Implementation of General Linear Methods for Stiff Ordinary Differential Equations

Shirley Jun Ying Huang

A thesis submitted for the degree of

Doctor of Philosophy

Department of Mathematics

The University of Auckland

2005

Abstract

A new type of general linear method for the numerical solution of stiff differential equations has been discovered recently. These methods are characterised by a property known as "inherent Runge-Kutta stability". This property implies that the stability matrix has only a single non-zero eigenvalue and that this eigenvalue is a rational approximation to the exponential function. This rational approximation is just like the stability function of a Runge-Kutta method. The theoretical properties of the new methods, such as stability and order, are surveyed. Also constructing practical general linear methods of this type for stiff problems is discussed.

The emphasis of the thesis is on implementation and numerical experiments. We have investigated several implementation questions such as starting methods, prediction of the stage values, an efficient iteration scheme, truncation error estimation and stepsize control. For each of these questions, numerical tests have been carried out.

To investigate a fixed order general linear method, a starting method is needed to provide the first incoming approximations. It is to be expected that a good prediction of the initial iteration in the Newton method calculations will reduce the number of iterations required. Since the *LU* factorizations are expensive, we have designed an efficient iteration scheme to reduce the computational cost. A reliable error estimator gives an optimal stepsize sequence. These implementation issues are addressed in detail.

Three general linear methods of order 2, 3 and 4 are selected as a basis for the numerical experiments. The numerical results provide evidence that the new type of general linear method is feasible as the basis for an efficient solver of stiff ordinary differential equations.

Acknowledgements

I would like to thank all the people who have helped me make this thesis possible. It is not possible to name all here but I will name just a few.

Firstly, I wish to express the most sincere thanks to my supervisor, Prof. John Butcher, for introducing me to this fascinating and challenging area of research, for his guidance throughout my doctoral study, for his patience, encouragement, understanding, friendship and many other aspects. I have benefited not only from his academic knowledge but also his attitude toward the meaning of life. I suspect I will never be able to repay him for what he has done for me.

Secondly, I would like to thank my advisor Dr. Robert Chan for his continual encouragement, friendship and many helpful discussions.

I would also like to record my gratitude to Dr. Allison Heard, Dr. Will Wright, Dr. Helmut Podhaisky, Mrs Nicolette Rattenbury and Ms Angela Tsai. They were always there whenever I needed to rehearse for my presentations, discuss my research and proof-read my thesis. They helped me to improve my English and gave me numerous useful comments at various stages of writing this thesis.

I am also grateful to Dr. David Chen, Dr. Tina Chan, Dr. Alona Ben-Tal, Mr. Steffen Schulz, Ms. Jane Lee and Ms. Priscilla Tse for their friendship and encouragement. I would especially like to acknowledge Dr. Philip Sharp for the discussions about Fortran programming.

The weekly workshop in numerical analysis at the University of Auckland has been extremely helpful for me in understanding many important concepts, improving my presentation skills and building my confidence.

It is my pleasure to thank the University of Auckland for offering me a PhD scholarship in the first three years of my study, the University of Auckland Research Office for financial support towards my overseas conference travel, Prof. John Butcher's Marsden Fund project funded by the Royal Society of New Zealand for supporting my overseas conference travel and the New Zealand Institute of Mathematics & its Applications (NZIMA) Numerical Analysis program for financial support in the last year of my study and the Mathematics department PhD student fund for helping me attending New Zealand conferences.

Finally, I express my deep appreciation to my family. I thank my parents, who have sacrificed so much for my education and have encouraged me towards doctoral study. Without their emotional support, I am sure I would not have been able to finish my study. I would like to especially thank my husband, Dr. Jun Yu, for his understanding, encouragement and discussions. Finally, I want to express my heartful thanks to my dear daughter Jessica for giving me the happiness, encouragement, love and precious time to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In science, social science and engineering, differential equations often have to be solved. Although in some cases they can be solved analytically, unfortunately, the majority of differential equations are too complicated to have analytic solutions. Even when analytic solutions can be found, they are not always useful in practice since the computational cost involved is too great.

For example, it can be shown that the following problem:

$$\begin{aligned} y'(x) &= 1 - 2xy, \qquad x \in [0, T], \\ y(0) &= 0, \end{aligned}$$

has the analytic solution:

$$y(x) = e^{-x^2} \int_0^x e^{t^2} dt. \tag{1.1}$$

If y is evaluated at several values of x, one has to do numerical integration that could be computationally intensive. Another example is to solve a linear ordinary differential equation analytically. We often have to find roots of the corresponding characteristic equation. This requires one to solve high degree algebraic equations and hence can be difficult.

The above examples show that solving ordinary differential equations analytically is not always feasible. On the other hand, quite often the analytical solution is not required. In many cases one only needs to approximate the solutions at some discrete points or approximate the analytical solution. As a consequence, numerical methods for solving ordinary differential equations are very useful and practical. However, different problems need different methods. In this thesis, we study some numerical methods for ordinary differential equations with a given initial condition, y_0, the so-called initial value problems (IVP),

$$\begin{aligned} y'(x) &= f(x, y(x)), \quad f : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N \\ y_0 &= y(x_0). \end{aligned}$$

Without loss of generality, by defining $z_i(x) = y_i(x)$ (i =1,2,$\cdots$,N) and $z_{n+1}(x) = x$, we can express the problem in autonomous form

$$\begin{aligned} z'(x) &= g(z(x)), \quad g : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1} \\ z_0 &= z(x_0). \end{aligned}$$

In this thesis we only discuss autonomous IVP.

In this chapter, we will first survey some well known numerical methods for ordinary differential equations, such as the Euler method, Runge-Kutta methods, linear multistep methods and general linear methods. We then discuss a special group of problems called stiff problems.

1.1 Euler method

Numerical methods for the solution of ordinary differential equations have been developed over one hundred years. Butcher published a survey paper in 2000 [4]. A classical way of solving initial value problems numerically is by the method of Euler [19]. To solve the differential equation

$$y'(x) \;=\; f(y(x)), \quad y_0 = y(x_0) \tag{1.2}$$

and output a solution at a particular point $\bar{x}$, firstly, we divide the interval $[x_0, \bar{x}]$ into a sequence of grid points, $x_n, n = 1, 2, \ldots, N$ where $x_n = x_{n-1} + h_n$ and h_n is called the stepsize. The Euler method computes y_n as an approximation to $y(x_n)$ at x_n where $y(x_n)$ represents the exact solution. The Euler method generates approximations using the following formula

$$y_n = y_{n-1} + h_n f(y_{n-1}), \quad n = 1, 2, \cdots, N, \quad y_0 = y(x_0). \tag{1.3}$$

This is the explicit Euler method. The next formula is referred to as the implicit Euler method when the equation system is nonlinear,

$$y_n = y_{n-1} + h_n f(y_n), \quad n = 1, 2, \ldots, N, \quad y_0 = y(x_0). \tag{1.4}$$

Note that the imlicit Euler method requires the solution of a nonlinear equation system which will cause the high computational cost.

We now review questions of convergence and stability for the Euler method.

1.1.1 Convergence and order

For any point x_n in a fixed interval, let Y denote the approximation of $y(x_n)$ and H denote the stepsize. By using the Euler method, we obtain a sequence of solutions, $(Y_1, Y_2, \ldots)$ and a corresponding sequence of step sizes, $(H_1, H_2, \ldots)$. For the convergence of Euler method, we then have the following theorem.

Theorem 1.1 *[5] If $H_k \to 0$ as $k \to \infty$ and $Y_k(x_0) \to y(x_0)$ as $k \to \infty$, then Y_k converges uniformly to y as $k \to \infty$.*

For simplicity, we suppose h_n is a constant h and consider the solution only at a fixed point $\bar{x}$. Let $Y(\bar{x})$ represent the numerical solution and $y(\bar{x})$ represent the exact solution. There are two numerical errors need to be considered: the local error and the global error. The local error is the error introduced in a single step of the integration, while the global error is the overall error caused by many integration steps. Since global error is accumulated by local error, we consider local error. At a particular step, we could get two type of errors, the round-off error and the truncation error. The round-off error is due to the limitation of a computer, while the truncation error is presented because of the infinite Taylor series. In this thesis we only talk about the truncation error. For a point x_n, we assume that the numerical solution is the same as exact solution, that is $Y(x_n) = y(x_n)$, then at point x_{n+1}, we define the local truncation error as

$$E(x_{n+1}) = Y(x_{n+1}) - y(x_{n+1}) \tag{1.5}$$

Theorem 1.1 shows that $\|Y(\bar{x}) - y(\bar{x})\|/h$ is bounded as $h \to 0$. For some positive integer p, if we have $\|Y(\bar{x}) - y(\bar{x})\|/h^p$ bounded, then we would define "the numerical solution to be of order p". If p is the highest value such that $\|Y(\bar{x}) - y(\bar{x})\|/h^p$ is bounded, we would define "the order of the numerical solution to be p". By comparing the Euler solution $Y(\bar{x})$ with the Taylor series of the exact solution at $\bar{x}$ given by

$$\begin{aligned} y(\bar{x}) &= \sum_{i=0}^{\infty} \frac{h^i}{i!} y^{(i)}(\bar{x}-h) \\ &= y(\bar{x}-h) + hy'(\bar{x}-h) + O(h^2), \end{aligned}$$

we get

$$Y(\bar{x}) - y(\bar{x}) = O(h^2). \tag{1.6}$$

This gives the order of the Euler method as at least 1.

1.1.2 Stability

Applying the explicit Euler method to a linear differential equation given by

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

where L can be complex, then we have

$$y_n = (1+hL)y_{n-1} = \cdots = (1+hL)^n y_0.$$

We write $z = hL$. The numerical solution y_n is bounded if and only if $|1+z| \le 1$. The region, $|1+z| \le 1$, called the stability region, is shown in Figure 1.1.

For the implicit Euler method, we have

$$y_n = \frac{1}{1-hL} y_{n-1} = \cdots = \left(\frac{1}{1-hL}\right)^n y_0.$$

Therefore, the stability region is $|1-z| \ge 1$. Figure 1.2 shows the stability region for the implicit Euler method.

The Euler method is regarded as an object of theoretical study but its suitability for practical computation is limited by the need for an unreasonably small

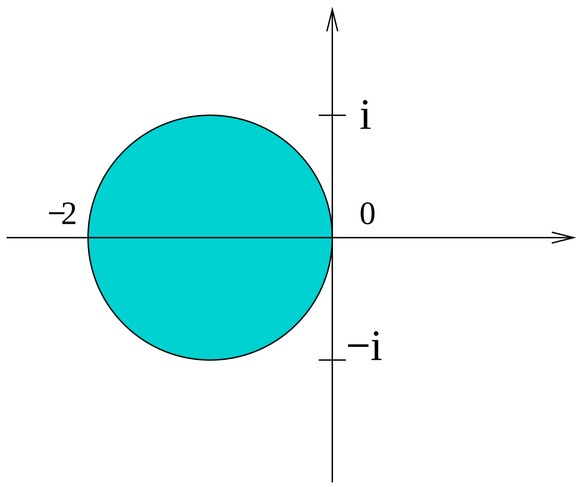

Figure 1.1: Stability region of explicit Euler method (shaded area).

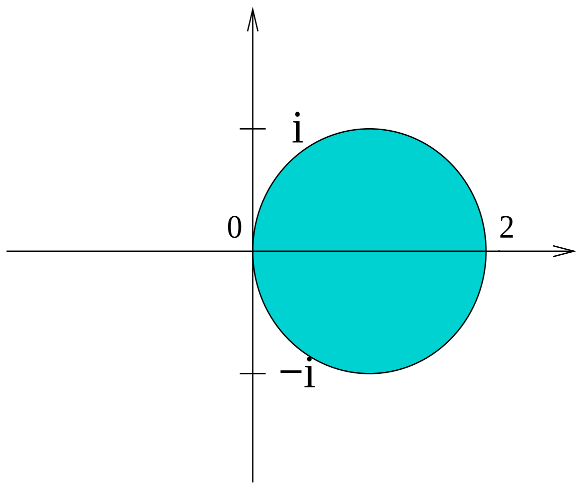

Figure 1.2: Stability region of implicit Euler method (unshaded area).

stepsize to reach a satisfactory accuracy. Some generalizations in various directions have been studied. Traditional higher order methods mainly lie in two large classes: multistep methods and multistage methods. Multistage methods involve calculating the derivative of f several times inside each step, and the most important multistage methods are Runge-Kutta methods. Multistep methods advance the solution approximation by using a linear combination of the information generated in past steps. For stiff problems, the most important of the multistep methods are the backward differentiation formula (BDF) methods. We now review these two classes of methods separately.

1.2 Runge-Kutta methods

The first important generalization of the Euler method is the use of Runge-Kutta methods. Runge [35] proposed the idea of extending the Euler method by allowing for a multiplicity of evaluations of function f within one step, then using the information obtained to match a Taylor series expansion up to some higher order. Later, Heun [27] and Kutta [29] made further contributions in this area. Kutta [29] completely characterized the order 4 Runge-Kutta methods and proposed the methods of order 5.

For an ordinary differential equation system $y' = f(y)$, a general Runge-Kutta method is of the form

$$Y_i = y_{n-1} + h\sum_{j=1}^{s} a_{ij} f(Y_j), \quad i = 1, \ldots, s, \tag{1.7}$$

$$y_n = y_{n-1} + h\sum_{i=1}^{s} b_i f(Y_i), \tag{1.8}$$

where the quantities Y_1, Y_2, …, Y_s are called stage values. They are approximations to solution values $y(x_{n-1} + c_i h)$ at points $x_{n-1} + c_i h$. The integer s is the number of stages of the method. The c_i represents the position of the internal stages within one step. The Runge-Kutta formula can be conveniently represented by the following "Butcher tableau" [5],

$$\begin{array}{c|c} c & A \\ \hline & b^T. \end{array}$$

where $A = \{a_{ij}\}$, $b^T = \{b_i\}$ and $c = \{c_i\}$. The set of numbers a_{ij} are the coefficients used to find the internal stages using linear combinations of the stage derivatives. The components of the vector b^T are coefficients which represent how the numerical solution at this step depends on the derivatives of the internal stages. The vector $c = [c_1, c_2, \cdots, c_s]^T$ is called the abscissae.

There are two main kinds of Runge-Kutta methods, implicit and explicit. If matrix A is strictly lower triangular, i.e., the internal stages can be calculated without depending on later stages, then the method is called an explicit method. Otherwise the internal stages depend not only on the previous stages but also on the current stage and later stages, and then the method is called an implicit method. The implicit method involves Newton iterations to evaluate the stage values. Those methods for which the matrix A is lower triangular are called diagonally implicit methods. The methods are known as "fully implicit" if the matrix A is not lower triangular. The following are some examples of Runge-Kutta methods. In the section we will review some important properties of the Runge-Kutta methods.

- A three stage order three explicit Runge-Kutta method,

$$\begin{array}{c|ccc}
0 & 0 & & \\
\frac{2}{3} & \frac{2}{3} & 0 & \\
\frac{2}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\
\hline
 & \frac{1}{4} & 0 & \frac{3}{4}
\end{array}$$

- A three stage order four implicit Runge-Kutta method (Lobatto IIIC),

$$\begin{array}{c|ccc}
0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\
\frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\
1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\
\hline
 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6}
\end{array}$$

- A two stage order two diagonally implicit Runge-Kutta method,

$$\begin{array}{c|cc}
1-\frac{1}{2}\sqrt{2} & 1-\frac{1}{2}\sqrt{2} & 0 \\
1 & \frac{1}{2}\sqrt{2} & 1-\frac{1}{2}\sqrt{2} \\
\hline
 & \frac{1}{2}\sqrt{2} & 1-\frac{1}{2}\sqrt{2}
\end{array}$$

1.2.1 Stability

The stability properties of numerical methods are important for achieving a good approximation to the true solution. When a numerical method is used, then at each mesh point, there are differences between the exact solution and the numerical solution. That is the local truncation error. Sometimes the accumulation of errors will cause instability and the numerical solution will no longer follow the path of the 'true' solution. Therefore, a method must satisfy the stability condition so that the numerical solution converges to the exact solution. We now review the stability of Runge-Kutta methods.

Consider the standard linear test problem

$$\begin{aligned} y'(x) &= Ly(x), \qquad (1.9) \\ y_0 &= 1, \end{aligned}$$

where L is a complex number. Applying an s-stage Runge-Kutta method, we use the matrix form to present the internal stages and the approximation to the solution as

$$Y = y_{n-1}e + hLAY, \qquad (1.10)$$

$$y_n = y_{n-1} + hb^T LY, \qquad (1.11)$$

where $e = [1, 1, \cdots, 1]^T$ and $Y = [Y_1, Y_2, \cdots, Y_s]^T$. We write $z = hL$ and calculate Y from the first equation then insert Y into the the second equation. The equation becomes

$$\begin{aligned} y_n &= y_{n-1} + zb^T(I - zA)^{-1}y_{n-1}e \\ &= \left(1 + zb^T(I - zA)^{-1}e\right) y_{n-1} \\ &= R(z)y_{n-1} \end{aligned}$$

where $R(z) = 1 + zb^T(I - zA)^{-1}e$ is a function of z.

Definition 1.2 *[26] $R(z) = 1 + zb^T(I - zA)^{-1}e$ is the stability function of the method. The set*

$$S = \{z \in C : |R(z)| \leq 1\}$$

is called the stability region of the method.

For an explicit method, the coefficients of A, $a_{ij} = 0$ for $j \geq i$. In this case, we have

$$R(z) = 1 + z\sum_j b_j + z^2 \sum_{j,k} b_j a_{jk} + z^3 \sum_{j,k,l} b_j a_{jk} a_{kl} + \cdots$$

where the degree of $R(z)$ is less than or equal to s. Since the exact solution of the test problem (1.9) is $\exp(z)$, $y_1 = R(z)y_0$ must satisfy

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

to the order of p for an order p method. This proves the following theorem.

Theorem 1.3 *[26] If an explicit Runge-Kutta method is of order p, then*

$$R(z) = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^p}{p!} + O(z^{p+1}).$$

Therefore, all explicit Runge-Kutta methods with $p = s$ have the stability function

$$R(z) = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^s}{s!}. \tag{1.12}$$

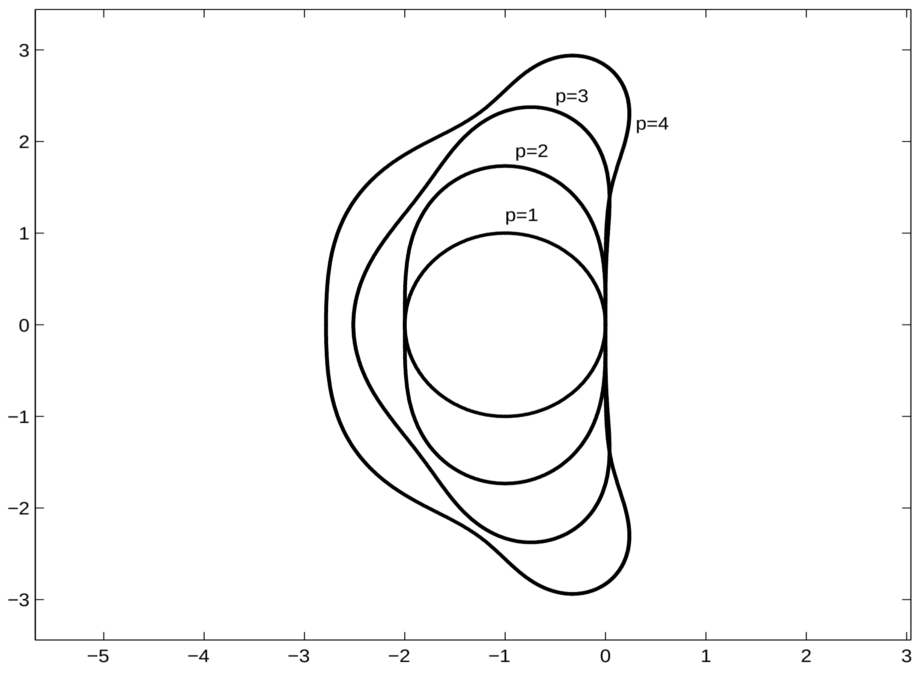

Figure 1.3: Stability regions of explicit Runge-Kutta methods.

Figure 1.3 shows the stability regions of order $p = 1$, 2, 3, and 4 explicit Runge-Kutta methods. It can be seen that explicit Runge-Kutta methods corresponding to (1.12) have bounded stability regions.

For an s-stage implicit Runge-Kutta method, we have the following theorem.

Theorem 1.4 *[26] The stability function of an implicit Runge-Kutta method is given by*

$$R(z) = \frac{\det(I - zA + zeb^T)}{\det(I - zA)}.$$

Proof. Rewrite the equations (1.10) and (1.11) of Runge-Kutta method as a linear system

$$\begin{pmatrix} I - zA & 0 \\ -zb^T & 1 \end{pmatrix} \begin{pmatrix} Y \\ y_n \end{pmatrix} = y_{n-1} \begin{pmatrix} e \\ 1 \end{pmatrix}.$$

By applying Cramer's rule and simplifying, we obtain

$$R(z) = \frac{\det\left(I + z(eb^T - A)\right)}{\det(I - zA)}.$$

For the implicit Runge-Kutta methods, we write $R(z)$ as a rational function with the degree of the numerator and denominator less than or equal to s. It has the form

$$R(z) = \frac{P(z)}{Q(z)}.$$

For an order p implicit Runge-Kutta method we have

$$\exp(z) - R(z) = Cz^{p+1} + O(z^{p+2})$$

where constant C is called the error constant.

In Figure 1.3 we can see that the stability regions of explicit Runge-Kutta methods are bounded. Very often when approximating the solutions of some special problems such as stiff problems a larger stability region is required to ensure an efficient computation. Therefore, a special property of some Runge-Kutta methods is introduced. The following definition of A-stability characterizes the stability of certain Runge-Kutta methods.

Definition 1.5 *A Runge-Kutta method is said to be "A-stable" if its stability region contains the non-positive half plane ($Re(z) \leq 0$).*

A-stability allows a Runge-Kutta method to have an unbounded stability region. This is a great advantage for some problems with a large negative value of z $(= hL)$, especially for stiff problems. A related concept is L-stability. L-stability was first studied by Ehle [18] and is defined as follows.

