1,max} \times tol,$$

otherwise $y_{max} < 1$ and we use the absolute comparison

$$E_{DIRK} < tol$$

This is a very simple form of error control, only one tolerance value need be specified We have found it very effective on all problems considered in this thesis

3.6.2 Algorithm Composite Integration scheme.

We give the variable step algorithm based on the theory developed so far for this method

ALGORITHM COMPOSITE INTEGRATION SCHEME;
BEGIN
Given an absolute (atol) and or relative (rtol) tolerance,
SET $t_n = a$, the starting time,
COMPUTE the Jacobian at the starting time,
WHILE $t_n < finaltime$

[4]In fact Alexander demands that $E_{DIRK} < tol/10$, our choice is 2^3 reflecting the possibility of doubling the stepsize on the next step

```
BEGIN
   IF (h_{n+1} = 2 × h_n) OR (r > 0 85) OR
      (the Jacobian has not been updated for the previous 15 steps) THEN
      RE-COMPUTE the Jacobian and the iteration matrix,
   SET the iteration counter i to 0,
   SET maximum iteration limit imax,
   WHILE NOT(converged) AND (i < imax)
      COMPUTE y_{n+γ} using (3.13),
   IF converged
      RE-SET the iteration counter i to 0,
      WHILE NOT(converged) and (i < imax)
         COMPUTE y_{n+1} using (3 14),
      IF converged
         COMPUTE the error estimate E_{n+1}
            using (3 9) and (3 10),
         COMPUTE e_n = rtol|y_{n+1}| + atol;
         COMPUTE the mean square norm
         r^2 = 1/N { Σ_{i=1}^{N} (E_{n+1,i}/e_{n+1,i})^2 };
         IF r ≥ 1 THEN (reject the step)
            h_{n+1} = h_n/2,
         ELSE (accept the step)
            y_n = y_{n+1}
            IF r > 1/2 THEN
               SET h_{n+1} = h_n,
            ELSE
               SET h_{n+1} = r^{1/3} × h_n
            ENDIF,
            {Remark 3 6 also applies here also }
         ENDIF,
      ENDIF,
   ENDIF,
   IF NOT(converged) (reject the step)
      SET h_{n+1} = h_n/4
   END { WHILE}
END {Algorithm Composite Integration scheme }
```

Note We follow Carroll [18], and set the iteration limit imax to 5 We also point out that *remarks 3 6* and *3 7* also apply to the Composite Integration scheme for exactly the same reasons as the DIRK(2,2) scheme.

Remark 3 9 The convergence criteria are those given in subsection 3 5 1

Remark 3 10 This algorithm is very similar to the algorithm given by Carroll [18] There is one significant difference, *(i e)* we reject the step if the *Newton* iteration fails This alteration is essential for solving Differential Algebraic Equations (DAEs), which we consider in later Chapters.

3.6.3 Estimating the initial steplength.

To complete both algorithms we provide a means of estimating the initial steplength We follow Shampine & Watt [67] and suppose the error in a first order method is h times the error in a $zero^{th}$ order method

We place an upper bound h_{input} (given below) on the size of the initial step, which indicates the general scale of the problem, along with preventing any difficulty due to $\mathbf{f}(0, \mathbf{y}(0)) = 0$ We use their estimate of the initial step size as

$$h_0 = min\left(h_{input}, \frac{1}{4}\left\{\frac{tol}{\max_{i=1}^{n} f_i(0, \mathbf{y}(0))}\right\}^{1/2}\right)$$

and set $h_{input} = tol/10$

3.7 Numerical experiments.

The performance of the two variable step algorithms outlined in previous sections are evaluated against several test problems that have arisen in the literature Specifically we test our methods on problems B1, B5, C1, C5, D1, D2 and E3 of the well known stiff ODE test problems of Enright *et al* [27] We reproduce these problems here for completeness Also considered are two other problems which we refer to as P1 and P2 respectively In this section our approach will be to define each problem and discuss its performance before moving to the next problem in our test set

We feel our choice of test problems is representative of those that have appeared in the literature. In particular problems B1, B5, C1 and C5 have also been solved Alexander [1], Enright *et al* [27] and Carroll [18] We have also included D1, D2 and E3, as Carroll [18] observed that his code found it difficult to solve these problems It is our intention to discuss the problems encountered more fully as each problem is dealt with

In our implementation of the Composite Integration scheme, we have followed Carroll [18] and taken $\theta = 0\ 55$ He suggests that this value is close to optimal for the problems arising in the literature In fact, this value was originally suggested by Hall & Watt [38] It provides a compromise between second order accuracy and stability in the θ–scheme (3 5)

In the numerical experiments to follow, we measure the statistics given in the *Key Table* below at different tolerance values for both methods The methods are then evaluated *w r t* these statistics This method of testing and evaluation is analogous to Enright *et al* [27]

NSTEP	*No. of Integration Steps*
NFE	*No of Function Evaluations*
NJE	*No of Jacobian Evaluations*
GERR	*Global Error*

Key Table

We also provide results for the problems using two other polyalgorithms based on BDF formulae, the LSODI package of Hindmarsh & Painter [43] and the DASSL solver of Petzold [56]. The reason for including these results are twofold (a) they are

used for comparison purposes,
(b) they are two of the DAE solvers which we describe and whose performance we evaluate in Chapter 6
We point out that error control for both LSODI and DASSL is accomplished by assigning the absolute (atol) and relative (rtol) tolerances to scalar values and adopting a mixed form of error control.

In our implementation of the DIRK(2,2) scheme we remark that two iteration matrices are used on every step However only one Jacobian of the system of ODEs is evaluated In the figures we quote for the DIRK(2,2) scheme we have adopted the convention of supplying only the actual number of Jacobians evaluated during the integration of a particular problem. The number of LU-decompositions is therefore twice the NJE value

We finally mention that the exact solution for the one-step methods was generated using the NAG routine D02EAF with a tolerance of 10^{-8}. While the exact solution for both routines LSODI and DASSL, was generated by calling these routines with absolute and relative tolerances set to 10^{-8}. Also we point out that in the succeeding discussions, the following notation is used

DIRK(2,2): The DIRK(2,2) scheme outlined in Algorithm 3 6 1

Comp Int The Composite scheme given in Algotithm 3 6 2

LSODI BDF-based code by Hindmarsh [43], described more fully in Chapter 6

DASSL BDF-based code by Petzold [56], described in Chapter 6

Carr Carrolls version of the Composite scheme [18]

TRAPEX Trapezoidal Rule with extrapolation based error control, implemented by Enright *et al* [27]

IMPRK A two-stage fourth-order Implicit RK method implemented by Enright *et al* This code also uses extrapolation based error control

Alex Alexanders DIRK(2,2) scheme with extrapolation based error control This code uses full Newton iteration to solve the nonlinear equations Alexander also measures error in RMS norm All other references to error quoted are in maximum norm

Problem B1.

$$\begin{aligned} y_1' &= -y_1 + y_2 \\ y_2' &= -100y_1 - y_2 \\ y_3' &= -100y_3 + y_4 \\ y_4' &= -10000y_3 - 100y_4 \end{aligned}$$

with initial values

$$y_1 = 1 \quad y_2 = 0 \quad y_3 = 1 \quad y_4 = 0$$

and $t \in [0, 20]$

This problem is linear with non real eigenvalues $-1 \pm 10i$, $-100 \pm 100i$. We mention that Enright *et al* [27] comment that most methods require a large number of step changes, both increases and decreases on this problem Therefore we expect that most methods will use a large number of Jacobian evaluations In Table 3 1 to follow, we give our test results for this problem.

N B The Composite Integration scheme does not compute a solution to the same accuracy as the other methods

Problem B1

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int*	*LSODE*	*DASSL*
NSTEP	127	151	227	159
NFE	822	880	478	334
NJE	26	21	36	25
GERR	3.0×10^{-2}	1.0×10^{-1}	3.0×10^{-6}	10^{-8}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	265	619	393	1391
NFE	1710	3690	716	802
NJE	53	85	45	24
GERR	1.0×10^{-4}	1.1×10^{-2}	5.0×10^{-8}	10^{-9}

Table 3.1

The DIRK(2,2) scheme performs moderately well on this problem with a reasonable level of accuracy In Table 3 2 we reproduce Alexander's results for this problem along with those for Enright's IMPRK scheme We mention that Alexander only published statistics for this method at a tolerance of 10^{-2}

Problem B1

$Tol = 10^{-2}$	*Alex*	*IMPRK*
NSTEP	67	37
NFE	435	443
NJE	28	12
GERR	8.4×10^{-3}	1.8×10^{-2}

$Tol = 10^{-4}$	*Alex*	*IMPRK*
NSTEP		142
NFE		1751
NJE		26
GERR		2.0×10^{-4}

Table 3 2

As can be seen from Table 3 2 the IMPRK scheme is least expensive However the error in all Enright's codes [27] is the maximum error/unit-step encountered over the integration interval It may therefore be less stringent than the other forms of control on this problem Enright *et al* comment that error/unit-step control is usually less problem dependant than other forms and therefore more suitable for test comparisons

The performance of the Composite Integration scheme can be directly compared with the results published by Carroll [18] He also compares his algorithm with SDBASIC and TRAPEX given in Enright *et al* SDBASIC is a variable step variable order (VSVO) multistep code using methods of orders four to nine We feel therefore that it is unreasonable to compare SDBASIC with our second order schemes However the TRAPEX algorithm does provide a reasonable level of comparision. Before we reproduce the statistics for Carroll's code and TRAPEX we mention that Carroll does not provide global error values on some problems Where this statistic is unavailable we have ommitted it

Problem B1		
$Tol = 10^{-2}$	*Carr*	*TRAPEX*
NSTEP	98	69
NFE	464	511
NJE	19	20
GERR		2.6×10^{-2}

$Tol = 10^{-4}$	*Carr*	*TRAPEX*
NSTEP	409	204
NFE	1814	1502
NJE	67	29
GERR		2.0×10^{-4}

Table 3.3

Table 3.3 duplicates the figures published by Carroll [18] and Enright *et al* [27] for this problem. It can be seen from Table 3.1 that our implementation of the Composite scheme is more expensive than the methods quoted in Table 3.3. We expect that the TRAPEX code will be less expensive as it requires less function evaluations/step. All methods require roughly the same number of Jacobian evaluations with both our algorithm and Carroll's requiring about five function evaluations/step.

Before moving on to the next problem we mention the performance of LSODI and DASSL on this problem. Both methods perform quite well at the tolerance values considered with moderately more Jacobian evaluations.

Problem B5

$$\begin{aligned} y_1' &= -10y_1 + \alpha y_2 \\ y_2' &= -\alpha y_1 - 10y_2 \\ y_3' &= -4y_3 \\ y_4' &= -y_4 \\ y_5' &= -0.5y_5 \\ y_6' &= -0.1y_6 \end{aligned}$$

with initial values

$$y_i = 1 \quad i = 1(i)6,$$

$\alpha = 100$ and $t \in [0, 20]$.

This problem is linear and has non real eigenvalues. This problem is known to cause severe difficulties for BDF-based codes as the transient eigenvalues he in an unstable region for higher order BDF formulae. Indeed we can see that the performance of both LSODI and DASSL is very poor as Table 3.4 demonstrates at the higher tolerance value.

For the DIRK(2,2) scheme we again compare the performance of this algorithm with the Alex and IMPRK codes in Table 3.5.

The performance of both codes listed in Table 3.5 is very reasonable on this problem. They both use a moderate number of steps, function evaluations and Jacobian evaluations, with reasonable levels of global error. Compared to these results the performance of the DIRK(2,2) scheme in Table 3.4 is very poor. We have noticed that our DIRK(2,2) scheme appears to behave poorly in the presence of wildly oscillating solutions. That is, solutions with large imaginary eigenvalues which are not rapidly damped out. This observation is born out by the statistics given in Table 3.6 for problems B3 and B4. These problems are identical to B5 except that the parameter

Problem B5				
$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	761	69	125	235
NFE	4626	370	253	434
NJE	81	11	16	12
GERR	3.7×10^{-3}	1.2×10^{-2}	1.0×10^{-3}	8.0×10^{-1}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	1046	278	2379	500
NFE	6330	1326	3762	1008
NJE	106	31	145	7
GERR	9.8×10^{-5}	1.0×10^{-3}	1.0×10^{-5}	5.0×10^{-5}

Table 3.4

Problem B5		
$Tol = 10^{-2}$	*Alex*	*IMPRK*
NSTEP	52	30
NFE	342	361
NJE	15	12
GERR	2.7×10^{-2}	5.0×10^{-3}

$Tol = 10^{-4}$	*Alex*	*IMPRK*
NSTEP		88
NFE		1057
NJE		13
GERR		7.0×10^{-5}

Table 3.5

α is set to 8 and 25 for B3 and B4 respectively. Thus the solutions to these problems do not oscillate as wildly as the solutions to B5.

It is apparent from Table 3.6 that the performance of our code on problems B3 and B4 is similar to both Alexander's [1] and Enright *et al* [27] on problem B5. It therefore appears that our code is unsuitable for problems with extremely large imaginary eigenvalues. We comment that similar behaviuor is observed at the higher tolerance value.

Again we compare the Composite scheme with Carroll's code and TRAPEX in Table 3.7.

Both TRAPEX and Carroll's algorithm (*c.f.* Table 3.7) once again prove much more efficient than our Composite scheme. We mainly attribute this discrepency to the conservative approach we have adopted to handling a failed Newton iteration step. Our code is on average least expensive on Jacobian evaluations/step, but as the global error is larger the method is the least successful at integrating this problem.

Problem C1:

$$\begin{aligned} y_1' &= -y_1 + y_2^2 + y_3^2 + y_4^2 \\ y_2' &= -10y_2 + 10(y_3^2 + y_4^2) \\ y_3' &= -40y_3 + 40y_4^2 \\ y_4' &= -100y_4 + 2 \end{aligned}$$

$Tol = 10^{-2}$	*B3*	*B4*
NSTEP	58	68
NFE	354	414
NJE	11	12
GERR	5.1×10^{-3}	5.0×10^{-3}

Table 3.6

Problem B5

$Tol = 10^{-2}$	*Carr*	*TRAPEX*
NSTEP	49	41
NFE	261	297
NJE	9	14
GERR		2.0×10^{-2}

$Tol = 10^{-4}$	*Carr*	*TRAPEX*
NSTEP	199	178
NFE	852	1265
NJE	20	17
GERR		2.0×10^{-5}

Table 3.7

with initial values

$$y_i = 1 \quad i = 1(i)4$$

and $t \in [0, 20]$.

This problem exhibits nonlinear coupling from the transient to the smooth components. The stiffness ratio is 100 and all eigenvalues are real. Once again the BDF based codes behave very well on this problem in terms of the statistics we give in Table 3.8.

For this problem, as with problem C5 to be considered later, we compare our results for the DIRK(2,2) scheme with those of Alexander [1] and Cash [21]. The results reproduced here from Cash [21] are for a fourth order Strongly S-stable scheme with a third order embedded scheme to estimate the error. The results are summarisied in Table 3.9, where we use Cash to denote Cash's scheme and only give figures for the 10^{-2} tolerance value.

Both second order schemes DIRK(2,2) (*c.f* Table 3.8) and Alexander's produce comparable statistics at the low tolerance value. The fourth order method of Cash produces a greater number of function evaluations/step. We would expect this behaviour from this method. Our scheme uses half the number of Jacobian evaluations reflecting our design criteria that the method should be cheap *w.r.t.* this statistic. It is therefore about twice as efficient as the other two methods on this problem. Statistics are unavailable from Alexander [1] at the 10^{-4} tolerance, we therefore do not include any further comparisions.

The Composite scheme has also produced very good results for this problem. Analysis of the statistics reveals our code to be more expensive than Carroll's (see Table 3.10). The TRAPEX code gives rise to similar behaviour as the statistics given in Table 3.10 demonstrate. The difference between the figures for the Composite scheme and Carroll's code is again due to our conservative approach to dealing with

Problem C1

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	22	32	44	46
NFE	132	128	132	97
NJE	5	9	15	13
GERR	3.0×10^{-4}	3.6×10^{-2}	3.0×10^{-6}	4.0×10^{-5}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	56	85	98	107
NFE	342	538	210	224
NJE	12	10	19	19
GERR	2.0×10^{-9}	7.5×10^{-5}	3.0×10^{-7}	2.0×10^{-6}

Table 3.8

Problem C1

$Tol = 10^{-2}$	*Alex*	*Cash*
NSTEP	20	27
NFE	139	526
NJE	11	11
GERR	2.3×10^{-3}	4.7×10^{-3}

Table 3.9

failed Newton iteration steps. This fact may also account for the poor performance of TRAPEX on this problem as compared to Carroll's implementation.

We note that the TRAPEX uses about two Jacobian evaluations/step, making it very uncompetitive overall.

Problem C5:

$$\begin{aligned} y_1' &= -y_1 + 2 \\ y_2' &= -10y_2 + \beta y_1^2 \\ y_3' &= -40y_3 + 4\beta(y_1^2 + y_2^2) \\ y_4' &= -100y_4 + 10\beta(y_1^2 + y_2^2 + y_3^2) \end{aligned}$$

with initial values

$$y_i = 1 \quad i = 1(i)4$$

$\beta = 100$ and $t \in [0, 20]$.

This problem exhibits nonlinear coupling from the smooth to the transient components. The remaining characteristics are similar to those of problem C1. The BDF-based codes again perform very well on this problem as Table 3.11 shows.

We make the same comparisions for this problem as we made the previous problem. Table 3.12 summerises the results of both Alexander's and Cash's codes. Once again

Problem C1

$Tol = 10^{-2}$	*Carr*	*TRAPEX*
NSTEP	22	9
NFE	73	101
NJE	8	16
GERR		$1\,0 \times 10^{-3}$

$Tol = 10^{-2}$	*Carr*	*TRAPEX*
NSTEP	68	20
NFE	336	261
NJE	11	16
GERR		$2\,0 \times 10^{-5}$

Table 3 10

Problem C5

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int.*	*LSODI*	*DASSL*
NSTEP	73	43	42	48
NFE	438	273	103	107
NJE	12	10	10	14
GERR	$6\,0 \times 10^{-4}$	$6\,7 \times 10^{-3}$	$1\,0 \times 10^{-4}$	$2\,0 \times 10^{-4}$
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	210	153	99	103
NFE	1290	1064	228	208
NJE	32	30	20	21
GERR	$1\,0 \times 10^{-9}$	$1\,5 \times 10^{-4}$	$4\,0 \times 10^{-6}$	$2\,0 \times 10^{-6}$

Table 3 11

the lower tolerance value is quoted because Alexander only gives results at this value and Cash uses a fourth order code. Comparisions with Cash's scheme are therefore a little unrealistic as the tolerance is reduced

Again all methods behave reasonably well on this problem as can be inferred from Tables 3 11 and 3 12 Based on the statistics it appears that this type of problem is quite amenable to solution by most stiff solvers

The Composite scheme is also cheap on this problem Comparision with Carroll's code and TRAPEX whose performance figures are reproduced in Table 3 13, reveal our code to be significantly cheaper.

The reason our code is significantly cheaper on this problem is that when our code fails the Newton iteration step we reduce the stepsize by a factor of four, re-evaluate the Jacobian and re-take the step This feature enhances behaviour on some nonlinear problems

Problem D1

$$\begin{aligned} y_1' &= 0\,2(y_2 - y_1) \\ y_2' &= 10y_1 - (60 - 0\,125y_3)y_2 + 0\,125y_3 \\ y_3' &= 1 \end{aligned}$$

with initial values

$$y_1 = 0, y_2 = 0, y_3 = 0$$

and $t \in [0, 400]$

This problem is nonlinear with real eigenvalues Specifically we compare our results given in Table 3 14 with Carroll's code and Enright *et al* IMPRK and TRAPEX codes given in Table 3 15

On this problem the Composite scheme proves more expensive than the methods listed in Table 3 15 At the higher tolerance similar behaviour is observed, the problem proving difficult for the Composite scheme and those methods given in Table 3.15 Once again the reason for the large number of Jacobians required by the Composite scheme is primarily due to the conservative approach adopted to failed Newton iteration steps. The Composite scheme once again fails to compute a solution to the same accuracy as the other methods.

Comparing the DIRK(2,2) scheme with the IMPRK method, we see that the former is considerably more efficient This fact, once again adds weight to our claim that the DIRK(2,2) algorithm seems quite cheap on problems with real eigenvalues Indeed the DIRK(2,2) scheme compares favourably with both LSODI and DASSL, whose performance is once again excellent at both tolerances

Problem C5

$Tol = 10^{-2}$	*Alex*	*Cash*
NSTEP	27	78
NFE	188	1333
NJE	13	26
GERR	$5\,0 \times 10^{-5}$	$7\,7 \times 10^{-3}$

Table 3 12

Problem C5

$Tol = 10^{-2}$	*Carr*	*TRAPEX*
NSTEP	234	56
NFE	1792	1598
NJE	19	35
GERR		$1\,2 \times 10^{-2}$

$Tol = 10^{-4}$	*Carr*	*TRAPEX*
NSTEP	1029	298
NFE	5471	9257
NJE	164	163
GERR		$1\,9 \times 10^{-4}$

Table 3 13

Problem D1

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP	29	212	23	52
NFE	154	1036	97	107
NJE	6	56	9	19
GERR	$2\,0 \times 10^{-4}$	$3\,9 \times 10^{-3}$	$1\,0 \times 10^{-3}$	$1\,0 \times 10^{-6}$
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int.*	*LSODI*	*DASSL*
NSTEP	50	369	55	114
NFE	318	2420	164	231
NJE	11	121	13	23
GERR	$2\,0 \times 10^{-6}$	$2\,5 \times 10^{-5}$	$2\,0 \times 10^{-5}$	1.0×10^{-6}

Table 3 14

Problem D1

$Tol = 10^{-2}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	130	19	20
NFE	806	785	659
NJE	23	36	65
GERR		$3\,0 \times 10^{-3}$	$2\,4 \times 10^{-4}$

$Tol = 10^{-4}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	567	46	67
NFE	3287	1983	2009
NJE	99	152	55
GERR		$7\,0 \times 10^{-5}$	6.3×10^{-3}

Table 3 15

Problem D2

$$\begin{aligned} y_1' &= -0\,04y_1 + 0\,01y_2y_3 \\ y_2' &= 400y_1 - 100y_2y_3 - 3000y_2^2 \\ y_3' &= 3000y_2^2 \end{aligned}$$

with initial values

$$y_1 = 1, \quad y_2 = 0, \quad y_3 = 0$$

and $t \in [0, 40]$

Again this problem is similar to D1 in that it is nonlinear with real eigenvalues We make comparisions similar to those made for problem D1. Table 3 16 lists the statistics generated by our methods applied to this problem while Table 3 17 reproduces the statistics taken from the literature.

Problem D2

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int.*	*LSODI*	*DASSL*
NSTEP	41	145	36	43
NFE	246	646	103	86
NJE	9	43	13	16
GERR	$1\,0 \times 10^{-2}$	$2\,3 \times 10^{-3}$	$7\,0 \times 10^{-3}$	$9\,0 \times 10^{-3}$
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	79	711	92	94
NFE	474	3505	209	191
NJE	13	239	19	23
GERR	$6\,0 \times 10^{-3}$	$5\,5 \times 10^{-5}$	$3\,0 \times 10^{-4}$	$1\,0 \times 10^{-4}$

Table 3.16

On this problem our Composite scheme (*c f* Table 3.16) produces figures similar to those listed in Table 3 17 for Carroll's code at both tolerance values listed The other methods listed perform much better, so we conclude that this problem is unsuitable for solution by the Composite scheme.

The DIRK(2,2) scheme employed solves the problem reasonably efficiently Again the method is proving suitable for this problem which does not oscillate wildly We comment also that the BDF based methods perform extremely well on this problem indicating their appropriateness for solving mildly oscillatory systems of ODEs

Problem E3

$$\begin{aligned} y_1' &= -(55 + y_3)y_1 + 65y_2 \\ y_2' &= 0\,0785(y_1 - y_2) \\ y_3' &= 0\,1y_1 \end{aligned}$$

with initial values

$$y_1 = 1, \quad y_2 = 1, \quad y_3 = 0$$

Problem D2

$Tol = 10^{-2}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	130	11	11
NFE	622	237	387
NJE	22	29	89
GERR		0.0×10^{-0}	1.0×10^{-3}
$Tol = 10^{-4}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	561	21	16
NFE	2692	593	385
NJE	80	40	25
GERR		1.0×10^{-5}	4.3×10^{-4}

Table 3.17

and $t \in [0, 500]$.

This is the last problem from the Enright *et al* test set. It is nonlinear with non-real eigenvalues. In particular the eigenvalues values stay close to the real axis and therefore the problem does not possess highly oscillatory solutions. We endeavour to make the same comparisions as we did for the previous problem.