Definition 1.6 *A method is called L-stable if it is A-stable and the stability function satisfies*

$$lim_{|z|\to\infty}|R(z)| = 0.$$

Runge-Kutta methods have been widely studied. There are some advantages and disadvantages of Runge-Kutta methods. One disadvantage of using implicit Runge-Kutta methods lies in its high implementation costs as we need to use Newton's method to calculate the stage values at each step. It is also expensive to estimate the local error. Advantages of Runge-Kutta methods include higher accuracy and ease of changing stepsize and order. Furthermore, "A-stable" Runge -Kutta methods with any order can be found. This property of "A-stability" is required for solving stiff ordinary differential equations which we will discuss later.

1.3 Linear multistep methods

The second important generalization of the Euler method is the use of linear multistep methods. In 1883, Bashforth and Adams [1] proposed the idea of extending the Euler method by using several previous solutions and derivative values in computing the updated solution. The general form of a linear k-step method for an ODE $y' = f(y)$ is

$$\begin{aligned} y_n &= \alpha_1 y_{n-1} + \alpha_2 y_{n-2} + \cdots + \alpha_k y_{n-k} \\ &\quad + h[\beta_0 f(y_n) + \beta_1 f(y_{n-1}) + \cdots + \beta_k f(y_{n-k})], \end{aligned} \tag{1.13}$$

where y_n is the numerical approximation to the exact solution at the point x_n and α_1, α_2, …, α_k, β_0, β_1, …, β_k are fixed numbers. The values of α_1, α_2, …, α_k, β_0, β_1, …, β_k are chosen to obtain the highest possible order and characterize a method. If $\alpha_1 = 1$ and $\alpha_i = 0$, $i = 2, \ldots, k$ and $\beta_0 = 0$, the methods are known as the Adams-Bashforth methods. If β_0 is not 0, the methods are known as Adams-Moulton methods [31]. If $\beta_0 \neq 0$ and $\beta_i = 0$, $i = 1, \cdots, k$, the methods are known as Backward-Difference Formulae (BDF). Here are some examples of linear multistep methods.

- Adams-Bashforth methods (order 2)

  $y_n = y_{n-1} + h\left(\frac{3}{2}f(y_{n-1}) - \frac{1}{2}f(y_{n-2})\right)$

- Adams-Moulton methods (order 2)

  $y_n = y_{n-1} + \frac{h}{2}\left(f(y_n) + f(y_{n-1})\right)$

- Backward-Difference Formulae (order 3)

  $y_n = \frac{18}{11}y_{n-1} - \frac{9}{11}y_{n-2} + \frac{2}{11}y_{n-3} + h\frac{6}{11}f(y_n)$

1.3.1 Consistency, stability and convergence

If a method is used to solve an ordinary differential equation at the point x_{n-1}, and the numerical solution tends to the exact solution as the stepsize h tends to zero at point x_n, which is $x_{n-1} + h$, the method is said to be consistent. This condition guarantees the local accuracy of the numerical method. However this condition does not guarantee convergence since the difference between the numerical approximation and exact solution, the local truncation error (say at x_n), may accumulate as n increases and overflow. Therefore the method must satisfy the stability conditions.

To obtain consistency conditions a pair of polynomials have been introduced [17]. The polynomials are defined as follows

$$\alpha(z) = 1 - \alpha_1 z - \alpha_2 z^2 - \cdots - \alpha_k z^k, \tag{1.14}$$

$$\beta(z) = \beta_0 + \beta_1 z + \beta_2 z^2 + \cdots + \beta_k z^k. \tag{1.15}$$

Using a linear multistep method to solve the simple differential equation $y'(x) = 0$ with exact solution $y(x) = 1$, we have

$$\alpha_1 + \alpha_2 + \cdots + \alpha_k = 1,$$

which is equivalent to $\alpha(1) = 0$. This property is named “pre-consistency”.

We then apply a pre-consistent method to another trivial differential equation $y'(x) = 1$ with $y(0) = 0$. For step number n, it is found that

$$n\left(1 - \sum_{i=1}^{k} \alpha_i\right) = \sum_{i=0}^{k} \beta_i - \sum_{i=1}^{k} i\alpha_i.$$

As the method is pre-consistent the left hand side equals to 0, and the right hand side can be written as $\beta(1) - \alpha'(1)$. This leads to the following definition.

Definition 1.7 *A linear multistep method is consistent if the coefficients of the* *method satisfy*

$$\sum_{i=1}^{k} \alpha_i = 1, \tag{1.16}$$

$$\sum_{i=0}^{k} \beta_i = \sum_{i=1}^{k} i\alpha_i. \tag{1.17}$$

The stability condition of a linear multistep method is given by the following definition.

Definition 1.8 *[5] A linear multistep method is stable if all solutions to the difference equation*

$$y_n = \alpha_1 y_{n-1} + \alpha_2 y_{n-2} + \cdots + \alpha_k y_{n-k}$$

are bounded as $n \to \infty$.

The following theorem characterizes convergence of a linear multistep method.

Theorem 1.9 *[16] A linear multistep method is convergent if and only if it is stable and consistent.*

As for Runge-Kutta methods, linear multistep methods have both advantages and disadvantages. Linear multistep methods have low implementation costs because only one function evaluation per step is needed, and the error estimation also has a low cost. Disadvantages include that changing stepsize is a complicated procedure and variable order implementation can be inefficient because only methods of low orders (< 3) can be A-stable [17].

1.4 General linear methods

The two classes of traditional numerical methods for ordinary differential equations have been reviewed separately. Both of these classes have well known advantages and disadvantages. The linear multistep methods have advantages in terms of computational cost, but difficult questions are associated with the stability of the approximations and obtaining a high degree of accuracy. The multistage methods (such as Runge-Kutta methods) have high computational costs and complicated accuracy questions, but they have good stability. To combine the multivalue nature of linear multistep methods and the multistage nature of the Runge-Kutta methods, many generalizations can be found. For example, Hybrid methods are generalized multistep methods allowing some evaluations of f in each step [21]. Rosenbrock type methods are generalized Runge-Kutta methods that linearize the nonlinear systems of the implicit Runge-Kutta methods in order to reduce the computational cost. General linear methods were introduced by Butcher [6] as a unifying framework for the traditional methods. General linear methods combine many essential features of linear multistep methods and multistage methods. In this section we give an introduction.

The formulation of general linear methods used in this thesis follows Burrage and Butcher [2]. They used a partitioned $(s+r)\times(s+r)$ matrix to represent a general linear method. This matrix contains four matrices, A, B, U and V, and has the form

$$\left[\begin{array}{c|c} A_{s\times s} & U_{s\times r} \\ \hline B_{r\times s} & V_{r\times r} \end{array}\right].$$

The coefficients of these matrices indicate the relationships among various numerical quantities that arise in the computation. The structure of the leading coefficient matrix A determines the implementation costs of these methods. For an N-dimensional differential equation system (1.2), a general linear method can be defined by the following equations

$$\begin{aligned} Y_i &= \sum_{j=1}^{s} a_{ij} h f(Y_j) + \sum_{j=1}^{r} u_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \cdots, s, \\ y_i^{[n]} &= \sum_{j=1}^{s} b_{ij} h f(Y_j) + \sum_{j=1}^{r} v_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \cdots, r, \end{aligned}$$

where Y_i is the internal stage value as in a Runge-Kutta method, $f(Y_i)$ is the corresponding stage derivative, $y_i^{[n-1]}$ is the incoming approximation, and $y_i^{[n]}$ is the outgoing approximation in step number n.

For convenience, the above equations can be represented in the following matrix form,

$$\left[\begin{array}{c} Y_1 \\ \vdots \\ Y_s \\ \hline y_1^{[n]} \\ \vdots \\ y_r^{[n]} \end{array}\right] = \left[\begin{array}{c|c} A \otimes I & U \otimes I \\ \hline B \otimes I & V \otimes I \end{array}\right] \left[\begin{array}{c} hF_1 \\ \vdots \\ hF_s \\ \hline y_1^{[n-1]} \\ \vdots \\ y_r^{[n-1]} \end{array}\right],$$

or

$$Y = (A \otimes I) hF + (U \otimes I) y^{[n-1]}, \tag{1.18}$$

$$y^{[n]} = (B \otimes I) hF + (V \otimes I) y^{[n-1]}, \tag{1.19}$$

where $Y = [Y_1, Y_2, \cdots, Y_s]^T$, $F = [F_1, F_2, \cdots, F_s]^T$, $F_i = f(Y_i)$ for $i = 1, \ldots, s$, $y^{[n-1]} = \left[y_1^{[n-1]}, y_2^{[n-1]}, \cdots, y_r^{[n-1]}\right]^T$ and $y^{[n]} = \left[y_1^{[n]}, y_2^{[n]}, \cdots, y_r^{[n]}\right]^T$. Note tha in equation (1.18), (1.19) and through out the thesis, we have used a simplified notation. For example, AhF represents $A \otimes hF$.

We use four integers to characterize a general linear method. They are p, q, r and s.

- s: the number of stages as in Runge-Kutta methods.
- r: the number of quantities passed between steps. r and s are used to measure the complexity of the method.
- p: the order of the method. It is a measure of the accuracy of the method. It means that the method approximates the solution to the order of p. If $y(\bar{x}_n)$ denotes the exact solution, then we have
$$y(\bar{x}_n) = y(x_n) + O(h^{p+1}).$$
- q: the stage order. This is a measure of the accuracy of the intermediate stages. It means that stage value Y_i approximates the solution at $x_{n-1}+c_ih$ to the order of q where $c = [c_1, c_2, \cdots, c_s]^T$ is the abscissae vector. That is
$$Y_i = y(x_{n-1} + c_ih) + O(h^{q+1}).$$

In this thesis, we are looking for the methods that are easy to implement and interpret. Like many high order Runge-Kutta methods, we consider methods with $q = p$. One of the advantages is that it is easy to obtain a local error estimate with $q = p$. This assumption also helps to avoid order reduction [33]. In Chapter 2 we will discuss how to find practical general linear methods.

1.4.1 Consistency, stability and convergence

We first apply a general linear method to the trivial differential equation $y'(x) = 0$ with initial condition $y(x) = y_0$. We know that the solution should be exact at both the beginning and end of each step. This suggests the following definition.

Definition 1.10 *[7] A general linear method is "pre-consistent" if there exists a vector u such that*

$$\begin{aligned} Uu &= e, \\ Vu &= u, \end{aligned}$$

where vector u is the "pre-consistency vector".

Secondly, we apply the general linear method to another differential equation $y'(x) = 1$ with solution $y = x - x_0$. Again the numerical solution should be exact at both the beginning and end of each step. This gives the consistency definition.

Definition 1.11 *[7] A general linear method is "consistent" if it is pre-consistent with pre-consistency vector u and there exists a vector v such that*

$$Be + Vv = u + v.$$

As for linear multistep methods, a concept of stability is necessary. We define the stability of a matrix.

Definition 1.12 *The matrix V is stable if there exists a constant T such that $||V^n||_\infty \leq T$ for all $n = 1, 2, \ldots$.*

When we apply a general linear method to the problem $y' = 0$, we want the solution to be bounded. We have the output approximations

$$y^{[n]} = Vy^{[n-1]} = V^n y^{[0]}.$$

This leads to the following definition.

Definition 1.13 *A general linear method is stable if V is a stable matrix (power-bounded).*

The convergence we discuss here means the ability of a method to represent the exact solution of a differential equation as the number of steps tends to infinity. We have the following definition.

Definition 1.14 *A general linear method is "convergent" if for any initial value problem*

$$y'(x) = f(y(x)), \quad y(x_0) = y_0,$$

subject to the Lipschitz condition $||f(y) - f(z)|| \leq L||y - z||$, there exists a non-zero vector u, such that if $y_i^{[0]} = u_i y(x_0) + O(h)$, with $i = 1, 2, \ldots, r$, then $y_i^{[n]} = u_i y(x_0 + nh) + O(h)$, for all n subject to bounded nh.

This condition guarantees that we can obtain accurate numerical solutions at any fixed point by taking h sufficiently small, for given sufficiently accurate initial approximations.

Stability and consistency are together necessary and sufficient for convergence. This result can be expressed by the following theorem.

Theorem 1.15 *[16] A general linear method is convergent only if it is stable and consistent.*

As we know, Runge-Kutta methods can achieve an arbitrarily high order and still be A-stable. We would like to obtain general linear methods with the same stability regions as the Runge-Kutta methods. In this thesis, we will use a class of general linear methods for a special group of problems, stiff problems. This new type of general linear method is characterized by the property known as "inherent Runge-Kutta stability" (IRKS). We will discuss this "IRKS" in Chapter 2 in detail.

1.4.2 Order of accuracy

For a general linear method, there are r quantities passed between steps. A starting procedure is required to obtain the incoming approximation, $y^{[0]}$, from the given initial value $y(x_0)$. We write the internal stage values of the starting method as $\bar{Y}_1$, $\bar{Y}_2$, …, $\bar{Y}_{\bar{s}}$ and the derivatives of these stages as $f(\bar{Y}_1)$, $f(\bar{Y}_2)$, …, $f(\bar{Y}_{\bar{s}})$ where $\bar{s}$ is the number of stages for the starting method. This $\bar{s}$ may or may not be equal to s, the number of stages of the method. A starting method can be represented by the following partitioned $(\bar{s}+r)\times(\bar{s}+\bar{r})$ matrix

$$S = \left[\begin{array}{c|c} S_{11} & S_{12} \\ \hline S_{21} & S_{22} \end{array}\right],$$

where r is the number of approximations which are required for $y^{[0]}$ and $\bar{r}$ is the number of the given initial conditions.

The starting method is given by

$$\bar{Y} = hS_{11}\bar{F} + S_{12}y_0, \tag{1.20}$$

$$y^{[0]} = hS_{21}\bar{F} + S_{22}y_0, \tag{1.21}$$

where

$$\bar{Y} = \begin{bmatrix} \bar{Y}_1 \\ \bar{Y}_2 \\ \vdots \\ \bar{Y}_{\bar{s}} \end{bmatrix}, \quad \bar{F} = \begin{bmatrix} f(\bar{Y}_1) \\ f(\bar{Y}_2) \\ \vdots \\ f(\bar{Y}_{\bar{s}}) \end{bmatrix}. \tag{1.22}$$

To have the pre-consistency condition, we choose $S_{22} = u$ and $S_{12} = \bar{e}$.

Denote S as the starting method, M as the method and E as the exact solution from x_0 to x_1. A method starting with method S and completing with a step of method M can be expressed by MS. Order of accuracy can now be defined relative to S as the greatest p for which $(MS)y_0 - (SE)y_0 = O(h^{p+1})$. Furthermore, $(MS)y_0 - (SE)y_0$ is considered as the local error. This procedure is illustrated in Figure 1.4.

Generally speaking, if the general linear method in this thesis has order of p, then each of the r quantities of $y^{[0]}$ is accurate to order p.

The local truncation error can be worked out using Figure 1.4. With this type of local truncation error, we can present the global error approximation by repeating the diagram as many times as the method takes steps. Figure 1.5 shows the global error estimate.

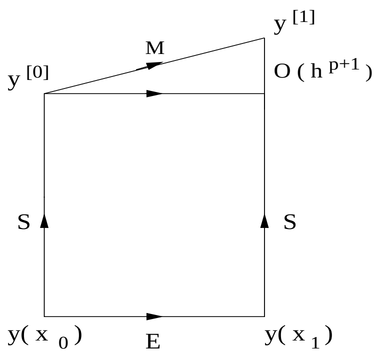

Figure 1.4: Order of accuracy.

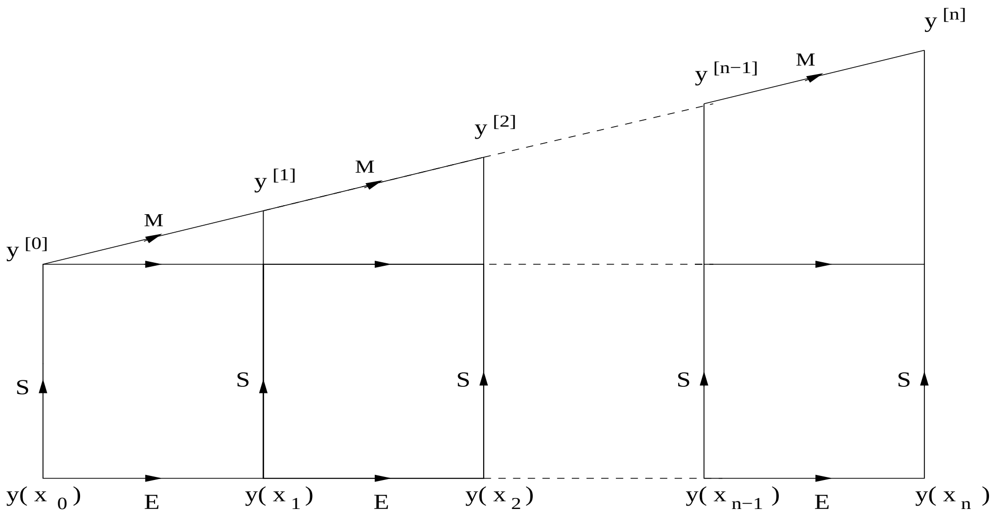

Figure 1.5: Global error.

In order to recover the final answer, a finishing method is needed to undo the work of starting method S. This finishing method is denoted by F. If we apply F to $y^{[n]}$, then we should get $y(x_n) + O(h^{p+1})$ as long as nh is bounded. For the methods in this thesis, however, one component of $y^{[n]}$ directly approximates the solution at x_n. In this case, we can save the cost of using a finishing method.

General linear methods make up a class of numerical methods for ODEs. In attempting to find some practical new methods from this large class of methods, certain assumptions have been made to fit our requirements. In Chapter 2 we will discuss how to construct this new type of general linear method.

1.5 Stiff problems

Among ordinary differential equations, there is a special group of problems called stiff problems which require some special methods to solve them efficiently. One may ask, what is a stiff problem? In this section, we give the definition of stiffness and some examples of stiff problems. Furthermore we will discuss the difficulties involved in solving stiff problems.

It is well known that many real problems can be formulated using ordinary differential equations. Some of these problems contain both slowly and rapidly decaying transients in the solution. Therefore the corresponding variables in the ODE system also vary at widely different rates. In order to obtain the 'true' solution one must use a sufficiently small stepsize for the fast decaying variable. This group of problems is called "stiff" while other problems are referred to as "non-stiff".

we can look at a simple example which is given by Gear [22],

$$y'(x) = L(y(x) - f(x)) + f'(x) \quad , L \ll 0 \tag{1.23}$$

where $f(x)$ is a smooth, slowly varying function. For this ODE, we have the solution

$$y = (y(x_0) - f(x_0))e^{Lx} + f(x). \tag{1.24}$$

Note that the two components in the solution, for large x, $(y(x_0) - f(x_0))e^{Lx}$ will be insignificant comparing to $f(x)$. Therefore, this example is considered as a stiff problem.

To understand the stiff problems better, we Consider the following linear problem

$$y'(x) = My(x), \tag{1.25}$$

where M is an $N \times N$ matrix which has distinct eigenvalues λ_j $(j = 1, \ldots, N)$. The solution of the above equation is of the form

$$y(x) = [v_1, \cdots, v_s][C_1 e^{(\lambda_1 x)}, \cdots, C_N e^{(\lambda_N x)}]^T, \tag{1.26}$$

where the v_i are the eigenvectors and the C_i are constants which depend on the initial conditions. If $Re(\lambda_j) < 0$ for $j = 1, \cdots, N$, then $C_i e^{(\lambda_i x)} \to 0$ when $x \to \infty$. Consider the case where these eigenvalues are very different, for example, one of the eigenvalues has a very negative real part relative to the others. This means that

$$R = \frac{\text{Max}|Re\lambda_j|}{\text{Min}|Re\lambda_j|} \gg 1, \quad Re(\lambda_j) < 0, \quad j = 1, \cdots, N. \tag{1.27}$$

Denote $\text{Max}|Re\lambda_j|$ by λ_0. Hence the exponential function $\exp(\lambda_0 x)$ decays to zero much more rapidly than that based on the eigenvalue, $\text{Min}|Re\lambda_j|$. Such behaviour leads to great difficulties in obtaining an accurate numerical solution. The property which makes numerical computation complicated and difficult is referred to as "stiffness". The ratio R is called the "stiffness ratio" which provides a measure of stiffness.

The above discussion is based on the linear system, it is useful to generalize the stiffness ratio for nonlinear systems. For nonlinear systems the Jacobian is used to determine the stiffness. If the eigenvalues of Jacobian $J = \frac{\partial f}{\partial y}$ satisfy the same condition as in (1.27), then the problem is a stiff problem. However, even for linear systems, sometimes the stiffness ratio is not a good measure of stiffness. For example, if only one of the eigenvalues was non-zero one would not be able to calculate the stiffness ratio. As a result, we cannot tell whether or not the problem is stiff even if the negative part of the non-zero eigenvalue is reasonably big. To be more general, there is a generic definition which applies to both linear and nonlinear problems.

Definition 1.16 *If a numerical method is forced to use, in an interval of integration, a stepsize is forced to be excessively small relative to dominant time-scale of the solution to get a smooth approximation of the exact solution in that interval, then the problem is said to be stiff in that interval.*

A paper by Garfinkel and Marbach [20] gives a good review of stiff differential equations. There are many well known stiff problems in the literature. An example is Robertson's problem. Robertson's problem is a three dimensional chemical reaction system,

$$\begin{aligned} A &\xrightarrow{0.04} B \\ B+B &\xrightarrow{3\times10^7} C+B \\ B+C &\xrightarrow{10^4} A+C \end{aligned}$$

where A, B and C are three species. In mathematics we use y_1, y_2 and y_3 to denote the concentrations of A, B and C, respectively. This chemical process can be represented by the following differential equations [34],

$$\begin{aligned} y_1' &= -0.04y_1 + 10^4 y_2 y_3, \\ y_2' &= 0.04y_1 - 10^4 y_2 y_3 - 3\times10^7 y_2^2, \\ y_3' &= 3\times10^7 y_2^2, \end{aligned}$$

with initial values,

$$\begin{aligned} y_1(0) &= 1, \\ y_2(0) &= 0, \\ y_3(0) &= 0. \end{aligned}$$

Since this problem is a nonlinear problem, we look at the Jacobian of the problem. The Jacobian is

$$J(y) = \begin{pmatrix} -0.04 & 10^4 y_3 & 10^4 y_2 \\ 0.04 & -10^4 y_3 - 6\times10^7 y_2 & -10^4 y_2 \\ 0 & 6\times10^7 y_2 & 0 \end{pmatrix}.$$

When $x = 0$, the Jacobian has eigenvalues of -0.04, 0 and 0, and hence stiffness does not occur. It has been observed that the second component, y_2, reaches $y_2 \approx 3.65 \times 10^{-5}$ in a very short time, and then goes back very slowly to zero again. A very short period of time is represented by a very small stepsize in a numerical solution. At point $(y_1 = 1, y_2 = 3.65 \times 10^{-5}, y_3 = 0)$ the eigenvalues become 0, -0.0405, and -2189.6. We can see that the stiffness ratio $R \gg 1$. This makes the system of equations stiff. In Chapter 4, we will use this system to test our new methods designed for solving stiff problems.

1.6 Solving stiff problems

According to Definition 1.16, in order to get a smooth approximation of the solution we need to use very small stepsize for stiff problems. In practice we want to use large stepsizes to reduce computational costs. To understand why stiff problems are difficult to solve and how to solve them, we use the following example as an illustration.

Example 1.1 *Consider the linear problem*

$$\begin{aligned} y_1' &= -8y_1 + 7y_2, \\ y_2' &= 42y_1 - 43y_2 \end{aligned}$$

with

$$\begin{aligned} y_1(0) &= 1 \\ y_2(0) &= 8. \end{aligned}$$

The coefficient matrix has two eigenvalues, $\lambda_1 = -1$ and $\lambda_2 = -50$. The stiffness ratio $R = 50$. This is a mildly stiff problem. The exact solution of the problem is

$$\begin{aligned} y_1 &= 2\exp(-x) - \exp(-50x), \\ y_2 &= 2\exp(-x) + 6\exp(-50x). \end{aligned}$$

If we use an order two explicit Runge-Kutta method to solve this problem, the stepsize needs to satisfy the following condition

$$|h\lambda_{max}| \leq 2$$

for the method to be stable. The stability region requires $h \leq 0.04$. *If we choose* $h = 0.04$, *when* $x = 0.4$, *the fast component* $\exp(-50 * 0.4) = \exp(-20)$ *contributes almost nothing to the solution compared with the slow component* $\exp(-1 * 0.4) = \exp(-0.4)$. *On the other hand, we need 100 steps to reach* $x = 4$ *with stepsize* $h = 0.04$. *This means that on the one hand a bigger stepsize is needed for efficient calculation, but on the other hand, a smaller stepsize is needed to ensure stability. The example shows that to solve stiff problems the stability of a good method should impose no limitation on the stepsize, and hence it requires a large stability region.*

As discussed in the previous sections, an 'A-stable' method has the stability region which contains the non-positive half plane. As a result, 'A-stability' is a desirable property when solving stiff problems. To illustrate this, consider the following implicit Euler method,

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

Applying it to the linear problem

$$y' = Ly,$$

where $L < 0$ is a constant. We get

$$y_{n+1} = \frac{y_n}{1 - Lh}.$$

We find that as $h \to \infty$, $y_{n+1} \to 0$, which means the method is absolutely stable for any $z = Lh$ in the left half plane. (Fig 1.2). It has been proved that A-stable methods must be implicit methods by Dahlquist [17].

In this thesis we only consider the implicit methods since we are concerned only with stiff problems. Implicit Runge-Kutta methods (multistage methods) and BDF methods (multistep methods) have been widely used for solving stiff problems. A new type of general linear method consisting of implicit methods designed for solving stiff problems will be introduced in Chapter 2.

1.7 The cost of using implicit methods

For an implicit method, when we calculate the stage values, we need to solve the system of equations,

$$Y_i \;=\; \psi + \Psi(Y_1, Y_2, \cdots, Y_s), \qquad i = 1, 2, \ldots, s,$$

where Ψ is a function depending on the method and ψ presents some known values. To illustrate, we consider an implicit Runge-Kutta method to solve an N dimensional differential equation system. When we calculate the stage values, we need to solve the following equation system,

$$Y_i \;=\; y_{n-1} + \sum_{j=1}^{s} a_{ij} h f(Y_j) \qquad i = 1, 2, \ldots, s. \tag{1.28}$$

Using the Newton method we need to go through the following steps to get the stage values:

- give a 'predicted' value of the stage value, Y_i,

- calculate a sequence of iterations $Y_i^{[k+1]}$, which approximates Y_i using the formula
$$Y_i^{[k+1]} = Y_i^{[k]} - e_i^{[k]}$$
with
$$(I_s \otimes I_N - hA \otimes J)e_i^{[k]} = \phi_i,$$
where J is the Jacobian matrix with the form of
$$J = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_N} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial y_1} & \frac{\partial f_N}{\partial y_2} & \cdots & \frac{\partial f_N}{\partial y_N} \end{bmatrix},$$
and
$$\phi_i = Y_i - y_{n-1} - \sum_{j=1}^{s} a_{ij} h f(Y_j),$$
- test convergence: when to stop the Newton iteration.

From these three steps we can see that the cost of the stage evaluation comes from the following calculations,

- evaluation of Jacobian,
- LU factorization for the matrix $I_s \otimes I_N - hA \otimes J$,
- solving linear systems, $LUe_i^{[k]} = \phi_i$.

The iteration matrix $I_s \otimes I_N - hA \otimes J$ is very complicated, especially when the matrix A is a full matrix and s and N are large. For an s stage method on an N dimensional problem we have sN unknowns, therefore the cost of LU factorization requires s^3N^3 operations and the back substitution for solving the linear system is s^2N^2 operations. The total cost would be $C_1(s^3N^3) + C_2(s^2N^2)$ operations where C_1 and C_2 are constants. To lower the computational cost in the evaluation of the stage values, we consider methods which have a lower triangular structure with equal diagonal elements. With this special structure of matrix A, the total cost is reduced to $C_3(N^3s) + C_4(N^2s)$ operations where C_3 and C_4 are constants. Another possibility is to keep the Jacobian matrix and LU factorization unchanged over several iterations, several stages and even several steps. We will discuss the technical details in Chapter 3.

Chapter 2

A new type of general linear method for stiff problems

General linear methods were originally proposed by Butcher [6] in 1966 as a framework for studying stability, consistency and convergence for the traditional methods. However, very few methods which are significantly different from traditional methods have been developed in this group. As we mentioned in the introduction, the two traditional classes of methods for stiff ordinary differential equations are implicit Runge-Kutta methods and backward differentiation formulas (BDF) methods. They both have advantages and disadvantages. To take the advantages and overcome the disadvantages, a new class of general linear methods for stiff problem has been discovered recently [13]. These methods are not only multistep, but also multistage methods. In this chapter we first review some known general linear methods in Section 1, and then construct this new group of general linear methods for stiff problems in subsequent sections. These new methods have a special property known as the Inherent Runge-Kutta stability.

2.1 DIMSIMs

Recall that general linear methods are characterized by four matrices A, U, B and V. As for Runge-Kutta methods, the structure of the leading coefficient matrix A determines the implementation cost of these methods. Therefore the phrases such as 'diagonally-implicit' and 'singly-implicit' can be directly borrowed from the Runge-Kutta terminology. To lower the cost of implementation, the matrix A is restricted to being of lower triangular form as for an explicit or a diagonally implicit Runge-Kutta method. If the matrix A is lower triangular with its diagonal elements all equal, then we can calculate Y_1, Y_2, ..., Y_s separately and sequentially by a modified Newton iteration scheme. Furthermore, we can use the same matrix of partial derivatives for every stage. In 1993 Butcher [8] introduced a class of methods known as Diagonally Implicit Multistage Integration Methods (DIMSIMs) which have considerable potential for efficient implementation. The aim of these methods is to overcome some of the disadvantages of Runge-Kutta methods, linear multistep methods and other successful known methods. These disadvantages include the high implementation costs for implicit Runge-Kutta methods and the lack of high order A-stable linear multistep methods. A further advantage of DIMSIMs is that parallelism is available and achievable.

Due to various considerations, such as the need for efficient implementation and easy error estimation, certain restrictions on the four matrices are required. These methods are characterized by the following requirements:

- The matrix A has lower triangular form with all diagonal elements equal to a constant value so as to lower implementation costs.

- The stage order should be close or equal to the order of the method to avoid order reduction.
- The matrix V should be power bounded for convergence. A stronger requirement is that V has only one non zero eigenvalue which guarantees zero stability (that is, stability in the sense of Dahlquist [16]).
- Low cost of implementation, effective local error estimation and high order interpolatory output should be available.
- The incoming and outgoing quantities should be related to the exact solution by weighted Taylor series.
- Modified methods should exist for changing stepsizes.

A method from the DIMSIMs group can then be defined by

$$\begin{aligned} Y_i &= \sum_{j=1}^{s} a_{ij} h f(Y_j) + \sum_{j=1}^{r} u_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \dots, s, \\ y_i^{[n]} &= \sum_{j=1}^{s} b_{ij} h f(Y_j) + \sum_{j=1}^{r} v_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \dots, r, \end{aligned}$$

where

$$Y_i = y(x_{n-1} + c_i h) + O(h^{q+1}),$$

and the incoming quantities and outgoing quantities are expressed by the weighted Taylor series,

$$\begin{aligned} y_i^{[n-1]} &= \sum_{k=0}^{p} \alpha_{ik} y^{(k)}(x_{n-1}) h^k + O(h^{p+1}), \\ y_i^{[n]} &= \sum_{k=0}^{p} \alpha_{ik} y^{(k)}(x_n) h^k + O(h^{p+1}). \end{aligned}$$

According to the structure of the leading matrix A, DIMSIM methods can be divided into the following four sub-families known as types:

- Type 1. Matrix A is required to be of lower triangular form and to have zero on its diagonal. These methods are intended for nonstiff problems on a sequential computer.
- Type 2. Matrix A is assumed to be of lower triangular form with a constant on the diagonal. These methods are for stiff problems on a sequential computer.
- Type 3. Matrix A is required to be the zero matrix. These methods are for nonstiff problems in a parallel environment.
- Type 4. Matrix A is required to be a diagonal matrix. These methods are for stiff problems on a parallel computer.

The four types of DIMSIMs are shown in Table 2.1.

Several derivations of DIMSIMs have been presented in papers, [8] and [11]. Furthermore, Butcher and Jackiewicz have discussed some implementation questions of these methods for non-stiff problems in their paper [12]. Singh [38] has implemented parallel DIMSIMs in her PhD thesis. It has been found that these DIMSIM methods perform well for a wide range of problems [10] [39].

Type	Structure of A	Problem	Computation
type 1	$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ a_{s1} & a_{s2} & \cdots & 0 \end{bmatrix}$	nonstiff	Sequential
type 2	$\begin{bmatrix} \lambda & 0 & \cdots & 0 \\ a_{21} & \lambda & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ a_{s1} & a_{s2} & \cdots & \lambda \end{bmatrix}$	stiff	Sequential
type 3	$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$	nonstiff	Parallel
type 4	$\begin{bmatrix} \lambda & 0 & \cdots & 0 \\ 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \lambda \end{bmatrix}$	stiff	Parallel

Table 2.1: Four types of DIMSIMs

2.2 A new type of general linear method

Among these four types of DIMSIMs, we are mostly interested in type 2 methods since we are working on stiff problems. In this Chapter, we introduce a new type of general linear method which has the structure of DIMSIM type 2 and possesses a special property called Inherent Runge-Kutta stability. This means the stability region is exactly the same as for a Runge-Kutta method. This group of general linear methods is designed to solve stiff differential equations with efficient implementation.

2.2.1 Choices of coefficients

As we introduced in Chapter 1, p and q are the order and stage order of a general linear method respectively. High stage order is an important property that we need to consider. One advantage of high stage order is that it is easier to obtain a local error estimate. High stage order also helps to avoid order reduction. In this thesis, we consider methods with $q = p$. Therefore, the internal stage values satisfy

$$Y_i = y(x_{n-1} + c_i h) + O(h^{p+1}).$$

The other two integers, s and r, are the number of internal stages and the number of incoming and outgoing approximations respectively. We specifically consider methods for which $r = s = p + 1$. In this case, the quantities passed from step to step have the form

$$y_i^{[n-1]} = \sum_{k=0}^{p} \alpha_{ik} y^{(k)}(x_{n-1}) h^k + O(h^{p+1}),$$

$$y_i^{[n]} = \sum_{k=0}^{p} \alpha_{ik} y^{(k)}(x_n) h^k + O(h^{p+1}).$$

The choice of $s = p + 1$ makes it possible to obtain a balance between the stability requirements and acceptable error constants. The advantage of the choice of $r = p + 1$ is that we can interpret the method in such a way that the quantities passed between steps represent Nordsieck vectors. Nordsieck vectors were introduced by Nordsieck in 1962 [32] for ease of changing the stepsize in Adams methods. Later Gear [23] promoted and used the Nordsieck vector in the code DIFSUB. The same technique can be applied to this new type of general linear method [9].

A Nordsieck vector has the form

$$\tilde{y}^{[n]} = \begin{bmatrix} y(x_n) \\ hy'(x_n) \\ \vdots \\ h^p y^{(p)}(x_n) \end{bmatrix}.$$

Using a transformation matrix T, we have

$$\begin{aligned} y^{[n-1]} &= T\tilde{y}^{[n-1]} \\ y^{[n]} &= T\tilde{y}^{[n]} \end{aligned}$$

where

$$T = \begin{bmatrix} \alpha_{10} & \alpha_{11} & \cdots & \alpha_{1p} \\ \alpha_{20} & \alpha_{21} & \cdots & \alpha_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{p0} & \alpha_{p1} & \cdots & \alpha_{pp} \end{bmatrix}.$$

The method becomes

$$\begin{aligned} Y &= AhF + UT\tilde{y}^{[n-1]}, \\ T\tilde{y}^{[n]} &= BhF + VT\tilde{y}^{[n-1]}. \end{aligned}$$

This leads to

$$\begin{aligned} Y &= AhF + \tilde{U}\tilde{y}^{[n-1]}, \\ \tilde{y}^{[n]} &= \tilde{B}hF + \tilde{V}\tilde{y}^{[n-1]}, \end{aligned}$$

where $\tilde{U} = UT$, $\tilde{B} = T^{-1}B$ and $\tilde{V} = T^{-1}VT$. This transformation matrix will bring the method into the Nordsieck form. Without loss of any generality we use the Nordsieck form directly in our methods and still use B, U and V to present our methods from now on.

To find a suitable method, the following simplifying assumptions are required:

- Since our aim is to find methods for stiff problems, we restrict A to being a lower triangular form as for a diagonally implicit Runge-Kutta method to lower the cost of implementation. The diagonal elements of A are all equal to a positive constant λ as for the DIMSIMs type 2. This type of method can be used in a sequential environment for solving stiff problems. For these methods, matrix A has the form

$$A = \begin{bmatrix} \lambda & 0 & \cdots & 0 \\ a_{21} & \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \cdots & \lambda \end{bmatrix}.$$

- In Chapter 1, we showed that if matrix V is a stable matrix then we have a stable general linear method. Furthermore, if matrix V is of rank one then zero stability is guaranteed. To satisfy these requirements we choose the matrix, V, to have a simple structure. Let the first column of V equal the basis vector e_1, that is $Ve_1 = e_1$. Furthermore, matrix V is also required to have only one non-zero eigenvalue, which means the matrix $\tilde{V}$, formed by deleting the first row and first column of the matrix V, has all the eigenvalues equal to zero. That is

$$V = \begin{bmatrix} 1 & v_{12} & \cdots & v_{1r} \\ 0 & v_{22} & \cdots & v_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_{r2} & \cdots & v_{rr} \end{bmatrix} = \begin{bmatrix} 1 & v_{12} & \cdots & v_{1r} \\ 0 & & & \\ \vdots & & \tilde{V} & \\ 0 & & & \end{bmatrix}$$

where $\rho(\tilde{V}) = 0$.

- High stage order is preferred for stiff problems to avoid order reduction. Our choice is to consider methods with stage order equal to the order of the method, that is $q = p$. This choice is convenient for calculating the local error and at the same time, allows an easier interpolation which will be shown in Chapter 3.

- The quantities passed from step to step represent the exact solution by a weighted Taylor series. Since we are looking for methods which are competitive with other well known methods, we will consider variable stepsize implementations. Nordsieck form is used here since it will make some aspects of implementation, such as changing stepsize, easy and efficient. In

Nordsieck form, the components of $y^{[n]}$, namely $y_1^{[n]}$, $y_2^{[n]}$, $\cdots$, and $y_{p+1}^{[n]}$ approximate $y(x_n)$, $hy'(x_n)$, $\cdots$, and $h^p y^{(p)}(x_n)$ respectively. That is

$$\begin{bmatrix} y_1^{[n]} \\ y_2^{[n]} \\ \vdots \\ y_{p+1}^{[n]} \end{bmatrix} \approx \begin{bmatrix} y(x_n) \\ hy'(x_n) \\ \vdots \\ h^p y^{(p)}(x_n) \end{bmatrix}$$

2.2.2 Stability

We now discuss the stability property of the new type of general linear methods for stiff problems. Recall that a general linear method can be written in matrix form as follows

$$Y = AhF + Uy^{[n-1]}, \tag{2.1}$$

$$y^{[n]} = BhF + Vy^{[n-1]}. \tag{2.2}$$

The linear stability behaviour of the methods is defined by applying the method to the standard linear test problem $y' = Ly$, where L is a possibly complex number. Substituting the problem into the first equation and writing $z = Lh$, we get

$$Y = (I - zA)^{-1}Uy^{[n-1]},$$

where $I - zA$ is nonsingular. Then substituting Y into the second equation, we have

$$y^{[n]} = My^{[n-1]}$$

where $M(z) = V + zB(I - zA)^{-1}U$. $M(Z)$ is called the stability matrix of the general linear method. If we apply the method to stiff problems, in order to get a smooth solution, we may need to use an excessively small stepsize since the real part of L is very negative for a stiff problem. This leads to the following definition.

Definition 2.1 *A general linear method is A-stable if $I - zA$ is nonsingular and the stability matrix $M(z)$ is a stable matrix for all $z \in C^-$.*

As for Runge-Kutta methods, we are able to define L-stability.

Definition 2.2 *A general linear method is L-stable if it is A-stable and $\rho(M(\infty)) = 0$.*

2.2.3 Order conditions

We present the order conditions of this new type of general linear method by using a step from x_{n-1} to x_n with stepsize h. For the new type of general linear methods, we consider that all the output approximations have the same order. The other assumption on these methods is that the stage order q equals to the order of the method, p.

To express these requirements a theorem has been presented using functions of a complex variable [8]. The following theorem is equivalent to the theorem in paper [8] but it is for methods in Nordsieck form [40].

Theorem 2.3 *A general linear method with coefficient matrices A, U, B and V in Nordsieck form, has stage order q equal to the order of the method p iff*

$$\exp(cz) = zA\exp(cz) + UZ + O(z^{p+1}), \tag{2.3}$$

$$\exp(z)Z \quad = \quad zB\exp(cz) + VZ + O(z^{p+1}), \tag{2.4}$$

where $\exp(cz)$ *denotes the vector for which the ith component is equal to* $\exp(c_i z)$*. It has the form*

$$\exp(cz) = \begin{bmatrix} \exp(c_1 z) \\ \exp(c_2 z) \\ \vdots \\ \exp(c_s z) \end{bmatrix}.$$

Proof. To satisfy the required order of p for the stages, the stage values which are the approximations to the solution at points $x_{n-1} + hc_i$, $i = 1, 2, \cdots, s$, should have the form

$$Y_i = y(x_{n-1} + hc_i) + O(h^{p+1}). \tag{2.5}$$

Using Taylor series expansions about point x_{n-1} we get

$$Y_i = y(x_{n-1}) + c_i h y'(x_{n-1}) + \cdots + \frac{(c_i h)^p}{p!} y^{(p)}(x_{n-1}) + O(h^{p+1}). \tag{2.6}$$

Since the incoming approximation $y^{[n-1]}$ is in Nordsieck form, we present the above equation in vector form and in terms of $y^{[n-1]}$. Equation (2.6) becomes

$$Y = Cy^{[n-1]} + O(h^{p+1}), \tag{2.7}$$

where C is a Vandermonde matrix given by

$$C = \begin{bmatrix} 1 & c_1 & \frac{c_1^2}{2!} & \cdots & \frac{c_1^p}{p!} \\ 1 & c_2 & \frac{c_2^2}{2!} & \cdots & \frac{c_2^p}{p!} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & c_s & \frac{c_s^2}{2!} & \cdots & \frac{c_s^p}{p!} \end{bmatrix}.$$

For each corresponding stage derivative we have

$$\begin{aligned} hf(Y_i) &= hy'(x_{n-1}+hc_i)+O(h^{p+2}) \\ &= \sum_{k=1}^{p+1} \frac{c_i^{k-1}}{(k-1)!} y^{(k)}(x_{n-1})h^k + O(h^{p+2}) \\ &= \sum_{k=1}^{p} \frac{c_i^{k-1}}{(k-1)!} y^{(k)}(x_{n-1})h^k + O(h^{p+1}). \end{aligned} \tag{2.8}$$

Again, the above equation can be represented in vector form as

$$hF = CKy^{[n-1]} + O(h^{p+1}), \tag{2.9}$$

where K is a shifting matrix given by

$$K = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

For an order p method the outgoing approximation at this step has the form

$$y_i^{[n]} = h^i y^{(i)}(x_{n-1}+h) + O(h^{p+1}), \quad i = 1, \ldots, p \tag{2.10}$$

while the incoming approximation has the form

$$y_i^{[n-1]} = h^{i-1} y^{(i-1)}(x_{n-1}) + O(h^{p+1}), \quad i = 1, \ldots, p. \tag{2.11}$$

We do the Taylor series expansion about x_{n-1} for the outgoing approximation and present it in terms of $y^{[n-1]}$. We then get

$$y^{[n]} = Ey^{[n-1]} + O(h^{p+1}), \tag{2.12}$$

where E is a Toeplitz matrix given by

$$E = \begin{bmatrix} 1 & \frac{1}{1!} & \frac{1}{2!} & \cdots & \frac{1}{p!} \\ 0 & 1 & \frac{1}{1!} & \cdots & \frac{1}{(p-1)!} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{1!} \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = \exp(K).$$

Substituting equations (2.7), (2.9) and (2.12) into the general linear method equations (2.1) and (2.2), we get the following order conditions

$$\begin{aligned} Cy^{[n-1]} &= ACKy^{[n-1]} + Uy^{[n-1]} + O(h^{p+1}), \\ Ey^{[n-1]} &= BCKy^{[n-1]} + Vy^{[n-1]} + O(h^{p+1}). \end{aligned}$$

We define a basis vector Z by

$$Z = \begin{bmatrix} 1 \\ z \\ \vdots \\ z^p \end{bmatrix}.$$

Let $h^k y^{(k)}(x_{n-1})$ be identified with z^k. Then the Nordsieck vector $y^{[n-1]}$ can be represented by vector Z. Therefore the order conditions become

$$CZ = ACKZ + UZ + O(z^{p+1}), \tag{2.13}$$

$$EZ = BCKZ + VZ + O(z^{p+1}). \tag{2.14}$$

Note that the matrices K, C and E have the following special properties,

$$KZ = zZ + O(z^{p+1}),$$

$$\begin{aligned} CZ &= \exp(cz) + O(z^{p+1}), \\ EZ &= \exp(z)Z + O(z^{p+1}). \end{aligned}$$

Substitute these properties into the order condition equations (2.13) and (2.14) and the result follows.