Problem E3

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	31	148	39	34
NFE	186	647	118	78
NJE	7	41	16	14
GERR	3.0×10^{-2}	8.8×10^{-4}	2.0×10^{-3}	2.0×10^{-3}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	57	287	85	88
NFE	342	1585	177	195
NJE	11	85	17	15
GERR	2.0×10^{-3}	5.0×10^{-5}	1.0×10^{-4}	1.0×10^{-4}

Table 3.18

Comparing our results in Table 3.18 with those published by Carroll and replicated in Table 3.19, similar behaviour is observed at both tolerance values. Both algorithms are however less efficient than IMPRK and TRAPEX. Based on our results we again conclude that this scheme is unsuitable for this problem.

Looking at the DIRK(2,2) scheme we see that it compares favourably with all other results quoted, espcially in terms of function evaluations. Again the nature of this problem proves amenable to to solution by our DIRK(2,2) scheme. Finally we

Problem E3

$Tol = 10^{-2}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	129	12	9
NFE	407	247	217
NJE	31	26	40
GERR		1.0×10^{-3}	3.0×10^{-3}
$Tol = 10^{-4}$	*Carr*	*TRAPEX*	*IMPRK*
NSTEP	396	21	18
NFE	1412	555	427
NJE	130	38	18
GERR		$4\,0 \times 10^{-5}$	$1\,8 \times 10^{-3}$

Table 3 19

mention LSODI and DASSL on this problem We can see from Table 3 18 that both methods perform very well once again

Problem P1

$$y_i' = -\beta_i y_i + y_i^2 \quad i = 1(1)4$$

with

$$\beta_1 = -1000, \ \beta_2 = -800, \ \beta_3 = -10 \ \textit{and} \ \beta_4 = -0\,1$$

with initial values

$$y_i = -1 \quad i = 1(i)4$$

and $t \in [0, 20]$ The exact solution of this problem is

$$y_i(t) = \frac{\beta_i}{1 - (1 + \beta_i)\, e^{\beta_i t}}$$

This problem is a Ricatti type equation. Our results are displayed in Table 3 20 while Table 3 21 shows the performance of Carroll's version of the Composite scheme on this problem

The results from both tables demonstrate that all methods applied to this problem behave similarly The problem is solved by all methods quite efficiently with reasonable values for global error.

Problem P2

$$\begin{aligned} y_1' &= -0\,04 y_1 + 10^4 y_2 y_3 \\ y_2' &= 0\,04 y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2 \\ y_3' &= 3 \times 10^7 y_2^2 \end{aligned}$$

with initial values

$$y_1 = 1, y_2 = 0, y_3 = 0$$

and $t \in [0, 40]$

Problem P1

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	26	32	85	50
NFE	156	150	177	110
NJE	6	8	17	20
GERR	5.0×10^{-3}	5.0×10^{-3}	1.0×10^{-3}	6.0×10^{-3}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	60	95	131	110
NFE	366	594	300	234
NJE	12	11	27	19
GERR	1.0×10^{-3}	3.0×10^{-4}	2.0×10^{-5}	2.0×10^{-5}

Table 3.20

Carr

Tol	10^{-2}	10^{-4}
NSTEP	37	115
NFE	174	672
NJE	10	13
GERR	1.6×10^{-2}	4.4×10^{-4}

Table 3.21

This problem has been considered by many authors including Hall & Watt [38], Prothero & Robinson [60] and Carroll [18]. Our results are presented in Table 3.22. We also quote Carroll's results in Table 3.23.

The figures quoted in both Tables 3.22 and 3.23 indicate that our codes produce similar results to those of Carroll. All codes solve the problem efficiently. We therefore conclude that this problem is suitable for solution by the stiff ODE codes considered here.

Finally, to sum up we adopt the approach of Carroll [18] providing totals for each statistic in Table 3.24. This table summarises the results given in a convenient form for general discussion. We have not included a summary of the global error. In all, statistics for five methods are diplayed, the four codes we have tested, DIRK(2,2), Comp. Int., LSODI and DASSL, along with Carroll's version of the Composite scheme denoted by Carr.

Table 3.24 clearly shows that our fixed order schemes are uncompetitive when compared to the BDF based codes in terms of steps and function evaluations. The Composite scheme performs worst on the problems considered. However we point out that the problems chosen were those those distinguished by Carroll [18] to be the 'worst case' set available from Enright *et al* stiff ODE test set. It is therefore reasonable for us to observe poorest performance on these problems. We mention

Problem P2

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP	24	35	46	14
NFE	144	116	114	86
NJE	5	10	36	7
GERR	1.0×10^{-2}	1.2×10^{-3}	5.0×10^{-2}	6.0×10^{-3}

$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODE*	*DASSL*
NSTEP	45	60	37	24
NFE	270	266	56	50
NJE	7	16	15	7
GERR	3.0×10^{-3}	8.9×10^{-5}	3.0×10^{-5}	1.0×10^{-4}

Table 3.22

Carr

Tol	10^{-2}	10^{-4}
NSTEP	35	54
NFE	99	230
NJE	8	12
GERR	3.6×10^{-3}	1.1×10^{-4}

Table 3.23

that the figures quoted for Carroll's code should in fact be NSTEP-1 less than those quoted here His code uses three function evaluations on every step However two will suffice as the function call on the previous step at time $t_{n-1} + h$ will be very close to the value of the function at t_n on the current step Ofcourse the former implementation is effectively PECE which is more stable (c *f* Hall & Watt [38]) than the latter PEC implementation In fact we would recommend the former when solving DAEs which we consider in subsequent chapters

The DIRK(2,2) algorithm also fairs badly overall But most of the fault lies with problem B5 In fact this problem accounts for over half of the total figures quoted for this method in table 3.24 As we have already stated, the highly oscillatory nature of the solutions proves to be a problem for our code. Recall we demonstrated that problems B3 and B4 which were similar to B5, but the smaller imaginary eigenvalues proved easy for our code to handle.

In conclusion, the fixed order algorithms we have discussed can provide a competitive alternative to the polyalgorithms LSODE and DASSL based on BDF formulae in certain instances In particular the number of Jacobian evaluations required by the fixed order algorithms is low, at low tolerance This is particularly important since a Jacobian evaluation requires N^2 function evaluations by finite differences This is a crucial factor in the efficiency of these methods when applied to the systems of time

Totals for all problems

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int.*	*LSODE*	*DASSL*	*Carr*
NSTEP	1134	1244	667	681	864
NFE	6904	4278	1575	1467	4698
NJE	157	210	168	140	149

$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODE*	*DASSL*	*Carr*
NSTEP	1864	2665	3369	1531	3398
NFE	11400	15024	5882	3123	16766
NJE	259	630	320	158	596

Table 3 24

dependant Partial Differential Equations (PDEs).

Chapter 4

Differential Algebraic Equations (DAEs).

4.1 Introduction

The general first order differential system described by

$$\mathbf{F}(t, \mathbf{y}(t), \mathbf{y}'(t)) = 0 \qquad t \in [a, b] \tag{4 1}$$

is called a vector implicit system of ODEs or simply an implicit system of ODEs [1] These systems look similar to standard explicit first order ODE systems, which we have dealt with in earlier Chapters and of course include explicit first order systems as a special case

If we assume $\mathbf{F}(\ )$, has continuous first partial derivatives, we can differentiate (4 1) *w r.t* t as follows

$$\frac{\partial \mathbf{F}}{\partial \mathbf{y}'}\mathbf{y}'' + \frac{\partial \mathbf{F}}{\partial \mathbf{y}}\mathbf{y}' + \frac{\partial \mathbf{F}}{\partial t} = 0 \tag{4 2}$$

Letting $\mathbf{y} = \mathbf{y}_1$ and $\mathbf{y}' = \mathbf{y}_2$, we have

$$\begin{aligned} \mathbf{y}'_1 &= \mathbf{y}_2 \\ \frac{\partial \mathbf{F}}{\partial \mathbf{y}_2}\mathbf{y}'_2 &= -\left\{\frac{\partial \mathbf{F}}{\partial \mathbf{y}_1}\mathbf{y}_2 + \frac{\partial \mathbf{F}}{\partial t}\right\} \end{aligned}$$

Since $\mathbf{F}(\ )$ has continuous first partial derivatives, we can assume $\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}_2}\right)^{-1}$ exists and is bounded Therefore we can rewrite the above system in explicit form [2]

$$\begin{aligned} \mathbf{y}'_1 &= \mathbf{y}_2 \\ \mathbf{y}'_2 &= -\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}_2}\right)^{-1}\left\{\frac{\partial \mathbf{F}}{\partial \mathbf{y}_1}\mathbf{y}_2 + \frac{\partial \mathbf{F}}{\partial t}\right\} \end{aligned}$$

Implicit ODE systems where $\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}'}\right)$ is singular are called Differential Algebraic Equations (DAEs).

[1] We assume that $\mathbf{y}$ and $\mathbf{y}'$ are mappings from $\mathbf{R} \rightarrow \mathbf{R}^n$

[2] This follows from the Implicit Function Theorem of Vector Calculus, see Marsden & Tromba [48]

In this *thesis* we concern ourselves with the study and development of numerical ODE methods for DAEs of the form

$$E(t,\mathbf{y})\,\mathbf{y}' = \mathbf{f}(t,\mathbf{y}) \tag{4 3}$$

where E is a square matrix usually singular Systems of this form are called Linearly Implicit DAEs, because of their linear dependence on $\mathbf{y}'$

There are two special cases of the Linearly Implicit DAE (4 3) that have been studied in the literature.

(a) The Linear Constant Coefficient DAE

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t) \tag{4 4}$$

and

(b). The Linear Non-Constant Coefficient DAE

$$E(t)\mathbf{y}' = A(t)\mathbf{y} + \mathbf{g}(t) \tag{4 5}$$

We devote sections 3 & 4 respectively, of this Chapter, to reviewing the literature on these forms

Other forms have also appeared in the literature, Gear [33] and Petzold & Lotstedt [58], [59] considered Semi-explicit DAEs which have the following structure

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t,\mathbf{y},\mathbf{z}) \\ 0 &= \mathbf{g}(t,\mathbf{y},\mathbf{z}) \end{aligned} \tag{4 6}$$

Brenan & Engquist [4] have considered a special form of the Semi-explicit DAE, called the Triangular (Hessenberg) form given by

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t,\mathbf{y},\mathbf{z}) \\ 0 &= \mathbf{g}(t,\mathbf{y}) \end{aligned} \tag{4 7}$$

Note All the above forms have been studied both analytically and numerically in the literature, however we intend to primarily concern ourselves with their analytic aspects in this Chapter

Returning to the Linearly Implicit equation (4 3), we point out that there is no loss of generality in considering systems of this form, since we can easily transform the general Implicit ODE into a DAE by letting $\mathbf{z} = \mathbf{y}'$ The Implicit ODE then becomes

$$\begin{aligned} \mathbf{y}' &= \mathbf{z} \\ 0 &= \mathbf{F}(t,\mathbf{y},\mathbf{z}) \end{aligned} \tag{4 8}$$

and the equation is now linear in $\mathbf{y}'$, the equation is also in Semi-explicit form

Example 4 1. Consider the following Implicit ODE

$$(y')^2 + y'y = 0, \qquad t \in [0,\infty] \tag{4 9}$$

with $y(0) = 1$ Letting $y' = z$, we get

$$\begin{aligned} y' &= z \\ 0 &= z^2 + z\,y \end{aligned} \tag{4 10}$$

The first point to notice about (4.9) is that only one initial condition is supplied. This would seem reasonable from our knowledge of ODEs, since the equation is first order. However the equation has two solutions, the first is $y(t) = e^{-t}$ and the second is the constant function $y(t) = 1$. Therefore our example problem will not possess a unique solution unless further constraints are imposed. The following definitions, see Campbell [13], help us to characterize the set of conditions, which must be inmposed in order for a DAE to possess a unique solution.

Definition 4.1: An initial vector $\mathbf{y}_a$ is said to be a consistent initial vector for (4.11) at a point $t(a)$, if (4.1) possesses at least one solution.

Definition 4.2: The equation (4.1) is said to be solvable at a point $t(a)$, if a unique solution exists for each consistent initial vector.

Thus in (4.9), $y(0)$ is a consistent initial value, but the equation is not solvable from this point. However in (4.10), it is a trivial task to generate consistent initial values. We substitute the initial $y(0)$ into the algebraic equation and solve the resulting quadratic equation for z, giving two possible initial values, $z = 1$ and -1.

The purpose of this Chapter then, is the study of DAEs. Our discussion centres on two important issues, namely the concept of an *index* or degree of complexity of a DAE and the characterization of consistent initial conditions. We therefore are primarily concerned with analytic solutions of DAEs, where they can be obtained. However certain numerical aspects will be considered where we feel they are appropriate. A fuller treatment of the issues involved in the solution of DAE systems will be given in the next Chapter. We begin by considering what are regarded as the simplest DAE systems, in the sense that they can be solved by ODE methods and are also completely understood analytically.

4.2 Infinitely Stiff ODEs.

We begin this section by considering a special case of the stiff ODE systems discussed in earlier Chapters. In particular we examine the pair of scalar ODEs:[3]

$$\begin{aligned} y'(t) &= f(t, y(t), z(t)) \qquad t \in [a, b] \\ \epsilon z'(t) &= g(t, y(t), z(t)) \end{aligned} \tag{4.11}$$

with $y(a)$ and $z(a)$ given at the initial point $t = a$. In the above equations we assume that $f(\cdot)$, $g(\cdot)$, $y(\cdot)$ and $z(\cdot)$ are $O(1)$, while ϵ is a small parameter different from *zero*. We also assume that these functions are continuous throughout the interval and satisfy the conditions laid down for (2.1). Under these conditions, the stiffness of (4.11) is determined by ϵ and the stiffness ratio is of order $(1/\epsilon)$.

The scalar *Infinitely Stiff ODE,* is a generalization of the stiff system (4.11), obtained by considering the limiting case $\epsilon = 0$, giving the semi-explicit DAE system

$$\begin{aligned} y' &= f(t, y(t), z(t)) \qquad t \in [a, b] \\ 0 &= g(t, y(t), z(t)) \end{aligned}$$

with $y(a)$ and $z(a)$ given.

[3]Recall, this is the scalar singular perturbation problem, introduced in Chapter 1.

Remark 4 1. It was this relationship between *Infinitely Stiff ODEs* and *semi-explicit DAEs,* that prompted Gear [32] to propose solving these problems using standard stiff ODE integration schemes, based on the implicit numerical methods of Chapter 2

We complete the characterization of *Infinitely Stiff ODEs,* by giving a formal definition. We denote the differential "state" variables by the vector $\mathbf{y}(t)$ and the algebraic "non-state" variables by the vector $\mathbf{z}(t)$

Note We shall often refer to the algebraic subsystem in the definition, as the constraints of the system.

Definition 4 3 (Infinitely Stiff ODE systems) The Differential Algebraic System

$$\begin{aligned} \mathbf{y}'(t) &= \mathbf{f}(t, \mathbf{y}(t), \mathbf{z}(t)) \qquad t \in [a, b] \\ 0 &= \mathbf{g}(t, \mathbf{y}(t), \mathbf{z}(t)) \end{aligned} \tag{4.12}$$

where

$$\mathbf{y}(t) \text{ and } \mathbf{z}(t) \quad \mathbf{R} \rightarrow \mathbf{R}^n \text{ and } \mathbf{R}^m, \text{ respectively}$$

and

$$\mathbf{f}(\cdot) \text{ and } \mathbf{g}(\cdot) \quad \mathbf{R}^{n+m+1} \rightarrow \mathbf{R}^n \text{ and } \mathbf{R}^m, \text{ respectively}$$

possess a unique solution for consistent initial conditions on $\mathbf{y}(a)$ and $\mathbf{z}(a)$, provided $\mathbf{f}(\,)$ and $\mathbf{g}(\,)$ satisfy the Lipshitz conditions [4]

$$\|\mathbf{f}(t, \mathbf{y}_1, \mathbf{z}) - \mathbf{f}(t, \mathbf{y}_2, \mathbf{z})\| \leq L_1 \|\mathbf{y}_1 - \mathbf{y}_2\|$$

$$\|\mathbf{g}(t, \mathbf{y}, \mathbf{z}_1) - \mathbf{g}(t, \mathbf{y}, \mathbf{z}_2)\| \leq L_2 \|\mathbf{z}_1 - \mathbf{z}_2\|$$

for all $t \in [a, b]$

Remark 4 2 These conditions are minimal and also require that the Jacobian of the non-state variables $\frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ be non-singular, (see Cameron [9])

Since the Jacobian of the non-state variables exists and is bounded $\forall\, t \in [a, b]$, we can differentiate the constraint and apply the *Implicit Function Theorem.* [48] This transforms the constraint equations into a differential system, as follows[5]

$$0 = \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \mathbf{y}' + \frac{\partial \mathbf{g}}{\partial \mathbf{z}} \mathbf{z}'$$

giving

$$\mathbf{z}' = \left(\frac{\partial \mathbf{g}}{\partial \mathbf{z}} \right)^{-1} \left\{ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \mathbf{f}(\mathbf{y}, \mathbf{z}) \right\}$$

This differential system can now be solved by the techniques used in earlier Chapters

Using the *Implicit Function Theorem* in this way appears to answer all our needs for this problem, but it does have serious drawbacks. Firstly, the transformation is analytic and tedious to compute This can be overcome, to some extent, by decoupling the differential and algebraic subsystems in (4 12). The individual state and non-state subsystems are then solved independently, at each step of the time interval, using a suitable numerical integration scheme for the state components and an inner Newton

[4] We drop the dependence $\mathbf{y}(t)$ and $\mathbf{z}(t)$ on t for clarity

[5] This transformation is identical to that used in (4 2) Once again we drop the dependance on t for clarity

iteration for the non-state components Cameron [10], in his thesis solved chemical systems in this way using functional iteration for the differential subsystem of (4 12) The approach seems reasonable since the state equations are non-stiff However, for tightly coupled state and non-state subsystems the performance of this approach was poor We note, that the effect of increased coupling between the two subsystems, is equivalent to a virtual instantaneous change from a non-stiff to an *infinitely stiff* ODE system This to some extent explains the existence of Dirac δ functions in the solution of DAEs We will return to this topic later in this Chapter.

The second major drawback associated with the *Implicit Function Theorem* is *sparsity* If the original DAE system is sparse, then the transformation outlined will not, in general, preserve the original system structure. The resulting ODE system may be dense and therefore the *storage and calculation* of Jacobian matrices required for numerical solution is greatly increased

The last question we address regarding (4 12) is the existence of a consistent set of initial conditions[6] which guarantee a unique solution. In general this is a non trivial task However the question can be satisfactorily answered for (4.12) It is reasonable to assume that the initial conditions for the state equations are automatically consistent in (4 12) The non-state initial values can be easily computed by substituting the state values into the algebraic equations and using a Newton iterative scheme on these equations only This automatically generates consistent non-state initial values It is also expedient to use a *damped Newton iteration* for this purpose, as conditions for a descent direction may not be automatically satisfied for a full Newton iteration scheme In this situation the full Newton scheme may diverge or possibly "hunt" around a saddle point

4.3 Linear Constant Coefficient DAEs

In this section we review the structure of the linear constant coefficient DAE given by

$$A\,\mathbf{y}'(t) + B\,\mathbf{y}(t) = \mathbf{g}(t) \qquad t \in [a, b] \tag{4 13}$$

with $\mathbf{y}(a)$ and $\mathbf{y}'(a)$ given A and B are assumed to be $n \times n$ dimensional matrices, both possibly singular and $\mathbf{y}(t)$ and $\mathbf{y}'(t)$ are mappings from $\mathbf{R} \rightarrow \mathbf{R}^n$

In particular we define the concept of *index* for (4 13) and derive the general solution In the literature, (4 13) has been referred to by different names Sincovec *et al* [68] follow Luenberger [47] and call (4 13) a Descriptor System, while Campbell [13] and Newcomb [51] call (4 13) a Singular System and a set of semi-state equations respectively We shall use the title *linear constant coefficient DAE,* which is now common in the literature

When the matrices A or B are singular, the structure of (4 13) can be completely understood via a *canonical* form, called the Kronecker Canonical Form (KCF) for the matrix pencil $(A + \lambda B)$, with λ a scalar. In fact if the matrix $(A + \lambda B)^{-1}$ exists and is bounded, then (4 13) will have a solution. We formalize this statement with a definition of *solvability,* see Campbell [13].

Definition 4 4. The linear constant coefficient DAE (4 13) is solvable *iff*

$$\det(A + \lambda B) \neq 0$$

[6] We have assumed consistent initial conditions

Remark 4 3 When $(A+\lambda B)$ is singular for all values of λ in (4 13), then either no solutions or infinitely many solutions exist Fortunately the numerical ODE methods we propose in this work reject these problems automatically Our methods factorize a linear system of the form $(A+h\beta B)$, where h is the stepsize and β is a scalar which depends on the method. This matrix is singular for all values of h

Sincovec *et al* [68] apply a non-singular row scaling matrix P and non-singular change of variables matrix Q to (4.13) as follows

$$PAQ\ Q^{-1}\mathbf{y}' + PBQ\ Q^{-1}\mathbf{y} = P\mathbf{g}(t) \tag{4 14}$$

To gain further insight into equation (4.14), it is necessary to define the concept of *nilpotency* for an $n \times n$ matrix.

Definition 4 5 An arbitrary square matrix A is nilpotent, if there exists an integer $m > 0$, such that $A^{m-1} \neq 0$, but $A^m = 0$. The integer m is defined as the *index of nilpotency,* or simply the *index,* for the matrix A.

Remark 4.4. In the case where A is empty, (*the zero by zero matrix,*) we assume $0^0 = I$, the identity matrix

The transformations outlined in (4 14) reduces the DAE (4 13) to the following equivalent system.

$$\begin{aligned} \mathbf{x}_1' + C\mathbf{x}_1 &= \mathbf{f}_1(t) \quad \mathbf{x}_1(t_0) = \mathbf{x}_{1,0} \\ E\mathbf{x}_2' + \mathbf{x}_2 &= \mathbf{f}_2(t) \quad \mathbf{x}_2(t_0) = \mathbf{x}_{2,0} \end{aligned} \tag{4 15}$$

with

$$Q^{-1}\mathbf{y} = [\mathbf{x}_1, \mathbf{x}_2]^T \text{ and } P\mathbf{g} = [\mathbf{f}_1, \mathbf{f}_2]^T$$

and E is a nilpotent matrix of index $m \geq 0$ In general, the matrix E is composed of *Jordan blocks* of the form

$$\begin{bmatrix} 0 & & \cdot & \cdot & \cdot & 0 \\ 1 & 0 & & \cdot & \cdot & \cdot \\ & 1 & 0 & \cdot & & \cdot \\ \cdot & \cdot & & & & \\ \cdot & \cdot & & & & \cdot \\ & & & & 1 & 0 \end{bmatrix}$$

and m is the size of the largest of these blocks If $m = 0$, the system is transformed into an explicit first order ODE system

Remark 4 5 The transformation just outlined completely de-coupled (4 13) into a purely differential part and a purely differential algebraic part We follow the literature and consider the latter case only, as differential systems have been completely dealt with earlier

Example 4 2 (An m = 2 system)

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x}' + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x} = \begin{pmatrix} 0 \\ \sin(t) \end{pmatrix}$$

We can easily solve systems of this form by starting with the last equation to obtain

$$x_2(t) = \sin(t) \text{ and } x_1(t) = x_2'(t) = -\cos(t)$$

Remark 4 6 Luenberger [47] calls systems of this form *Pure predictors,* as the solution is independent of initial conditions Clearly this system is very different from a traditional ODE system, where the constants of integration are uniquely specified by the initial conditions For *Pure Predictor* systems, like the $m = 2$ system above, no constants of integration arise so that the system is independent of initial values

We say the solvability of (4 13) is equivalent to the existence of non-singular matrices P and Q, which transform (4.13) into (4 15) The solution of (4 15), (see Sincovec *et al*), is

$$\begin{aligned} \mathbf{x}_1 &= e^{tC}\,\mathbf{x}_{1,0} + \int_{t_0}^{t} e^{(t-s)C}\,\mathbf{f}_1(s)\,ds \\ \mathbf{x}_2 &= -\sum_{i=0}^{m-1} E^i\,\mathbf{f}_2^{(i)}(t) \end{aligned}$$

with $\mathbf{f}_2^{(i)}$ the i^{th} derivative of $\mathbf{f}_2$ The solution $\mathbf{x}_1$ is well the known integral solution of a differential system For the differential/algebraic subsystem, we verify that $\mathbf{x}_2$ is correct for the $m = 2$ DAE system considered An easy calculation for this example yields

$$\mathbf{x} = \begin{pmatrix} 0 \\ -\sin(t) \end{pmatrix} + \begin{pmatrix} -\frac{d}{dt}\sin(t) \\ 0 \end{pmatrix}$$

which agrees with the solution given earlier

Another interesting, albeit similar characterization has been suggested by Campbell & Meyer [11]. They introduce the notion of a *Drazin* generahzed inverse of a singular matrix and use the concept to generate a solution of (4 13)

Definition 4 6 For any singular matrix A with index $m > 0$ there exists a non-singular matrix P such that

$$A = P \begin{bmatrix} C & 0 \\ 0 & N \end{bmatrix} P^{-1}$$

where C is non-singular and N is nilpotent of index m The *Drazin* Pseudo Inverse of A (written A^D)[7] is then

$$A^D = P \begin{bmatrix} C^{-1} & 0 \\ 0 & 0 \end{bmatrix} P^{-1}.$$

Campbell & Meyer [11] consider the following *commutative* matrices

$$\mathcal{A} = (A + \lambda B)^{-1} A \quad \text{and} \quad \mathcal{B} = (A + \lambda B)^{-1} B$$

Remark 4 7 The commutativity of $\mathcal{A}$ and $\mathcal{B}$ is easily verified By considering the equation

$$\mathcal{A}\,\mathcal{B} = \mathcal{B}\,\mathcal{A}$$

and pre-multiplying both sides by A^{-1} on the left and $(A + B)$ on the right Then, by taking inverses of both sides, equality holds trivially

[7] If A is non-singular, then both the Drazin and ordinary inverses are identical and $A\,A^D = I$

Campbell & Meyer [11] define a general solution to (4 13) in terms of the matrix exponential based on the Drazin inverse of $\mathcal{A}$ For the homogeneous problem

$$A\mathbf{x}' + B\mathbf{x} = 0$$

the solution is

$$\mathbf{x}(t) = e^{-\mathcal{A}^D B(t-t_0)}\, \mathcal{A}\, \mathcal{A}^D\, \mathbf{q}$$

where $\mathbf{q} \in \mathbf{R}^n$ is a vector of initial values.

Example 4.3 Consider the homogeneous linear constant coefficient DAE with

$$A = \begin{bmatrix} 1 & 0 & -2 \\ -1 & 0 & 2 \\ 2 & 3 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 2 \\ -27 & -22 & -17 \\ 18 & 14 & 10 \end{bmatrix}$$

A is singular, but $(A + B)$ is invertible, thus $\lambda = 1$ and

$$\mathcal{A} = 1/3 \begin{bmatrix} -3 & -5 & -4 \\ 6 & 5 & -2 \\ -3 & 2 & 10 \end{bmatrix}$$

The eigenvalues of $\mathcal{A}$ are { 0, 1, 3 }, so that $\mathcal{A}^D$ may be computed from the eigenvectors of $\mathcal{A}$ via a similarity transformation as

$$\mathcal{A}^D = 1/27 \begin{bmatrix} -27 & -41 & -28 \\ 54 & 77 & 46 \\ -27 & -34 & -14 \end{bmatrix}$$

and the general solution follows from above Campbell & Meyer [11] also generahze these results to the inhomogeneous case.

In this section, we have developed two useful characterizations of (4 13) that have appeared in the literature The theory given in Sincovec *et al* is useful in demonstrating how (4 13) naturally de-couples into differential and differential algebraic subsystems This will be our take off point for dealing with these systems later. It is also useful in defining the concept of *index* . Campbell & Meyer [11] and Campbell [13], introduce the idea of a generalized inverse as a generalization of the ordinary inverse of a matrix This proved useful in defining a solution to (4 13) in the classical *exponential* sense and will prove to be useful later when we discuss initial conditions for this problem In the next section, we shall attempt to generalize the ideas introduced here to the non-constant coefficient DAE, where we shall see that these notions will be carried over in a limited sense

4.4 Linear non constant coefficient DAE systems

We explore the linear non constant coefficient DAE system

$$A(t)\,\mathbf{y}'(t) + B(t)\,\mathbf{y}(t) = \mathbf{g}(t) \qquad t \in [a, b] \tag{4 16}$$

with $\mathbf{y}(a)$ and $\mathbf{y}'(a)$ given, by making a brief review of the important contributions that have appeared in the literature, to understanding the structure of this problem

It is our intention to summarize the combined works of Campbell, Gear and Petzold, [13], [14], [16], [17] and [35], in an effort to generalize the results of the previous section to (4 16), we begin with a definition of solvability for this problem.

Definition 4 7. (Campbell & Petzold [17]) We shall say (4.16) is *analytically solvable* on the interval $[a, b]$ if for any sufficiently smooth input function g(t), there exist solutions to (4 16) and these solutions are defined $\forall\ t \in [a, b]$ and are uniquely determined at any time $t \in [a, b]$.

Remark 4 8 Campbell & Petzold point out that (4.16) fails to be analytically solvable if it has a *turning point,* that is, a point in time where the dimension of the manifold of solutions changes, since at these points solutions may fail to exist or be unique (*c f example 4 1 at* $t = 0$)

When the coefficient matrices are not constant as in (4.16), we can define two forms of index The simplest is the *local index* This is the index of the corresponding constant coefficient problem obtained by considering (4 16) at some fixed point in time We can also define the *global index* of (4 16) when it exists in terms of a semi-canonical form, see Gear & Petzold [35]

Once again we consider a change of variables $\mathbf{y} = Q(t)\,\mathbf{x}$ and a row scaling *P(t)*, where *Q(t)* and *P(t)* are non-singular $\forall\ t \in [a, b]$ Applying *P(t)* and *Q(t)* to (4 16) gives

$$P(t)\,A(t)\,Q(t)\,\mathbf{x}' + \{P(t)\,B(t)\,Q(t) + P(t)\,A(t)\,Q'(t)\}\,\mathbf{x} = P(t)\,\mathbf{g}(t)$$

and transforms (4 16) to the semi-canonical form

$$\begin{aligned} \mathbf{x}'_1 + C(t)\,\mathbf{x} &= \mathbf{f}_1(t) \\ N(t)\,\mathbf{x}'_2 + \mathbf{x}_2 &= \mathbf{f}_2(t) \end{aligned} \qquad (4\ 17)$$

where C(t) is non-singular, N(t) is nilpotent and *lower triangular,* as in the time invariant case considered in the last section. Following Campbell & Petzold [17], we say the system is in *Standard Canonical Form* (SCF) and the index-m of (4 16) is the index of nilpotency of N(t) In particular, if N(t) is time invariant we say the system has global index m and (4 17) is in *Strong Standard Canonical Form* (SSCF)

Note that the global index is the local index of the semi-canonical form above, when N(t) is time invariant

We also point out that the SSCF is the canonical form considered in Sincovec *et al* [68], Petzold [55] and Gear & Petzold [35] We mention in passing that Campbell & Petzold [17] provide examples to demonstrate that analytic solvability does not imply the existence of a SSCF as had been originally thought It does however imply the existence of an SCF When the SSCF does not exist, it therefore is not possible to define the concept of global index When the SSCF exists, the global index determines the behaviour of the solution In this case, we know that n_1 independent initial values can be chosen, where n_1 is the dimension of the differential part of the system and the driving term can be subject to $m - 1$ differentiations

Remark 4 9. The local index in some sense governs the behaviour of numerical ODE methods applied to (4 16) For example, if the local matrix pencil is singular, then numerical ODE methods cannot solve the problem, because they will be faced with the solution of a singular linear system In understanding why ODE methods break down, it is natural to ask how the local and global indices are related Gear & Petzold [35] provide the following theorem which answers this question

Theorem 4 1 If the local index is not greater than 1, then it is not changed by a smooth transformation If the local index is greater than 1, then a smooth non-constant transformation of variables in (4 16) will yield a system whose local index is 2, unless additional constraints are satisfied by the transformation. A restricted set of transformations will cause the index to be greater than 2 or the pencil to be singular When the transformation to semi-canonical form is used, this shows the relationship between the local and global indices.

In this section, we have tried to show how the concepts we introduced earlier can be generalized in a useful way to the non-constant coefficient problem We have demonstrated that the concept of index can be generalized via a suitable canonical form, but the characterization is more restricted for (4 16). However, if a global index exists then the non-constant coefficient problem will in a certain sense have a linear DAE subsystem embedded within it

4.5 The general Implicit Differential Equation

We return in this section to the general Implicit ODE introduced in section 4 1,

$$\mathbf{F}(t, \mathbf{y}(t), \mathbf{y}'(t)) = 0, \qquad t \in [a, b], \tag{4 18}$$

with $\mathbf{y}(t)$ and $\mathbf{y}'(t)$ being vector mappings from $\mathbf{R} \rightarrow \mathbf{R}^n$ and $\mathbf{F}(\ )$ a mapping from $\mathbf{R}^{2n+1} \rightarrow \mathbf{R}^n$

It is our intention to tie together the theory of the previous three sections in a more meaningful and appropriate way for this work We intend to develop what is perhaps the best known and simplest definition of index of nilpotency for (4 18), which has appeared in the literature This definition also applies to all other forms of DAE considered earlier. The general idea, is to take the constraint equations and differentiate them to generate an equivalent ODE system via suitable manipulations We give an example to illustrate the technique on a set of Euler-Lagrange equations for the Simple Pendulum This example has appeared in several papers including Gear [33], Petzold & Lotstedt [59] and Pantelides [54]

Example 4 4 In the following second order system (x, y) represents the position of the Pendulum, g the *acceleration due to gravity* and T is a *Lagrange Multiplier* (representing the tension in the string) The equations of motion are

$$\begin{aligned} x'' &= -Tx \\ y'' &= -Ty - g \\ 0 &= x^2 + y^2 - 1 \end{aligned}$$

and the initial values are chosen to satisfy the constraint, (*i e.* any position on the unit circle) We can easily put this in first order form (4 18) by letting $x' = u$ and $y' = v$ giving the following system of semi-explicit DAEs

$$\begin{aligned} x' &= u \\ y' &= v \\ u' &= -Tx \\ v' &= -Ty - g \\ 0 &= x^2 + y^2 - 1 \end{aligned} \tag{4 19}$$

To transform (4 19) into a differential system, we repeatedly differentiate the constraint $w\ r\ t$ time, as follows

$$2\,x\,x' + 2\,y\,y' = 0$$

Substituting for x' and y', with u and v, respectively gives

$$x\,u + y\,v = 0$$

Differentiating this equation and using suitable substitutions we get

$$u^2 + v^2 - T - y\,g = 0$$

One further differentiation of this equation yields the following differential equation for T

$$T' + 3\,v\,g + 2\,T\,(u\,x + v\,y) = 0$$

giving

$$T' = -3\,v\,g$$

since $u\,x + v\,y = 0$, from above We can replace the constraint equation in (4.19), by the ODE for T and a full ODE system results This example prompts the following definition of index for the implicit differential equation and it holds in general It is due to Rheinboldt [61]

Definition 4 8 The index of (4 18) is the minimum number of differentiations of the constraint equations, required to reduce (4 18) to an ODE system

From this definition, it is clear that the index of a DAE represents its degree of complexity, as each extra differentiation increases the number of degrees of freedom of the resulting system by one, eventually reducing the index of the system to zero In this case, a degree of freedom is the introduction of an extra constant of integration, as the requirement for the constraints to be satisfied at the initial point is lifted from the DAE system

The method outlined for finding the index of (4 18) and generating a reduced index-0 problem has been proposed as an algorithm Gear & Petzold [35], Petzold & Lotstedt [58], [59], Gear [33] and Pantelides [54] However this technique has the disadvantage that it may introduce additional instabilities into the problem This can be overcome by stabilizing the problem, that is, by taking a linear combination of the constraint equations with their first and second deriavatives This is called Baumgarte stabilization [30] For the Simple Pendulum equations above, the constraints are
(1) Index-3 position constraint

$$x^2 + y^2 - 1 = c_3(x, y) = 0$$

(2) Index-2 velocity constraint

$$x\,u + y\,v = c_2(x, y, u, v) = 0$$

(3) Index-1 force constraint (Tension)

$$u^2 + v^2 - T - y\,g = c_1(x, y, u, v, T) = 0$$

The constraint equation in the original system is now replaced by the stabilized Index-1 constraint

$$c_1(x,y,u,v,T) + \alpha\, c_2(x,y,u,v) + \beta\, c_1(x,y) = 0$$

Fuhrer [30] suggest that for initial values $x = 1, y = u = v = T = 0$, the system will have a period of exactly 2 seconds In this case he proposes that the stabilizing factors are $\alpha = 50$, and $\beta = 625$. The constants c_1, c_2, c_3, are then chosen to satisfy the constraint at the initial point

Gear [33] has adopted a similar approach. He reintroducs the algebraic equations generated by differentiation into the derived ODE system, making the resulting system overdetermined His method removes the constants of integration introduced by earlier differentiation and simultaneously lowers the index. An example based on the Euler-Lagrange equations for the Simple Pendulum equations is given in Gear [33].

4.6 Initial conditions for DAE systems

In the introduction, we indicated that initial conditions (ic's) for a DAE system may be inconsistent, that is they may fail to satisfy the system at the initial time with the possibility of non-unique solutions at the initial point. We propose to examine this question more fully in this section, reviewing those contributions to the literature that have increased our understanding in this area Our primary sources of reference are Campbell [13], Smcovec *et al* [68] and Pantelides [54]

4.6.1 Initial conditions for the linear constant coefficient problem

We begin by introducing the following index-1 one DAE system which is taken from Smcovec *et al* [68]

Example 4 5

$$\begin{aligned} x_1' + x_2' &= x_1 + x_2 & x_1(0) &= 2 \\ 0 &= x_1 - x_2 - 5 & x_2(0) &= 1 \end{aligned}$$

Letting $y_1 = x_1 + x_2$ and $y_2 = x_1 - x_2$, this system is equivalent to

$$\begin{aligned} y_1' &= y_1 & y_1(0) &= 3 \\ 0 &= y_2 - 5 & y_2(0) &= 1 \end{aligned}$$

We can see from this system, that the i.c on y_2 is not consistent and it appears that no solution exists with this i c However, if we choose to neglect the i c on the algebraic equation we can obtain the following solutions [8]

$$x_1 = \frac{3\,e^t - 5}{2} \qquad x_2 = \frac{3\,e^t + 5}{2}$$

It is therefore reasonable to think of this system as having lost one degree of freedom associated with its i c 's, due to the presence of the constraint

[8]These solutions are the differentiable or smooth solutions, as they can also be obtained by differentiating the constraints

Sincovec *et al*, apply the Backward Euler to this problem and show that the approximate solution obtained is identical to the solution of this problem specified with consistent i c 's on all but the first step, where a sudden jump occurs in the behaviour of the solution. They also comment that a jump at the initial step gives an indication of inconsistent i.c.'s and point out that this situation holds in general for the linear constant coefficient DAE.

Campbell [13] gives a complete analysis of the linear constant coefficient problem Recall from section 3 that we can put this problem into a canonical form and it is sufficient for us to consider the nilpotent equation[9]

$$E\,\mathbf{z}' + \mathbf{z} = \mathbf{f}_2 \qquad t \in [0,\infty] \tag{4 20}$$

where E is the nilpotent operator. Campbell [13] investigates necessary conditions on z, so that a solution (4 20) can be obtained for inconsistent i c's He applied *Laplace transforms* to (4.20) assuming $\mathbf{z}$ and $\mathbf{f}_2$ to be sufficiently smooth $\forall t \geq 0$ We will denote the *Laplace transform* of a function $f(t)$ by $\hat{f}(s)$ Applying *Laplace transforms* to (4 20), gives

$$\hat{\mathbf{z}}(s) = (s\,E - I)^{-1}\,E\,\mathbf{z}(0) + (s\,E - I)^{-1}\,\hat{\mathbf{f}}_2(s)$$

since the index of E is m, we have

$$\hat{\mathbf{z}}(s) = -\sum_{i=0}^{m-1} s^i\,E^{i+1}\,\mathbf{z}(0) + \sum_{i=0}^{m-1} s^i\,E^i\,\hat{\mathbf{f}}_2^{(i)}(s)$$

Taking inverse *Laplace transforms* and denoting the i^{th} distributional derivative of the *Dirac delta function* by $\delta^{[i]}$ we can write the solution of (4 20) as

$$\mathbf{z}(t) = -\sum_{i=0}^{m-1} E^i\,\mathbf{f}_2^{(i)}(t) \; - \sum_{r=0}^{m-2} \delta^{[r]}\,E^{r+1}\left\{\mathbf{z}(0) + \sum_{j=0}^{m-r-2} E^j\,\mathbf{f}^{(j)}(0)\right\} \tag{4.21}$$

Therefore, if $\mathbf{z}(t)$ is continuous on $t \geq 0$, we get

$$\mathbf{z}(0) = -\sum_{i=0}^{m-1} E^i\,\mathbf{f}_2^{(i)}(0) \tag{4 22}$$

and the solution is simply

$$\mathbf{z}(t) = -\sum_{i=0}^{m-1} E^i\,\mathbf{f}_2^{(i)}(t) \tag{4 23}$$

If however z(0) does not satisfy (4 22), then (4 21) provides a solution to (4 20) This solution is impulsive at the origin It explains why a numerical method ODE method will generate the exact solution (4 23) on all steps except the first. In some sense then, these problems admit an infinite boundary layer at the origin, of negligible duration. Sincovec *et al* have shown that this impulse is smoothed out using a k-step BDF method in (m-1)k+1 steps, for an *index m* linear constant coefficient problem For numerical work, where we are primarily concerned with smooth differentiable solutions, the ic's we specify are unimportant, since their distributional nature will

[9]We change the interval because we wish to introduce Laplace transforms

not be exhibited by the numerical method Thus we can say the solution is unique for the problem being solved numerically when the solution exists

Remark 4.10 Campbell [13] discusses the use of linear state and non-state feedback to eliminate impulsive behaviour by choosing a control $\mathbf{u} = K\,\mathbf{z} + \mathbf{v}$ for (4 20), so that the resulting system is index-1. The solution of this new system is then unique in the ordinary sense

4.6.2 Initial values for the general problem

Pantelides [54], in a recent paper discussed the consistent initialization of the general problem

$$\mathbf{F}(t, \mathbf{u}, \mathbf{u}', \mathbf{v}) = 0 \tag{4.24}$$

where the state variables are labelled by $\mathbf{u}$ and non-state variables by $\mathbf{v}$ A set of ic's $(\mathbf{u}_0, \mathbf{u}'_0, \mathbf{v}_0)$ are consistent for (4.24), if they satisfy the system at the initial point, that is, if

$$\mathbf{F}(t, \mathbf{u}_0, \mathbf{u}'_0, \mathbf{v}_0) = 0$$

This condition is necessary but not always sufficient Differentiating some or even all of the original constraints produces new equations which must also be satisfied by the ic's However this need not constrain the initial vector further The index-1 case is an obvious example of this

Pantelides [54] proposes a graph theoretical algorithm for analyzing the structure of (4 24), to determine the minimal subset of equations whose differentiation may yield useful information, in the sense that they impose further constraints on the vector of ic's The algorithm generates a bi-partite matching between the equations of the system (both original equations and those derived by differentiation) and variables of (4 24). This assignment uniquely determines the set of consistent i c 's if they exist We give an example of the application of this technique but the full algorithm can be found in Pantelides [54]. We assume the reader is familiar with the basic notions of graph theory such as nodes, edges, colourings, matchings and augmenting paths

Example 4 6 Consider the following DAE system

$$\begin{aligned} x' &= -y \\ y' &= z \\ 0 &= x - y - 1 \end{aligned} \tag{4 25}$$

which is index-2 An exact solution which satisfies these equations is

$$x(t) = 1 + e^{-t}, \quad y(t) = e^{-t} \quad z(t) = -e^{-t}$$

We intend to generate an extended system by differentiating the constraints, whose solution is identical to the solution of (4 25) Pantelides graph colouring algorithm proceeds as follows for this problem, we denote the equation nodes in our graph by f_i and the variables by v_i .

1 Construct a graph relating the equations to variables in the problem Only include variables whose derivatives do not appear, the following graph fig 3 1(a) results

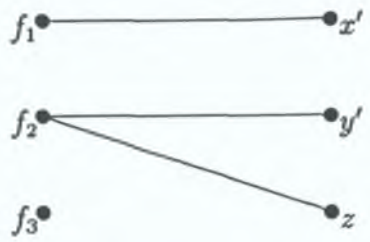

fig 3.1(a).

2. Apply a depth first search procedure to the above graph to see if it has an augmenting path. For the above graph no augmenting path can be found.
3. We differentiate f_3, giving

$$f_4 = x' - y' = 0$$

and introduce it into our graph as a new node. The resulting graph is given in fig 3.1(b).

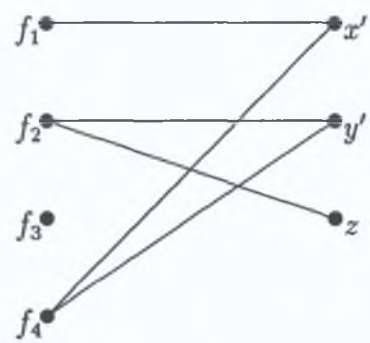

fig 3.1(b).

4. Applying a depth first search to this graph yields an augmenting path with maximal matching

$$(f_1 \cdots x' \cdots f_4 \cdots y' \cdots f_2 \cdots z).$$

Node f_3 is left exposed giving four equations in the five variables (x, y, x', y', z). Thus one initial variable can be chosen arbitrarily, subject to the constraint that the four equations are non singular w.r.t. to the remaining four initial values.

Remark 4.11: This method is completely general. Once again however it is necessary to explicitly differentiate the equations. As we have previously pointed out, this is unreasonable in practice.

4.7 Finding the index of DAE systems

Previous sections of this Chapter have outlined the importance of the concept of index for a DAE. In the next Chapter we shall see how the index also determines the numerical behaviour of an ODE method for solving DAEs. We have also identified one

method for finding the index for the general problem by differentiating the constraint equations This method is of little use in the numerical context, as it is virtually an impossible task to generate a set of ODEs for a large system of constraints Because the index is important numerically, a number of contributions have appeared in the literature which attempt to find or estimate it for a specific type of DAE It is our intention in this section to discuss the devices that have been proposed for this purpose and to outline some of the difficulties associated with finding the index using these devices

We return again to the linear constant coefficient problem (4 13) The index of this problem is the index of the matrix pencil $(A + \lambda B)$ and can be found via a *Generalized Singular Value Decomposition* (GSVD) or more precisely *Cosine-Sine* decompositions of partitioned orthonormal matrices, see Moler & Stewart [50] Since the singular values are the square roots of the eigenvalues, the number of zero singular values determines the dimension of the nullspace of a general matrix Generating a *Singular Value Decomposition* (SVD) involves the determination of the rank of a matrix The dimension of the nullspace of the associated matrix pencil is therefore the index of the system

More recently Kangstrom [44] has further improved the GSVD to include the computation of the Kronecker structure of (4 13) His algorithm, the Re-iterating GSVD or (RGSVD), generates the KCF of (4 13) Thus the understanding of (4 13) is complete as the index can be found using a finite algorithm However GSVD algorithm is $O(N^3)$ where N is the dimension of (4 13) This would seem a reasonable amount of computation, but for large problems, such as a system of coupled PDEs, it is not computable within a reasonable amount of time In fact, this is equivalent to or greater than the amount of computation that is associated with numerically integrating the system over a reasonably finite time horizon Thus, in all but the smallest of problems, it is quite inefficient to use a GSVD to find the index The GSVD is completely inefficient for the nonlinear problem, as the index may change over the time interval It therefore does not seem a reasonable alternative in practice

Sincovec *et al* [68] proposed a method for computing the *index m* of (4 13) using a backtracking function Recall from the previous section that we stated that a k step BDF method converged to the solution of (4 13) in *(m-1)k + 1* steps. They proposed the following technique for finding m

Create two problem instances of (4 13) having two different sets of initial conditions Integrate both problem instances with a k step BDF method, using a fixed stepsize, until both solutions agree to within round-off level Let the number of steps required be NSTEP We then have

$$m = (NSTEP - 1)/k + 1$$

The motivation behind this technique is that (4 13) admits distributional solutions which will be smoothed out by the integration method

While this technique is very appealing, our experience has shown us that the device is very unreliable We know of no integration routine which employs this technique for estimating the index of a problem and therefore discount it

Because the problem of rank (index) determination is ill-conditioned, other methods have been sought to find the index which avoid computing the rank of a system Duff & Gear [25] proposed a graph theoretical algorithm for finding the *structural*

index of a system Their motivation for doing this is that the index is often determined by the pattern of non zero entries in the Jacobians of a DAE system $\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}'}\right)$ and $\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}}\right)$ Since systems of index-2 or less can be solved by ODE methods, it is valuable to know if the index of a system is greater than two Duff & Gear [25] provide an algorithm for answering this question for systems of the form

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t,\mathbf{y}) + G\mathbf{z} \\ H\mathbf{y} &= A\mathbf{z}. \end{aligned}$$

If the dimension of the differential part is n and the algebraic part is m, then a necessary and sufficient condition for the index to be less than three is

$$rank \begin{bmatrix} A \\ NGH \end{bmatrix} = m$$

where N is an r by r matrix for which $NA = 0$. In other words, the rows of A span the left null space of A While we are not concerned with the details of the algorithm here, we provide an example to show how the notion of structural index is useful.