2.2.4 Inherent Runge-Kutta stability

Inherent Runge-Kutta stability is the key property of these new methods. Recall that the stability matrix of a general linear method has the form $M(z) = V + zB(I - zA)^{-1}U$. The stability region can be defined as

$$R = \left\{ z \in C, \exists K : \text{such that } \; ||M(z)||^n \leq K, \forall n \geq 1 \right\}.$$

This means that all the eigenvalues, w, of $M(z)$ must satisfy $|w| \leq 1$. The characteristic polynomial, which is known as the stability function of $M(z)$, can be written as

$$\begin{aligned} P(w) &= \det(wI - M(z)) \qquad (2.15) \\ &= w^r + P_1(z)w^{r-1} + \cdots + P_r(z), \end{aligned}$$

where the degrees of the complex functions $P_1(z)$, $\cdots$, $P_r(z)$ are at most s. Theoretically speaking, if we solve $P(w) = 0$, we could find the stability region for general linear methods as for Runge-Kutta methods. However, $P(w)$ is a very complicated function when the order of the method is large. It is difficult to obtain suitable conditions for these methods. As we know Runge-Kutta methods have very good stability properties for stiff problems. To take advantage of these nice properties, a stability condition called "inherent Runge-Kutta stability" [40] is imposed for this new type of general linear method. The idea is that these methods have the same stability region as the equivalent Runge-Kutta method. This leads to the following definition.

Definition 2.4 *A general linear method is said to possess Runge-Kutta stability if the stability function has the form*

$$P(w) = det(wI - M(z)) = w^{r-1}(w - R(z)),$$

where $R(z)$ is a rational function which has the same significance as the stability function of a Runge-Kutta method.

To ensure a general linear method has Runge-Kutta stability, we need to find sufficient conditions on the methods. Some work has been done in this area [13] [41]. As we mentioned earlier, we assume that the first column of V satisfies $Ve_1 = e_1$ for this new type of method. First we define another shifting matrix J, which is

$$J = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix},$$

and then define "inherent Runge-Kutta stability" in the following way.

Definition 2.5 *A general linear method satisfying $Ve_1 = e_1$ has the property of inherent Runge-Kutta stability if*

$$det(wI - V) \quad = \quad (w-1)w^{r-1}, \tag{2.16}$$

$$\begin{aligned} BA &\equiv JB, && (2.17)\\ BU &\equiv JV - VJ, && (2.18) \end{aligned}$$

where the notation $\equiv$ denotes equality of two matrices, except for the first row.

In the definition we note that matrix $\tilde{V}$, which is a matrix formed by deleting the first row and first column of matrix V, has all the eigenvalues in the open unit disc. This condition on the matrix V guarantees the method is stable. We now have the following theorem [13] that gives inherent Runge-Kutta stability a practical significance.

Theorem 2.6 *If a general linear method has the property of inherent Runge-Kutta stability, then the characteristic polynomial of the stability matrix, $M(z) = V + zB(I - zA)^{-1}U$ has the form*

$$P(w) = (w - R(z))w^{r-1}.$$

Proof. Instead of considering the characteristic equation of stability matrix M, we consider a matrix which is similar to $M(z)$.

$$\begin{aligned} M &\sim (I - zJ)M(I - zJ)^{-1}\\ &= (I - zJ)\left[V + zB(I - zA)^{-1}U\right](I - zJ)^{-1}. \end{aligned}$$

Substituting equations (2.17) and (2.18) into right hand side, we have

$$\begin{aligned} & (I - zJ)V(I - zJ)^{-1} + z(I - zJ)B(I - zA)^{-1}U(I - zJ)^{-1}\\ = \; & (I - zJ)V(I - zJ)^{-1} + z(B - zJB)(I - zA)^{-1}U(I - zJ)^{-1}\\ \equiv \; & (I - zJ)V(I - zJ)^{-1} + zB(I - zA)(I - zA)^{-1}U(I - zJ)^{-1}\\ \equiv \; & ((I - zJ)V + z(JV - VJ))(I - zJ)^{-1}\\ = \; & V. \end{aligned}$$

This means that $(I - zJ)M(I - zJ)^{-1}$ is identical to V except for the first row. By assumption matrix V has only one non zero eigenvalue, therefore, $M(z)$ has only one non zero eigenvalue, which is $R(z)$. The aim of having the stability region for the method to be the same as for a Runge-Kutta method is achieved.

Recently, the shifting matrix J has been generalized to a matrix called a doubly companion matrix [14] which has the form

$$X = \begin{bmatrix} -\alpha_1 & -\alpha_2 & -\alpha_3 & \cdots & -\alpha_{p-1} & -\alpha_p & -\alpha_{p+1} - \beta_{p+1} \\ 1 & 0 & 0 & \cdots & 0 & 0 & -\beta_p \\ 0 & 1 & 0 & \cdots & 0 & 0 & -\beta_{p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & -\beta_3 \\ 0 & 0 & 0 & \cdots & 1 & 0 & -\beta_2 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -\beta_1 \end{bmatrix},$$

where α_i and β_i are coefficients. The following theorem shows that J is a special case of X [40].

Theorem 2.7 *Given a general linear method in Nordsieck form with $q = p$, the most general matrix W satisfying*

$$\begin{aligned} BA &\equiv WB, \\ BU &\equiv WV - VW, \end{aligned}$$

is the doubly companion matrix X, that is $W = X$.

2.3 The choice of λ values

In the previous section, we have introduced the property of inherent Runge-Kutta stability for this new type of method. The inherent Runge-Kutta stability property guarantees that the stability function of these methods is the same as for diagonally implicit Runge-Kutta methods. Therefore, for an order p method, the stability function is of the form

$$R(z) = \frac{N(z)}{(1-\lambda z)^{p+1}},$$

where the numerator, $N(z)$, is a polynomial. The degree of $N(z)$ is $p+1$ in general and reduced to p here to make sure the method is L-stable. The $N(z)$ is chosen so that

$$R(z) = \exp(z) + Cz^{p+1} + O(z^{p+2}),$$

where the coefficient C is the error constant. This leads to

$$\begin{aligned} N(z) &= \exp(z)(1-\lambda z)^{p+1} + O(z^{p+1}) \qquad (2.19) \\ &= \left(1 + z + \frac{z^2}{2} + \cdots + \frac{z^p}{p!}\right)(1-\lambda z)^{p+1} + O(z^{p+1}). \end{aligned}$$

By calculating $N(z)$ we can find the error constants for different orders as shown in Table 2.3 (up to order 5).

Since we wish to construct this type of general linear method for stiff problems, we aim for A-stable methods. To satisfy the requirement of A-stability, as for a Runge-Kutta method, the stability function needs to satisfy

$$|R(z)| \leq 1,$$

p	C
1	$\frac{1}{2}-2\lambda+\lambda^2$
2	$\frac{1}{6}-\frac{3}{2}\lambda+3\lambda^2-\lambda^3$
3	$\frac{1}{24}-\frac{2}{3}\lambda+3\lambda^2-4\lambda^3+\lambda^4$
4	$\frac{1}{120}-\frac{5}{24}\lambda+\frac{5}{3}\lambda^2-5\lambda^3+5\lambda^4-\lambda^5$
5	$\frac{1}{720}-\frac{1}{20}\lambda+\frac{5}{8}\lambda^2-\frac{10}{3}\lambda^3+\frac{15}{2}\lambda^4-6\lambda^5+\lambda^6$

Table 2.2: Error constants for different orders.

for all the z in the left half complex plane. If we let $z = iy$, then $|R(z)| \le 1$ is equivalent to the E-polynomial being greater than or equal to zero, where

$$\begin{aligned} E &= |(1-\lambda iy)^{p+1}|^2 - |N(iy)|^2 \\ &= (1-\lambda iy)^{p+1}(1+\lambda iy)^{p+1} - N(iy)N(-iy). \end{aligned}$$

The following example illustrates how to find the value of λ.

Example 2.1 *Let $p = 2$. From equation (2.19) we get*

$$N(z) = 1 + (1-3\lambda)z + (\tfrac{1}{2} - 3\lambda + 3\lambda^2)z^2 + O(z^3).$$

Substituting $N(z)$ into the E-polynomial, we have

$$(1+y^2\lambda^2)^3 - \Big(\big[1 - y^2(\tfrac{1}{2} - 3\lambda + 3\lambda^2)\big]^2 + (1-3\lambda)^2y^2\Big) \ge 0.$$

Rearranging the equation we obtain

$$\left(3\lambda^4 - (\frac{1}{2} - 3\lambda + 3\lambda^2)^2\right)y^4 + \lambda^6y^6 \ge 0,$$

which is equivalent to

$$3\lambda^4 - (\frac{1}{2} - 3\lambda + 3\lambda^2)^2 \ge 0.$$

Rearranging the inequality, we obtain

$$\lambda \in [0.18042530, 2.18560009].$$

We should choose λ in this interval for order 2 methods so that the method has inherent Runge-Kutta stability and A-stability.

Table 2.3 shows the possible values for an order p method to have A-stability (up to order 5) [40].

The λ values that we use in this thesis are chosen from this table. In Table 2.3 we also note that the error constant is a function of λ. To obtain a suitable method we want the error constants to be reasonably small. Figure 2.1 shows the relationship between the error constant and the interval of λ for the order 2 method. The range of X-axis is the interval of λ. The solid line shows how the error constant C changes in this interval. For example, we choose $\lambda = \frac{1}{4}$, then $C = -\frac{2}{192}$, which is reasonably small.

p	interval for λ
1	$[0.29289321, 1.70710678]$
2	$[0.18042530, 2.18560009]$
3	$[0.22364780, 0.57281606]$
4	$[0.24799464, 0.67604239]$
5	$[0.18391465, 0.33414236]$

Table 2.3: The possible values of λ for different orders.

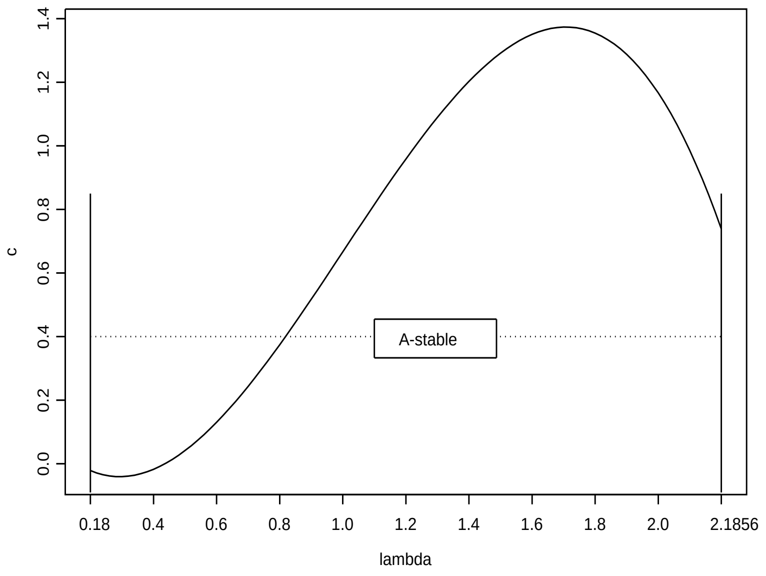

Figure 2.1: The error constant $C = \frac{1}{6} - \frac{3}{2}\lambda + 3\lambda^2 - \lambda^3$ and the interval of A-stability for $p = 2$.

2.4 Finding coefficients of A, U, B, and V

From previous sections, we know that there are some assumptions on this new type of method. Furthermore, these new methods need to satisfy order and stage order conditions. We now illustrate how to find the coefficients of matrices A, U, B and V step by step.

Theorem 2.3 in Section 2 has shown that for a general linear method to have the stage order equal to the order of the method, equations (2.13) and (2.14) must hold. By equating the coefficients of powers of z, we get

$$\begin{aligned} U &= C - ACK, \\ V &= E - BCK. \end{aligned}$$

This means that if we find suitable matrices A and B, then U and V can be found easily.

We know that these methods must also satisfy the inherent Runge-Kutta stability conditions. Therefore for a given doubly companion matrix X, we have the following conditions,

$$\begin{aligned} BA &\equiv XB, \\ BU &\equiv XV - VX, \\ \rho(\tilde{V}) &= 0. \end{aligned}$$

We note that if we are able to find matrix B, then A is found by using the above conditions.

As we have mentioned before, we consider methods with $\rho(A) = \lambda$. Using some special properties of the doubly companion matrix X [40] we have the following theorem on the first inherent Runge-Kutta stability condition.

Theorem 2.8 *Given the coefficients β_1, β_2, ..., the coefficients α_1, α_2, ..., α_{p+1}, are chosen so that the characteristic polynomial of X is of the form*

$$\det(wI - X) = (w - \lambda)^{p+1}.$$

Then $BA \equiv XB$ implies

$$BA = XB. \tag{2.20}$$

This theorem proved in [40] indicates that once we know matrix B we know matrix A (assuming that B is nonsingular). Another result is that the α_i actually depend on the β_i. Therefore only the β_i are free parameters.

In order to find the coefficients of matrix B, some useful properties of the doubly companion matrices together with our assumptions on the methods and inherent Runge-Kutta stability conditions have been used. Wright [40] has given a detailed approach to finding the coefficients of matrix B in his PhD thesis. A Maple program is available in [40]. This approach of finding the coefficients of four matrices is called Wright's approach.

There is another earlier approach which is introduced by Butcher [13]. We call this Butcher's approach. In the paper, the author used the order conditions and the inherent Runge-Kutta stability condition to find the coefficients of matrix A, then B, U and V. In this approach, the method needs to be converted into Nordsieck form with a transformation matrix. These two approaches can be transformed into each other [40]. In the thesis, we select example methods from both approaches.

2.5 Examples of the new type of methods

To obtain these specially designed methods for stiff problems, we need to choose some parameters. The parameters are listed below.

- Order of the methods, p. The p value can be 1, 2, 3, 4 and 5. In this thesis, we only test methods up to order 4.
- The value of λ. For a given order method, the λ value can only be chosen from Table 2.3 to ensure A-stability for stiff problems.
- The absicissae vector c. The c_i represent the positions of the internal stages. We prefer $c_i \in [0, 1]$. Practically $c_1 = 0$ and $c_s = 1$ have been chosen to lower implementation costs.
- The vector β in the doubly companion matrix X if the first approach is used.

Some examples of new methods are presented below.

Example 2.2 *An order* 1 *method with* 2 *stages. The absicissae vector is chosen to be* $c = [\frac{1}{2}, 1]^T$. *We choose* λ *to be* $\frac{3}{10}$ *which satisfies all the requirements of A-stability and makes the error constant* ($C = -\frac{1}{100}$) *reasonably small. The method is presented as follows.*

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{cc|cc} \frac{3}{10} & 0 & 1 & \frac{1}{5} \\ \frac{21}{50} & \frac{3}{10} & 1 & \frac{7}{25} \\ \hline \frac{21}{50} & \frac{3}{10} & 1 & \frac{7}{25} \\ 0 & 1 & 0 & 0 \end{array}\right]$$

Example 2.3 *$s = 3$, $p = 2$ with $c = [0, \frac{1}{2}, 1]^T$. We choose $\lambda = \frac{1}{4}$, therefore $C = -\frac{7}{192}$. Wright's approach is used here to find this method. The method is*

$$\left[\begin{array}{ccc|ccc}
\frac{1}{4} & 0 & 0 & 1 & -\frac{1}{4} & 0 \\
\frac{1}{4} & \frac{1}{4} & 0 & 1 & 0 & 0 \\
\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 1 & 0 & \frac{1}{8} \\
\hline
\frac{1}{2} & -\frac{1}{8} & \frac{1}{2} & 1 & \frac{1}{8} & \frac{1}{16} \\
\frac{1}{2} & -\frac{1}{2} & 1 & 0 & 0 & \frac{1}{4} \\
0 & -2 & 2 & 0 & 0 & 0
\end{array}\right]$$

Example 2.4 *An order 3 and 4 stage method with $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$. $\lambda = \frac{1}{4}$ and $C = -\frac{1}{256}$. We use Butcher's approach. This method is presented by*

$$\left[\begin{array}{cccc|cccc}
\frac{1}{4} & 0 & 0 & 0 & 1 & -\frac{1}{4} & 0 & 0 \\
\frac{5}{6} & \frac{1}{4} & 0 & 0 & 1 & -\frac{3}{4} & -\frac{1}{36} & -\frac{5}{648} \\
\frac{109057}{33000} & \frac{1701}{2200} & \frac{1}{4} & 0 & 1 & -\frac{20137}{5500} & -\frac{4003}{19800} & -\frac{17509}{356400} \\
\frac{368999}{154000} & \frac{21071}{30800} & \frac{11}{56} & \frac{1}{4} & 1 & -\frac{24319}{9625} & -\frac{3357}{30800} & -\frac{22171}{554400} \\
\hline
\frac{11419277}{5832000} & -\frac{824833}{1166400} & \frac{5303}{23328} & \frac{827}{3888} & 1 & -\frac{341047}{162000} & -\frac{116611}{1166400} & -\frac{619133}{20995200} \\
\frac{529}{1620} & -\frac{17}{162} & -\frac{35}{162} & \frac{41}{36} & 0 & -\frac{13}{90} & \frac{13}{324} & -\frac{91}{5832} \\
\frac{677}{225} & -\frac{197}{45} & -\frac{23}{18} & \frac{19}{6} & 0 & -\frac{13}{25} & \frac{13}{90} & -\frac{91}{1620} \\
6 & -9 & 0 & 3 & 0 & 0 & 0 & 0
\end{array}\right].$$

To verify that these methods have order $q = p$ and the property of inherent Runge-Kutta stability, we use the order 2 method as an example.

Example 2.5 *Considering a step with stepsize h, we have*

$$\left[\begin{array}{c} Y_1 \\ Y_2 \\ Y_3 \\ \hline y(x_n) \\ hy'(x_n) \\ h^2y''(x_n) \end{array}\right] = \left[\begin{array}{ccc|ccc} \frac{1}{4} & 0 & 0 & 1 & -\frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 1 & 0 & \frac{1}{8} \\ \hline \frac{1}{2} & -\frac{1}{8} & \frac{1}{2} & 1 & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{2} & -\frac{1}{2} & 1 & 0 & 0 & \frac{1}{4} \\ 0 & -2 & 2 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} hF_1 \\ hF_2 \\ hF_2 \\ \hline y(x_{n-1}) \\ hy'(x_{n-1}) \\ h^2y''(x_{n-1}) \end{array}\right].$$

Then, for stage 1 with $c_1 = 0$, we have

$$Y_1 = \frac{1}{4}hf(Y_1) + y(x_{n-1}) - \frac{1}{4}hy'(x_{n-1}) \tag{2.21}$$

According to the definition of the method, the left hand side of equation (2.21) is

$$Y_1 = y(x_{n-1} + c_1h) + O(h^3) = y(x_{n-1}) + O(h^3).$$

Expanding $f(Y_1)$ in Taylor series, the right hand side of equation (2.21) is

$$\begin{aligned} &= \frac{1}{4}hf(Y_1) + y(x_{n-1}) - \frac{1}{4}hy'(x_{n-1}) \\ &= \frac{1}{4}\left(hy'(x_{n-1}) + c_1h^2y''(x_{n-1}) + O(h^3)\right) + y(x_{n-1}) - \frac{1}{4}hy'(x_{n-1}) \\ &= y(x_{n-1}) + O(h^3) \end{aligned}$$

This means that stage 1 has stage order 2. For stage 2 with $c_2 = \frac{1}{2}$, we have

$$Y_2 = \frac{1}{4}hf(Y_1) + \frac{1}{4}hf(Y_2) + y(x_{n-1}) \tag{2.22}$$

The left hand side of (2.22) is

$$\begin{aligned} Y_2 &= y(x_{n-1}+c_2h)+O(h^3) \\ &= y(x_{n-1})+\frac{1}{2}hy'(x_{n-1})+(\frac{h}{2})^2\frac{y''(x_{n-1})}{2!}+O(h^3) \\ &= y(x_{n-1})+\frac{1}{2}hy'(x_{n-1})+\frac{1}{8}h^2y''(x_{n-1})+O(h^3). \end{aligned}$$

The right hand side of (2.22) is

$$\begin{aligned} &= \frac{1}{4}\left(hy'(x_{n-1})+c_1h^2y''(x_{n-1})+O(h^3)\right) \\ &\quad +\frac{1}{4}\left(hy'(x_{n-1})+c_2h^2y''(x_{n-1})+O(h^3)\right)+y(x_{n-1}) \\ &= y(x_{n-1})+\frac{1}{2}hy'(x_{n-1})+\frac{1}{8}h^2y''(x_{n-1})+O(h^3). \end{aligned}$$

This verifies that stage 2 *has stage order* 2.

For stage 3 *with* $c_3 = 1$, *we have*

$$Y_3 = \frac{1}{2}hf(Y_1)+\frac{1}{4}hf(Y_2)+\frac{1}{4}hf(Y_3)+y(x_{n-1})+\frac{1}{8}h^2y''(x_{n-1}) \tag{2.23}$$

The left hand side of (2.23) is

$$Y_3 = y(x_{n-1}+c_3h)+O(h^3) = y(x_{n-1})+hy'(x_{n-1})+\frac{h^2}{2!}y''(x_{n-1})+O(h^3)$$

and the right hand side of (2.23) is

$$\begin{aligned} &= \frac{1}{2}\left(hy'(x_{n-1})+c_1h^2y''(x_{n-1})+O(h^3)\right) \\ &\quad +\frac{1}{4}\left(hy'(x_{n-1})+c_2h^2y''(x_{n-1})+O(h^3)\right) \\ &\quad +\frac{1}{4}\left(hy'(x_{n-1})+c_3h^2y''(x_{n-1})+O(h^3)\right) \\ &\quad +y(x_{n-1})+\frac{1}{8}h^2y''(x_{n-1}) \\ &= y(x_{n-1})+hy'(x_{n-1})+\frac{1}{2}h^2y''(x_{n-1})+O(h^3). \end{aligned}$$

This verifies that stage 3 *has stage order* 2.