Example 4 7· Assume

$$HG = \begin{bmatrix} \alpha & \beta \end{bmatrix} \begin{bmatrix} 0 \\ \gamma \end{bmatrix} = \beta\gamma$$

and

$$A = [0]$$

Choose N such that $NA = 0$, a suitable N is [1]. We then have

$$rank \begin{bmatrix} A \\ NHG \end{bmatrix} = \begin{bmatrix} 0 \\ \beta\gamma \end{bmatrix} = 1$$

and the structural index of the problem is two. We point out that this is example 4 6 considered earlier, with the coefficients of the variables chosen arbitrarily

While it may be useful to obtain the structural index using this algorithm, Duff & Gear [25] mention that it may take exponential time on some problems

The importance of this Chapter lies in finding solutions of DAE systems We have reviewed what we feel are the main approaches to generating analytic solutions that have appeared in the literature Our whole understanding can be encapsulated in the concept of index or degree of complexity of a DAE, which is also vital for later numerical work The inadequacy of the tools we have outlined for finding the index leads us to rely completely on analytic techniques, in particular differentiation, which is unreasonable in practice, unless the system possess some simple structure Since the index is vital both for numerical work and for an analytic understanding of these problems, it is the central non-numeric concept which will pervade the remainder of this work We however, have not included methods for its computation in DAE solvers which will be developed later, as it does not fall directly within the objectives of this work It does however remain an outstanding research question, which must be satisfactorily addressed in order for DAEs to be efficiently solved by current techniques.

Chapter 5

Numerical Aspects of Solving DAEs.

5.1 Introduction

Having considered DAEs from an analytic point of view, we return in this Chapter to pertinent questions regarding their numerical solution Perhaps the most intriguing aspect of DAEs is that numerical ODE methods can be successfully used to solve these problems, which are very different to ODEs. In the early sections of this Chapter, we will discuss why it is possible to solve some types of DAE with numerical methods [1] and not others Once again the index or degree of complexity of a DAE determines this In solving ODEs, the error and stability of the numerical scheme give us a complete picture of the behaviour of the scheme on a specific problem We intend to see how effective these concepts are when we solve DAEs using ODE methods by looking at specific examples

The other vital question involved in applying implicit numerical methods to the solution of DAEs, is the stability of the iteration scheme In fact we shall see that an effective iteration scheme is the primary key to obtaining solutions. While we propose standard methods based on *Newton* iteration, a technique is outlined in the final Chapter based on a second order *tensor* approximation which we feel could be of considerable value in this context

The type of problem we consider in this Chapter for numerical analysis has the form [2]

$$E\,\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \qquad t \in [a, b] \tag{5 1}$$

subject to given initial conditions $\mathbf{y}(a) = \mathbf{y}_a$ We assume that $\mathbf{f}(\,)$ and $\mathbf{y}(t)$ are *n-dimensional vector mappings* with $\mathbf{f}(\,)$ having continuous first partial derivatives The index of (5 1) is determined only by the index of nilpotency of E, which we denote by $m \geq 1$ This equation is our starting point for analysis of local error In the next section, we follow Petzold [55] and consider the application of the Backward Euler (BE) method introduced in Chapter 2, to the solution of this problem

[1] We use the term numerical method to mean numerical ODE method

[2] Our numerical integration routines have been coded to handle problems of the form $E(t, \mathbf{y}(y))\,\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$ These problems are difficult to analyze theoretically We therefore have restricted our analysis to the case where the L H S is linear

5.2 Errors in Solving DAEs numerically

Suppose we start with exact solution values for our numerical scheme at time t_n. What then is the error after one step in solving (5 1) with the BE method ?
Taking one step of size h, we obtain the numerical solution by replacing the derivative $\mathbf{y}'(t_{n+1})$ by the linear combination $\frac{\mathbf{y}_{n+1}-\mathbf{y}_n}{h}$, giving

$$E\,(\mathbf{y}_{n+1} - \mathbf{y}_n) = h\,\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) \tag{5 2}$$

The exact solution at time t_n, expanded about t_{n+1} is

$$\mathbf{y}(t_n) = \mathbf{y}(t_{n+1}) - h\,\mathbf{y}'(t_{n+1}) + \frac{h^2}{2}\mathbf{y}''(\xi)$$

where $t_n \leq \xi \leq t_{n+1}$ Thus

$$\mathbf{y}'(t_{n+1}) = \frac{1}{h}\left\{\mathbf{y}(t_{n+1}) - \mathbf{y}(t_n) + \frac{h^2}{2}\mathbf{y}''(\xi)\right\}.$$

Substituting this expression in (5 1) we get

$$E\left\{\mathbf{y}(t_{n+1}) - \mathbf{y}(t_n) + \frac{h^2}{2}\mathbf{y}''(\xi)\right\} = h\,\mathbf{f}(t_{n+1}, \mathbf{y}(t_{n+1}) \tag{5 3}$$

Let us denote the error at the point t_n by $\mathbf{e}_n$, therefore

$$\mathbf{y}(t_{n+1}) = \mathbf{y}_{n+1} + \mathbf{e}_{n+1},$$

giving

$$\mathbf{f}(t_{n+1}, \mathbf{y}(t_{n+1})) = \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1} + \mathbf{e}_{n+1})$$

Expanding the RHS of this equation about $\mathbf{y}_{n+1}$, we get

$$\mathbf{f}(t_{n+1}, \mathbf{y}(t_{n+1})) = \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}_{n+1}}\right)\mathbf{e}_{n+1} + h\;o\;t$$

Denoting the matrix of partial derivatives $(\partial\,\mathbf{f}/\partial\,\mathbf{y}_{n+1})$ by A,[3] substituting the above expression into (5 3) and subtracting the result from (5 2), we get the following error equation

$$(E - h\,A)\,\mathbf{e}_{n+1} = E\,\mathbf{e}_n - h^2/2\;E\,\mathbf{y}''(\xi) + h\;o\;t \tag{5 4}$$

To gain further insight into (5 4) we assume $\mathbf{f}(\,)$ is locally linear In this situation (5.4) contains no higher order terms and we can generate the following closed form expression for $\mathbf{e}_{n+1}$

$$\mathbf{e}_{n+1} = (E - h\,A)^{-1}\,E\,\mathbf{e}_n - \frac{h^2}{2}\,(E - h\,A)^{-1}\,E\,\mathbf{y}''(\xi) \tag{5 5}$$

[3]In earlier chapters we used J to denote the Jacobian matrix of partial derivatives We have decided to use A in this chapter to keep our theory consistent with previously published literature on DAEs

Thus if we start our integration scheme off with exact initial values, the error after one step is

$$\mathbf{e}_1 = -\frac{h^2}{2}\,(E - h\,A)^{-1}\,E\,\mathbf{y}''(\xi) \tag{5 6}$$

Petzold [55] points out that there are several consequences of (5 6) and we briefly review them here to demonstrate the difficulties which may arise in solving some problems It is these difficulties which set limitations on the use of ODE schemes for solving DAEs

Consider the $m = 2$ linear constant coefficient problem

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{y}' + \mathbf{y} = \begin{pmatrix} 0 \\ g(t) \end{pmatrix}$$

we have

$$\mathbf{y}''(\xi) = \begin{pmatrix} -g'''(\xi) \\ g''(\xi) \end{pmatrix}$$

and

$$(E - h\,A)^{-1}\,E = \begin{bmatrix} 0 & -1/h \\ 0 & 0 \end{bmatrix}$$

so that the error after one step is

$$\mathbf{e}_1 = \begin{pmatrix} \frac{h}{2}\,g''(\xi) \\ 0 \end{pmatrix}$$

The first thing to note about this result, is that the algebraic equation is solved exactly We expect this as an *Implicit* method will be exact for an algebraic equation if it employs a *Newton iterative* scheme. The important point here however, is that the error in state variables is $O(h)$ and not $O(h^2)$, as predicted for the BE method applied to ODEs Thus an error estimate of the usual form based on $(h^2/2)\,\mathbf{y}''(\xi)$ would be asymptotically a gross underestimate This situation can obviously wreak havoc with any step selection algorithm which assumes errors are $O(h^{k+1})$ for a method of $O(h^k)$ It is possible to reduce the error $\mathbf{y}_{n+1}$ by decreasing h, in this case, provided a suitable error estimate is available to accomplish this task.

Unfortunately, there are several even more severe problems in solving systems of nilpotency $m \geq 3$. For the linear constant coefficient problem

$$y_2' = y_1 \quad y_3' = y_2 \quad y_3 = g(t)$$

we have

$$(E - h\,A)^{-1}\,E = \begin{bmatrix} 0 & -1/h & -1/h^2 \\ 0 & 0 & -1/h \\ 0 & 0 & 0 \end{bmatrix}$$

and the error after one step is

$$\mathbf{e}_{n+1} = \begin{pmatrix} g''(\xi)/2 + \frac{h}{2}\,g'''(\xi) \\ \frac{h}{2}\,g''(\xi) \\ 0 \end{pmatrix}$$

Once again the algebraic equation is solved exactly but the state variables cause difficulties The error in y_2 is $O(h)$, which can be controlled as in the $m = 2$ case

But the error in y_1 depends on terms independent of h Thus we cannot choose h small enough, so that the error in the solution after one step, starting with exact solution values is small

The results for these two problems appear to conflict with those of Sincovec *et al* [68] They show that when a k step constant stepsize BDF method is applied to the linear constant coefficient problem with $k < 7$, the solution is $O(h^k)$ accurate globally after a maximum of $(m-1)\,k+1$ steps, regardless of initial conditions For the $m = 3$ system just considered, a simple calculation yields the following solution for $y_{1,n+1}$

$$y_{1,n+2} = \frac{1}{h^2}\,(g_{n+2} - 2\,g_{n+1} + g_n), \tag{5 7}$$

which is a second order approximation for the exact solution

$$y_1(t_{n+2}) = g''(t_{n+2}).$$

Petzold [55] points to a qualification on the theorem given in Sincovec *et al* [68] That is, the convergence of the solution only applies to the end point of some fixed interval of integration This is because the first $m-1$ solution values, contain *impulsive* components which do not become arbitrarily small when h is decreased. The results for later steps depend only on the function $\mathbf{g}(t)$ at past steps and not on the starting values, so that the solution converges in any interval bounded away from the starting point

A second constraint on the result of Sincovec *et al,* is that it only applies to constant stepsizes Gear & Petzold [35] show that when the ratio of adjacent stepsizes is not bounded, the BE method fails to pick up the divided difference (5 7) correctly and the error in the $m = 3$ case has the form

$$\mathbf{e}_{n+1} = 1/2\ (1 - h_n/h_{n+1})\ \mathbf{g}''(\xi) + O(h_{n+1}) \tag{5 8}$$

With ODE codes the stepsize taken on the current accepted step is fixed and the stepsize required for the next step, is chosen to achieve the desired level of local accuracy In this model, the error in the $m = 2$ case does not go to zero as h is reduced, while in the $m = 3$ case, the error diverges as shown by (5 8), where the error behaves like $O(h_{n+1}^{-1})$ Gear & Petzold provide the following theorem which shows that even under the assumption of adjacent stepsizes remaining bounded, order reduction can occur for BDF methods

Theorem 5 1 If the k-step BDF method is applied to the linear constant coefficient DAE with k < 7 and the ratio of adjacent stepsizes is bounded, then the global error is $O(h^q)$ where $q = min(k, k-m+2)$.

We remark that for second order methods on index-1 and -2 DAEs, no order reduction occurs by this theorem

From the examples, it can be inferred (see Gear & Petzold [35]) that a problem of index no greater than $k+1$ can be solved by a *k-step* BDF method However the above discussion shows that this is not the case and variable step BDF formulae are not suitable for DAEs with arbitrary index In [35], it is also shown, that the asymptotic expansion of the global error in the BDF formulae make it possible for the linear constant coefficient problem to be solved by extrapolation methods applied to fixed step BDF methods.

Our discussion so far has only considered the BE and BDF methods In fact Marz [49] has studied the general linear multistep methods applied to index-1 DAE systems. She has shown that the coefficients of the LMM must satisfy an extra set of conditions, (which happen to be satisfied by the BDF formulae), for the method to be convergent with the expected order of accuracy. Hence, it is not entirely surprising that Implicit RK methods should suffer some order reduction on DAEs. Petzold [57] gives a set of necessary and sufficient conditions to ensure the the local truncation error of an RK method attains a given order for the index-1 problem It is fortunate that the DIRK(2,2) method of Chapter 2 attains the expected order of local accuracy $O(h^k)$, as these conditions are effectively the order conditions of Crouziex [23] coupled with L-stability. Recently Brenan & Petzold [5] have studied IRK methods applied to nonlinear semi-explicit index-2 system. By examining the accuracy and stability of a method they derive a set of necessary and sufficient conditions to ensure that a method is accurate to a given order on these systems

5.3 Error Estimates for DAEs.

In this section we shall examine several potential error estimates for DAE systems Our aim is to find an estimate which accurately reflects the behaviour of the error for index-1 and -2 DAEs Gear [31] and Gear & Brown [34] proposed solving systems of the form

$$\mathbf{f}(t, \mathbf{y}, \mathbf{y}') + P\,\mathbf{v} = 0 \qquad t \in [a, b] \tag{5.9}$$

where $\mathbf{y}$ and $\mathbf{y}'$ are vectors of length p_1, P is an $n \times (n - p_1)$ matrix and $\mathbf{f}$ is a vector function of length n In (5 9) the algebraic variables $\mathbf{v}$ appear linearly Both [32] and [34] make no attempt to estimate errors in $\mathbf{v}$ This make sense, since $\mathbf{v}$ on every step is completely determined by $\mathbf{y}$, thus errors in $\mathbf{v}$ do not cause errors in $\mathbf{y}$ Petzold [55] shows that the index-3 linear constant coefficient problem can be put in this form In this case error control is not attempted on y_1, which is the component with largest error after a stepsize decrease She also points out, that an ODE code may behave very differently if the algebraic variables do not appear linearly. For these reasons we are led to discount this technique in favour of estimates to be discussed in the remainder of this section

Sincovec *et al* [68] observed, for the linear constant coefficient problem, that the error in the non-state components has a different asymptotic behaviour to that of the state components In addition, errors in non-state components only affect the solution locally and are not propagated globally to the state components Let us denote the ordinary ODE error estimate by $\mathbf{e}_n$ and the DAE estimate by $\mathbf{e}_n^*$ The estimate proposed in [68] has the form

$$\mathbf{e}_n^* = M\,\mathbf{e}_n \tag{5 10}$$

M is called the *state variable projection matrix* and its purpose is to *filter out* the non-state values from the ODE error estimate M has the form

$$M = \lim_{h \to 0} M(h, \jmath) \tag{5 11}$$

with

$$M(h, \jmath) = ((E - h\,A)^{-1})\,E)^{\jmath}$$

and $j \geq m$, the nilpotency of the DAE system

Remark 5 1 This estimate is easily computed, as the *LU-decomposition* of $E - h\,A$ is already available from the iterative scheme for solving the nonlinear equation, (c *f section 5*). However implementing the estimate has a serious drawback, in that it requires a knowledge of the index of the DAE system which, as we have seen in Chapter 4, can be difficult to compute

Petzold [55] proposed an error estimate similar to (5 10). In her paper, she quoted results from Sachs-Davies [63], that error estimates for second derivative ODE methods (see Hall & Watt [38]) are asymptotically correct as $h \rightarrow 0$ and are reliable and efficient for very stiff ODEs The estimates have the form

$$\epsilon_n^{**} = W^{-1}\,\epsilon_n \tag{5.12}$$

where W is the iteration matrix for the second derivative method Petzold [55] suggests using the iteration matrix for the Implicit numerical scheme, in place of W, based on the fact, that the local contribution to the global error for the BE scheme on the linear constant coefficient DAE is

$$(E - h\,A)^{-1}\,E\,(h^2/2)\,\mathrm{y}''(\xi).$$

The error estimate (5 12) then becomes

$$\epsilon_n^{**} = (E - h\,A)^{-1}\,E\,\epsilon_n \tag{5 13}$$

This is precisely the estimate (5 10), (5 11) proposed by Sincovec for use on the index-1 problem. Petzold [55] also shows that (5 13) accurately reflects the behaviour of errors for all k-step BDF formulae, with $k \leq 6$ and $m \leq 2$ Based on this observation, Petzold suggests that, by using this error estimate, DAEs with $m \leq 2$ can be adequately handled by ODE integration methods with only slight modification This is also our primary reason for restricting the one step schemes of Chapter 3 to solving $m \leq 2$ DAEs In the next Chapter we incorporate this estimate into our schemes.

Recently Petzold & Lotstedt [59] have proposed a generalization of this estimate for the semi-explicit system

$$\begin{aligned} x' &= f(t,x,y) \\ 0 &= g(t,x,y) \end{aligned} \tag{5 14}$$

with iteration matrix

$$B = \begin{bmatrix} 1 - h\beta\frac{\partial f}{\partial x} & -h\beta\frac{\partial f}{\partial y} \\ -h\beta\frac{\partial g}{\partial x} & -h\beta\frac{\partial g}{\partial y} \end{bmatrix}$$

They observe that part of the error e in x *and* y, due to the truncation error $h\,\tau_n$ is

$$\mathbf{e} = -B^{-1}\,(\partial f/\partial x)\,h\,\tau_n$$

They propose the following estimate

$$\epsilon_n^{***} = -B^{-1}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\epsilon_n,$$

which is identical to the estimate (5 12) (with $W = B$)

5.4 Linear Stability for DAEs

In this section we apply classical linear stability to DAE systems, we consider the linear version of (5 1) that is

$$E\,\mathbf{y}' = D\,\mathbf{y} \qquad t \in [a,b] \tag{5 15}$$

where D is a matrix of eigenvalues $\lambda_i, 1 \leq i \leq n$. If we apply the BE to this system we obtain

$$(E - h\,D)\,\mathbf{y}_{n+1} = E\,\mathbf{y}_n$$

and we once again keep the ratio

$$\|R(D)\| = \|\mathbf{y}_{n+1}\|/\|\mathbf{y}_n\|$$

bounded by 1. For the BE method, we have

$$\frac{\|\mathbf{y}_{n+1}\|}{\|\mathbf{y}\|} = \|(E - h\,D)^{-1}\,E\| \tag{5 16}$$

as amplification factor measured in some norm We have adopted the l_2 norm which measures the spectral radius [4]. For a matrix A, the l_2 norm is defined as (see Butcher [6])

$$\|A\|_2 = \rho((A\,A^T)^{1/2})$$

where $\rho(\ )$ is the spectral radius. While any matrix norm would be suitable, the true amplification factor of a matrix is directly related to the size of the spectral radius and this is precisely the quantity we wish to measure. We intend to look at some simple examples to gain some further insight

Consider the index-1 problem

$$\begin{aligned} y_1' &= \lambda_1\,y_1 \\ 0 &= \lambda_2\,y_2 \end{aligned}$$

with

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

In order to consider stability, we introduce a parameter $\epsilon > 0$ and analyze the following system

$$\begin{aligned} y_1' &= \lambda_1\,y_1 \\ \epsilon\,y_2' &= \lambda_2\,y_2 \end{aligned}$$

with

$$E = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix}.$$

From (5 16), we compute

$$A = (E - h\,D)^{-1}\,E = \begin{bmatrix} \frac{1}{1-z_1} & 0 \\ 0 & \frac{\epsilon}{\epsilon - z_2} \end{bmatrix}$$

[4]Recall that the spectral radius is the magnitude of the largest eigenvalue in modulus, (see Krysig [45], page 350)

and denote $h\lambda_i$ by z_i Taking the limit as $\epsilon \to 0$, we get

$$A = \begin{bmatrix} \frac{1}{1-z_1} & 0 \\ 0 & 0 \end{bmatrix}$$

The l_2 norm of this matrix is easily computed as

$$\rho(a) = 1/(1 - z_1)$$

This is the ordinary stability function for the differential equation in y_1 and is bounded by $1, \forall z < 0$, with $Re(z_1) < 0$. In this case no difficulties arise, as the algebraic components do not affect the stability of the system Therefore we expect the same behaviour on this problem as we would for an ODE system.

The Index-2 case proves less amenable to the foregoing analysis Consider the system

$$\begin{aligned} y_2' &= y_1 \\ \epsilon y_1' &= y_2. \end{aligned}$$

In the limit as $\epsilon \to 0$, we have

$$E = \begin{bmatrix} 0 & 1 \\ 0 & \epsilon \end{bmatrix}$$

and this matrix has Index-2 However, the ODE system from which it is derived does not naturally decouple into its constituent components, as in the Index-1 example The resulting solutions have the form $a \sinh(t) + b \cosh(t)$ and stability classical analysis is useless We know from the previous section, that errors in Index-2 systems may not decrease as h is decreased As our stability analysis is inadequate even on the simplest Index-2 system, it is therefore impossible to guarantee that an integration method will perform satisfactorily on problems of Index-m> 1 However, if we are mindful of these limitations, limited success can be obtained on Index-2 systems, as our test results of the next Chapter demonstrate

The question of stability of DAEs has received little attention in the literature To our knowledge, this question has only been addressed in the contributions of Griepentrog & Marz [37], Gear & Petzold [35] and Petzold [57] In Gear & Petzold [35], the BE is examined on linear non-constant coefficient DAEs, where it is shown that error amplification is not damped out They therefore reject this methods for solving DAEs in general and suggest, for this reason, that higher order method should also be rejected Petzold [57] considered stability for RK methods on the index-1 problem We mentioned earlier, that she demonstrated that order reduction can occur for some RK methods on these problems She arrives at this conclusion, by accessing the stability and the contribution of local error to global error on each step Petzold points out that her results are similar to those of Frank, Schneid & Ueberhuber [29] for RK methods applied to ODEs

5.5 Implementation of Implicit Schemes for DAEs

In Chapter 2, we outlined how implicit numerical methods are applied to ODEs and subsequently solved by a modified Newton iterative scheme In this section we intend

to generalize this work to solve DAE systems. In contrast to Chapter 2, we propose two different approaches to framing our problem, so that a solution can be found to the resulting nonlinear equations.

The type of problem we have designed our methods to solve can be written in the form (5 1)[5] Applying the LMM (2 20) to this problem yields the following nonlinear system of equations, to be solved at time t_{n+k} for the unknown $\mathbf{y}_{n+k}$

$$E\left\{\mathbf{y}_{n+k}+\sum_{j=1}^{k-1}\alpha_j\mathbf{y}_{n+j}\right\}-h\beta_k\mathbf{f}_{n+k}-h\sum_{j=1}^{k-1}\beta_j\mathbf{f}_{n+k}=0 \tag{5 17}$$

Applying a modified Newton iteration to (5 17) gives the following system to be applied iteratively

$$B\Delta\mathbf{y}_{n+k}^{i+1}=h\beta_k\mathbf{f}_{n+k}^{i}-E\mathbf{y}_{n+k}^{i}+\mathbf{g} \tag{5 18}$$

with

$$\mathbf{y}_{n+k}^{i+1}=\Delta\mathbf{y}_{n+k}^{i+1}+\mathbf{y}_{n+k}^{i}$$

Once again, $\mathbf{g}$ is the vector of past information, but in this case has the form

$$\mathbf{g}=h\sum_{j=1}^{k-1}\beta_j\mathbf{f}_{n+j}-E\sum_{j=1}^{k-1}\alpha_j\mathbf{y}_{n+j},$$

while the iteration matrix is given by

$$B=E-h\beta_k\left(\frac{\partial\mathbf{f}}{\partial\mathbf{y}_{n+k-1}}\right)$$

We will return to the iteration matrix later. We first point out an important difference between (5 18) and (2 22) regarding the structure of $\mathbf{g}$, the vector of past information

In (5 18), we have the linear combination $E\sum_{j=1}^{k-1}\alpha_j\mathbf{y}_{n+j}$ instead of $\sum_{j=1}^{k-1}\alpha_j\mathbf{y}_{n+j}$, which arose in the ODE case This removes the state effect from the non-state variables in the iterative scheme, we illustrate the idea with an example

Example 5 1 Consider the application of θ-scheme to the solution of the following system

$$\begin{aligned} y_2' &= f(t,y_1,y_2)\\ 0 &= g(t,y_1,y_2) \end{aligned} \tag{5 19}$$

For this system using the θ-scheme, (5 17) becomes

$$E(\mathbf{y}_{n+1}-\mathbf{y}_n)=h\left[(1-\theta)\mathbf{f}_n+\theta\mathbf{f}_{n+1}^{i}\right]$$

and the iteration scheme is

$$B\,\Delta\mathbf{y}_{n+1}^{i+1}=h\left[(1-\theta)\mathbf{f}_n+\theta\mathbf{f}_{n+1}^{i}\right]-E(\mathbf{y}_{n+1}^{i}-\mathbf{y}_n)$$

For the system (5 19) the R H S of the iteration scheme is

$$\begin{pmatrix} h\left[(1-\theta)f_{1,n}+\theta f_{1,n+1}^{i}\right]-y_{2,n+1}^{i}+y_{2,n}\\ h\left[(1-\theta)f_{2,n}+\theta f_{2,n+1}^{i}\right] \end{pmatrix}$$

[5]In fact our codes can handle a linearly implicit R H S , as we previously remarked

Thus, for the second equation, we are only applying an ordinary Newton scheme to the nonlinear equation

$$0 = f_2(t, y_1, y_2).$$

In this way, we are able to mix the differential and algebraic equations in one iterative scheme, taking full advantage of the natural coupling that exist between the variables in the system. This technique was first proposed by Gear [32] for the solution of ODEs by BDF methods. This approach has been used in several DAE integration routines including our own schemes (*c.f. Chapter 6*) and those of [9], [22] and [18]

We call the formulation outlined above, the Direct formulation of the problem Another means of treating DAEs numerically by ODE methods is to use Residual formulation. Here we define a residual vector for (5.1) as

$$\mathbf{r}(t, \mathbf{y}, \mathbf{y}') = E\mathbf{y}' - \mathbf{f}(t, \mathbf{y}) = 0 \tag{5 20}$$

and approximate the derivative by a linear combination of back values. Using the BE on (5.13), we get the following nonlinear system of equations to be solved at each time step, for $\mathbf{y}_{n+1}$

$$\mathbf{r}\left(t_{n+1}, \mathbf{y}_{n+1}, (\mathbf{y}_{n+1} - \mathbf{y}_n)/h\right) = 0.$$

In this case the iterative scheme is

$$B\,\Delta\mathbf{y}_{n+1}^{i+1} = -\mathbf{r}\left(t_{n+1}, \mathbf{y}_{n+1}^{i}, (\mathbf{y}_{n+1}^{i} - \mathbf{y}_n)/h\right)$$

with

$$B = E - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}_n}$$

The important point about this formulation of the problem, is that it is not necessary explicitly generate *E and J, the Jacobian of* $\mathbf{f}$, required by the iteration matrix B and add them together It is the iteration matrix itself that is computed by finite differencing, thereby halving the work involved Most production codes designed to solve DAEs, such as DASSL [56] , LSODI [43] and SPRINT [3], use this formulation They require the user to supply two routines, one to compute E, that is, only those terms involving $\mathbf{y}'$ and a second routine for the full residual

5.6 The Iteration Matrix and Scaling

The iteration matrix that arises in solving ODEs, as we have seen in Chapter 2, has the form

$$B_{ODE} = I - h\beta J$$

where β is a parameter that depends on the method and J is the Jacobian of $\mathbf{f}(\ )$ in (2 1) When a numerical ODE method uses this iteration matrix to solve the nonlinear equations (2 22), it is usual to decrease h if the resulting iteration fails to converge reasonably quickly, or the error on the current step is outside the tolerance Thus as $h \to 0$, the condition number[6] $\mathcal{K}(B) \to 1$, since $B_{ODE} \to I$ Therefore the resulting *LU-decomposition* of B_{ODE} becomes more stable and we expect that

[6]The condition number is the ratio in absolute value of the largest to the smallest eigenvalues of a matrix, (see Krysig, [45])

$\mathbf{y}_{n+k-1} = \mathbf{y}^0_{n+k}$ to become a better approximation to $\mathbf{y}_{n+k}$ With ODEs we are fortunate, since as $h \to 0$, $B_{ODE} \to I$. However, for DAEs, this does not occur In the DAE, case the iteration matrix is

$$B_{DAE} = E - h\beta J$$

If we apply the usual ODE arguments to B_{DAE}, we run into serious problems It is the structure on E, that causes these problems in solving the nonlinear system In particular we are concerned with what happens, when our error estimate fails to lie within the tolerance or the Newton iteration fails In this case, reducing h causes

$$E - h\beta J \to E$$

which is singular. Thus we may be faced with the *LU-decomposition* of a singular system, causing the code to fail completely, without giving any indication of the cause of failure. This can easily occur. For example, Petzold [55] has shown that steep gradients in a solution can cause error estimates to be unbounded as $h \to 0$ Thus the initial guess may not improve as $h \to 0$, causing the corrector iteration to diverge A code faced with this situation has no way of deciding whether the problem is due to error estimates, or to poor conditioning of the iteration matrix One thing is clear, that is, if a code fails to converge on a Newton step or fails the error test while attempting a time step, we should be careful about how we choose h to ensure that the next integration step will be accepted Petzold [55] tries to overcome this difficulty by using a more robust iteration scheme such as *Damped Newton* In her code, DASSL [56], she simply multiplies the correction at each step of the Newton iteration by 0 75 instead of 1

Recently Petzold & Lotstedt [59] have suggested scaling the iteration matrix, so that the iterative scheme is more stable Consider the application of a q-stage RK method to the solution of (5 1) Recall with an RK method, we replace $y'(t_n + c_i\, h)$ by unknowns k_i $(1 \leq i \leq q)$ and the solution $y(t_n + c_i\, h)$, is obtained by writing it as a linear combination of $y(t_n)$ and the derivatives k_i The coefficients of the method are then chosen so that the scheme has the desired level of local accuracy For example, the 1^{st}-order RK method (the Backward Euler method) gives

$$E\,\mathrm{k} = h\,\mathbf{f}(t_{n+1}, \mathbf{y}_n + \mathrm{k})$$

Linearizing this equation gives (*c f* Rosenbrock technique Chapter 2)

$$(E - h\,J)\,\mathrm{k} = h\,\mathbf{f}(t_{n+1}, \mathbf{y}_n)$$

Let us see how scaling can be applied to this system If (5 1) is index-1, then E has block form

$$\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$$

where I, is an $n - r \times n - r$ identity matrix Thus

$$(E - h\,J) = \begin{bmatrix} I - h\,J_{11} & -h\,J_{12} \\ -h\,J_{21} & -h\,J_{22} \end{bmatrix}$$

where J_{ij}, is the Jacobian of the block of equations $\mathbf{f}_i$ *w.r.t.* variables $\mathbf{y}_j$ This system is easily scaled, we simply multiply the bottom r rows by $1/h$. Since we are

not scaling variables, but only equations, the effect of this scaling should improve overall accuracy in solving the nonlinear system. However our experience on index-1 DAEs has shown us that this scaling is unnecessary on the test problems which we consider in the next Chapter. In fact we have had no apparent problems with conditioning of the index-1 problems which were tackled by our one-step codes.

If our problem is index-2, then E will have a sub-block of the form

$$\begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}$$

and

$$(E - h\,J) = \begin{bmatrix} -h\,J_{11} & I - h\,J_{12} \\ -h\,J_{21} & -h\,J_{22} \end{bmatrix}$$

Once again, it would seem appropriate to scale the algebraic system by $1/h$, for the index-2 case In [59], it is shown that round off errors proportional to $1/h$ and $1/h^2$ are introduced into the state and non-state variables respectively, without scaling With this scaling, the errors are multiplied by a factor of h Thus in the index-2 case, we still should be careful how we choose h, so that these errors will not dominate the solution

Perhaps the most important feature of the proposed scaling is to control the size of round off error in the state variables, while, at the same time, the algebraic variables may contain errors proportional to $1/h$. These may be tolerable, since the error in the non-state components may not be propagated throughout the solution interval. In this case, we must be careful to accurately solve the algebraic equations at the final time Also, we must not include algebraic variables in error tests, as large errors in non-state components may cause unnecessary failure of the integration scheme The estimate proposed earlier automatically takes care of this restriction In [59], it is mentioned that Painter [53] used this scaling in solving the *Navier-Stokes* equations which, on discretization, are index-2. Painter found the scaling to be valuable when the code was starting with a small stepsize, or when it was integrating over a discontinuity in a derivative

5.7 Initial conditions for Numerical Schemes

In this section we propose a strategy for the initialization of DAE systems Recall from Chapter 4, that we outlined the available analytic approaches to guarantee a consistent set of initial conditions for DAEs We pointed out, that consistency in this context meant that initial values for the variables and the derivatives should satisfy the equations at the starting point This was only a necessary condition We also required the derivatives to satisfy the derived ODE system for sufficiency and mentioned the algorithm of Pantelides [54] as a tool for generating a set of consistent initial conditions However the real difficulty with the techniques which we proposed were their analytic nature, which ruled out using them in a numerical context

Campbell [15] and Newcomb [51] considered a system of the form

$$A\,\mathbf{x}' + B(\mathbf{x}) = \mathbf{g}(t) \tag{5.21}$$

which is virtually identical to the type of system (5 1), given the assumption that $B(\mathbf{x})$ is sufficiently differentiable. Following on their work, we consider the following limit

$$\lim_{\delta \to 0} \frac{1}{\delta} \{A(\mathbf{x} - \mathbf{x}_0) + \delta\, B(\mathbf{x}) - \delta\, \mathbf{g}\} = 0.$$

Expanding $B(\mathbf{x})$ about $\mathbf{x}_0$, we have

$$\lim_{\delta \to 0} \frac{1}{\delta} \{A(\mathbf{x} - \mathbf{x}_0) + \delta\, [B(\mathbf{x}_0) + B_1(\mathbf{x} - \mathbf{x}_0) + B_2 + \cdots - \mathbf{g}]\}$$

where B_1 is the Jacobian of B at $\mathbf{x}_0$ and B_j, $j \geq 2$, is *j-linear* in $(\mathbf{x} - \mathbf{x}_0)$ Thus B_2 is a quadratic form in $(\mathbf{x} - \mathbf{x}_0)$ We can write this limit as

$$\lim_{\delta \to 0} (A + \delta B_1)/\delta \left\{(\mathbf{x} - \mathbf{x}_0) + (A + \delta B_1)^{-1}\, \delta\, [B(\mathbf{x}_0) + B_2 + \cdot\ \cdot - \mathbf{g}]\right\} = 0$$

or

$$\lim_{\delta \to 0} (A + \delta B_1)^{-1}\, \delta\, [B(\mathbf{x}_0) + B_2 + \quad - \mathbf{g}] = 0$$

since, if the system is solvable, the KCF is invertible and continuity ensures that $\mathbf{x} = \mathbf{x}_0$, as $\delta \to 0$ If we neglect higher order terms in $\mathbf{x} - \mathbf{x}_0$, this limit suggests the following possible iterative scheme

$$\mathbf{x}_1 = \mathbf{x}_0 - (A + \delta B_1(\mathbf{x}_0))^{-1} \delta\, [B(\mathbf{x}_0) - \mathbf{f}_1]$$

which is, in, essence one step with the BE method We use precisely this technique with our DIRK(2,2) scheme in the next Chapter Recall, this scheme has a BE first stage which we solve using the Rosenbrock technique In the case of the Composite Integration Scheme we additionally provide a simple *Damped Newton* iteration We point out, that using our schemes in this way automatically generates the initial values for the derivatives. Thus, with the Direct formulation outlined, it is not necessary to explicitly provide initial values for derivatives, as is the case in the Residual approach This is our primary reason for adopting the Direct formulation

In closing this Chapter then, we remind the reader that, in solving DAEs numerically, errors behave differently to the ODE case In particular the higher index problems are virtually impossible to solve, even with constant stepsizes. While the stability of the numerical scheme should guarantee linear error growth, we may not have this for simple linear problems It is the conditioning of the iteration matrix, as $h \to 0$ however, that is the real deficiency of numerical schemes in this context Unless this problem can be overcome, ODE methods will always remain experimental for general DAE systems It is our opinion that this question can be satisfactorily addressed, by the *tensor* method which we outline in the final Chapter All things considered, it is quite remarkable that ODE software is so successful in solving DAEs. In the next Chapter we solve several DAE systems by the one-step and multistep methods which were introduced in Chapter 3 We intend, in so far as is possible, to demonstrate the versatility of these methods for handling such complex systems

Chapter 6

Numerical schemes for solving DAEs.

6.1 Introduction

In Chapter 3, we developed and provided test results for one step numerical ODE methods. We demonstrated that, at low tolerances, these simple schemes provided efficient alternatives to BDF methods espcially in terms of Jacobian evaluations. This Chapter parallels the work of Chapter 3. We extend our one-step methods to the solution of DAE systems of the form

$$E\,\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)) \qquad t \in [a, b] \tag{6.1}$$

with $\mathbf{y}(a) = \mathbf{y}(t_a)$. We will evaluate the performance of the one-step methods against two special purpose software packages designed for numerically solving DAE systems: the LSODI package of Hindmarsh [43] and the DASSL integrator developed by L. Petzold [56]. Both of these packages are based on BDF formulae and we will consider them later in this Chapter.

In this Chapter, it is our intention to incorporate into the one-step methods, some of the improvements suggested in the last Chapter. Recall that the difficulties which arise are due to poor error estimation, preventing the iteration matrix from becoming singular and providing a robust iteration scheme. We will compare the performance of the one-step codes, with both LSODI and DASSL on a selection of test problems. We mention here that the test set of Enright *et. al.* (DETEST) [27], was choosen in solving ODEs. However no such test set is available for DAEs. We solve a selection of problems that have appeared in the literature, along with some of the ODEs solved in Chapter 3, recast as DAEs. We therefore have a benchmark for measuring performance. That is, a method should solve an ODE re-cast as a DAE, without any loss of overall efficiency or accuracy.

6.2 DIRK(2,2) scheme for DAEs.

The general technique of applying Runge-Kutta methods to (6.1), is to approximate the unknown $\mathbf{y}_{n+1}$ by a linear combination of $\mathbf{y}_n$ and its derivatives, at intermediate stages in the interval t_n to t_{n+1}. The DIRK(2,2) scheme applied to (6.1) gives the

following equations

$$\begin{aligned} E\,\mathbf{k}_1 &= h\,\mathbf{f}(t_n + \alpha h, \mathbf{y}_n + \alpha \mathbf{k}_1) \\ E\,\mathbf{k}_2 &= h\,\mathbf{f}(t_n + h, \mathbf{y}_n + (1-\alpha)\mathbf{k}_1 + \alpha \mathbf{k}_2) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + (1-\alpha)\mathbf{k}_1 + \alpha \mathbf{k}_2 \end{aligned} \tag{6 2}$$

Once again, we are concerned with the structure of the error and stability of these schemes. We mentioned in Chapter 5 that Petzold [57] has considered order results for the general IRK formula on index-1 DAEs We will not review her results here, but analyze the solution of the linear constant coefficient problem

$$E\,\mathbf{y}' = A\,\mathbf{y} \tag{6 3}$$

using the DIRK(2,2) scheme (6 2).

Applying the DIRK(2,2) scheme (6 2) to the linear problem (6 3) and denoting the matrix $E - \alpha h A$ by B, we have

$$\mathbf{k}_1 = hB^{-1}A\mathbf{y}_n$$

and

$$\mathbf{k}_2 = hB^{-1}A\mathbf{y}_n + h^2(1-\alpha)B^{-1}AB^{-1}A\mathbf{y}_n$$

giving

$$\mathbf{y}_{n+1} = \left[I + hB^{-1}A + h^2\alpha(1-\alpha)B^{-1}AB^{-1}A\right]\mathbf{y}_n \tag{6 4}$$

Subtracting the exact solution

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h\mathbf{y}'(t_n) + \frac{h^2}{2}\mathbf{y}''(t_n) + \frac{h^3}{6}\mathbf{y}^{(3)}(\xi)$$

from (6 4), letting $\mathbf{e}_n = \mathbf{y}_n - \mathbf{y}(t_n)$ and multiplying through by B, we get

$$\begin{aligned} B\mathbf{e}_{n+1} &= B\mathbf{e}_n + hA\mathbf{y}_n + h^2\alpha(1-\alpha)AB^{-1}A\mathbf{y}_n \\ &\quad -hB\mathbf{y}'(t_n) - \frac{h^2}{2}B\mathbf{y}''(t_n) - \frac{h^3}{6}B\mathbf{y}^{(3)}(\xi). \end{aligned}$$

Splitting up the matrix B into E and $-\alpha h A$ we can write the above equation as

$$\begin{aligned} B\mathbf{e}_{n+1} &= B\mathbf{e}_n + hA\mathbf{y}_n + h^2\alpha(1-\alpha)AB^{-1}A\mathbf{y}_n \\ &\quad -hE\mathbf{y}'(t_n) + \alpha h^2 A\mathbf{y}'(t_n) - \frac{h^2}{2}E\mathbf{y}''(t_n) \\ &\quad +\alpha\frac{h^3}{2}A\mathbf{y}''(t_n) - h^3 6B\mathbf{y}^{(3)}(\xi) \end{aligned}$$

Since

$$E\mathbf{y}' = A\mathbf{y} \;\Rightarrow\; E\mathbf{y}'' = A\mathbf{y}' \;\Rightarrow\; E\mathbf{y}^{(3)} = A\mathbf{y}''$$

we have

$$\begin{aligned} B\mathbf{e}_{n+1} &= (B + hA)\mathbf{e}_n \\ &\quad +\frac{h^2}{2}\left\{2\alpha(1-\alpha)AB^{-1}E\mathbf{y}'_n - (1-2\alpha)A\mathbf{y}'(t_n)\right\} + O(h^3) \end{aligned}$$

Assuming that $\mathbf{e}_n = 0$ we have

$$B\mathbf{e}_{n+1} = \frac{h^2}{2}\left\{2\alpha(1-\alpha)A(B^{-1}E)\mathbf{y}'_n - (1-2\alpha)A\mathbf{y}'(t_n)\right\} + O(h^3)$$

then setting $B^{-1}E = I$ in the r h.s of this expression, our error has the correct form *iff*

$$2\alpha(1-\alpha) = 1-2\alpha$$
$$\Rightarrow \alpha = 1 \pm 1/\sqrt{2}.$$

Thus the error estimate suggested in (5 13), is reasonable in this case and we propose using the following estimate for DAEs

$$\mathbf{e}_{DAE} = B^{-1}E\mathbf{e}_{ODE}. \tag{6 5}$$

Returning to (6 4) and assuming that $A = D$, a diagonal matrix of eigenvalues, we require for stability that

$$\|R\| = \|I - hB^{-1}D + h^2\alpha(1-\alpha)B^{-1}DB^{-1}D\| < 1.$$

Once again in the index-1 case the stability is determined by the differential variables, since if

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \text{ and } D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

then we require

$$|R| = 1 - h\frac{1}{1-\alpha z_1} + h^2\alpha(1-\alpha)\frac{1}{(1-\alpha z_1)^2} < 1$$

which is the A-stability polynomial for the ODE case

Finally for the DIRK(2,2) integrator, we extend the Rosenbrook implementation given in Chapter 2, to systems of the form (6 1) Again we linearize (6.2) about $\mathbf{y}_n$ as follows·

$$E\,\mathbf{k}_1 = h\mathbf{f}(t_n + \alpha h, \mathbf{y}_n + \alpha\mathbf{k}_1)$$

giving

$$E\,\mathbf{k}_1 = h\,\mathbf{f}(t_n + \alpha h, \mathbf{y}_n) + \alpha h J\mathbf{k}_1$$

where J is the Jacobian of $\mathbf{f}(t_n, \mathbf{y}_n)$ The second stage is handled in the same way, expanding about $\mathbf{y}_n + (1-\alpha)\mathbf{k}_1$, to give

$$E\,\mathbf{k}_2 = h\mathbf{f}(t_n + h, \mathbf{y}_n + (1-\alpha)\mathbf{k}_1) + \alpha h J\mathbf{k}_2$$

and we use the same Jacobian of $\mathbf{f}(\,)$, for both stages We then compute

$$\mathbf{y}_{n+1} = \mathbf{y}_n + (1-\alpha)\mathbf{k}_1 + \alpha\mathbf{k}_2$$

as before The algorithm for the DAE case, is therefore identical to the ODE algorithm except for the following

1 The error estimate (6.5) replaces the ODE error estimate

2 We place a lowerbound on the stepsize to enhance the stability of the integration scheme and prevent the iteration from becoming singular We propose the following

$$h = \max\left(h_{min}, \frac{1}{|f_{i,max}|}\right)$$

where $f_{i,max}$, represents the scale of the problem at any time and h_{min}, is a lowerbound in the stepsize, supplied by the user

3 We use the implementation outlined above for solving DAE systems

6.3 The Composite Integration Scheme for DAEs.

We return to the Composite Integration scheme introduced in Chapter 3 in this section and apply it to the DAE system

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t) \tag{6 6}$$

Recall from Chapter 5 the application of the θ-scheme to a DAE system For the system (6.6) the θ-stage of the integration using the Composite scheme is

$$E\mathbf{y}_{n+\gamma} = E\mathbf{y}_n + \gamma h[(1-\theta)(A\mathbf{y}_n + \mathbf{g}_n) + \theta(A\mathbf{y}_{n+\gamma} + \mathbf{g}_{n+\gamma})]$$

Denoting the iteration matrix $E - \gamma\theta hA$ by B we have

$$\mathbf{y}_{n+\gamma} = B^{-1}E\mathbf{y}_n + \gamma h(1-\theta)B^{-1}A\mathbf{y}_n + \gamma h(1-\theta)B^{-1}\mathbf{g}_n + \gamma\theta hB^{-1}\mathbf{g}_{n+\gamma}$$

With BDF methods we approximate the derivative by a linear combination of past solution values, thus for the BDF stage of the Composite scheme we have

$$E\{\alpha_0\mathbf{y}_n + \alpha_1\mathbf{y}_{n+\gamma} + \alpha_2\mathbf{y}_{n+1}\} = hA\mathbf{y}_{n+1}$$

Substituting the expression derived above for $\mathbf{y}_{n+\gamma}$ into the above equation we obtain

$$\begin{aligned} B\mathbf{y}_{n+1} &= -\frac{\alpha_0}{\alpha_2}E\mathbf{y}_n \\ &\quad -\frac{\alpha_1}{\alpha_2}\{EB^{-1}E\mathbf{y}_n + \gamma h(1-\theta)EB^{-1}(A\mathbf{y}_n + \mathbf{g}_n) + \gamma\theta hEB^{-1}\mathbf{g}_{n+\gamma}\} \\ &\quad +\frac{h}{\alpha_2}\mathbf{g}_{n+1} \end{aligned}$$

which gives using (6 6)

$$\begin{aligned} B\mathbf{y}_{n+1} &= -\frac{\alpha_0}{\alpha_2}E\mathbf{y}_n \\ &\quad -\frac{\alpha_1}{\alpha_2}\{EB^{-1}E\mathbf{y}_n + \gamma h(1-\theta)EB^{-1}E\mathbf{y}'_n + \gamma\theta hEB^{-1}\mathbf{g}_{n+\gamma}\} \\ &\quad +\frac{h}{\alpha_2}\mathbf{g}_{n+1}. \end{aligned} \tag{6 7}$$

It is not our intention to give a complete analysis of the error in this case, instead we look at the index-2 problem

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\mathbf{y}' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\mathbf{y} + \begin{pmatrix} 0 \\ -e^t \end{pmatrix}$$

with $y_1(0) = y_2(0) = 1$ and exact solution $y_1(t) = y_2(t) = e^t$ For this problem we have $E^2 = 0$ so that (6 7) becomes

$$B\mathbf{y}_{n+1} = -\frac{\alpha_0}{\alpha_2}E\mathbf{y}_n - \frac{\alpha_1}{\alpha_2}\gamma\theta hEB^{-1}\mathbf{g}_{n+\gamma} + \frac{h}{\alpha_2}\mathbf{g}_{n+1}$$

A simple calculation then yields that

$$B\mathbf{y}_{n+1} = -\frac{\alpha_0}{\alpha_2}\begin{pmatrix} y_{2,n} \\ 0 \end{pmatrix} - \frac{\alpha_1}{\alpha_2}\begin{pmatrix} e^{t_n\gamma h} \\ 0 \end{pmatrix} - \frac{h}{\alpha_2}\begin{pmatrix} 0 \\ e^{t_n h} \end{pmatrix}$$

The exact solution at $t_n + h$ which we multiply by the matrix B for simplicity, is

$$B\begin{pmatrix} e^{t_n+h} \\ e^{t_n+h} \end{pmatrix} = e^{t_n+h}\begin{pmatrix} 1-h/\alpha_2 \\ -h/\alpha_2 \end{pmatrix}$$

Subtracting this expression from the approximate solution and substituting e^{t_n} for $y_{2,n}$, we get

$$B\mathbf{e}_{n+1} = e^{t_n}\begin{bmatrix} -\frac{\alpha_0}{\alpha_2} - \frac{\alpha_1}{\alpha_2}e^{\gamma h} - \left(1-\frac{h}{\alpha_2}\right)e^h \\ 0 \end{bmatrix}. \tag{6 8}$$

Expanding the exponentials in h in equation (6 8) and using the order conditions from Chapter 3, we obtain

$$B\mathbf{e}_{n+1} = \frac{h^2}{2\alpha_2}\begin{bmatrix} (2-\alpha_1\gamma^2-\alpha_2) + \frac{h}{3}(3-\alpha_1\gamma^3-\alpha_2) \\ 0 \end{bmatrix} e^{t_n}.$$

Thus

$$\mathbf{e}_{n+1} = -\frac{h}{2}\begin{bmatrix} (2-\alpha_1\gamma^2-\alpha_2) + \frac{h}{3}(3-\alpha_1\gamma^3-\alpha_2) \\ 0 \end{bmatrix} e^{t_n} \tag{6 9}$$

We can see from equation (6 9) that the error is $O(h)$ in the state component while the algebraic equation is solved exactly This result is in keeping with analysis given in section 5 2, where we showed that the Backward Euler does not attain the expected order of accuracy on DAEs

Finally, by multiplying through by $B^{-1}E$ we have

$$B^{-1}E\mathbf{e}_{n+1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0$$

Therefore the use of Petzold's error estimate (4 5) on this problem results in no error control, even though the observed errors are $O(h)$. Thus we recommend that index-2 problems are solved with constant stepsizes only

We do not consider stability for the Composite Integration scheme However we expect that no stability problems will arise for the index-1 case, since in this case stability is determined by differential variables only.