For the first quantity passed between steps, we have

$$y(x_n) = \frac{1}{2}hf(Y_1) - \frac{1}{8}hf(Y_2) + \frac{1}{2}hf(Y_3) + y(x_{n-1}) + \frac{1}{8}hy'(x_{n-1}) + \frac{1}{16}h^2y''(x_{n-1}).$$

The left hand side is

$$y(x_n) = y(x_{n-1}+h) = y(x_{n-1}) + hy'(x_{n-1}) + \frac{h^2}{2}y''(x_{n-1}) + O(h^3),$$

and the right hand side is

$$\begin{aligned} &= \frac{1}{2}\big(hy'(x_{n-1}) + c_1h^2y''(x_{n-1})\big) \\ &\quad -\frac{1}{8}\big(hy'(x_{n-1}) + c_2h^2y''(x_{n-1})\big) \\ &\quad +\frac{1}{2}\big(hy'(x_{n-1}) + c_3h^2y''(x_{n-1})\big) + O(h^3) \\ &\quad +y(x_{n-1}) + \frac{1}{8}hy'(x_{n-1}) + \frac{1}{16}h^2y''(x_{n-1}) \\ &= y(x_{n-1}) + hy'(x_{n-1}) + \frac{h^2}{2}y''(x_{n-1}) + O(h^3). \end{aligned}$$

For the second quantity passed between steps, we have

$$hy'(x_n) = \frac{1}{2}hf(Y_1) - \frac{1}{2}hf(Y_2) + hf(Y_3) + \frac{1}{4}h^2y''(x_{n-1}).$$

The left hand side is

$$hy'(x_n) = hy'(x_{n-1}+h) = hy'(x_{n-1}) + h^2y''(x_{n-1}) + O(h^3),$$

and the right hand side is

$$\begin{aligned} &= \frac{1}{2}\big(hy'(x_{n-1}) + c_1h^2y''(x_{n-1})\big) \\ &\quad -\frac{1}{2}\big(hy'(x_{n-1}) + c_2h^2y''(x_{n-1})\big) \\ &\quad +\big(hy'(x_{n-1}) + c_3h^2y''(x_{n-1})\big) + O(h^3) + \frac{1}{4}h^2y''(x_{n-1}) \\ &= hy'(x_{n-1}) + h^2y''(x_{n-1}) + O(h^3). \end{aligned}$$

For the third quantity passed between steps, we have

$$h^2 y''(x_n) = -2hf(Y_2) + 2hf(Y_3).$$

The left hand side is

$$h^2 y''(x_n) = h^2 y''(x_{n-1} + h) = h^2 y''(x_{n-1}) + O(h^3),$$

and the right hand side is

$$\begin{aligned} &= -2\left(hy'(x_{n-1}) + c_2 h^2 y''(x_{n-1})\right) \\ &\quad +2\left(hy'(x_{n-1}) + c_3 h^2 y''(x_{n-1})\right) + O(h^3) \\ &= h^2 y''(x_{n-1}) + O(h^3). \end{aligned}$$

Recall that the method has inherent Runge-Kutta stability. This means the stability matrix, M, should have only one non zero eigenvalue and the non zero eigenvalue should have the form

$$R(z) = \frac{(1-\lambda z)^3(1+z+z^2) + O(h^3)}{(1-\lambda z)^3} = \frac{1 + \frac{1}{4}z - \frac{1}{16}z^2 + O(h^3)}{(1-\frac{1}{4}z)^3}.$$

Now we calculate the stability matrix of the method. Substituting A, U, B and V into $M = V + zB(I - zA)^{-1}U$, we then get

$$M = \begin{bmatrix} \frac{z^3-6z^2-8z-64}{(z-4)^3} & \frac{z^3-8z^2+56z-32}{4(z-4)^3} & -\frac{3z+4}{16(z-4)} \\ \frac{2z(z^2-4z-32)}{(z-4)^3} & \frac{2z(z+4)}{(z-4)^3} & \frac{z+4}{4(z-4)} \\ \frac{8z^2(z-8)}{(z-4)^3} & \frac{8z^2}{(z-4)^3} & -\frac{z}{z-4} \end{bmatrix}.$$

The eigenvalues of M are

$$0, 0, \frac{4(z^2-4z-16)}{z^3-12z^2+48z-64}.$$

The only non zero eigenvalue is the same as $R(z)$. *This verifies that the method has inherent Runge-Kutta stability.*

Chapter 3

Implementation

In Chapter 2, we reviewed the theoretical properties of the new type of general linear method for stiff problems, such as stability and order. We have also discussed how to construct practical methods. In this chapter, we concentrate on the implementation issues for these new general linear methods designed in the previous chapter. These methods are quite new in the literature and this is the first time they have been submitted to extensive numerical testing. Theoretically, these methods have several advantages such as inherent Runge-Kutta stability and high stage order, over the traditional methods. Understanding the performance of these methods is desirable from practical point of view. Our aim is to explore the implementation questions and carry out numerical experiments on these new methods. To implement the methods in an efficient way, we need to consider several important questions.

- Iteration scheme: since these methods are implicit methods, we need to evaluate the stage values using some variant of Newton iteration. In Section

3.1 we will explain how we do the Newton iterations and develop an iteration scheme for our experiments.

- Stage predictors: in the Newton iteration process an estimated initial iteration value is needed. A good estimate will reduce the total number of iterations. In Section 3.2, three different predictors will be presented and compared.

- Error estimation: we estimate the local truncation error to ensure the output result is of acceptable accuracy.

- Stepsize control: we change the stepsize to control the local truncation error for efficient solution.

- Starting method: due to the multivalue nature of these methods, a starting method is required. We design a starting method to provide the value of the initial Nordsieck vector.

- Interpolation: in Section 3.6 we show how to use interpolation to calculate an output value at a specific point.

- Test problems: we review the test problems used here for assessing the performance on various aspects of the new methods. These problems are listed in Section 3.7.

The rest of this chapter will be concerned with these questions.

3.1 Iteration scheme

In Chapter 1 we noted that the non-linear equations defining the stage values have to be solved by an iterative procedure, such as a Newton-type method. We call it as modified Newton procedure. To be simple, we still use the name, Newton procedure in the thesis. In this section, we give details of the Newton iteration process.

Consider an s stage new type of general linear method for an ordinary differential equation which is an N dimensional system. The stages can be expressed by the following equation,

$$Y_i = \sum_{j=1}^{s} a_{ij} h f(Y_j) + \sum_{j=1}^{r} u_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \ldots, s, \tag{3.1}$$

with $a_{ii} = \lambda$ and $a_{ij} = 0$ for $i < j$. This is equivalent to

$$Y_i - \lambda h f(Y_i) = \sum_{j=1}^{i-1} a_{ij} h f(Y_i) + \sum_{j=1}^{r} u_{ij} y_j^{[n-1]}, \qquad i = 1, 2, \ldots, s. \tag{3.2}$$

Note that the right hand side of equation (3.2) contains information from the previous step and from the stages which are already evaluated. Letting rhs denote the right hand side of equation (3.2), we have

$$Y_i - \lambda h f(Y_i) = \mathrm{rhs}_i \qquad i = 1, 2, \ldots, s. \tag{3.3}$$

Introduce a variable, η, and write equation (3.3) as

$$\phi(\eta) = \eta - \lambda h f(\eta) - \mathrm{rhs}_i = 0.$$

Then the problem becomes to solve $\phi(\eta) = 0$. Applying the Newton method here, we then solve a linear system with the iteration matrix

$$I - h\lambda J = LU,$$

where the matrices L and U are the LU factorization and J is the Jacobian matrix.

The cost of the stage evaluation of a full Newton iteration scheme includes the following:

- evaluation of the Jacobian. It is typically expensive to compute numerically.
- LU factorization. The cost is $C_1N^3s^3$ operations because there are sN unknowns.
- solving linear systems. The back substitution cost is $C_2N^2s^2$ operations. These C_1 and C_2 are constants.

Implementations of all three items are costly. Our aim is to minimize the total computational cost to achieve an efficient code. In Chapter 2, we have shown that the A matrix is chosen to have a lower triangular form for the reason of lower implementation cost. This choice allows us to solve the nonlinear system stage by stage, separately and sequentially. We also choose all the diagonal elements of the matrix A equal to λ. The advantage of this choice is that only one Jacobian evaluation and matrix factorization is needed for the Newton iteration. This special structure of matrix A reduces the total cost to $C_1(N^3) + C_2(N^2s)$ operations.

To illustrate how the stages are evaluated, we use stage 1 as an example.

Example 3.1 *For stage* 1, *we have the following equation,*

$$Y_1 = \lambda hf(Y_1) + \sum_{j=1}^{r} u_{1j}y_j^{[n-1]}.$$

Therefore function ϕ is

$$\phi(\eta) = \eta - \lambda h f(\eta) - \text{rhs},$$

where

$$\text{rhs} = \sum_{j=1}^{r} u_{1j} y_j^{[n-1]}.$$

The problem is then equivalent to finding the root of $\phi(\eta) = 0$. The following procedure is designed to use Newton iteration to calculate the stage values.

```
carryon = true
  while carryon
      Evaluate e by solving $(I - \lambda h J)e = \phi(\eta^{[j]})$
      $\eta^{[j+1]} = \eta^{[j]} - e$
      carryon=$||e|| > \epsilon$
  end while
$Y_1 = \eta^{[j+1]}$
```

where ϵ is the error tolerance for terminating the Newton iterations and $\eta^{[j]}$ is the j-th Newton update. This ϵ depends on the tolerance used in the stepsize controller. $\eta^{[j+1]}$ is the approximation to Y_1 when the convergence condition is satisfied. Other stages, because of the special triangular form of matrix A, satisfy similar equations, and the same procedures are used.

In the Newton method, the criterion for stopping the iterations is

$$||\eta^{[j+1]} - Y_i|| < \text{Tol},$$

where Y_i is the exact solution and Tol is the tolerance for accuracy of stage approximations. In practice this Y_i value is not known yet. We need to modify the criterion. Defining a convergence rate at the jth iteration by $r^{[j]}$, we have

$$r^{[j]} = \frac{||\eta^{[j+1]} - \eta^{[j]}||}{||\eta^{[j]} - \eta^{[j-1]}||}.$$

We assume that the upper bound of $r^{[j]}$ is r and that $r < 1$. After a large number of iterations, say N iterations, $\eta^{[j+N]}$ converges to Y_i, we then have

$$\begin{aligned} ||\eta^{[j+1]} - Y_i|| &= ||\eta^{[j+1]} - \eta^{[j+2]} + \eta^{[j+2]} - \cdots + \eta^{[j+N]} - Y_i|| \\ &\leq ||\eta^{[j+1]} - \eta^{[j+2]}|| + ||\eta^{[j+2]} - \eta^{[j+3]}|| + \cdots + ||\eta^{[j+N]} - Y_i|| \\ &\approx r||e^{[j]}|| + r^2||e^{[j]}|| + \cdots + r^N||e^{[j]}|| \\ &= r(1 + r + \cdots + r^{N-1})||e^{[j]}|| \\ &= r\frac{1-r^N}{1-r}||e^{[j]}|| \\ &\approx \frac{r}{1-r}||e^{[j]}||, \end{aligned}$$

where $e^{[j]} = \eta^{[j+1]} - \eta^{[j]}$. For a given tolerance ϵ, if

$$\frac{r}{1-r}||e^{[j]}|| < \text{Tol}$$

then

$$||\eta^{[j+1]} - Y_i|| < \text{Tol}.$$

In practice the upper bound r is not known either. To derive a criterion for stopping iterations, we assume that

$$\frac{r}{1-r} < 10,$$

which gives $r < 10/11$, and guarantees that the iterations converge, otherwise we reject this step. Then if

$$||e^{[j]}|| \leq \frac{\text{Tol}}{10},$$

it follows that $||\eta^{[j+1]} - Y_i|| <$ Tol holds, and we accept $\eta^{[j+1]}$ as Y_i. Therefore, we let $\epsilon = \frac{\text{Tol}}{10}$.

Using the convergence rate, we can generalize the criterion for stopping iterations. Assuming that we have approximately the same convergence rate for several iterations, then at the nth iteration

$$\frac{||e^{[n+1]}||}{||e^{[n]}||} \approx \frac{||e^{[n]}||}{||e^{[n-1]}||}.$$

If $||e^{[n+1]}|| \leq \epsilon$ holds at the $n+1$th iteration, then at the nth iteration, we can relax the convergence condition to

$$||e^{[n]}||^2 < \epsilon ||e^{[n-1]}||.$$

If the ratio, $\frac{||e^{[n+1]}||}{||e^{[n]}||}$ at the $n+1$th iteration is too big, say $K = 2$, we consider the iteration to have diverged and we then reject this iteration. Figure 3.1 shows the procedure for testing the convergence.

In Section 3, we will use the stage derivatives to calculate the error estimator. The stage derivatives are one order higher than the stages. In order to achieve the goal that the error estimator is comparable to $Tolerance$ which is provided by users, we need to carry out Newton iterations to a higher degree of accuracy. This means that we should choose a smaller tolerance for the accuracy of the stage approximations than the given tolerance, $Tolerance$, of the method. Since Tol, the tolerance for accuracy of stage approximations, does not appear in the computer codes, we only use ϵ. The results of numerical experiments with $\epsilon = Tolerance/10$, $\epsilon = Tolerance/100$ and $\epsilon = Tolerance/1000$ will be presented in Chapter 4.

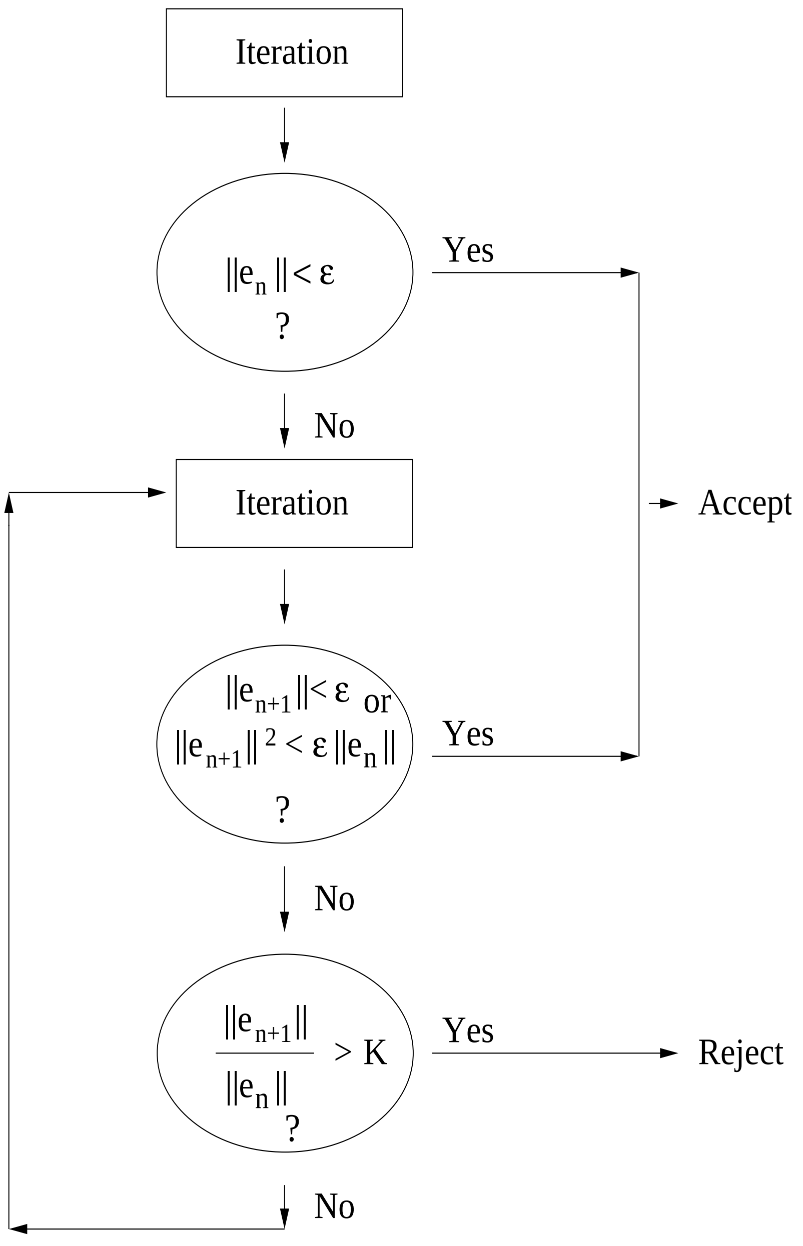
Iteration
$\|e_n\| < \varepsilon$?
Yes
No
Iteration
Accept
$\|e_{n+1}\| < \varepsilon$ or $\|e_{n+1}\|^2 < \varepsilon \|e_n\|$?
Yes
No
$\frac{\|e_{n+1}\|}{\|e_n\|} > K$?
Yes
Reject
No

Figure 3.1: The scheme for convergence testing

The most costly step in the procedure is the LU factorization of $I - \lambda hJ$ and the back substitutions. The operational cost is N^3s^3 for doing the LU factorization for $I - h\lambda J$. The back substitution takes N^2s^2 operations. These computational costs are very high, especially when the dimension of the system, N, is large and the Jacobian matrix is dense or a method with more stages is used. In order to reduce these costs, we wish to use the same LU and J as much as possible provided that the convergence is achieved. This strategy is adopted by many solvers to reduce the computational costs. In this thesis, the "old" J and LU which might have been calculated in previous steps is used to do the Newton iterations. However, if we keep the "old" J and LU too long, it may cause slow convergence or divergence. Therefore an iteration scheme is needed to determine when and how often we should evaluate a new Jacobian or do an LU factorization and how to deal with convergence failure. In this thesis, we present the following algorithm to obtain better efficiency.

In this scheme, for a stage Y_i, attempt the Newton iteration with "old" J and LU from previous steps. If it is convergent, move to the next stage or step; if not, we carry out an LU factorization of $I - h\lambda J$ with current h. With the updated LU factorization we then redo the Newton iterations. If the Newton process is convergent, move to the next stage or step and keep this update for next stage or step to use; if it is not convergent, update the J and redo the LU factorization of $I - h\lambda J$. We use the most recently updated LU factorization to do the iteration again; if it is convergent, move to the next stage or step and keep the most recent update for next stages or steps to use, otherwise we need to lower h to $\frac{h}{2}$ and redo this step from the beginning. Figure 3.2 shows the algorithm for Newton iteration.

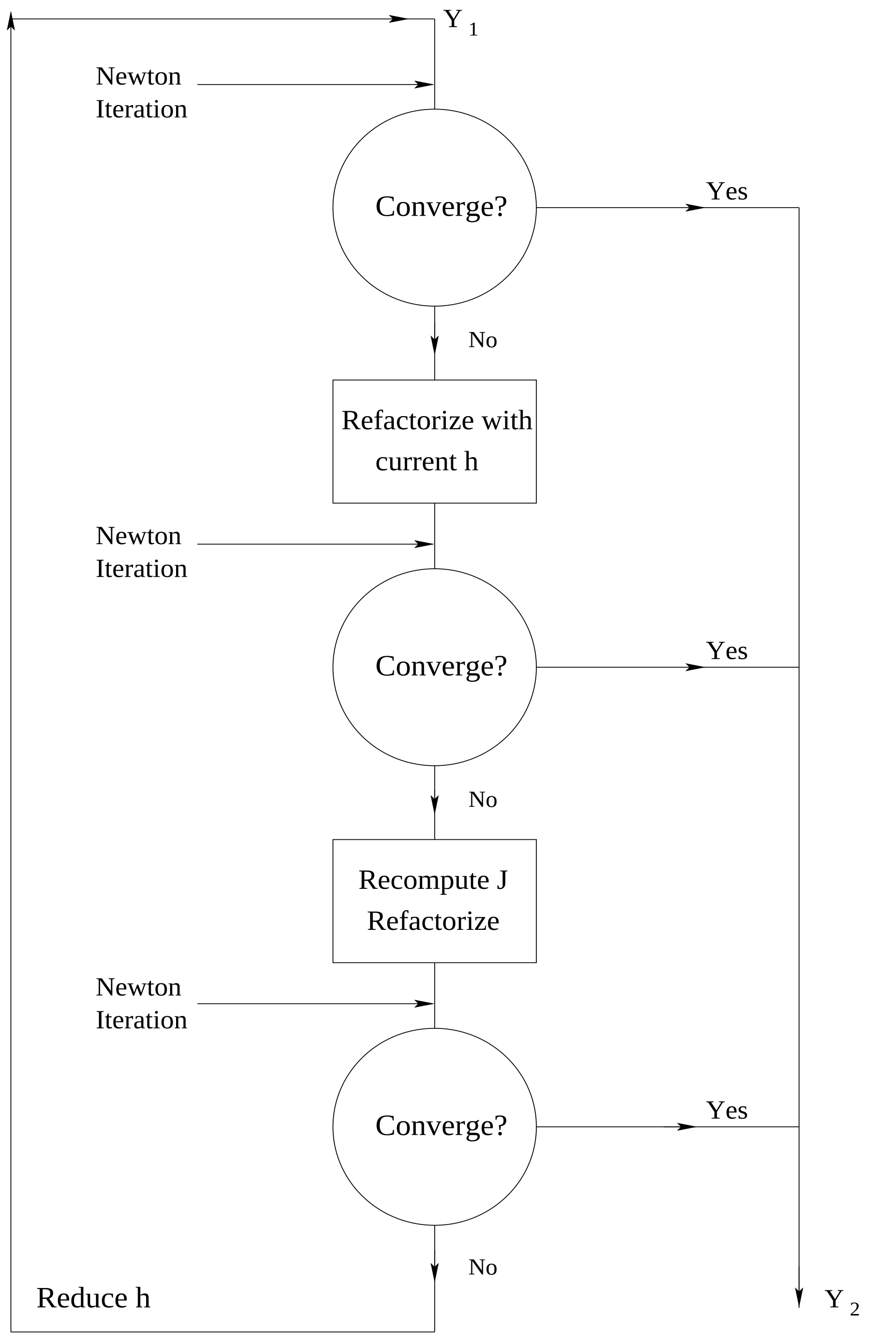

Figure 3.2: Scheme for controlling updates.