The implementation of the Composite scheme is similar to that outlined in Chapter 2 In this case, the nonlinear equations to be solved on the θ- stage are

$$E(\mathbf{y}_{n+\gamma} - \mathbf{y}_n) - \gamma h[(1-\theta)\mathbf{f}_n + \theta\mathbf{f}_{n+\gamma}] = 0$$

Applying Newton's method to this system gives the following iterative scheme for the unknown $\mathbf{y}_{n+\gamma}$

$$B\Delta\mathbf{y}^{i+1}_{n+\gamma} = \gamma h[(1-\theta)\mathbf{f}_n + \theta\mathbf{f}^i_{n+\gamma}] - E(\mathbf{y}^i_{n+\gamma} - \mathbf{y}_n)$$

with

$$\Delta\mathbf{y}^{i+1}_{n+\gamma} = \mathbf{y}^{i+1}_{n+\gamma} - \mathbf{y}^i_{n+\gamma}$$

and starting values

$$\mathbf{y}^0_{n+\gamma} = \mathbf{y}_n \quad \text{and} \quad \mathbf{f}^0_{n+\gamma} = \mathbf{f}_n$$

While the BDF scheme gives the following nonlinear equations to be solved at each stage

$$E(\alpha_0 \mathbf{y}_n + \alpha_1 \mathbf{y}_{n+\gamma} + \alpha_2 \mathbf{y}_{n+1}) - h\mathbf{f}_{n+1} = 0.$$

Once again Newtons method gives

$$B\Delta\mathbf{y}_{n+1}^{i+1} = h\mathbf{f}_{n+1}^{i} - E(\alpha_0 \mathbf{y}_n + \alpha_1 \mathbf{y}_{n+\gamma} + \alpha_2 \mathbf{y}_{n+1}^{i})$$

with

$$\Delta\mathbf{y}_{n+1} = \mathbf{y}_{n+1}^{i+1} - \mathbf{y}_{n+1}^{i}$$

and starting values

$$\mathbf{y}_{n+1}^{0} = \mathbf{y}_{n+\gamma} \quad \text{and} \quad \mathbf{f}_{n+1}^{0} = \mathbf{f}_{n+\gamma}.$$

Also both stages have the common iteration matrix

$$B = E - \gamma\theta h \frac{\partial \mathbf{f}_n}{\partial \mathbf{y}_n}.$$

Based on the results outlined above, we propose the following changes to the algorithm given in Chapter 3 for the Composite Integration scheme:
1. An error estimate of the form

$$\mathbf{e}_{DAE} = (E - \gamma\theta hJ)^{-1})E\mathbf{e}_{ODE}$$

where J is the Jacobian of $\mathbf{f}(\cdot)$ at $\mathbf{y}_n$
2. Place a lowerbound on the stepsize for the index-2 problems identical to the one given earlier for the DIRK(2,2) scheme. Note this is reasonable since $\gamma\theta = \alpha$, the parameter of the DIRK(2,2) scheme.
3. Replace the original implementation with that outlined earlier in this section.
4. Provide a simple form of damping in the iterative process, similar to that given in DASSL [56]. That is, add 0.75 times the correction vector on the θ- stage during the first step, for integrating index-2 problems.

6.4 ODEPACK & LSODI.

ODEPACK is a "systematized collection" of Fortran routines for the numerical solution of differential systems. The philosophy behind the concept is to provide a set general purpose routines with a standard user interface and common internal structure which make the routines more flexible, more portable and easier to install in software libraries. The first routine developed to conform with this philosophy was a package based on the GEAR [31] and GEARB [40] ODE codes, called LSODE (*Livermore Solver for ODEs* [43] written by A. C. Hindmarsh in 1975. LSODE combines the capabilities both GEAR & GEARB in that it solves explicitly given non-stiff and stiff ODEs of the form $\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t))$. In the stiff case, it treats the Jacobian matrix $\partial\mathbf{f}/\partial\mathbf{y}$ as either full or banded and as either user supplied or generated internally by differencing. LSODE is therefore a direct decendent of the GEAR package and also uses BDF formulae of orders $1 \leq k \leq 5$. Other routines in the ODEPACK family include LSODES, the general sparse Jacobian matrix solver written jointly with A. H. Sherman. LSODA, written jointly with L. Petzold, switches automatically between stiff and non-stiff methods (the suffix A stands for automatic). LSODAR, is

a version of LSODA having a root finding capability for a set of functions $\mathbf{g}(t, \mathbf{y})$ of independent and dependent variables in the ODE system This is sometimes called the g-stop feature It can be helpful in particle tracking where it is desireable to know when a particle reaches the walls of a container The last member of this family is the LSODI package, the linearly implicit solver. We shall discuss this code in more detail in the remainder of this section Before we go on to deal with LSODI, we mention that all the routines in ODEPACK use basically the same stepsize and order changing mechanism that is used in the GEAR package with slight modifications

LSODI [43] was written jointly by A. C. Hindmarsh & J F Painter at the Lawerence Livermore National Lab in California, U.S A . LSODI treats systems of the linearly implicit form $A(t, \mathbf{y})\,\mathbf{y}' = \mathbf{g}(t, \mathbf{y})$, where A is a square matrix LSODI allows A to be singular, but the user must then input consistent initial values for $\mathbf{y}$ and $\mathbf{y}'$ In the singular case we have a DAE system Then the user must be cautious about formulating a well posed problem, as LSODI is not designed to be robust in this case. LSODI is based on and supersedes GEARIB [41] and is only suitable for index-1 DAE systems

A numerical method for the linearly implicit system

$$A(t, \mathbf{y})\mathbf{y}' = \mathbf{g}(t, \mathbf{y}) \tag{6 10}$$

can be developed from the BDF formulae

$$\begin{aligned} \mathbf{y}_n &= h\beta_0 \mathbf{y}'_n + \sum_{i=1}^{k} \alpha_i \mathbf{y}_{n-i} \\ &= \mathbf{a}_n + h\beta_0 \mathbf{f}(t_n, \mathbf{y}_n) \end{aligned} \tag{6 11}$$

where the order of the method is k, with $(1 \leq k \leq 5)$ and $\beta_0 > 0$ Multiplying both sides by $A(t_n, \mathbf{y}_n)$, replacing $A(t_n, \mathbf{y}_n)\,\mathbf{y}'_n$ by $\mathbf{g}(t_n, \mathbf{y}_n)$ and solving the resulting implicit relation for $\mathbf{y}_n$, we obtain the following implicit relation using (6 11)

$$\mathbf{S}(\mathbf{y}) = A(t_n, \mathbf{y})\,\{\mathbf{y} - \mathbf{a}_n\} - h\beta_0 \mathbf{g}(t_n, \mathbf{y})$$

to be solved for $\mathbf{y} = \mathbf{y}_n$, where $\mathbf{a}_n$ is a constant vector This system is solved using a modified Newton iteration LSODI introduces a residual vector

$$\mathbf{r}(\mathbf{y}) = \mathbf{g}(t_n, \mathbf{y}) - A(t_n, \mathbf{y})\mathbf{s}$$

of values which the user is to supply Here $\mathbf{s}$ represents an approximation $\mathbf{y}'_n$ and $\mathbf{s}$ is specifically defined to be

$$\mathbf{s} = \frac{\mathbf{y}_n^{(0)} - \mathbf{a}_n}{h\beta_0}.$$

That is, $\mathbf{s}$ is the predicted value of $\mathbf{y}'_n$ that corresponds to the prediction $\mathbf{y}_n^{(0)}$ through the original formula $\mathbf{y}_n^{(0)} = \mathbf{a}_n + h\beta_0 \mathbf{s}$ $\mathbf{S}(\mathbf{y})$ and $\mathbf{r}(\mathbf{y})$ are then related by

$$\mathbf{S}(\mathbf{y}) = A(t_n, \mathbf{y})(\mathbf{y} - \mathbf{y}_n^{(0)}) - h\beta_0 \mathbf{r}(\mathbf{y})$$

LSODI in fact solves this system the associated iteration matrix is

$$P = \mathbf{S}'(\mathbf{y}_n^{(0)}) = A(t_n, \mathbf{y}_n^{(0)}) - h\beta_0 \mathbf{r}'(\mathbf{y}_n^{(0)})$$

where $\mathbf{r}'(\mathbf{y}_n^{(0)})$ denotes the Jacobian of r( ), that is, $\mathbf{r}'(\ ) = \partial \mathbf{r}/\partial \mathbf{y}$ Clearly in the case $A = I$ the identity matrix, the matrix P reduces to the usual ODE iteration matrix

The LSODI package and interface provide the following useful features·

(a) The matrices involved can be either treated as either full or banded by use of a method flag

(b) The dependence of A on y is automatically and inexpensively accounted for whether the partial derivatives are supplied or generated internally

(c) When A is singular the user need only supply the initial value of y' but no later values. If A is nonsingular then LSODI can be used to compute the initial value of y' using a flag

(d) To the maximum extent possible, LSODI shares the same user interface as LSODE and so reflects all the advantages over GEARIB that LSODE has over GEAR & GEARB, in terms of flexibility, convenience and portability.

The differences between LSODI and LSODE occur primarily in the user interface. In LSODI it is necessary to supply a routine to compute the residual function $\mathbf{r}(\mathbf{y}) = \mathbf{g}(t,\mathbf{y}) - A(t,\mathbf{y})\mathbf{s}$ and another routine to add the matrix A to a given array, while the Jacobian of r( ) *w r.t* y can be optionally supplied The use of r as the basic user-supplied quantity, as opposed to constructing r from g and A, is designed to allow for both computational and storage economies Usually the user can construct r without explicitly forming A, thus saving considerably on storage

Later in this Chapter, we discuss the performance of LSODI on a selection of test problems We have also considered it in Chapter 3 for solving ODEs, recall that the method proved very reliable and efficient on all problems considered, except those with eigenvalues close to the imaginary axis It is worth pointing out here, that LSODI proves itself an efficient solver on all index-1 DAEs we consider later in this Chapter and is equally efficient at solving these in ODE or DAE form

6.5 DASSL.

DASSL [56] is an acronymn for Differential/Algebraic System Solver, a Fortran code designed by Linda Petzold for the numerical integration of general implicit systems of differential equations of the form

$$\mathbf{F}(t,\mathbf{y},\mathbf{y}') = \mathbf{0} \tag{6 12}$$

with consitent initial conditions

$$\mathbf{y}(0) = \mathbf{y}_0 \quad \mathbf{y}'(0) = \mathbf{y}'_0$$

The underlying idea behind DASSL is to replace the derivative in (6 12) by a BDF difference approximation and solve the resulting nonlinear equations at each time step by Newtons method For the purpose of illustration, the first order BDF formula (*i e* the Backward Euler) gives the following nonlinear system

$$\mathbf{F}\left(t_n, \mathbf{y}_n, \frac{\mathbf{y}_n - \mathbf{y}_{n-1}}{\Delta t_n}\right) = 0 \tag{6 13}$$

to be solved at each time step by Newton's method

DASSL obtains an initial guess for $\mathbf{y}_n$ by evaluating a polynomial which interpolates the solution at the last $k+1$ points $t_{n-1}, t_{n-2}, \cdots, t_{n-(k+1)}$ at the current time t_n. An initial guess for $\mathbf{y}'$ is obtained by evaluating the derivative of this polynomial at t_n. Newton's method is then used to generate $\mathbf{y}_n$ as in (6.13) and the derivative is approximated by k^{th} order BDF formula, instead of the backward difference of $\mathbf{y}_n$. When the stepsize is not constant, DASSL uses the fixed leading coefficient form of the BDF formulae. Petzold [56] comments that these formulae tend to be more stable than the fixed coefficient formulae used in LSODI and are more efficient than the variable coefficient formulae used in EPISODE [40] in some cases. In DASSL these polynomials are represented in terms of scaled divided differences and the details can be found in Petzold [56].

The equation (6.13) can be rewritten as

$$\mathbf{F}(t, \mathbf{y}, \hat{\alpha}\mathbf{y} + \beta) = 0 \tag{6.14}$$

where $\hat{\alpha}$ is a constant that changes whenever the stepsize or order changes, β is a vector which depends on the solution at past times and $t, \mathbf{y}, \hat{\alpha}, \beta$ are evaluated at t_n. The nonlinear equation (6.14) is solved in DASSL by a modified Newton method as follows:

$$\mathbf{y}^{i+1} = \mathbf{y}^i - \gamma B \mathbf{F}(t, \mathbf{y}^i, \hat{\alpha}^i \mathbf{y}^i + \beta) \tag{6.15}$$

with the iteration matrix B computed as $\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}'} + \alpha \frac{\partial \mathbf{F}}{\partial \mathbf{y}}\right)$ and used for as many time steps as possible. In general the value of α when B was last computed is different from $\hat{\alpha}$ the value required at t_n. If α is very different from $\hat{\alpha}$ the (6.14) may not converge. In DASSL the constant γ is chosen to speed up convergence and is given by

$$\gamma = \frac{2}{1 + \hat{\alpha}/\alpha}.$$

This relaxation process has also been used by Dew & Walsh [22] and by Berzins et. al. [3] in their SPRINT solver.

The stepsize and order for the next step are determined using basically the same strategies as in Shampine & Gordon [66]. DASSL estimates the error that would have been made if the last few steps had been made with a constant stepsize at the current order k and at orders *k-2, k-1* and *k+1* . If these estimates increase, then k is increased; if they decrease, the order is lowered. The new stepsize is chosen so that the error estimate based on taking constant stepsizes at order k satisfies the error test.

DASSL also provides a *damped Newton* iteration in conjunction with a Backward Euler step to compute the initial values of $\mathbf{y}'$. Thus, in contrast to LSODI, the approach is applicable even if $\partial \mathbf{F}/\partial \mathbf{y}'$ is singular and the system is differential/algebraic. This capability is also available in the SPRINT solver of Berzins *et. al.* [3]. Recall that our one step schemes automatically generate the derivatives at the starting point because we have adopted the direct formulation outlined in Chapter 5.

The user-interface to DASSL is similar to that of LSODI in that it uses a residual formulation. In using DASSL, it is necessary to define the residual vector $\Delta = \mathbf{F}(t, \mathbf{y}, \mathbf{y}')$, thus Δ is the amount by which $\mathbf{F}$ fails to be zero for the inputs $t, \mathbf{y}, \mathbf{y}'$. The interface is a little more straightforward than that of LSODI in that the user can optionally supply the Jacobian of the full residual. Thus one routine is required

instead of the two required in LSODI DASSL has most of the other features of the ODEPACK codes. However it also includes a flag for dealing with discontinuities in the solution if the user has knowledge of the position of these points in the indepentant variable

The performance of DASSL along with the other integration schemes developed in this thesis will be considered in the next section

6.6 DAE test problems and results.

Perhaps the most irritating feature about DAEs is the lack of published results in the literature on the performance of ODE methods for solving this type of problem Gear [32] and Cameron [9] have both published test results and problems in this area. However the number of problems considered is small This contrasts completely with the ODE case, where comparisions have been made for all types of method using the stiff test set of Enright *et. al.* [27] Because of the lack of problems, we have constructed several of our own problems and taken a small number of others from the literature.

Our approach for constructing index-1 problems is simply to recast some of the stiff problems considered in Chapter 3 as DAEs In particular we have chosen Problems B5, C5, D1, E3 and P2 Additionly we consider an eight dimensional example solved by both Gear [32] and Cameron [9] We have also included a test problem given to us by C Fuher [30], which displays peculiar behaviour when solved in ODE and DAE form The final index-1 problem is taken from Roche [62] and is a modified version of the Pendulum equations. We also mention that Problem P2 is the example problem provided with both the LSODI and DASSL packages. We also consider two index-2 problems, the first can be found in Brenan & Petzold [5] It is a linear non-constant coefficient problem The second problem is also borrowed from [5], it in an index-2 version of the Pendulum equations.

We mention that the method of testing and the presentation of our results for the recast ODEs is the same as that outlined in Chapter 3 However for some of the later problems considered, we also supply tables illustrating accuracy achieved by the one step methods

6.6.1 Recast Index-1 problems.

Problem B5

$$\begin{aligned} y_1' &= y_7 \\ y_2' &= y_8 \\ y_3' &= -4y_3 \\ y_4' &= -y_4 \\ y_5' &= -0\,5y_5 \\ y_6' &= -0\,1y_6 \\ 0 &= -10\,y_1 + \alpha y_2 + y_7 \\ 0 &= \alpha y_7 + 10y_2 + y_8 \end{aligned}$$

with initial values

$$y_i = 1 \quad i = 1(1)6 \quad y_7 = 90 \quad y_8 = -110$$

and $\alpha = 100$ as in Chapter 3.

Recall from Chapter 3 that Problem B5 is taken from Enright *et. al.* [27]. It is linear with non real eigenvalues. In particular this problem is known to cause difficulties for BDF based codes, as transient eigenvalues lie in an unstable region for the higher order BDF formulae.

In the DAE version of Problem B5, given above, we have chosen to replace the rhs's of the first two equations of the system with two new variables y_7 and y_8. In order to make the system consistent again we introduce two new algebraic equations for these new variables. Now variables y_1, y_2, y_7 and y_8 will oscillate wildly. Therefore we expect that this problem should be more difficult to solve than its ODE counterpart.

Problem B5

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	866	2766	1783	363
NFE	5256	13872	3117	734
NJE	93	1380	116	14
GERR	9.4×10^{-4}	5.9×10^{-1}	1.0×10^{-2}	2.0×10^{-3}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	1514	2176	2459	1015
NFE	9150	18524	4190	2054
NJE	148	1067	152	23
GERR	4.8×10^{-5}	2.7×10^{-1}	4.0×10^{-3}	4.0×10^{-5}

Table 6.1

The DIRK(2,2) scheme proves less efficient on the DAE version of B5 as the results in Table 6.1 demonstrate. In the ODE case, (*c.f. Table 3.4*), recall at both tolerance values the method produced poor results and we attributed the difficulties encountered to this particular problem. It must still be stated that the DAE performance is considerably worse than that quoted in Alexander [1] for this problem. Again we mention that we solved B5 as a DAE with $\alpha = 8$ and 25[1]. It is clear from the results presented in Table 6.2., that the method is solving both versions of B5 efficiently when α is reduced. This is exactly the behaviour we observed in the ODE case. We are therefore led to question the validity of Alexander's results [1] on this problem, as he uses the same scheme with a slightly modified implementation.

The Composite Integration scheme finds this problem particularly difficult. It does not compute the solution to the same accuracy as the other methods. We also computed the solution to this problem using the Composite scheme without the addition of Petzold's error estimate (5.13). In this instance the the figures were identical to those quoted in Table 6.1. It therefore appears that the structure of the problem is causing the Newton iterative scheme to misbehave and we attribute the

[1] Recall that these are Problems B3 and B4 respectively from Enright *et. al.* [27].

DIRK(2,2) on Problem B5		
$Tol = 10^{-2}$	$\alpha = 8$	$\alpha = 25$
NSTEP	50	131
NFE	300	792
NJE	13	21
GERR	1.0×10^{-3}	5.0×10^{-4}

$Tol = 10^{-4}$	$\alpha = 8$	$\alpha = 25$
NSTEP	149	250
NFE	894	1500
NJE	19	27
GERR	9.0×10^{-5}	2.7×10^{-5}

Table 6.2

poor performance to this fact. While some deterioration is to be expected based on the other figures in Table 6.1, we feel the figures quoted are excessive.

Finally, comparing the figures given in Table 3.4 with those of Table 6.1 for both LSODI and DASSL, the anticipated deterioration in performance is bourne out. In fact LSODI's performance is a good deal poorer at both tolerance values. DASSL performs roughly twice as bad as it did in the ODE case, but still a good deal better than LSODI. We attribute this difference to the fact that DASSL uses the fixed leading coefficient form of BDF formulae. Petzold [56] suggest that these formulae may be more stable than other versions BDF formulae.

Problem C5.

$$\begin{aligned} y_1' &= y_5 \\ y_2' &= y_6 \\ y_3' &= -40y_3 + 4\beta(y_1^2 + y_2^2) \\ y_4' &= -100y_4 + 10\beta(y_1^2 + y_2^2 + y_3^2) \\ 0 &= y_1 + y_5 - 2 \\ 0 &= 10y_2 - \beta y_1^2 + y_6 \end{aligned}$$

The initial values are

$$y_i = 1 \quad i = 1(1)5 \quad y_6 = 10$$

and $\beta = 20$ once again.

This problem is nonlinear with real eigenvalues. The ODE version exhibits nonlinear coupling from the smooth to the transient components. In the DAE version above, we have chosen to replace the rhs's of equations 1 and 2 with new variables y_5 and y_6. The rhs's in turn are re-introduced as algebraic equations. This retains the coupling from smooth to transient components, except that now the coupling is through the intermediate variables we have artificially introduced.

Firstly, we consider the DIRK(2,2) scheme. Comparing the results given in Table 3.11 with those of Table 6.3, we observe that the method is efficient at solving this problem in either ODE or DAE form. In either case, very similar statistics were produced. The Composite Integration scheme also compares favourably as does both LSODI and DASSL. All methods considered found this problem easy to handle, producing virtually identical statistics in both ODE and DAE forms. However we again remark that the Composite scheme does not compute a solution to the same accuracy as the other schemes.

Problem D1.

$$y_1' = 0.2(y_2 - y_1)$$

Problem C5

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	73	40	42	49
NFE	438	234	132	107
NJE	12	11	11	14
GERR	2.0×10^{-5}	1.1×10^{-2}	4.0×10^{-5}	2.0×10^{-5}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	184	161	115	112
NFE	1128	1064	295	236
NJE	29	42	21	24
GERR	8.0×10^{-9}	3.5×10^{-4}	2.0×10^{-7}	2.0×10^{-6}

Table 6.3

$$\begin{aligned} y_2' &= 10y_1 - (60 - 0.125y_3)\, y_2 + 0.125y_3 \\ 0 &= y_3 - t \end{aligned} \tag{6.16}$$

with

$$y_i = 0 \quad i = 1(1)3.$$

This problem is nonlinear with real eigenvalues. The DAE is simply generated from the ODE by replacing the 3^{rd} differential equation $y_3' = 1$, by the algebraic equation $y_3 - t = 0$.

Problem D1

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	32	244	23	30
NFE	192	1421	97	79
NJE	7	75	9	15
GERR	4.0×10^{-2}	3.4×10^{-3}	1.0×10^{-3}	7.0×10^{-4}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	108	519	55	68
NFE	666	3553	164	147
NJE	15	168	13	19
GERR	1.0×10^{-3}	1.1×10^{-4}	2.0×10^{-5}	1.0×10^{-5}

Table 6.4

This problem caused particularly severe difficulties for the Composite Integration scheme, as the figures in Table 3.14 show. Similar behaviour is observed for the method solving the problem cast as a DAE, as the statistics in Table 6.4 indicate. The method still requires a large number of steps and function evaluations demonstrating that this problem is unsuitable for solution using the Composite scheme. In order to

further explain the weakness of the Composite scheme on this problem, we set $\theta = 1$, giving a DIRK(2,2) scheme We expected that performance of the resulting scheme would be similar to the results quoted for the DIRK(2,2) scheme given in Table 6 4 While the results of Table 6 5 for this experiment show a good deal of improvement, they still fall short of expected performance Therefore we are led to the conclusion that the poor performance is due to the different implementations used.