In this thesis, the Lapack routines DGETRF and DGETRS [42] were used for the LU or PLU factorization and back substitution.

To produce an efficient solver, we are also concerned with the evaluation of the stage derivatives. After the Newton iterations, we have accepted a converged value as the stage value Y_i. We now need to compute the stage derivative value $f(Y_i)$. A natural way is substituting $x_n + hc_i$ and Y_i in the function f, and then we use these F_i $(i = 1, 2, \ldots, s)$ values to estimate the error. Since we are solving stiff problems, if we calculated F in this way, the error would be increased [37]. It is therefore necessary to evaluate the F_i values using the following formula

$$f(Y_i) = \frac{Y_i - \mathrm{rhs}_i}{\lambda h}.$$

3.2 Stage predictors

In the Newton iteration process, we need a starting value $\eta^{[0]}$ as an approximation to η. The convergence of the Newton iteration is sensitive to the starting values of the stages. A better starting value gives better efficiency. If this first approximation is very close to the solution value, the number of Newton iterations may be reduced. As a result, the total cost is reduced. Another advantage of a good prediction is that actual convergence is more likely to be successful. Therefore, we get fewer rejections. For the methods introduced in this thesis, we consider step number $n-1$ being accepted, and then have the Nordsieck vector $y^{[n-1]}$,

$$y^{[n-1]} = \begin{bmatrix} y_1^{[n-1]} \\ y_2^{[n-1]} \\ y_3^{[n-1]} \\ \vdots \end{bmatrix} \approx \begin{bmatrix} y(x_{n-1}) \\ hy'(x_{n-1}) \\ h^2y''(x_{n-1}) \\ \vdots \end{bmatrix}.$$

Our aim is to use this known information to calculate the first approximation of η. In this thesis we propose three different stage predictors to estimate the value of $\eta^{[0]}$. They are a Taylor expansion predictor, a Newton interpolation predictor and an Hermite interpolation predictor.

Taylor expansion predictor

For the Taylor expansion, we use the known values from the previous step without any additional computational cost. We choose special values for the elements of the abscissae vector c, namely $c_1 = 0$ and $c_s = 1$. As a result, we have $Y_s = y(x_{n-1} + c_s h) = y(x_n)$ for step number $n-1$, and $Y_1 = y(x_n + c_1 h) = y(x_n)$

for step number n. Therefore, the first approximation for Y_1 of step number n can be the last stage value, Y_s, from step number $n-1$. For other stages, we use the Taylor expansion since we know higher order derivatives from the Nordsieck vector $y^{[n-1]}$ of the previous step. Therefore, for stage i $(i > 1)$, we have

$$\begin{aligned} Y_i &= y(x_{n-1} + c_i h) + O(h^{p+1}) \\ &= y(x_{n-1}) + \sum_{j=1}^{p} \frac{c_i^j}{j!} h^j y^{(j)}(x_{n-1}) + O(h^{p+1}) \\ &= \sum_{j=0}^{p} \frac{c_i^j}{j!} y_{j+1}^{[n-1]} + O(h^{p+1}) \\ &\approx \sum_{j=0}^{p} \frac{c_i^j}{j!} y_{j+1}^{[n-1]}. \end{aligned}$$

The Taylor expansion predictors for order 2, order 3 and order 4 methods are presented in Table 3.1.

order	Taylor expansion predictor
2	$Y_i^{[0]} = y_1^{[n-1]} + c_i y_2^{[n-1]} + \frac{c_i^2}{2!} y_3^{[n-1]}$
3	$Y_i^{[0]} = y_1^{[n-1]} + c_i y_2^{[n-1]} + \frac{c_i^2}{2!} y_3^{[n-1]} + \frac{c_i^3}{3!} y_4^{[n-1]}$
4	$Y_i^{[0]} = y_1^{[n-1]} + c_i y_2^{[n-1]} + \frac{c_i^2}{2!} y_3^{[n-1]} + \frac{c_i^3}{3!} y_4^{[n-1]} + \frac{c_i^4}{4!} y_5^{[n-1]}$

Table 3.1: Taylor expansion predictors for order 2, order 3 and order 4 methods $(i > 1)$.

Newton interpolation predictor

To improve the accuracy of the prediction, the second stage predictor uses Newton interpolation to find values of $\eta_i^{[0]}$. It has been noticed that the later stages require more iterations to converge than the first stage. Therefore, we wish to use the information from the previous stages to improve the quality of the predictor.

Newton interpolation has the following general formula

$$\phi(t) \approx \phi(t_1) + (t - t_1)\phi(t_1, t_2) + (t - t_1)(t - t_2)\phi(t_1, t_2, t_3), \tag{3.4}$$

where

$$\begin{aligned} \phi(t_1, t_2) &= \frac{\phi(t_2) - \phi(t_1)}{t_2 - t_1} \\ &\approx \phi'(t_1) \quad \text{when} \quad t_2 \to t_1 \end{aligned}$$

and

$$\begin{aligned} \phi(t_1, t_2, t_3) &= \frac{\phi(t_2, t_3) - \phi(t_1, t_2)}{t_3 - t_1} \\ &\approx \frac{\phi''(t_1)}{2!} \quad \text{when} \quad t_3, t_2 \to t_1. \end{aligned}$$

To illustrate how we use this technique, we present an example below.

Example 3.2 *Consider an $s = 4$, $p = 3$ method with abscissae vector $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$. After step number $n - 1$ is completed, the Nordsieck vector $y^{[n-1]}$ is known. At step number n, the Newton procedure is used to calculate stage values. We know that $Y_1 \approx y(x_{n-1} + c_1 h) = y(x_{n-1})$, therefore we use $Y_1^{[0]} = Y_4^{[n-1]}$ as the prediction of Y_1. Then we do the Newton procedure to obtain the values of Y_1 and $f(Y_1)$.*

For $Y_2^{[0]}$, we use the Taylor expansion predictor which gives

$$\begin{aligned} Y_2^{[0]} &= y(x_{n-1} + \frac{1}{3}h) + O(h^{p+1}) \\ &\approx y_1^{[n-1]} + c_2 y_2^{[n-1]} + \frac{c_2^2}{2!} y_3^{[n-1]} + \frac{c_2^3}{3!} y_4^{[n-1]} \\ &= y_1^{[n-1]} + \frac{1}{3} y_2^{[n-1]} + (\frac{1}{3})^2 \frac{y_3^{[n-1]}}{2!} + (\frac{1}{3})^3 \frac{y_4^{[n-1]}}{3!}. \end{aligned}$$

This $Y_2^{[0]}$ is used in the Newton procedure to calculate Y_2 and $hf(Y_2)$.

For $Y_3^{[0]}$, we know Y_1, Y_2 and $hf(Y_2)$ from the previous two stages. This means that we can find $\phi(\frac{2}{3})$ with given $\phi(0)$, $\phi(\frac{1}{3})$ and $\phi'(\frac{1}{3})$. Letting $t = \frac{2}{3}$, $t_1 = \frac{1}{3}$, $t_2 = \frac{1}{3}$ and $t_3 = 0$, and then using the formula in equation (3.4), we have

$$\begin{aligned} \phi\left(\frac{2}{3}\right) &= \phi\left(\frac{1}{3}\right) + \left(\frac{2}{3} - \frac{1}{3}\right)\phi\left(\frac{1}{3}, \frac{1}{3}\right) + \left(\frac{2}{3} - \frac{1}{3}\right)^2 \phi\left(\frac{1}{3}, \frac{1}{3}, 0\right) \\ &= \phi\left(\frac{1}{3}\right) + \frac{1}{3}\phi'\left(\frac{1}{3}\right) + (\frac{1}{3})^2 \frac{\phi\left(\frac{1}{3}, 0\right) - \phi\left(\frac{1}{3}, \frac{1}{3}\right)}{0 - \frac{1}{3}} \\ &= \phi(0) + \frac{2}{3}\phi'\left(\frac{1}{3}\right). \end{aligned}$$

This leads to

$$Y_3^{[0]} = Y_1 + \frac{2}{3} hf(Y_2). \tag{3.5}$$

For $Y_4^{[0]}$, similarly to stage 3, we have $\phi(\frac{1}{3})$, $\phi(\frac{2}{3})$ and $\phi'(\frac{2}{3})$ available and wish to find $\phi(1)$. With $t = 1$, $t_1 = \frac{2}{3}$, $t_2 = \frac{2}{3}$, and $t_3 = \frac{1}{3}$, we use the formula in equation (3.4) to get

$$\phi(1) = \phi\left(\frac{1}{3}\right) + \frac{2}{3}\phi'\left(\frac{2}{3}\right). \tag{3.6}$$

Hence we have

$$Y_4^{[0]} = Y_2 + \frac{2}{3} hf(Y_3). \tag{3.7}$$

For other methods with different c vectors we use the same formulae to estimate the $Y_i^{[0]}$ values. Tables 3.2 and 3.3 list the Newton interpolation predictors for the order 3 and order 4 methods respectively.

Stage	Predictor
$Y_1^{[0]}$	$Y_4^{[n-1]}$
$Y_2^{[0]}$	$y_1^{[n-1]} + \frac{1}{3}y_2^{[n-1]} + (\frac{1}{3})^2\frac{y_3^{[n-1]}}{2!} + (\frac{1}{3})^3\frac{y_4^{[n-1]}}{3!}$
$Y_3^{[0]}$	$Y_1 + \frac{2}{3}hf(Y_2)$
$Y_4^{[0]}$	$Y_2 + \frac{2}{3}hf(Y_3)$

Table 3.2: The Newton interpolation predictor for an order 3 method with $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$.

Stage	Predictor
$Y_1^{[0]}$	$Y_5^{[n-1]}$
$Y_2^{[0]}$	$y_1^{[n-1]} + \frac{1}{4}y_2^{[n-1]} + (\frac{1}{4})^2\frac{y_3^{[n-1]}}{2!} + (\frac{1}{4})^3\frac{y_4^{[n-1]}}{3!} + (\frac{1}{4})^4\frac{y_5^{[n-1]}}{4!}$
$Y_3^{[0]}$	$Y_1 + \frac{1}{2}hf(Y_2)$
$Y_4^{[0]}$	$Y_2 + \frac{1}{2}hf(Y_3)$
$Y_5^{[0]}$	$Y_3 + \frac{1}{2}hf(Y_4)$

Table 3.3: The Newton interpolation predictor for an order 4 method with $c = [0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1]^T$.

Hermite interpolation predictor

The third stage predictor uses an Hermite interpolation formula. It is based on information from the previous two stages. In this predictor, we use Y_{i-2}, $hf(Y_{i-2})$, Y_{i-1} and $hf(Y_{i-1})$ $(i > 2)$ to predict the value of $Y_i^{[0]}$. For the first two stages, we use $Y_1^{[0]} = y_1^{[n-1]}$ and $Y_2^{[0]} = y_1^{[n-1]} + c_2 y_2^{[n-1]} + \frac{c_2^2}{2!} y_3^{[n-1]} + \cdots$. For stage 3 and up, let

$$Y_i^{[0]} = aY_{i-2} + bhf(Y_{i-2}) + cY_{i-1} + dhf(Y_{i-1}) \tag{3.8}$$

where a, b, c and d are coefficients to be determined. For simplicity we only show $i = 3$ as an example. By expanding both sides of equation (3.8) in a Taylor series around x_n we obtain

$$\begin{aligned}
& y(x_n) + c_3 hy'(x_n) + \frac{c_3^2}{2!} h^2 y''(x_n) + \frac{c_3^3}{3!} h^3 y'''(x_n) + O(h^4) \\
= \;& a\left[y(x_n) + c_1 hy'(x_n) + \frac{c_1^2}{2!} h^2 y''(x_n) + \frac{c_1^3}{3!} h^3 y'''(x_n)\right] \\
+ \;& bh\left[y'(x_n) + c_1 hy''(x_n) + \frac{c_1^2}{2!} h^2 y'''(x_n)\right] \\
+ \;& c\left[y(x_n) + c_2 hy'(x_n) + \frac{c_2^2}{2!} h^2 y''(x_n) + \frac{c_2^3}{3!} h^3 y'''(x_n)\right] \\
+ \;& dh\left[y'(x_n) + c_2 hy''(x_n) + \frac{c_2^2}{2!} h^2 y'''(x_n)\right] + O(h^4).
\end{aligned}$$

Equating the two sides we get

$$\begin{aligned}
a + c &= 1, \\
ac_1 + b + cc_2 + d &= c_3, \\
a\frac{c_1^2}{2!} + bc_1 + c\frac{c_2^2}{2!} + dc_2 &= \frac{c_3^2}{2!}, \\
a\frac{c_1^3}{3!} + b\frac{c_1^2}{2!} + c\frac{c_2^3}{3!} + d\frac{c_2^2}{2!} &= \frac{c_3^2}{3!}.
\end{aligned}$$

There are 4 equations and 4 unknown variables. The coefficients can be obtained by solving this equation system.

Example 3.3 *As before, we consider the $s = 4$ and $p = 3$ method with $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$. For stage 1, we let $Y_1^{[0]} = y_1^{[n-1]}$. For stage 2 we use $Y_2^{[0]} = y_1^{[n-1]} + c_2 y_2^{[n-1]} + \frac{c_2^2}{2!} y_3^{[n-1]} + \frac{c_2^3}{3!} y_4^{[n-1]}$. For stage 3, we use Y_1, $hf(Y_1)$ and Y_2, $hf(Y_2)$. Substituting c_1, c_2 and c_3 values to the above equation system, we get*

$$Y_3^{[0]} = 5Y_1 + \frac{2}{3} hf(Y_1) - 4Y_2 + \frac{4}{3} hf(Y_2).$$

For stage 4, we carry out the same procedure as for the stage 3 and obtain

$$Y_4^{[0]} = 5Y_2 + \frac{2}{3} hf(Y_2) - 4Y_3 + \frac{4}{3} hf(Y_3).$$

Tables 3.4 and 3.5 present the Hermite interpolation predictors for order 3 and order 4 methods.

Stage	Predictor
$Y_1^{[0]}$	$y_1^{[n-1]}$
$Y_2^{[0]}$	$y_1^{[n-1]} + \frac{1}{3} y_2^{[n-1]} + (\frac{1}{3})^2 \frac{y_3^{[n-1]}}{2!} + (\frac{1}{3})^3 \frac{y_4^{[n-1]}}{3!}$
$Y_3^{[0]}$	$5Y_1 + \frac{2}{3} hf(Y_1) - 4Y_2 + \frac{4}{3} hf(Y_2)$
$Y_4^{[0]}$	$5Y_2 + \frac{2}{3} hf(Y_2) - 4Y_3 + \frac{4}{3} hf(Y_3)$

Table 3.4: The Hermite interpolation predictor for an order 3 method with $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$.

The numerical experiments for these three predictors will be presented in Chapter 4. It is possible to use higher order formula for later stages.

Stage	Predictor
$Y_1^{[0]}$	$y_1^{[n-1]}$
$Y_2^{[0]}$	$y_1^{[n-1]} + \frac{1}{4}y_2^{[n-1]} + (\frac{1}{4})^2\frac{y_3^{[n-1]}}{2!} + (\frac{1}{4})^3\frac{y_4^{[n-1]}}{3!} + (\frac{1}{4})^4\frac{y_5^{[n-1]}}{4!}$
$Y_3^{[0]}$	$5Y_1 + \frac{1}{2}hf(Y_1) - 4Y_2 + hf(Y_2)$
$Y_4^{[0]}$	$5Y_2 + \frac{1}{2}hf(Y_2) - 4Y_3 + hf(Y_3)$
$Y_5^{[0]}$	$5Y_3 + \frac{1}{2}hf(Y_3) - 4Y_4 + hf(Y_4)$

Table 3.5: The Hermite interpolation predictor for an order 4 method with $c = [0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1]^T$.

3.3 Error estimation

In order to implement variable stepsize, we need to estimate the local truncation error, a measure of the accuracy of the approximation at each step. Our goal is to estimate the local error and then develop a stepsize controller. In Chapter 1, we have discussed the local truncation error for general linear methods. The local truncation error is defined as the difference between the computed numerical result and the exact solution of the problem. Figure 1.4 in Chapter 1 illustrates the local error, which is defined as follows,

$$E_n = Ch^{p+1}y^{(p+1)}(x_n) + O(h^{p+2}).$$

The higher order term $O(h^{p+2})$ is ignored for the stepsize control.

In step number n, we have the Nordsieck vector, $y^{[n]}$, and the values of hF_i available. We prefer using the hF_i only. The reason is that each component of the Nordsieck vector contains an error of $O(h^{p+1})$. But hF_i has error of $O(h^{p+2})$

since Y_i has error of $O(h^{p+1})$. We then estimate the error E_n by taking a linear combination of the known stage derivatives, $hf(Y_i)$ $i = 1, 2, \ldots, s$. The principal term of E_n, $h^{p+1}y^{(p+1)}$, can be represented by

$$h^{p+1}y^{(p+1)} \approx d_1 hf[y(x_{n-1}+c_1h)] + \ldots + d_s hf[y(x_{n-1}+c_sh)] \tag{3.9}$$

where d_1, …, d_s are coefficients which can be found for a particular method by expanding $f[y(x_{n-1}+c_ih)]$ in a Taylor series around x_{n-1}. We now use an example to illustrate the idea.

Example 3.4 . *Apply a $p = 3$, $s = 4$ method with $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$, $\lambda = \frac{1}{4}$ and error constant $C = \frac{1}{256}$ to the ODE problem $y' = f(x, y)$. At step number n, Y_1, Y_2, Y_3, Y_4, $hf(Y_1)$, $hf(Y_2)$, $hf(Y_3)$ and $hf(Y_4)$ have been estimated through the Newton procedure. According to the order conditions for this type of method in Chapter 2, we have*

$$\begin{aligned} hf(Y_i) &= hf(y(x_{n-1}+c_ih)) + O(h^5) \\ &= hy'(x_{n-1}) + c_ih^2y''(x_{n-1}) + \frac{c_i^2}{2!}h^3y'''(x_{n-1}) + \frac{c_i^3}{3!}h^4y^{(4)}(x_{n-1}) + O(h^5). \end{aligned}$$

Substituting the $hf(Y_i)$ into equation (3.9) we have

$$\begin{aligned} h^4y^{(4)} &= d_1hy'(x_{n-1}) \\ &+ d_2\left(hy'(x_{n-1}) + \frac{1}{3}h^2y''(x_{n-1}) + \frac{(\frac{1}{3})^2}{2!}h^3y'''(x_{n-1}) + \frac{(\frac{1}{3})^3}{3!}h^4y^{(4)}(x_{n-1})\right) \\ &+ d_3\left(hy'(x_{n-1}) + \frac{2}{3}h^2y''(x_{n-1}) + \frac{(\frac{2}{3})^2}{2!}h^3y'''(x_{n-1}) + \frac{(\frac{2}{3})^3}{3!}h^4y^{(4)}(x_{n-1})\right) \\ &+ d_4\left(hy'(x_{n-1}) + h^2y''(x_{n-1}) + \frac{1}{2!}h^3y'''(x_{n-1}) + \frac{1}{3!}h^4y^{(4)}(x_{n-1})\right) + O(h^5). \end{aligned}$$

Equating the coefficients of $h^iy^{(i)}$ for both sides we get

$$\sum_{i=1}^{4} d_i = 0,$$

$$\begin{aligned}\sum_{i=2}^{4} d_i c_i &= 0,\\ \sum_{i=2}^{4} d_i \frac{c_i^2}{2!} &= 0,\\ \sum_{i=2}^{4} d_i \frac{c_i^3}{3!} &= 1.\end{aligned}$$

This is a linear system with 4 unknown variables and 4 equations. Solve this equation system and we get

$$d_1 = -27, \quad d_2 = 81, \quad d_3 = -81, \quad d_4 = 27.$$

Therefore, we have the following error estimator for this particular method,

$$E_n \approx C\left[-27hf(Y_1) + 81hf(Y_2) - 81hf(Y_3) + 27hf(Y_4)\right].$$

The formula for the error estimator depends on the individual method. Each method has its own error estimator based on the error constant and the abscissae vector of the method. Table 3.6 presents the error estimators for the selected order 2, order 3 and order 4 methods.

order	λ	Error estimator
2	$\frac{1}{4}$	$-\frac{28}{192}(hf(Y_1) - 2hf(Y_2) + hf(Y_3))$
3	$\frac{1}{4}$	$-\frac{27}{256}(-hf(Y_1) + 3hf(Y_2) - 3hf(Y_3) + hf(Y_4))$
4	$\frac{1}{4}$	$-\frac{13}{60}(hf(Y_1) - 4hf(Y_2) + 6hf(Y_3) - 4hf(Y_4) + hf(Y_5))$

Table 3.6: Error estimators for order 2, order 3 and order 4 methods with $c = [0, \frac{1}{2}, 1]^T$, $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$ and $c = [0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1]^T$ respectively.

3.4 Stepsize control

Using a constant stepsize may not be efficient for some problems since the numerical behaviour of these problems changes over the region of integration. Exceptionally, some problems change stiffness over the integration interval. Variable stepsize has been considered for these general linear methods. In order to control the stepsize, we need to estimate the error for each step. In the previous section we have presented how we estimate the local truncation error, E_n. The stepsize controller used in this thesis is a traditional stepsize controller which has been used by many numerical codes. It has the form

$$h_{n+1} = \hat{\theta}_n h_n,$$

where h_n is the stepsize for step n, h_{n+1} is the stepsize expected in the following step $n+1$ and $\hat{\theta}_n$ is a coefficient.