Compositescheme $\theta = 1$		
	$Tol = 10^{-2}$	$Tol = 10^{-4}$
NSTEP	131	289
NFE	817	2096
NJE	21	79
GERR	$3\,0 \times 10^{-3}$	1.9×10^{-4}

Table 6.5

The DIRK(2,2) scheme does moderately better on this problem in ODE form at the lower tolerance and proves itself twice as good at the higher tolerance value We feel this deterioration is due primarily to the change in structure of the problem which may affect the stability of the one step schemes Our reason for stating this is because this behaviour also manifests itself for the Composite scheme on this problem

Finally the BDF based codes solved both forms of D1 without any difficulty and with broadly similar statistics, indicating that these formulae have ideal stability properties for this problem

Problem E3

$$\begin{aligned} y_1' &= -(55 + y_3)y_1 + 65y_2 \\ y_2' &= 0\,0785(y_1 - y_2) \\ y_3' &= y_4 \\ 0 &= y_4 - 0.1y_1 \end{aligned}$$

with

$$y_1 = 1, \quad y_2 = 1, \quad y_3 = 0, \quad y_4 = 0\,1.$$

This problem is nonlinear with non real eigenvalues. The DAE version is derived from the ODE form by coupling the 3^{rd} differential equation to an algebraic equation incorporating a new variable y_4

The results we present in Table 6.6 show, that the DIRK(2,2) scheme solves the problem efficiently at both tolerances. Comparisions with the ODE case, (*c f Table 3 18*), show that similar statistics are reproduced in Table 6 6 The Composite scheme also produced similar figures albeit requiring quite a lot more steps and function evaluations than the other methods Lastly, both BDF codes solve the problem efficiently in either ODE or DAE form

Problem P2

$$\begin{aligned} y_1' &= -0\,04y_1 + 10^4 y_2\, y_3 \\ y_2' &= 0.04y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2 \\ 0 &= y_1 + y_2 + y_3 - 1 \end{aligned}$$

Problem E3

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP	31	157	33	33
NFE	186	707	106	76
NJE	7	39	12	12
GERR	1.0×10^{-1}	$9\,1 \times 10^{-4}$	$7\,0 \times 10^{-4}$	2.0×10^{-3}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	77	279	79	88
NFE	462	1558	195	190
NJE	15	77	17	15
GERR	$1\,0 \times 10^{-2}$	1.4×10^{-4}	$6\,0 \times 10^{-6}$	$5\,0 \times 10^{-5}$

Table 6.6

with

$$y_1 = 1, \quad y_2 = y_3 = 0$$

This is the well known chemical kinetics problem given in Chapter 3. Recall that as an ODE this problem has been considered by several workers In DAE form it is the original example problem supplied with both LSODI and DASSL We have taken this version of the problem directly from these codes

Problem P2

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	35	35	46	22
NFE	210	114	114	45
NJE	11	10	36	17
GERR	$2\,5 \times 10^{-2}$	$1\,2 \times 10^{-3}$	$4\,8 \times 10^{-3}$	2.0×10^{-3}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int*	*LSODI*	*DASSL*
NSTEP	87	60	37	41
NFE	522	267	56	90
NJE	14	16	15	17
GERR	$4\,8 \times 10^{-4}$	$1\,0 \times 10^{-4}$	$3\,0 \times 10^{-5}$	$4\,0 \times 10^{-5}$

Table 6 7

The DIRK(2,2) scheme proves slightly more efficient on the ODE problem as a comparision of figures in Tables 3 22 and 6 7 show It does however prove to be less efficient than the other method's. The reason for this is primarily due to the fact that this method is using the extrapolation based error estimate which requires six function evaluations per step. All other schemes integrated the problem efficiently, producing similar results regardless of problem formulation The other point worth noting about this problem is the improved performance of LSODI at the higher tolerance This we attribute to the method of error control we use, specifically, setting ierror = 1, rtol and atol as scalers set to 10^{-2} and 10^{-4} for the statistics quoted

The totals for each statistic are summerized here in Table 6 8 Table 6 8 also includes the totals for each statistic, when these problems are cast in both ODE and DAE form. These figures are based on the five problems considered above and the ODE figures are taken from Chapter 3. The benchmark we have adopted to measure the success of a numerical ODE based method on DAEs, is that the method should perform equally well on any problem regardless of its formulation Specifically, we require that any method will be efficient on any problem, cast in ODE or DAE form in terms of the statistics measured and that similar levels of global accuracy are obtained

Firstly, the results given for the Dirk(2,2) scheme in Table 6.8 are poorer for the DAE case. This again is primarily due to Problem B5 which in fact performs even worse as a DAE Overall one thing is apparent, that is the economy of the method in terms of Jacobian evaluations. Our implementation is bias toward a constant stepsize in order to achieve stability in the integration process. This reduces the number of step changes and consequently keeps the number of Jacobians required quite low

The Composite Integration scheme shows considerable change in performance on these problems However the excessive difference is due solely to problem B5 In fact this problem accounts for over $4/5^{th}$ of the total work on all problems Recall in Chapter 3 we stated that the selection of problems chosen included some of those that Carroll [18] found most difficult to solve with his scheme Our implementation has not improved on this situation

LSODI and DASSL both produce similar results regardless of problem formulation Once again DASSL proves the most efficient solver of those considered Based on these results it is apparent that these BDF based codes are excellent though it must be said that the one step schemes compare favourably in terms of Jacobian evaluations required Therefore these methods may be a worthwhile alternative to the BDF codes in some application areas

A final point we remark on here, is that the use of Petzold's error estimate (5 13) proved of little value When the correction vectors of an algorithm are kept bounded, both the ordinary estimate and Petzold's estimate behave identically on the index-1 problems considered above

6.6.2 Other Index-1 problems.

Problem P3 (Gear's problem [32])

$$\begin{aligned} y_i' &= s - (r - y_i^2) - \sum_{j=1}^{4} b_{ij} y_j \\ 0 &= y_5 - y_1 y_6 \\ 0 &= 2y_6 + y_6^3 - y_1 + y_7 - 1 - e^{-t} \\ 0 &= y_7 - y_8 + y_1 y_6 \\ 0 &= y_7 + y_8 + 5y_1 y_2 \end{aligned}$$

with

$$r - \sum_{j=1}^{4} y_j/2, \quad s = \sum_{j=1}^{4} (r - y_j^2)/2$$

$$t \in (0, 10^3)$$

Totals for all problems

$Tol = 10^{-2}$		*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	ODE	895	507	275	383
	DAE	373	2986	476	497
NFE	ODE	5770	2442	685	612
	DAE	2296	16348	3556	1041
NJE	ODE	176	128	87	76
	DAE	61	1515	164	72
$Tol = 10^{-4}$		*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	ODE	1142	1147	2255	829
	DAE	1422	2632	1019	1324
NFE	ODE	7374	6661	4387	1692
	DAE	8736	24966	4900	2717
NJE	ODE	169	283	210	73
	DAE	100	1370	218	98

Table 6.8

and initial conditions

$$y_i = -1, \quad i = 1(1)4, \quad y_5 = y_6 = 1, \quad y_7 = -2, \quad y_8 = -3,$$

also

$$b_{ij} = \begin{bmatrix} 447.5+\epsilon & -452.5+\epsilon & -47.5+\epsilon & -52.5-\epsilon \\ -452.5+\epsilon & 447.5+\epsilon & 52.5+\epsilon & 47.5-\epsilon \\ -47.5+\epsilon & 52.5+\epsilon & 447.5+\epsilon & 452.5-\epsilon \\ -52.5-\epsilon & 47.5-\epsilon & 452.5-\epsilon & 447.5+\epsilon \end{bmatrix}$$

with $\epsilon = 0.00025$.

Gear [32] originally constructed and proposed Problem P3. We applied the four methods outlined to this problem. As the results given in Table 6.9 show, the DIRK(2,2), Composite Integration scheme and LSODI algorithms all performed efficiently.

Originally we attempted to solve this problem with Carroll's version of the Composite scheme. It failed to adequately solve the problem. His version does not always reject the time step if the Newton scheme fails to converge. The resulting errors in the correction vector are therefore not picked up on the current time step. These are allowed to build up in the local error estimate until the local error estimate exceeds the tolerance. On this problem, we found that the corrections were large and grew too quickly for Carroll's version of the Composite scheme to control them. This caused the numerical solution to become unbounded and eventually overflow. To overcome this problem we decided to reject the step if the Newton iteration failed to converge. We then asked the code to decrease the stepsize by a factor of 4 and evaluate a new Jacobian and iteration matrix.

Cameron [9] also solved this problem using a fixed order DIRK(2,2) method identical to our scheme, but with a full Newton iterative scheme. A quick check on his

Problem P3

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*ADIRK(2,2)*
NSTEP	52	64	57	27
NFE	318	311	304	294
NJE	11	19	29	17
GERR	3.0×10^{-3}	4.7×10^{-3}	4.0×10^{-3}	9.0×10^{-5}

$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*ADIRK(2,2)*	Gear
NSTEP	135	139	92	165	168
NFE	834	925	383	1513	937
NJE	24	40	28	53	54
GERR	5.0×10^{-7}	3.5×10^{-4}	1.0×10^{-5}	4.0×10^{-6}	3.0×10^{-3}

Table 6.9

results (*c f* Table 6 9, ADIRK(2,2)) reveals that our results for all methods are better than Cameron's, in terms of function and Jacobian evaluations We also mention that Cameron solved this problem with variable order embedded DIRK codes We will not consider these results here, but remark that the variable order implementations were less efficient due to the greater overhead required to select the order and stepsize

Gear [32] also solved this problem at tolerances ranging from 10^{-4} to 10^{-8} We have reproduced his results at the 10^{-4} tolerance value Clearly the fixed order schemes perform equally well, but LSODI, which is a descendant of the Gear algorithm, proved to be over twice as efficient as the fixed order schemes LSODI is however a more finely tuned algorithm, in that it has improved error control capabilities over the Gear algorithm. This accounts to some extent for the improved performance.

The last point of interest we draw to the readers attention, is the performance of DASSL on this problem This code does generate a solution, but we halted the integration after the code reached $t = 50$ using 3000 integration steps at a tolerance of 10^{-2} We therefore have not supplied statistics for this method on this problem We also mention that the remarks we made above about Petzold's estimate apply here also

Problem P4 (see Fuher [30])

$$\begin{aligned} y_1' &= y_2 - a y_1^2 + \cos(t) \\ 0 &= y_2 - a y_1^2 \end{aligned}$$

with $t \in [0, 10\pi]$, $a = 200$ and $y_1 = y_2 = 0$

This problem has solutions

$$\begin{aligned} y_1 &= \sin(t) \\ y_2 &= 200 \sin^2(t) \end{aligned}$$

This problem was solved reasonably efficiently in DAE form as the statistics given in Table 6 10 demonstrate The highly oscillatory nature of the solution is the major

factor causing some numerical instability and the values of Global error to be large for one step schemes at the low tolerance.

Problem P4 as a DAE

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int.*	*LSODI*	*DASSL*
NSTEP	152	160	181	216
NFE	1068	1148	296	526
NJE	47	47	65	61
GERR	7.9×10^{0}	4.8×10^{0}	3.3×10^{-3}	6.0×10^{-2}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp Int.*	*LSODI*	*DASSL*
NSTEP	417	540	405	389
NFE	2754	4034	637	888
NJE	88	188	111	73
GERR	3.0×10^{-1}	6.6×10^{-3}	1.0×10^{-4}	1.0×10^{-5}

Table 6.10

In Table 6.11 we supply results for this problem recast as the ODE

$$\begin{aligned} y_1' &= y_2 - a y_1^2 + \cos(t) \\ y_2' &= 2a y_1 (y_2 - a y_1^2 + \cos(t)) \end{aligned}$$

with $t \in [0, 10\pi]$, $a = 200$ and $y_1(0) = y_2(0) = 0$

Firstly, as a DAE both one-step methods produced large absolute global errors with reasonably efficient performance statistics. In the case of the DIRK(2,2) scheme, the reasonable performance with poor error is explained by the fact that we used Petzold's error estimate (5.13). This in effect removes the algebraic component y_2, in the solution from the computation of error. It is this component which is oscillating wildly, from 0 to 200. In ODE form this component is included in error control, giving much improved accuracy and poorer performance.

The Composite scheme is behaving similarly to the DIRK(2,2) scheme in terms of the statistics measured. Global error however is still large for this scheme at the lower tolerance, indicating that the method is finding the problem hard to integrate. In fact the nature of this problem resembles that of Problem B5, which also proved difficult for this method.

The performance of the two other routines LSODI and DASSL was very different to that of the one step methods. In DAE form they produced accurate solution values with good performance characteristics. However as ODEs both methods failed to produce accurate solution values. It must be said that the large values of error quoted are again only in the y_2 component. Clearly both algorithms are unstable on this problem considered as an ODE. This can be partially explained by the fact that the eigenvalues of the system are lying on the imaginary axis in the complex plane. This region is known (see Chapter 2) to be unstable for higher order BDF formula. In fact small perturbations in the numerical solution might drive the eigenvalues into the right hand half of the complex plane, causing the solution to become unbounded. This appears to be happening to the numerical ODE solution in this case.

Problem P4 as an ODE

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp Int*	*LSODE*	*DASSL*
NSTEP	1507	1920	124	178
NFE	9414	15359	522	589
NJE	212	685	90	210
GERR	2.5×10^{1}	4.7×10^{-0}	3.0×10^{5}	8.0×10^{7}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int*	*LSODE*	*DASSL*
NSTEP	9965	4938	340	550
NFE	62334	40525	1394	2023
NJE	1380	863	340	550
GERR	9.0×10^{-3}	1.3×10^{0}	2.0×10^{3}	4.6×10^{2}

Table 6.11

Problem P5 (Roche [62], index-1 pendulum equations)

$$\begin{aligned} y_1' &= y_3 - y_1 y_6 \\ y_2' &= y_4 - y_2 y_6 \\ y_3' &= -y_1 y_5 \\ y_4' &= -y_2 y_5 - 1 \quad . \\ 0 &= y_3^2 + y_4^2 - y_2 - y_5 \\ 0 &= y_6 \end{aligned} \tag{6.17}$$

with $y_1 = 1$, $y_2 = y_3 = y_4 = y_5 = y_6 = 0$ and $t \in [0, 1]$

This is the last index-1 problem we consider, it is a version of the Pendulum Equations, introduced in Chapter 4, in modified index-1 form All methods solved this problem with no apparent problems except for the DIRK(2,2) scheme at the higher tolerance. The method requires about 6 function evaluations per step, this coupled with the fact that the method is conservative accounts for this difference

Roche [62] solved this problem with a constant step method in order to access the behaviour of the global error We repeated similar experiments for the Composite scheme Our results shown in Table 6.13 indicate that we are nearly obtaining an $O(h^2)$ level of global accuracy for the stepsizes considered We mention that this level of accuracy falls off as h is further decreased and we attribute this to rounding error It therefore appears that this scheme does not seem to suffer from the order reduction effects that occur for some methods as pointed out by Roche [62]

6.6.3 Index-2 problems.

Problem P6 (linear)
This problem is taken from Brenan & Petzold [5], it is a linear non-constant coefficient Index-2 DAE

$$y_1' = -e^{-t} y_1 + y_2 + y_4 + y_5 - e^{-t}$$

Problem P5

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	45	20	9	13
NFE	270	103	12	22
NJE	10	6	3	8
GERR	3.0×10^{-3}	6.6×10^{-3}	3.0×10^{-2}	1.0×10^{-1}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	326	37	19	22
NFE	2004	298	30	40
NJE	40	17	4	10
GERR	3.0×10^{-4}	2.2×10^{-4}	1.0×10^{-3}	2.0×10^{-3}

Table 6.12

Global errors of Composite Scheme

h	error	h^3
1.0^{-1}	1.6×10^{-3}	1.0×10^{-3}
5.0^{-2}	4.0×10^{-4}	1.3×10^{-4}
1.0^{-2}	1.7×10^{-5}	1.0×10^{-6}
5.0^{-3}	7.1×10^{-6}	1.3×10^{-7}

Table 6.13

$$
\begin{aligned}
y_2' &= -y_1 + y_2 - \sin(t)y_3 + y_5 - \cos(t) \\
y_3' &= \sin(t)y_1 + y_3 + \sin(t)y_4 - \sin^2(t) - e^{-t}\sin(t) \\
y_4' &= \cos(t)y_2 + y_3 + \sin(t)y_4 - e^{-t}(1 + \sin(t)) - \cos^2(t) - e^{-t} \\
0 &= y_1 \sin^2(t) + y_2 \cos^2(t) + (y_3 - e^t)(\sin(t) + 2\cos(t)) \\
&\quad + \sin(t)(y_4 - e^{-t})(\sin(t) + \cos(t) - 1) - \sin^3(t) - \cos^3(t)
\end{aligned}
$$

with exact solution

$$y_1 = \sin(t), \quad y_2 = \cos(t), \quad y_3 = e^t \quad y_4 = e^{-t} \quad y_5 = e^t \sin(t)$$

and $t \in [0, 1]$.

We initially remark, that at the higher tolerance DIRK(2,2) failed to take a first step with a singular iteration matrix. The difficulty here is that the error on the initial step is large. The reason for this is that two solutions are being computed in the one-step two-half-step error estimate. The discrepency between these is quite large when h is small for this problem. It is therefore impossible for the method to take a step once the tolerence is decreased. Petzold's error estimate also proved useful here, the method was unable to integrate the system without its use. LSODI was unable to solve the problem at either tolerance, the corrector iteration failed repeatedly. However when a one-step method did succeed in integrating the equations, it did so quite efficiently, finding no apparent problems with the higher index.

Problem P6

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP	54	65		28
NFE	348	659		73
NJE	13	31		14
GERR	7.0×10^{-3}	7.9×10^{-3}	1.0×10^{-0}	1.0×10^{-0}

$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP		544		634
NFE		5863		1297
NJE		258		336
GERR	1.0×10^{0}	2.4×10^{3}	1.0×10^{0}	1.0×10^{0}

Table 6.14

We also conducted fixed step experiments similar to those of Brenan & Petzold [5] for the one step schemes While the presentation of our results is different to their's, we point out that the results of Table 6.15 show that our methods are $O(h)$ accurate which is consistent with their results

Global errors of One Step schemes

h	*DIRK(2,2)*	*Comp Int*
1.3^{-1}	3.9×10^{-2}	7.5×10^{-2}
6.3^{-2}	1.6×10^{-2}	2.7×10^{-2}
3.1^{-2}	1.2×10^{-2}	1.1×10^{-2}
1.5^{-2}	7.4×10^{-3}	5.1×10^{-3}
7.8^{-3}	4.0×10^{-3}	2.4×10^{-3}
3.9^{-2}	2.0×10^{-3}	1.1×10^{-3}
1.9^{-3}	1.1×10^{-3}	5.8×10^{-3}
7.9^{-4}	5.3×10^{-4}	2.9×10^{-4}

Table 6.15

Problem P7 (Simple Pendulum Equations in index-2 form)
Once again these are taken from Brenan & Petzold [5] This is a nonlinear index-2 system of DAEs

$$\begin{aligned} y_1' &= y_3 - y_1 y_6 \\ y_2' &= y_4 - y_2 y_6 \\ y_3' &= -y_1 y_5 \\ y_4' &= -y_2 y_5 - 1 \\ 0 &= (1 - y_1^2 - y_2^2)/2 \\ 0 &= y_1 y_3 + y_2 y_4 \end{aligned}$$

with initial values $y_1 = 1$, $y_2 = y_3 = y_4 = y_5 = 0$ and $t \in [0, 1]$

Similar behaviour to the previous example is observed here, DIRK(2,2) failing at the higher tolerance and unable to successfully integrate the system without Petzold's estimate. LSODI was unable to take a first step at any tolerance and both the Composite scheme and DASSL integrated the problem efficiently at both tolerances as the figures in Table 6.16 indicate.

Problem P7

$Tol = 10^{-2}$	*DIRK(2,2)*	*Comp. Int.*	*LSODI*	*DASSL*
NSTEP	343	27		17
NFE	2118	173		47
NJE	44	9		13
GERR	2.7×10^{-2}	1.6×10^{-2}	1.0×10^{0}	1.7×10^{-2}
$Tol = 10^{-4}$	*DIRK(2,2)*	*Comp. Int*	*LSODI*	*DASSL*
NSTEP		80		78
NFE		620		200
NJE		23		66
GERR	1.0×10^{0}	9.0×10^{-3}	1.0×10^{0}	1.3×10^{-4}

Table 3.16

Once again we conducted fixed step experiments similar to those conducted for the previous problem As the figures given indicate the DIRK(2,2) scheme does not attain $O(h)$ accuracy. Thus it seem to be experiencing significant order reduction effects The Composite scheme however is approaching the $O(h)$ level of global accuracy which is quite good considering the nature of the problem

Global errors of One Step schemes

h	*DIRK(2,2)*	*Comp. Int*
1.3^{-1}	2.6×10^{-1}	7.7×10^{-3}
6.3^{-2}	1.4×10^{-1}	5.7×10^{-3}
3.1^{-2}	7.0×10^{-2}	3.2×10^{-3}
1.5^{-2}	3.0×10^{-2}	1.7×10^{-3}
7.8^{-3}	1.8×10^{-2}	1.1×10^{-3}
3.9^{-2}	7.6×10^{-3}	5.0×10^{-4}
1.9^{-3}	5.0×10^{-3}	3.0×10^{-4}
7.9^{-4}	2.0×10^{-3}	2.0×10^{-4}

Table 6.17

The results discussed for the index-2 systems clearly indicate that the conditioning of the iteration matrix is the key issue in solving higher index DAE sytems Unless this question can be successfully resolved, numerical ODE schemes will remain experimental for this type of problem A robust DAE solver therefore will probably have

to avoid Newton based iterative schemes. The Tensor approach outlined in the next Chapter, may help in overcoming this drawback in current DAE integration routines

Based on our results, we can also suggest that Petzold's error estimate (5 13) and its derivations given in Chapter 5 appear to be very useful Recall from above, that it was an essential ingredient for the one step methods developed in this thesis to solve the index-2 problems considered.

All the results quoted demonstrate that the one-step solvers are adequate and reasonably efficient for solving DAEs. However it must be pointed out that the LSODI and DASSL integration routines are more accurate and efficient at higher tolerances, as was the case for ODEs. Clearly the preference for index-1 problems should be the LSODI algorithm, because it is more reliable, although sometimes less efficient than the other methods considered For index-2 problems there appears to be only one choice in terms of reliability and accuracy, the DASSL algorithm. However the simplicity of the one step schemes along with their efficiency at low tolerances, may make them useful as Elliptic/Parabolic PDE integration routines using the Method of Lines. We therefore feel justified in saying that the schemes researched in this thesis provide an adequate alternative to BDF based codes for index-1 problems at low tolerance values

Chapter 7

Conclusions and Future Directions.

7.1 Introduction.

This thesis has studied the numerical solution of Ordinary and Algebraic Differential Equations In the first Chapter we set out the objectives of this study We identified the primary objective as the developement of efficient one step numerical methods for the solution of ODEs Associated with this, we pointed out our intention to study the theory of numerical schemes for ODEs.

Having completed our study of ODEs, we extended our brief to include DAEs Our objective was to extend the one step numerical schemes to handle DAEs In order to accomplish this task, we intended to cite recently published theory, which might aid our understanding of DAEs and their numerical solution.

Our intention then in this Chapter, is to evaluate our work against the objectives set out initially In the next section we will review our work and try to draw some conclusions Then in section 3, we will briefly consider the Tensor approach to solving nonlinear systems of equations as an alternative to Newton's method Finally we close the thesis with a look at some possible extensions of DAE type problems It is our belief that these problems have never been seen in the literature

7.2 Review and Conclusions.

In Chapter 2 we outlined the theory of stiff ODEs Concepts of convergence and order of accuracy were defined for numerical schemes applied to ODEs In particular we identified two well known types of numerical method, the Runge-Kutta (RK) methods and the Backward Differentiation Formulae (BDF), as special cases of the general Linear Multistep Method (LMM) We concentrated heavily on RK methods, defining stability concepts that have been well documented in the literature We gave a number of reasons why it is desirable for a one-step method to possess one of the many form of stability discussed But primarily we pointed out that stability would ensure linear error growth when solving stiff ODEs Classical methods such as the Euler method

$$y_{n+1} = y_n + h\, f(t_n, y_n)$$

failed to be efficient for solving these problems The reason we gave for this was that the stepsize had to be kept very small to ensure that the numerical solution converged to the true solution of the problem with the expected order of accuracy

We also considered practical aspects of solving ODEs in Chapter 2 Newton's method was applied to solve the nonlinear equations that arise from the application of an implicit numerical method to an ODE Recall that we pointed out that the size of the Lipshitz constant forced us to use a Modified Newton method rather than functional iteration on stiff ODEs. Then we considered error estimation for numerical schemes. The purpose of any practical error estimate is to instruct a numerical method to change the stepsize when conditions are desirable to do so. We considered three possible estimates that have been widely implemented. The BDF methods usually use the difference between the predicted and corrected solutions, while embedded and extrapolation techniques are used for RK methods. With any error estimate, the amount of work involved in its implementation is the primary factor in its choice However this must be measured against the simplicity of the estimate and its reliability We adopted the one-step-two-half-step extrapolation estimate for the DIRK(2,2) scheme for this reason That is because the technique is easily understood, well documented in the literature and proved reliable for other workers, such as Alexander [1] and Hall & Watt [38]

Chapter 3 introduced the one step schemes that are the backbone of this thesis We proved accuracy, A-, L-, S- and Strong S-stability for the DIRK(2,2) method, while we quoted Carroll [18] for accuracy requirements, A- and L-stability of the Composite Integration scheme Based on this theory, we developed two algorithms for the numerical solution of Stiff ODEs These algorithms were coded as variable step integration routines in Fortran Recall that the DIRK(2,2) implementation used an extrapolation based error estimate and Rosenbrock method for the solution of the nonlinear system The Composite Integration scheme used a modified Newton method for the nonlinear equations and an error estimate based upon a linear combination of available function values.