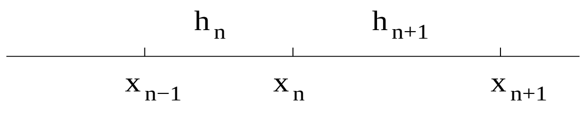

Figure 3.3: Variable stepsize

It has been mentioned that if one step is accepted the truncation error needs to satisfy

$$||E_n|| \leq Tolerance,$$

where $Tolerance$ is the tolerance for the method provided by the user. The main idea of stepsize control is that we wish the stepsize for the next step to be optimal. One way to achieve that is to choose the new stepsize to make the local error approximately the same as the given tolerance. For an order p method, at step n we have the local error

$$E_n = Ch_n^{p+1} y^{(p+1)}(x_n) = \psi_n y^{(p+1)}(x_n),$$

where $\psi_n = Ch_n^{p+1}$. At step $n+1$ we should have

$$\psi_{n+1} = Ch_{n+1}^{p+1}$$

and

$$E_{n+1} = Ch_{n+1}^{p+1} y^{(p+1)}(x_{n+1}) = \psi_{n+1} y^{(p+1)}(x_{n+1}).$$

To have the optimal stepsize, we want the norm of the local error approximately equal to the given $Tolerance$. This condition gives

$$\frac{Tolerance}{||E_n||} \approx \left(\frac{h_{n+1}}{h_n}\right)^{p+1},$$

which is equivalent to

$$h_{n+1} = \left(\frac{Tolerance}{||E_n||}\right)^{\frac{1}{p+1}} h_n.$$

We now introduce a safety factor γ. The purpose of this safety factor is to reduce the risk of rejecting the next step. Normally γ is a value between 0 and 1. In this thesis, we choose $\gamma = 0.9$. Therefore, we have

$$\hat{\theta}_n = \gamma \left(\frac{Tolerance}{||E_n||}\right)^{\frac{1}{p+1}}.$$

To change the stepsize smoothly and reduce the risk of unstable behaviour in the numerical solution, we need to increase or decrease the stepsize gradually. If the new stepsize is chosen too big then a larger error will result in rejection. If the new stepsize is chosen too small then it will take more steps to reach x_{end}. Furthermore, since our theorems of stability and order conditions are based on constant stepsize, the changes in stepsize should be reasonably small. As a result, we set a limit on the value of θ_n. At step number n, we have

$$\theta_n = \text{Min}\left(2, \text{Max}(\hat{\theta}_n, \frac{1}{2})\right),$$

where the two numbers, $\frac{1}{2}$ and 2, are typical choices and based on the experimentation. The numerical results using the traditional stepsize controller will be presented in Chapter 4. Recently, there has been an emphasis on using sound control theory principles [25] to control stepsize. In this thesis, we only use the traditional controller.

Recall that the Nordsieck vector $y^{[n]}$ at step number n consists of the output approximations which are calculated in terms of h_n, i.e.,

$$y_{out}^{[n]} \approx \begin{bmatrix} y(x_n) \\ h_n y'(x_n) \\ \vdots \\ h_n^p y^{(p)}(x_n) \end{bmatrix}.$$

For the following step number $n+1$, we will use the new stepsize $h_{n+1} = \hat{\theta}_n h_n$. Therefore the corresponding input approximations in terms of h_{n+1} are needed.

The new incoming approximations should have the form of

$$y_{in}^{[n+1]} \approx \begin{bmatrix} y(x_n) \\ h_{n+1}y'(x_n) \\ \vdots \\ h_{n+1}^p y^{(p)}(x_n) \end{bmatrix}.$$

To rescale the quantity $y_{out}^{[n]}$ to $y_{in}^{[n+1]}$, we introduce a transformation matrix $D(\theta)$. It has the form

$$D(\theta) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_n^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & \theta_n^p \end{bmatrix}.$$

We then have

$$y_{in}^{[n+1]} \approx D(\theta) y_{out}^{[n]}.$$

Zero stability with variable stepsize

Applying a new type of general linear method to the test problem $y'(x) = 0$, from step number n to step number $n+1$ we have

$$y^{[n+1]} = D(\theta) V y^{[n]}.$$

Note that we have $D(\theta)V$ in the equation compared with V for constant stepsize. Therefore we need to consider the effect of repeatedly multiplying by a matrix

$D(\theta)$ to matrix V in the integration interval. In the thesis, we do not carry out this analysis in full but consider only the question of repeatedly increasing the stepsize to the maximum value allowed in our code. This means that we need to consider the possible power-boundedness of $D(\theta)V$. We use the following example to illustrate the analysis.

Example 3.5 *Consider matrix V in the order 3 method given as an example in Chapter 2. Matrix V has the form*

$$V = \begin{bmatrix} 1 & -\frac{341047}{162000} & -\frac{116611}{1166400} & -\frac{619133}{20995200} \\ 0 & -\frac{13}{90} & \frac{13}{324} & -\frac{91}{5832} \\ 0 & -\frac{13}{25} & \frac{13}{90} & -\frac{91}{1620} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

We calculate the four eigenvalues of $D(\theta)V$. They are 1, 0, 0 and $\frac{13}{90}\theta(\theta-1)$. If $\frac{13}{90}\theta(\theta-1) < 1$, that is $\theta < 3.18$, then all the eigenvalues are less than or equal to 1 and the zero stability is satisfied. Therefore, $\theta_{max} = 2$ is a safe choice.

3.5 Starting methods

Recall that when we construct these general linear methods we use Nordsieck form for the quantities passed between steps. For an order p method, the first incoming approximation is

$$\begin{bmatrix} y(x_0) \\ h_0 y'(x_0) \\ \vdots \\ h_0^p y^{(p)}(x_0) \end{bmatrix}.$$

The problems that we wish to work on are initial value problems. There is only one initial value, which is $y(x_0) = y_0$, available. There are two ways to implement these methods. One way is variable order. In variable order implementation, we can start with order 1 or order 2 methods which require only this known initial value. The alternative approach is to use a fixed order implementation. Due to the complexity in the implementation of variable order and our aim of performing experiments on these methods, therefore, we mainly concentrate on implementing fixed order general linear methods in this thesis. In this case, to produce these first incoming approximation values using the given initial value, for each order, we need to construct a starting method which has the form

$$\left[\begin{array}{c|c} \widehat{A} & \widehat{U} \\ \hline \widehat{B} & \widehat{V} \end{array}\right].$$

The starting method may also advance the solution through one single step. This step can take the solution to $x_0 + h_0$ or approximate the solution at x_0 depending on the design of the starting method. In this thesis, the modified singly implicit Runge-Kutta methods with $p+1$ outputs are used as the starting methods. The idea of Runge-Kutta methods with multiple outputs was suggested by Gear [24]. In our case, the starting method should use the same λ value as the main method. Since we have only y_0 available, the matrices, $\widehat{U}$ and $\widehat{V}$ need to be chosen as $\widehat{U} = [1, 1, \cdots, 1]^T$ and $\widehat{V} = [1, 0, \cdots, 0]^T$.

The following example illustrates how to construct a starting method.

Example 3.6 *We want to build a four stage starting method for a four stage order* 3 *general linear method which is regarded as the main method. The* λ *value for the main method is* $\frac{1}{4}$. *Consider matrix* $\widehat{A}$ *which has the form*

$$\widehat{A} = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ \hat{a}_{21} & \lambda & 0 & 0 \\ \hat{a}_{31} & \hat{a}_{32} & \lambda & 0 \\ \hat{a}_{41} & \hat{a}_{42} & \hat{a}_{43} & \lambda \end{bmatrix},$$

and the abscissae vector $\hat{c} = [\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4]^T$. *To provide an order* 3 *first approximation for the main method, the stage orders of the starting method are chosen to be* 1, 2, 2 *and* 2 *respectively. According to the order conditions we have:*

- *for stage* 1, *the stage order is* 1, *and we have*

$$\hat{c}_1 = \lambda;$$

- *for stage* 2, *the stage order is* 2, *and we have*

$$\begin{aligned} \hat{a}_{21} + \lambda &= \hat{c}_2, \\ \hat{a}_{21}\lambda + \lambda\hat{c}_2 &= \frac{1}{2}\hat{c}_2^2; \end{aligned}$$

- *for stage* 3, *the stage order is* 2, *and we have*

$$\begin{aligned} \hat{a}_{31} + \hat{a}_{32} + \lambda &= \hat{c}_3, \\ \hat{a}_{31}\hat{c}_1 + \hat{a}_{32}\hat{c}_2 + \lambda\hat{c}_3 &= \frac{1}{2}\hat{c}_3^2; \end{aligned}$$

- *for stage* 4, *the stage order is* 2, *we have*

$$\begin{aligned}\hat{a}_{41} + \hat{a}_{42} + \hat{a}_{43} + \lambda &= \hat{c}_4,\\ \hat{a}_{41}\hat{c}_1 + \hat{a}_{42}\hat{c}_2 + \hat{a}_{43}\hat{c}_3 + \lambda\hat{c}_4 &= \frac{1}{2}\hat{c}_4^2.\end{aligned}$$

Since there are 7 *equations and* 10 *unknowns, we have three free parameters. Solving the first three equations we get* $\hat{c}_1 = \frac{1}{4}$, $\hat{c}_2 = \frac{1}{2} - \frac{\sqrt{2}}{4}$ *and* $\hat{a}_{21} = \frac{1}{4} - \frac{\sqrt{2}}{4}$. *For the free parameters, we choose* $\hat{c}_3 = \frac{1}{3}$, $\hat{c}_4 = 1$ *and* $\hat{a}_{42} = 0$ *for simplicity and convinence, hence we have* $\hat{a}_{31} = -\frac{4+7\sqrt{2}}{36}$, $\hat{a}_{32} = \frac{7+7\sqrt{2}}{36}$ *and* $\hat{a}_{43} = \frac{3}{4}$. *As a result,* $\hat{c}$ *of the starting method is* $[\frac{1}{4}, \frac{1}{2} - \frac{\sqrt{2}}{4}, \frac{1}{3}, 1]^T$ *and matrix* $\widehat{A}$ *is*

$$\begin{bmatrix} \frac{1}{4} & 0 & 0 & 0\\ \frac{1}{4} - \frac{\sqrt{2}}{4} & \frac{1}{4} & 0 & 0\\ -\frac{4+7\sqrt{2}}{36} & \frac{7+7\sqrt{2}}{36} & \frac{1}{4} & 0\\ 0 & 0 & \frac{3}{4} & \frac{1}{4}\end{bmatrix}.$$

The next step is to decide the coefficients of $\hat{B}$. As in the main method, $y^{[0]}$ is calculated by the following equation,

$$y^{[0]} = \hat{B}hf(Y) + \hat{V}y_0.$$

We wish the elements of $y^{[0]}$ to be accurate to at least order 3. Therefore, the information from the first stage is not to be used since stage 1 only has the order of 1. As a result, the matrix $\hat{B}$ is of the form

$$\hat{B} = \begin{bmatrix} 0 & \hat{b}_{12} & \hat{b}_{13} & \hat{b}_{14}\\ 0 & \hat{b}_{22} & \hat{b}_{23} & \hat{b}_{24}\\ 0 & \hat{b}_{32} & \hat{b}_{33} & \hat{b}_{34}\\ 0 & \hat{b}_{42} & \hat{b}_{43} & \hat{b}_{44}\end{bmatrix}.$$

By calculating the first component of the output value, $y_1^{[0]}$, we have

$$y_1^{[0]} = y(x_0 + rh) = \hat{b}_{12}hf(Y_2) + \hat{b}_{13}hf(Y_3) + \hat{b}_{14}hf(Y_4) + y_0.$$

Note that this formula gives an approximation for variable r. For example, if $r = 0$, the output values will be on the original point x_0 and if $r = 1$, the output values will be on the first point $x_1 = x_0 + h$.

Using the order conditions, and expanding each $f(Y_i)$ in Taylor series, we equate both sides and get

$$\begin{aligned} \sum_{i=2}^{4} \hat{b}_{1i} &= r, \\ \sum_{i=2}^{4} \hat{b}_{1i}\hat{c}_i &= \frac{r^2}{2!}, \\ \sum_{i=2}^{4} \hat{b}_{1i}\frac{\hat{c}_i^2}{2!} &= \frac{r^3}{3!}. \end{aligned}$$

For the second component of the output value, $y_2^{[0]}$, we have

$$y_2^{[0]} = hy'(x_0) = \hat{b}_{22}hf(Y_2) + \hat{b}_{23}hf(Y_3) + \hat{b}_{24}hf(Y_4).$$

As for the first component, we have following equations

$$\begin{aligned} \sum_{i=2}^{4} \hat{b}_{2i} &= 1, \\ \sum_{i=2}^{4} \hat{b}_{2i}\hat{c}_i &= r, \\ \sum_{i=2}^{4} \hat{b}_{2i}\frac{\hat{c}_i^2}{2!} &= \frac{r^2}{2!}. \end{aligned}$$

For the third and fourth components, $y_3^{[0]}$ and $y_4^{[0]}$, we have

$$\sum_{i=2}^{4} \hat{b}_{3i} = 0,$$

$$\begin{aligned}
\sum_{i=2}^{4} \hat{b}_{3i}\hat{c}_i &= 1, \\
\sum_{i=2}^{4} \hat{b}_{3i}\frac{\hat{c}_i^2}{2!} &= r,
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i=2}^{4} \hat{b}_{4i} &= 0, \\
\sum_{i=2}^{4} \hat{b}_{4i}\hat{c}_i &= 0, \\
\sum_{i=2}^{4} \hat{b}_{4i}\frac{\hat{c}_i^2}{2!} &= 1.
\end{aligned}$$

There are 12 equations and 12 unknowns, so we can solve the linear equation system to get the elements of $\hat{B}$. If $r = 0$, the elements of $\hat{B}$ are

$$\hat{B}_{r=0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -\frac{8}{7} + \frac{16\sqrt{2}}{7} & \frac{9}{14} - \frac{9\sqrt{2}}{7} & \frac{3}{2} - \sqrt{2} \\ 0 & \frac{32}{7} - \frac{64\sqrt{2}}{7} & \frac{27}{14} + \frac{36\sqrt{2}}{7} & -\frac{13}{2} + 4\sqrt{2} \\ 0 & -\frac{48}{7} + \frac{96\sqrt{2}}{7} & -\frac{36}{7} - \frac{54\sqrt{2}}{7} & 12 - 6\sqrt{2} \end{bmatrix}.$$

If $r = 1$, $\hat{B}$ is

$$\hat{B}_{r=1} = \begin{bmatrix} 0 & 0 & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{16}{7} + \frac{32\sqrt{2}}{7} & -\frac{45}{14} - \frac{18\sqrt{2}}{7} & \frac{11}{2} - 2\sqrt{2} \\ 0 & -\frac{48}{7} + \frac{96\sqrt{2}}{7} & -\frac{36}{7} - \frac{54\sqrt{2}}{7} & 12 - 6\sqrt{2} \end{bmatrix}.$$

Example 3.7 *The starting method for the five stage order four main method. The λ value is $\frac{1}{4}$ for the main method. To have order 4 first approximations,*

$y^{[0]}$, we wish to build up the order stage by stage. Therefore, we choose the stage orders of the starting method to be 1, 2, 2, 3, 3, 3 and 3 respectively. Note that for stage 4, we have

$$Y_4 = \hat{a}_{41}hf(Y_1) + \hat{a}_{42}hf(Y_2) + \hat{a}_{43}hf(Y_3) + \lambda hf(Y_4) + y_0.$$

We know that the error in $hf(Y_1)$ depends on $O(h^3)$ since the error in Y_1 is $O(h^2)$. We wish Y_4 to have stage order 3, which means that the error in Y_4 should be $O(h^4)$. To reach this goal, we do not want to use information from Y_1. In this case, $\hat{a}_{41} = 0$. The same principle can be applied to stages 5, 6 and 7, that is $\hat{a}_{51} = 0$, $\hat{a}_{61} = 0$ and $\hat{a}_{71} = 0$. Under these assumptions, matrix $\widehat{A}$ has the form

$$\widehat{A} = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ \hat{a}_{21} & \lambda & 0 & 0 & 0 & 0 & 0 \\ \hat{a}_{31} & \hat{a}_{32} & \lambda & 0 & 0 & 0 & 0 \\ 0 & \hat{a}_{42} & \hat{a}_{43} & \lambda & 0 & 0 & 0 \\ 0 & \hat{a}_{52} & \hat{a}_{53} & \hat{a}_{54} & \lambda & 0 & 0 \\ 0 & \hat{a}_{62} & \hat{a}_{63} & \hat{a}_{64} & \hat{a}_{65} & \lambda & 0 \\ 0 & \hat{a}_{72} & \hat{a}_{73} & \hat{a}_{74} & \hat{a}_{75} & \hat{a}_{76} & \lambda \end{bmatrix}.$$

and $\hat{c} = [\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4, \hat{c}_5, \hat{c}_6, \hat{c}_7]^T$.

Using the order conditions we have

- *for stage 1, the stage order is 1, and we have*

$$\hat{c}_1 = \lambda;$$

- *for stage* 2*, the stage order is* 2*, and we have*

$$\begin{aligned}\hat{a}_{21} + \lambda &= \hat{c}_2,\\ \hat{a}_{21}\lambda + \lambda\hat{c}_2 &= \frac{1}{2}\hat{c}_2^2;\end{aligned}$$

- *for stage* 3*, the stage order is* 2*, and we have*

$$\begin{aligned}\hat{a}_{31} + \hat{a}_{32} + \lambda &= \hat{c}_3,\\ \hat{a}_{31}\hat{c}_1 + \hat{a}_{32}\hat{c}_2 + \lambda\hat{c}_3 &= \frac{1}{2}\hat{c}_3^2;\end{aligned}$$

- *for stage* 4*, the stage order is* 3*, and we have*

$$\begin{aligned}\hat{a}_{42} + \hat{a}_{43} + \lambda &= \hat{c}_4,\\ \hat{a}_{42}\hat{c}_2 + \hat{a}_{43}\hat{c}_3 + \lambda\hat{c}_4 &= \frac{1}{2}\hat{c}_4^2,\\ \hat{a}_{42}\hat{c}_2^2 + \hat{a}_{43}\hat{c}_3^2 + \lambda\hat{c}_4^2 &= \frac{1}{3}\hat{c}_4^3;\end{aligned}$$

- *for stage* 5*, the stage order is* 3*, and we have*

$$\begin{aligned}\hat{a}_{52} + \hat{a}_{53} + \hat{a}_{54} + \lambda &= \hat{c}_5,\\ \hat{a}_{52}\hat{c}_2 + \hat{a}_{53}\hat{c}_3 + \hat{a}_{54}\hat{c}_4 + \lambda\hat{c}_5 &= \frac{1}{2}\hat{c}_5^2,\\ \hat{a}_{52}\hat{c}_2^2 + \hat{a}_{53}\hat{c}_3^2 + \hat{a}_{54}\hat{c}_4^2 + \lambda\hat{c}_5^2 &= \frac{1}{3}\hat{c}_5^3;\end{aligned}$$

- *for stage* 6*, the stage order is* 3*, and we have*

$$\begin{aligned}\hat{a}_{62} + \hat{a}_{53} + \hat{a}_{64} + \hat{a}_{65} + \lambda &= \hat{c}_6,\\ \hat{a}_{62}\hat{c}_2 + \hat{a}_{63}\hat{c}_3 + \hat{a}_{64}\hat{c}_4 + \hat{a}_{65}\hat{c}_5 + \lambda\hat{c}_6 &= \frac{1}{2}\hat{c}_6^2,\\ \hat{a}_{62}\hat{c}_2^2 + \hat{a}_{63}\hat{c}_3^2 + \hat{a}_{64}\hat{c}_4^2 + \hat{a}_{65}\hat{c}_5^2 + \lambda\hat{c}_6^2 &= \frac{1}{3}\hat{c}_6^3;\end{aligned}$$

- *for stage* 7*, the stage order is* 3*, and we have*

$$\begin{aligned}
\hat{a}_{72}+\hat{a}_{73}+\hat{a}_{74}+\hat{a}_{75}+\hat{a}_{76}+\lambda &= \hat{c}_7,\\
\hat{a}_{72}\hat{c}_2+\hat{a}_{73}\hat{c}_3+\hat{a}_{74}\hat{c}_4+\hat{a}_{75}\hat{c}_5+\hat{a}_{76}\hat{c}_6+\lambda\hat{c}_7 &= \frac{1}{2}\hat{c}_7^2,\\
\hat{a}_{72}\hat{c}_2^2+\hat{a}_{73}\hat{c}_3^2+\hat{a}_{74}\hat{c}_4^2+\hat{a}_{75}\hat{c}_5^2+\hat{a}_{76}\hat{c}_6^2+\lambda\hat{c}_7^2 &= \frac{1}{3}\hat{c}_7^3.
\end{aligned}$$

Solving the equations from stage 1 *and stage* 2*, we get* $\hat{c}_1$*, and* $\hat{a}_{21}$*. We choose* $\hat{c}_4 = \frac{1}{4}$*, and solve equations from stage* 3 *and stage* 4 *to get* $\hat{a}_{31}$*,* $\hat{a}_{32}$*,* $\hat{c}_3$*,* $\hat{a}_{42}$ *and* $\hat{a}_{43}$*. Let* $\hat{c}_5 = \frac{1}{2}$*, and solve equations from stage* 5 *to get* $\hat{a}_{52}$*,* $\hat{a}_{53}$ *and* $\hat{a}_{54}$*. For the equations from stage* 6*, we have two free parameters. Let* $\hat{c}_6 = \frac{3}{4}$ *and* $\hat{a}_{62} = 0$ *and solve the equations to get* $\hat{a}_{63}$*,* $\hat{a}_{64}$ *and* $\hat{a}_{65}$*. For stage* 7*, let* $\hat{c}_7 = 1$*,* $\hat{a}_{72} = 0$ *and* $\hat{a}_{73} = 0$*, and solve the equations to get* $\hat{a}_{74}$*,* $\hat{a}_{75}$ *and* $\hat{a}_{76}$*. Therefore,* $\hat{c}$ *is*

$$\hat{c} = \left[\frac{1}{4}, \frac{1}{2}-\frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4}-\frac{1}{6}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right]^T,$$

and $\widehat{A}$ *is of the form*

$$\begin{bmatrix}
\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{1}{4}-\frac{\sqrt{2}}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0\\
-\frac{31}{36}+\frac{17\sqrt{2}}{36} & \frac{4}{9}-\frac{2\sqrt{2}}{9} & \frac{1}{4} & 0 & 0 & 0 & 0\\
0 & \frac{3}{8}+\frac{9\sqrt{2}}{32} & -\frac{3}{8}-\frac{9\sqrt{2}}{32} & \frac{1}{4} & 0 & 0 & 0\\
0 & -\frac{9}{8}-\frac{3\sqrt{2}}{4} & \frac{129}{56}+\frac{45\sqrt{2}}{28} & -\frac{13}{14}-\frac{6\sqrt{2}}{7} & \frac{1}{4} & 0 & 0\\
0 & 0 & -\frac{261}{1288}-\frac{351\sqrt{2}}{2576} & \frac{25}{28}+\frac{9\sqrt{2}}{56} & -\frac{35}{184}-\frac{9\sqrt{2}}{368} & \frac{1}{4} & 0\\
0 & 0 & 0 & \frac{5}{12} & \frac{5}{12} & -\frac{1}{12} & \frac{1}{4}
\end{bmatrix}.$$