These algorithms were tested on a selection of problems from DETEST [27] The problems we chose, were those that Carroll's [18] implementation found difficult and those solved by Alexander [1] We demonstrated that our DIRK(2,2) code, proved as efficient as Alexander's, on all problems except B5 Our code proved considerably more efficient in terms of Jacobian evaluations, while it was less efficient $w\ r\ t$ the number of function evaluations required

The Composite Integration scheme was least competitive in terms of the statistics measured, but again was efficient in terms of Jacobian evaluations with reasonably good overall error behaviour Compared to Carroll's [18] results quoted in Chapter 3, we found our implementation to be slightly more efficient on the problems considered

Based on our numerical results of Chapter 3, it is clear that the one-step schemes meet the standards set out in our objectives in that they provide an efficient alternative to BDF methods, when solving stiff ODEs at low or moderate tolerances

In Chapter 4 we turned our attention to the second major topic of this study, DAEs We spent considerable time introducing the concept of index The connection between stiff ODEs and index-1 DAEs was demonstrated This was the reason we gave for applying stiff ODE numerical methods to the index-1 problem We then went on to review the literature on DAEs and outlined the transformation to Kronecker

canonical form for the linear constant coefficient DAE This transformation allowed us to define the concept of index, as the dimension of the nullspace of the differential operator for a DAE. The transformation was further generalized to the non-constant coefficient problem and the notion of global index was defined. Finally the index of a general DAE was defined in terms of the number of differentations required to generate an equivalent ODE system.

Initial conditions for DAEs were then considered We gave an example of Pantelide's algorithm [54] for generating consistent sets of i c's. The chapter closed with a look at possible methods for determining the index of a DAE system Recall that the only practical methods were graph-theoretic and these could have exponential running time. Consequently it is very difficult to estimate the index unless the problem possesses some structure. The reason why the index is so vital is that it determines the behaviour of a numerical method on a particular DAE

Numerical aspects were dealt with in Chapter 5. Again we were interested in numerical accuracy and stability. For index-1 DAE systems the behaviour is similar to the stiff ODE case, as we demonstrated by analyzing the Backward Euler on this problem However, the results of Petzold [55] for the linear constant coefficient problem show that errors may not decrease as the stepsize $h \to 0$, for higher index DAEs This, coupled with the fact that stability for numerical methods is not well understood for higher index DAEs, makes them unsuitable for solution by numerical ODE methods

The problem of error control can be overcome if a suitable error estimate is available. We recommended the use of the estimate introduced by Petzold [55] This estimate had the property that it only included state variables in the calculation of error Petzold [55] also showed that the estimate accurately reflects the local contribution to global error for BDF methods on index-2 problems

The most significant problems to overcome in solving DAEs are keeping the iteration matrix nonsingular and generating consistent initial conditions The Backward Euler method can be used for the purpose of finding i.c 's for linear DAEs. However no effective numerical techniques are available to handle this difficulty in general In order to ensure that ODE codes will be robust enough to handle DAEs, the question of singularity in the iteration scheme for the nonlinear equations must be addressed To date, no adequate techniques have been developed to overcome this problem Tensor methods, to be discussed in the next section, may provide a useful alternative to standard Newton schemes for this difficulty

Without an understanding of the theory of Chapters 4 and 5, it would be fruitless to attempt to solve DAEs Having considered the difficulties raised, we are aware that failure of an ODE code on DAEs is primarily due to two factors inadequate error estimates and poor conditioning of the iteration scheme These difficulties can be managed by the techniques outlined if they are understood However, since the tools to completely deal with these problems have not been perfected, it appears that a robust general DAE solver will take considerable effort to develop Our codes take into account the difficulties mentioned and attempt to manage them in as simple a manner as possible.

Chapter 6 returned to the one step schemes and outlined some modifications that would allow ODE based codes to handle index-1 and -2 DAE systems Recall that the modifications we suggested were simple in structure and interfered with the

original construction of the algorithms, given in Chapter 3, as little as possible In this Chapter we also outlined the LSODI [43] and DASSL [56] polyalgorithms. These were BDF-based variable order integration routines for solving DAE and Implicit ODE systems The primary difference between these methods and the one-step schemes was that we used the Direct formulation of the problem, while the polyalgorithms opted for the Residual formulation given in Chapter 5.

After discussing the methods, we gave a selection of test problems. Several of these were ODEs from DETEST [27], recast as DAEs Our results for these index-1 DAE systems demonstrated that the one step methods are efficient alternatives to BDF-based codes on most problems. All problems except B5 produced efficient and consistent results We also considered some other index-1 systems that have appeared in the literature. Again our one-step schemes proved as reliable and efficient as the polyalgorithms Only Fuher's problem [30] posed any real problems for the one step schemes It must also be stated that the BDF based codes also found difficulty with this particular problem For this reason we suggested that our schemes are an adequate alternative to the BDF-based codes and met with the objectives set out initially

On index-2 problems however the performance was quite good In fact our methods were able to solve the problems given at the lower tolerance Table's 6.15 and 6 17 demonstrated that the one step schemes produced levels of global accuracy consistent with theory, as predicted by Brenan & Petzold [5]. The LSODI algorithm was completely unable to handle the problems given In contrast, the DASSL routine was able to integrate the index-2 systems at both tolerance values efficiently Based on our observations therefore, we recommend this routine in preference to the one step schemes developed in this thesis It must be said, however, that the one step methods still prove useful for index-2 DAE systems Their simplicity allows easy modification Thus they can be included as integration routines in PDE solvers using the Method of Lines. We therefore feel that they should not be completely discounted as higher index DAE solvers

7.3 Tensor methods for solving Nonlinear Systems.

Tensor methods are a class of general purpose methods for solving systems of nonlinear equations They are intended to efficiently solve problems where the Jacobian matrix of the system at the solution is singular or ill-conditioned Their distinguishing feature, is that they base each iteration on a quadratic model of the nonlinear function In this section we summarize the work of Schnabel & Frank [64], in developing Tensor methods that are computationally efficient in space and time

Consider the general nonlinear system

$$\mathbf{f}(\mathbf{x}^{(*)}) = \mathbf{0} \tag{7.1}$$

where it is assumed that $\mathbf{f}(\ )$ is twice continuously differentiable and $\mathbf{x}^{(*)}$ is the solution to (7 1) In order to approximate the solution of (7 1), the standard approach is to base each iterate upon a linear model of $\mathbf{f}(\ )$ around the current iterate $\mathbf{x}^{(i)}$, thus

$$\mathbf{f}(\mathbf{x}^{(i)} + \mathbf{h}) = \mathbf{f}(\mathbf{x}^{(i)}) + J^{(i)}\mathbf{h} \tag{7 2}$$

where $\mathbf{h} \in \mathbf{R}^n$, is the correction vector and $J^{(i)} \in \mathbf{R}^{n \times n}$, is the Jacobian matrix of $\mathbf{f}(\ )$ at $\mathbf{x}^{(i)}$ Newton's method sets the next iterate $\mathbf{x}^{(i+1)}$ to the value of $\mathbf{x}^{(i)} + \mathbf{h}$, that solves (7.2) giving

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left(J^{(i)}\right)^{-1} \mathbf{f}(\mathbf{x}^{(i)}.$$

The main drawback of Newton's method, is that it fails to be quadratically convergent if $J^{(i)}$ is ill-conditioned or singular. Schanbel & Frank [64] point out that under these circumstances Newton's method is only linearly convergent with constant converging to 1/2 For example the behaviour of the sequence of iterates $x^{(k)}$ in the scalar problem is, (see Schnabel & Frank [64])

$$|x^{(k+1)} - x^{(k)}| = c^{(k)}|x^{(k)} - x^{(*)}|$$

with $lim_{k \to \infty} c^{(k)} = 1/2$ and $|x^{(*)} - x^{(0)}|$ being sufficiently small.

The main aim of Tensor methods is to provide a general purpose scheme that will have rapid convergence on ill-conditioned and singular problems They are based on expanding the linear model of $\mathbf{f}(\ )$ around $\mathbf{x}^{(i)}$ to the quadratic model

$$\mathbf{f}(\mathbf{x}^{(i)} + \mathbf{h}) = \mathbf{f}^{(i)} + J^{(i)}\mathbf{h} + \frac{1}{2}T^{(i)}\mathbf{hh} \tag{7 3}$$

where $T^{(i)} \in \mathbf{R}^{n \times n \times n}$ The three dimensional object $T^{(i)}$ is called a Tensor and we follow Schnabel & Frank [64] calling (7.3) a Tensor model The term $T^{(i)}\mathbf{hh}$ in (7 3) is defined by the quadratic form

$$\left(T^{(i)}\mathbf{hh}\right)_j = \mathbf{h}^t H_j \mathbf{h}$$

where H_j is the j^{th} horizontal face of $T^{(i)}$, that is the Hessian matrix associated with the j^{th} component function of $\mathbf{f}(\ )$

This model has a number of serious disadvantages

(a) n^3 second partial derivatives would have to be computed,

(b) $n^3/2$ additional storage locations would be needed compared to n^2 for the Newton model,

(c) to find a root of the model, each iteration would have to solve n quadratic equations in n unknowns, which requires an iterative process when $n > 1$,

(d) finally, the model may not have real roots

Schnabel & Frank [64] overcome these problems by avoiding the explicit calculation of the Tensor term in (7 3), they construct $T^{(i)}$ by asking the model to interpolate through previously computed values of the function $\mathbf{f}(\ )$ In particular they require that

$$\mathbf{f}(\mathbf{x}^{(-k)}) = \mathbf{f}(\mathbf{x}^{(i)}) + J^{(i)}\mathbf{s}^{(k)} + \frac{1}{2}T^{(i)}\mathbf{s}^{(k)}\mathbf{s}^{(k)} \tag{7 4}$$

$$k = 1, 2, 3, \quad , p$$

where

$$\mathbf{s}^{(k)} = \mathbf{x}^{(-k)} - \mathbf{x}^{(i)}$$

and $\mathbf{x}^{(-1)}, \cdot \quad , \mathbf{x}^{(-p)}$ are p past iterates that need not be consecutive. They use a modified Gram-Schmidt algorithm to select past iterates to include in the calculation of $T^{(i)}$, which requires about n^2 multiplications and additions. The equations (7.4)

are a set of $n\,p$ linear equations in n^3 unknowns Schanbel & Frank choose $T^{(i)}$ to be the solution to

$$minimize \|T^{(i)}\|_F$$

subject to

$$T^{(i)}\mathbf{s}^{(k)}\mathbf{s}^{(k)} = \mathbf{t}^{(k)} = 2\left(\mathbf{f}(\mathbf{x}^{(k)}) - \mathbf{f}(\mathbf{x}^{(i)}) - J^{(i)}\mathbf{x}^{(k)}\right)$$

where $\|\cdot\|_F$ is the Frobenius norm This has solution (see Schnabel & Frank [64])

$$T^{(i)} = \sum_{k=1}^{p} \mathbf{a}^{(k)}\mathbf{s}^{(k)}\mathbf{s}^{(k)}$$

with

$$\left(\mathbf{a}_j^{(1)}, \cdots, \mathbf{a}_j^{(p)}\right)^t = M^{-1}\left(\mathbf{t}_j^{(1)}, \cdots, \mathbf{t}_j^{(p)}\right)^t$$

and M is a positive definite matrix defined by

$$M_{j,l} = \left(\mathbf{s}_j^t\mathbf{s}_l\right)^2 \qquad 1 \le j, l \le p.$$

Substituting this in (7 2) gives

$$\mathbf{f}(\mathbf{x}^{(i)} + \mathbf{h}) = \mathbf{f}(\mathbf{x}^{(i)}) + J^{(i)}\mathbf{h} + 1/2 \sum_{k=1}^{p} \mathbf{a}^{(k)}\left(\mathbf{h}^t\mathbf{s}^{(k)}\right)^2 \tag{7 5}$$

Schnabel & Frank [64] subsequently solve these equations by noting that $\mathbf{f}(\mathbf{x}^{(i)} + \mathbf{h})$ is quadratic in a p-dimensional subspace spanned by $\mathbf{s}^{(k)}$ and linear in its orthogonal complement They apply an orthogonal transformation to partition the system so that the first $n - p$ components are linear and the remaining p components are quadratic Then they apply the QR algorithm to solve the Tensor model

Schanbel & Frank [64] give an algorithm for this process and comment that on singular systems the solution is usually well-posed They point out that linear part of their model is usually well-conditioned, while the ill-conditioning of the standard model is moved into the quadratic equations in the Tensor approach outlined This is also well-posed due to the tensor term Schnabel & Frank [64] also provide test results to demonstrate the efficiency of their approach They show that on singular problems their method is at least 30% more efficient than the standard method

Because this method appears to be very efficient on ill-conditioned problems, we feel that it would considerably improve the robustness of currently available DAE solvers We therefore think that it would be worth while to research this approach further in the context of Differential Algebraic Equations

7.4 Extensions of DAE problems.

The last problem we introduce into this thesis which, to our knowledge, has never appeared in the literature, is mixed differential, linear and nonlinear programming problems We present some possible versions of this problem which we feel could arise although we have no justification for making this assmption The linear versions could take the form

$$\max \quad F(t, y, y', z) = y' - f(t, y, z) \ .$$

subject to

$$ay + bz + g(t) < 0$$

where a, b are constants The nonlinear problem may be just the linear constraint replaced by one of the form

$$g(t, y, z) < 0.$$

Obviously more general versions of this problem can be constructed based on the DAE problems introduced in Chapter 4. While we have not analyzed these problems as they are outside the scope of the present work, certain questions can be immediately asked In particular existence and uniqueness need to be guaranteed for their solution. What analytic techniques are available for this type of problem. Is our knowledge of ODEs and DAE s useful in this context. Can our numerical methods be adapted to handle such problems, by possibly introducing some kind of Penalty Function technique to constrain the equations further, so that a solution can be found. Finally we ask does this type of problem ever occur in practice or is it just some kind of mathematical mutant that we have constructed We have not considered these questions but simply pose the problem as an interesting generalization of Ordinary and Algebraic Differential Equations considered in this work

Appendix A

Equivalence of DIRK(2,2) and Composite Integration schemes.

Recall the Composite Integration scheme The composite pair of formulae for the scalar first order ODE are

$$y_{n+\gamma} = y_n + \gamma h\,[(1-\theta)f_n + \theta f_{n+\gamma}] \tag{A.1}$$

and

$$\alpha_0 y_n + \alpha_1 y_{n+\gamma} + \alpha_2 y_{n+1} = h f_{n+1} \tag{A 2}$$

Consider the first stage (A 1), letting $k_1 = hf(t_n, y_n)$ this becomes

$$y_{n+\gamma} = y_n + \gamma(1-\theta)\,k_1 + \gamma\theta h\, f_{n+\gamma} \tag{A 3}$$

Now letting $k_2 = hf(t_n + \gamma h, y_{n+\gamma})$, we have

$$k_2 = \frac{y_{n+\gamma} - y_n - \gamma(1-\theta)k_1}{\gamma\theta}. \tag{A 4}$$

Substituting the expression for $y_{n+\gamma}$ in (A.3), into equation (A 4), we get

$$k_2 = hf(t_n + \gamma h, y_n + \gamma(1-\theta)k_1 + \gamma\theta k_2) \tag{A 5}$$

Taking the second stage (A 2), letting $z = \alpha_1 y_{n+\gamma} + \alpha_0 y_n$ and putting $k_3 = hf(t_n + h, y_{n+1})$, we obtain

$$y_{n+1} = (1/\alpha_2)\,\{k_3 - z\}$$

giving

$$k_3 = hf\left(t_n + h, \frac{1}{\alpha_2}\{k_3 - z\}\right). \tag{A 6}$$

Now substituting successively for the unknown z in terms of y_n, k_1, k_2 and using the order relations from Chapter 3, we get

$$k_3 = hf\left(t_n + h, y_n - \left(\frac{\alpha_1}{\alpha_2}\right)\gamma(1-\theta)k_1 - \left(\frac{\alpha_1}{\alpha_2}\right)\gamma\theta k_2 + \left(\frac{1}{\alpha_2}\right)k_3\right). \tag{A.7}$$

The Coefficient matrix for this method is therefore

$$\begin{array}{c|ccc} 0 & 0 & & \\ \gamma & \gamma(1-\theta) & \gamma\theta & \\ 1 & \frac{-\alpha_1}{\alpha_2}\gamma(1-\theta) & \frac{-\alpha_1}{\alpha_2}\gamma\theta & 1/\alpha_2 \\ \hline & \frac{-\alpha_1}{\alpha_2}\gamma(1-\theta) & \frac{-\alpha_1}{\alpha_2}\gamma\theta & 1/\alpha_2 \end{array} \tag{A 8}$$

Note All rows sum to 1 including the row of weights
Finally letting $\theta = 1$ and recalling that $\gamma\theta = 1/\alpha_2$, the above matrix becomes

$$\begin{array}{c|ccc} 0 & 0 & & \\ \gamma & 0 & \gamma & \\ 1 & 0 & 1-\gamma & \gamma \\ \hline & 0 & 1-\gamma & \gamma \end{array} \tag{A.9}$$

Since no information is required from the quadrature point t_n, we can rewrite this array as

$$\begin{array}{c|cc} \gamma & \gamma & \\ 1 & 1-\gamma & \gamma \\ \hline & 1-\gamma & \gamma \end{array} \tag{A 10}$$

This is the coefficient matrix of the DIRK(2,2) scheme of Chapter 3, since $\gamma\theta = \gamma = 1-1/\sqrt{2}$ Recall that this was the value choosen when we implemented the DIRK(2,2) scheme

Bibliography

[1] R Alexander, *Diagonally Implicit Runge-Kutta methods for Stiff ODEs*, SIAM J. Numer Anal , 14, 1977, 1006,1022.

[2] R E Bank *et al , Transient simulation of silicon devices and circuits*, IEEE ED-32, 1985, 1992-2007

[3] M Berzins *et al, SPRINT Software for time dependent problems,* University of Leeds, Rep. No 180, 1985

[4] K E Brenan & B. E Engquist, *Backward differentiation approximations of nonlinear differential-algebraic equations,* Dept of Comput Sci , Uppsala Univ Rep. No 101, Uppsala, Sweden, 1985

[5] K E Brenan & L. Petzold, *The numerical solution of higher index Differential Algebraic Equations by implicit methods,* SIAM J Numer Anal , 4, 1989, 976-996

[6] J C. Butcher, *Numerical Methods for Ordinary Differential Equations,* Wiley, 1987

[7] J. C. Butcher, *Coefficients for the study of Runge-Kutta integration processes,* J Austral Math Soc , 3, 1963, 50-64.

[8] J C Butcher, *Implicit Runge-Kutta processes,* Math & Comput., 18, 1964, 233-244

Ω

[9] I. T Cameron, *Solution of Differential Algebraic Systems using Diagonally Implicit Runge-Kutta methods,* IMA J of Numer Anal , 14, 1983, 273-288

[10] I T Cameron, *Numerical solution of differential-algebraic systems in process dynamics,* Ph D Thesis, University of London, 1982

[11] S L Campbell & C D Meyer Jr , *Generalized Inverses of Linear Transformations,* Pitman Publishing Co., 1979

[12] S L Campbell, *Singular Systems of Differential Equations,* Pitman Publishing, 1980

[13] S L Campbell, *Singular Systems of Differential Equations Vol 2,* Pitman Publishing, 1982.

[14] S L Campbell, *Index two linear time-varying singular systems of differential equations,* SIAM J Alg Disc Meth , 4, 1983, 237-243

[15] S L Campbell, *Consistent Initial Conditions for singular nonlinear systems,* IEEE CSSP-2, 1, 1983, 45-54

[16] S L Campbell, *One canonical form for the higher-index linear time-varying singular systems,* IEEE CSSP-2, 3, 1983, 311-326

[17] S. L. Campbell & L Petzold, *Canonical forms and solvable singular systems of differential equations,* SIAM J Alg Disc Meth , 4, 1983, 517-521.

[18] J Carroll, *A composite integration scheme for the numerical solution of systems of ordinary differential equations,* J Comput. & Appl Math , 25 1989, 1-13

[19] J. Carroll, Exponentially fitted one-step methods for the numerical solution of some stiff initial value problems, Ph D thesis, T C.D , Dublin, 1984

[20] J. Carroll, *The numerical solution of Ordinary, Parabolic and Elliptic/Parabolic Differential Equations,* DCU Research Report

[21] J R Cash, *Diagonally Implicit Runge-Kutta formula with error estimates,* J. Inst Maths & Applics., 24, 1979, 293-302

[22] T.S Chua & P M Dew, *The design of a variable step integrator for the simulation of gas transmission networks,* University of Leeds, Rep. No 157, 1982

[23] M Crouziex, *Sur l'approximation des equations differentialles operationnelles lineaires par des methodes de Runge-Kutta,* These presentee a l'Universite Paris VI, Paris 1975

[24] G G. Dahlquist, *A special stability problem for linear multistep methods,* BIT 3, 1963, 27-43

[25] I S. Duff & C. W. Gear, *Computing the structural index,* Tech Memorandum No 50, Math. & Comp Sci Div , Argonne Nat Lab 1985

[26] B Ehle, *On Pade approximations to the exponential function and A-stable methods for the solution of initial value problems,* Ph D Thesis, University of Waterloo, Ontario, Canada, 1969

[27] W. H Enright *et. al, Comparing Numerical Methods for Stiff Systems of ODEs,* BIT 15, 1975, 10-48

[28] E Fehlberg, *Klassische Runge-Kutta formeln funfter und siebenter ordnung mit schrittweitenkontrolle,* Computing J , 4, 1969

[29] R Frank, J Schneid and C W Ueberhuber, *Order results for stiff ODE systems,* SIAM J of Numer. Anal , 22, 1985, 515-534

[30] C Fuher, *private communication*

[31] C W Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, N J , U S A , 1971

[32] C W Gear, *Simultaneous numerical solution of differential-algebraic equations,* IEEE CT-18, 1971, 89-95

[33] C W Gear, *Differential-algebraic equation index transformation,* SIAM J. Sci Stat Comput , 9, 1988, 39-47

[34] C. W Gear & G Brown, *An implicit equation solver,* University of Illonois, 1973.

[35] C W. Gear & L. Petzold, *ODE methods for the solution of differential algebraic systems,* SIAM J of Numer Anal , 21, 1984, 716-728

[36] C. W Gear, B Leimkuhler & G K. Gupta, *Automatic integration of Euler-Lagrange equations with constraints,* J Comput Appl Math , 12 and 13, 1985, 77-90.

[37] E. Griepentrog & R Marz, *Differential Algebraic Equations and their Numerical Treatment,* Teuber Text zur Mathematik - Band 88, Leipzig, 1986.

[38] G Hall & J M Watt, *Modern Numerical Methods for Ordinary Differential,* Clarendon Press, Oxford, 1976

[39] A C Hindmarsh, *GEARB Solution of Ordinary Differential Equations having banded a Jacobian,* Rep UCID-30112, Lawerence Livermore Lab , 1977

[40] A C Hindmarsh & G D Byrne, *EPISODE, an experimental package for the integration of systems of ordinary differential equations,* Rep UCID-30112, Lawerence Livermore Lab , CA May 1975.

[41] A C Hindmarsh, *Preliminary documentation of GEARIB· Solutions of Implicit Systems of Ordinary Differential Equations with banded Jacobian,* Rpt UCID-30130, Lawerence Livermore Lab , CA, Feb 1976

[42] A. C Hindmarsh, *Preliminary Documentation of GEARBI Solution of ODE Systems with Block-Iterative treatment of the Jacobian,* Rpt UCID-30149, Lawerence Livermore Lab , CA Dec. 1976

[43] A C Hindmarsh, *ODEPACK, a systematized collection of ODE solvers,* Scientific Computing, R S Stapleman et al (eds), North Holland, Amsterdam, 1983

[44] B Kangstrom, *RGSVD-An algorithm for computing the kronecker structure and reducing subspaces of singular matrix pencils,* SIAM J. Sci Stat Comput , 7, 1986, 185-211

[45] W Krysig, *Advanced Engineering Mathematics,* Wiley, 1985

[46] J D Lambert, *Computational Methods in Ordinary Differential Equations,* Wiley, 1973

[64] R B Schnabel & P D. Frank, *Solving Systems of Nonlinear Equations by Tensor Methods,* in The State of the Art in Numerical Analysis, A Iserles & M J D Powell eds , Clarendon Press, Oxford, 1987.

[65] L. F Shampine, *Implementation of implicit formulas for the solution of ODEs,* SIAM J. Sci. Stat. Comput., 1, 1980, 103-118.

[66] L. F Shampine and M. K. Gordon, *Solution of Ordinary Differential Equations - The Initial Value Problem,* W H. Freeman, 1975

[67] L. F. Shampine and H. A. Watts, *Numerical Solution of ODEs,* Prentice Hall, 1975

[68] R F. Sincovec *et al, Analysis of Descriptor Systems using numerical algorithms,* IEEE AC-26, No. 1, 1981, 139-147.

[69] H. H Rosenbrock, *Some general implicit processes for the numerical solution of differential equations,* Computer J., 5, 1963, 329-331.

[70] J G. Verwer, *S-stability properties of generalized Runge-Kutta methods,* Numer. Math., 27, 1977, 359-370.