As we did for the order 3 *method, we wish the elements of* $y^{[0]}$ *to be accurate to* *at least order* 4*. In this case, the matrix* $\widehat{B}$ *is of the form*

$$\widehat{B} = \begin{bmatrix} 0 & 0 & 0 & \hat{b}_{14} & \hat{b}_{15} & \hat{b}_{16} & \hat{b}_{17} \\ 0 & 0 & 0 & \hat{b}_{24} & \hat{b}_{25} & \hat{b}_{26} & \hat{b}_{27} \\ 0 & 0 & 0 & \hat{b}_{34} & \hat{b}_{35} & \hat{b}_{36} & \hat{b}_{37} \\ 0 & 0 & 0 & \hat{b}_{44} & \hat{b}_{45} & \hat{b}_{46} & \hat{b}_{47} \\ 0 & 0 & 0 & \hat{b}_{54} & \hat{b}_{55} & \hat{b}_{56} & \hat{b}_{57} \end{bmatrix}.$$

As for the starting method for order 3 *method, we obtain the following equations for* $\widehat{B}$ *using the order conditions.*

For the first component of the output vector, $y_1^{[0]} = y(x_0 + rh)$*, and we have*

$$\begin{aligned} \sum_{i=4}^{7} \hat{b}_{1i} &= r, \\ \sum_{i=4}^{7} \hat{b}_{1i}\hat{c}_i &= \frac{r^2}{2!}, \\ \sum_{i=4}^{7} \hat{b}_{1i}\frac{\hat{c}_i^2}{2!} &= \frac{r^3}{3!}, \\ \sum_{i=4}^{7} \hat{b}_{1i}\frac{\hat{c}_i^3}{3!} &= \frac{r^4}{4!}. \end{aligned}$$

For the second component of the output vector, $y_2^{[0]} = hy'(x_0 + rh)$*, and we have*

$$\begin{aligned} \sum_{i=4}^{7} \hat{b}_{2i} &= 1, \\ \sum_{i=4}^{7} \hat{b}_{2i}\hat{c}_i &= r, \\ \sum_{i=4}^{7} \hat{b}_{2i}\frac{\hat{c}_i^2}{2!} &= \frac{r^2}{2!}, \end{aligned}$$

$$\sum_{i=4}^{7} \hat{b}_{2i}\frac{\hat{c}_i^3}{3!} = \frac{r^3}{3!}.$$

For the third component of the output vector, $y_3^{[0]} = h^2 y''(x_0 + rh)$, *and we have*

$$\begin{aligned}
\sum_{i=4}^{7} \hat{b}_{3i} &= 0,\\
\sum_{i=4}^{7} \hat{b}_{3i}\hat{c}_i &= 1,\\
\sum_{i=4}^{7} \hat{b}_{3i}\frac{\hat{c}_i^2}{2!} &= r,\\
\sum_{i=4}^{7} \hat{b}_{3i}\frac{\hat{c}_i^3}{3!} &= \frac{r^2}{2!}.
\end{aligned}$$

For the fourth component of the output vector, $y_4^{[0]} = h^3 y'''(x_0 + rh)$, *and we have*

$$\begin{aligned}
\sum_{i=4}^{7} \hat{b}_{4i} &= 0,\\
\sum_{i=4}^{7} \hat{b}_{4i}\hat{c}_i &= 0,\\
\sum_{i=4}^{7} \hat{b}_{4i}\frac{\hat{c}_i^2}{2!} &= 1,\\
\sum_{i=4}^{7} \hat{b}_{4i}\frac{\hat{c}_i^3}{3!} &= r.
\end{aligned}$$

For the fifth component of the output vector, $y_5^{[0]} = h^4 y^{(4)}(x_0 + rh)$, *and we have*

$$\begin{aligned}
\sum_{i=4}^{7} \hat{b}_{5i} &= 0,\\
\sum_{i=4}^{7} \hat{b}_{5i}\hat{c}_i &= 0,
\end{aligned}$$

$$\begin{aligned}\sum_{i=4}^{7} \hat{b}_{5i}\frac{\hat{c}_i^2}{2!} &= 0,\\ \sum_{i=4}^{7} \hat{b}_{5i}\frac{\hat{c}_i^3}{3!} &= 1.\end{aligned}$$

Solving the linear equation systems we get the elements of $\widehat{B}$. If $r = 0$, $\widehat{B}$ is

$$\widehat{B}_{r=0} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & -6 & 4 & -1 \\ 0 & 0 & 0 & -\frac{52}{3} & 38 & -28 & \frac{22}{3} \\ 0 & 0 & 0 & 48 & -128 & 112 & -32 \\ 0 & 0 & 0 & -64 & 192 & -192 & 64 \end{bmatrix}.$$

If $r = 1$,

$$\widehat{B}_{r=1} = \begin{bmatrix} 0 & 0 & 0 & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{4}{3} & 6 & -12 & \frac{22}{3} \\ 0 & 0 & 0 & -16 & 64 & -80 & 32 \\ 0 & 0 & 0 & -64 & 192 & -192 & 64 \end{bmatrix}.$$

For each main method, we can find a matching starting method. When building a starting method for a higher order method, we need to use increasingly many stages to build up the desired order for the output Nordsieck vector.

3.6 Interpolation

One may need an output value at a specific point x. Since we use variable stepsize, we cannot calculate the values exactly at that point unless the next stepsize, h, is forced to fit in the distance between x_n and x. One way to calculate the output value at point x is that a successful step from x_n to x_{n+1} is achieved and x lies between x_n and x_{n+1}, and then force $h = x - x_n$ and do one more step. The disadvantages of doing so are (a) if we need a lot of output points it becomes inefficient to force steps to fit in for output values; (b) as we need to take steps based on long time performance, forcing steps will interfere with this main aim. Therefore, interpolation is desirable to approximate the values for the output values. We assume that the solutions at x_n and x_{n+1}, which are $y(x_n)$ and $y(x_{n+1})$, have already been approximated using stepsize h, and point x lies between x_n and x_{n+1}. Let

$$h_1 = x - x_n, \quad h_2 = x_{n+1} - x.$$

We use Figure 3.4 to present the three points, x, x_n and x_{n+1}.

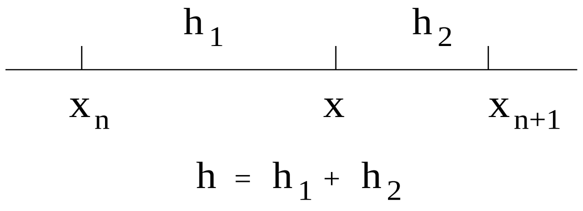

Figure 3.4: An output point between x_n and x_{n+1}.

In order to estimate the solution at point x, we use the information collected at

x_n and x_{n+1}. The Nordsieck vectors $y^{[n]}$ and $y^{[n+1]}$ have been calculated for the points x_n and x_{n+1}. Note that we need to transform the Nordsieck vector $y^{[n]}$ into $y^{[n]}_{in}$ with current stepsize h, which was discussed in the section of stepsize control. One natural way to achieve this is that we can use Taylor expansions of $y(x) = y(x_n + h_1)$ or $y(x) = y(x_{n+1} - h_2)$ to approximate the solution at point x. However, this approach only uses the information at one point to calculate $y(x)$. In order to obtain a good continuity (for example, continuity of y' at step points), we consider using $y(x_n)$, $hy'(x_n)$, $y(x_{n+1})$ and $hy'(x_{n+1})$. Let

$$y(x) = ay(x_n) + by(x_{n+1}) + chy'(x_n) + dhy'(x_{n+1}),$$

where a, b, c and d are coefficients to be determined. By expanding $y(x_n)$, $hy'(x_n)$, $y(x_{n+1})$ and $hy'(x_{n+1})$ in a Taylor series around x, we then have

$$\begin{aligned} y(x) &= a\left[y(x) - h_1y'(x) + \frac{h_1^2}{2}y''(x) - \frac{h_1^3}{3!}y'''(x)\right] \\ &+ b\left[y(x) + h_2y'(x) + \frac{h_2^2}{2}y''(x) + \frac{h_2^3}{3!}y'''(x)\right] \\ &+ ch\left[y'(x) - h_1y''(x) + \frac{h_1^2}{2}y'''(x)\right] \\ &+ dh\left[y'(x) + h_2y''(x) + \frac{h_2^2}{2}y'''(x)\right] + O(h^4). \end{aligned} \tag{3.10}$$

Introducing a variable $t = (x - x_n)/(x_{n+1} - x_n)$, which leads to $h_1 = th$ and $h_2 = (1-t)h$, we have the following equation system by equating both sides of equation (3.10),

$$\begin{aligned} a + b &= 1, \\ -t + b + c + d &= 0, \\ \frac{t^2}{2} + b\frac{1-2t}{2} - ct + d(1-t) &= 0, \\ \frac{-at^3 + b(1-t)^3}{3!} + \frac{ct^2 + d(1-t)^2}{2} &= 0. \end{aligned}$$

Solving the above equations we get

$$\begin{aligned} a &= 2t^3 - 3t^2 + 1, \\ b &= 3t^2 - 2t^3, \\ c &= t(1-t)^2, \\ d &= t^2(t-1). \end{aligned}$$

This interpolation scheme will be referred to as the first approach and can be used with any method.

In this thesis, we also try another approach which depends on the order of the methods. For an order 1 method, use $y(x_n)$ and $y(x_{n+1})$ to approximate $y(x)$. For order 2 and order 3 methods, use $y(x_n)$, $hy'(x_n)$, $y(x_{n+1})$ and $hy'(x_{n+1})$ to approximate $y(x)$. For order 4 and order 5 methods, use $y(x_n)$, $hy'(x_n)$, $h^2y''(x_n)$, $y(x_{n+1})$, $hy'(x_{n+1})$ and $h^2y''(x_{n+1})$ to approximate $y(x)$ and so on. This approach is called the second approach. Both approaches are actually based on Hermite interpolation formulae. Now we give an example to illustrate the second approach for an order 4 method.

Example 3.8 *Consider an order* 4 *method. Let* $x = x_n + th$ *where* $t \in (0,1)$ *and* $t = (x - x_n)/(x_{n+1} - x_n)$. *We use the following formula*

$$\begin{aligned} y(x) &= c_0(th)y(x_n) + c_1(th)hy'(x_n) + c_2(th)h^2y''(x_n) \\ &+ d_0(th)y(x_{n+1}) + d_1(th)hy'(x_{n+1}) + d_2(th)h^2y''(x_{n+1}), \end{aligned}$$

where c_i *and* d_i, $i = 0, 1, 2$ *are functions to be determined. Without loss of generality, we can set* $h = 1$, $x_n = 0$ *and* $x_{n+1} = 1$, *and then have*

$$y(t) = c_0(t)y(x_n) + c_1(t)y'(x_n) + c_2(t)y''(x_n) \tag{3.11}$$

$$+ \quad d_0(t)y(x_{n+1}) + d_1(t)y'(x_{n+1}) + d_2(t)y''(x_{n+1}),$$

where $y(t)$ is a polynomial of degree 5. Therefore, functions, $y(x) = 1$, $y(x) = x$, $y(x) = x^2$, $y(x) = x^3$, $y(x) = x^4$ and $y(x) = x^5$ satisfy equation (3.11). This leads to

$$\begin{aligned}
c_0 + d_0 &= 1,\\
c_1 + d_0 + d_1 &= t,\\
2c_2 + d_0 + 2d_2 + 2d_2 &= t^2,\\
d_0 + 3d_1 + 6d_2 &= t^3,\\
d_0 + 4d_1 + 12d_2 &= t^4,\\
d_0 + 5d_1 + 20d_2 &= t^5.
\end{aligned}$$

Solving this linear equation system, we have

$$\begin{aligned}
c_0 &= 1 - 10t^3 + 15t^4 - 6t^5,\\
c_1 &= t - 6t^3 + 8t^4 - 3t^5,\\
c_2 &= \frac{t^2 - 3t^3 + 3t^4 - t^5}{2},\\
d_0 &= 10t^3 - 15t^4 + 6t^5,\\
d_1 &= -4t^3 + 7t^4 - 3t^5,\\
d_2 &= \frac{t^3 - 2t^4 + t^5}{2}.
\end{aligned}$$

Numerical experiments on the two different interpolations will be presented in Chapter 4.

3.7 Test problems

Because this type of general linear method we introduced in the thesis is new, how well they perform in practice is still unknown. It is worthwhile to test them with some well known standard problems. Here we test the methods with the following three stiff problems.

- A stiff problem proposed by Prothero and Robinson [33], has the form

$$y'(x) = L(y - \phi(x)) + \phi'(x), \quad y_0 = y(x_0) = \phi(x_0),$$

where $\mathrm{Re}(L) \leq 0$. In our experiments we choose $L = -10^6$ and $\phi(x) = \sin(x)$.

- The Robertson problem introduced in Chapter 1.

- The Hires problem which was proposed by Schafer in 1975 [36]. It is known as 'High irradiance response'. The detailed description of this problem can be found in the reference paper. In mathematical language, the Hires problem is a stiff system of 8 dimensions. The problem is of the form

$$\begin{aligned}
\dot{y_1} &= -1.71y_1 + 0.43y_2 + 8.32y_3 + 0.00007, \\
\dot{y_2} &= 1.71y_1 - 8.75y_2, \\
\dot{y_3} &= -10.03y_3 + 0.43y_4 + 0.035y_5, \\
\dot{y_4} &= 8.32y_2 + 0.171y_3 - 1.12y_4, \\
\dot{y_5} &= -1.745y_5 + 0.43y_6 + 0.43y_7, \\
\dot{y_6} &= -280y_6y_8 + 0.69y_4 + 1.71y_5 - 0.43y_6 + 0.69y_7, \\
\dot{y_7} &= 280y_6y_8 - 1.81y_7,
\end{aligned}$$

$$\dot{y_8} = -280y_6y_8 + 1.81y_7,$$

with $0 \leq x \leq 321.8122$. The initial value is given by $y_0 = [1, 0, 0, 0, 0, 0, 0, 0.0057]^T$. The reference solution at the end of the integration interval ($x_{end} = 321.8122$) is from [30]. In [30], the solution is calculated by using RADAU5 with very small tolerance. The reference solution is

$$\begin{aligned}
y_{end}(1) &= 7.371312573325668 \times 10^{-4},\\
y_{end}(2) &= 1.442485726316185 \times 10^{-4},\\
y_{end}(3) &= 5.888729740967575 \times 10^{-5},\\
y_{end}(4) &= 1.175651343283149 \times 10^{-3},\\
y_{end}(5) &= 2.386356198831331 \times 10^{-3},\\
y_{end}(6) &= 6.238968252742796 \times 10^{-3},\\
y_{end}(7) &= 2.849998395185769 \times 10^{-3},\\
y_{end}(8) &= 2.850001604814231 \times 10^{-3}.
\end{aligned}$$

The numerical results will be presented in Chapter 4.

Chapter 4

Numerical experiments

In Chapter 2, we discussed the properties of the new type of general linear methods and presented the construction of these new methods. In Chapter 3, we discussed details of the implementation scheme and designed strategies for these methods. The focus of this Chapter is to experiment with implementation techniques using three standard test problems and we hope to draw conclusions about the practical performance of this type of general linear method for stiff problems.

4.1 Constant stepsize

To understand how these new general linear methods perform, we first use constant stepsize. With constant stepsize we are able to verify the accuracy and order of the methods. The test problem used here is the Prothero and Robinson problem [33]. It has the form of

$$y'(x) = L(y - \phi(x)) + \phi'(x), \quad y_0 = y(x_0) = \phi(x_0),$$

where $\mathrm{Re}(L) \leq 0$. It has the exact solution $y(x) = \phi(x)$. We choose $\phi(x) = \sin(x)$ and $L = -10^6$, which makes the problem stiff. The global error is calculated at the end point by comparing the numerical solution with the exact solution. Tables 4.1, 4.2 and 4.3 show the global errors versus the stepsizes at $x_{end} = 10$.

Full Newton iteration is used here since our aim of using a fixed stepsize is to verify the order of a method. The Prothero and Robinson problem is used to test the order reduction which occurs when we use some Runge-Kutta methods. Recall that $E = Ch^{p+1}y^{(p+1)} + O(h^{p+2})$. In the following tables, we report error divided by h^p in the third column. For each method, the numbers in third column have the same order. This verifies the order of the method is as excepted. As we discussed in Chapter 2, for this type of method, in order to avoid the order reduction we choose the stage order equal to the order of the method. Our numerical results confirm that there is no order reduction.

h	global error	global error$/h^2$
1	4.5×10^{-7}	4.5×10^{-7}
0.1	2.5×10^{-9}	2.5×10^{-7}
0.01	2.5×10^{-11}	2.5×10^{-7}
0.001	2.4×10^{-13}	2.4×10^{-7}
0.0001	2.2×10^{-15}	2.2×10^{-7}

Table 4.1: The errors versus stepsizes for an order 2 method.

h	global error	global error$/h^3$
1	3.2×10^{-8}	3.2×10^{-8}
0.1	2.7×10^{-11}	2.7×10^{-8}
0.01	3.1×10^{-14}	3.1×10^{-8}

Table 4.2: The errors versus stepsizes for an order 3 method.

h	global error	global error$/h^4$
1	3×10^{-8}	3.0×10^{-8}
0.1	4×10^{-12}	4.0×10^{-8}
0.01	3.3×10^{-16}	3.3×10^{-8}

Table 4.3: The errors versus stepsizes for an order 4 method.

4.2 Variable stepsize

Variable stepsizes are required for solving most ordinary differential equations for an efficient computation unless a specific problem requires a fixed stepsize. Variable stepsizes allow a solver to choose the most appropriate stepsize at a particular point for a given problem. In Chapter 3 we discussed how to change stepsize. To test the stepsize controller and the effects of other techniques we use these methods to solve the Robertson problem (Chapter 1). For this stiff problem, due to the physical nature of the problem, the three components have non-negative values and tend to 0, 0, 1, respectively as $x \to \infty$. In the following sections, we will use this problem to perform some experiments. Note that the parameters, such as the ratio of ϵ and $Tolerance$, θ_{min} and θ_{max}, are usually critically dependent on the test problem.

4.2.1 Effect of the initial stepsize

An initial stepsize, h_0, is needed when we use the stepsize controller. The choice of the initial stepsize will affect the total number of steps required for the integration interval. Too large an initial stepsize will cause too many rejected steps at the start of the integration while the solver tries to find a suitable stepsize, whereas if the initial stepsize is chosen to be too small then the total steps required to complete the integration interval will increase. The initial stepsize is problem dependent. Furthermore, a suitable initial stepsize depends not only on the tolerance used by the methods but also on the order of the methods. We perform the experiments on the Robertson problem to investigate the effects of the initial stepsize. For the Robertson problem, the stiffness occurs in a very short period of time. Therefore, most rejected steps lie in the first few steps. In our experiments, we investigate the first 20 steps. The numerical results are presented in the Tables 4.4, 4.5 and 4.6 for methods with different orders.

It is seen that for the Robertson problem, the initial stepsize should be small enough to avoid the rejections at the beginning for all three methods. Furthermore, for larger $Tolerance$ the initial stepsize can be bigger without too many rejected steps; for smaller $Tolerance$ we need to use a smaller initial stepsize to avoid rejected steps. To reduce the number of rejected steps, for the Robertson problem, we choose $h_0 = 10^{-4}$ as the initial stepsize for loose tolerance and

$h_0 = 10^{-6}$ for tight tolerance with methods of the order 2, 3 and 4.

$Tolerance$	h_0	Rejected steps
10^{-6}	10^{-2}	4
	10^{-3}	1
	10^{-4}	0
	10^{-5}	0
	10^{-6}	0
10^{-7}	10^{-2}	7
	10^{-3}	3
	10^{-4}	1
	10^{-5}	1
	10^{-6}	0
10^{-10}	10^{-2}	9
	10^{-3}	6
	10^{-4}	2
	10^{-5}	0
	10^{-6}	0

Table 4.4: The effects of initial stepsize, h_0, for an order 2 method.

$Tolerance$	h_0	Rejected steps
10^{-6}	10^{-2}	7
	10^{-3}	2
	10^{-4}	0
	10^{-5}	0
	10^{-6}	0
10^{-7}	10^{-2}	9
	10^{-3}	3
	10^{-4}	1
	10^{-5}	0
	10^{-6}	0
10^{-10}	10^{-2}	13
	10^{-3}	9
	10^{-4}	1
	10^{-5}	0
	10^{-6}	0

Table 4.5: The effects of initial stepsize, h_0, for an order 3 method.

$Tolerance$	h_0	Rejected steps
10^{-6}	10^{-2}	13
	10^{-3}	1
	10^{-4}	0
	10^{-5}	0
	10^{-6}	0
10^{-7}	10^{-2}	15
	10^{-3}	6
	10^{-4}	1
	10^{-5}	1
	10^{-6}	0
10^{-10}	10^{-2}	20
	10^{-3}	10
	10^{-4}	4
	10^{-5}	0
	10^{-6}	0

Table 4.6: The effects of initial stepsize, h_0, for an order 4 method.

To understand more about the effect of the initial stepsize, we check the stepsize for the first 20 steps. Using the order 2 method as an example, Table 4.7 shows how the stepsizes increase or decrease during the first 20 steps where $Tolerance = 10^{-10}$ and $h_0 = 0.001$.

step	h	θ
1	0.001	0.5
2	0.0005	0.5
3	0.00025	0.5
4	0.000125	0.5
5	0.0000625	0.5
6	0.00003125	0.7879
7	0.00002461	0.9989
$\vdots$	$\vdots$	$\vdots$

Table 4.7: The stepsize changing for $\theta_{min} = 0.5$ with $Tolerance = 10^{-10}$ and $h_0 = 0.001$.

The numerical results shows that for the first 6 rejected steps, the next stepsize is half of the current stepsize since we set the $\theta_{min} = 0.5$. It may be a feasible idea that we set a different limit of θ for the beginning of the calculation. Table 4.8 shows the result if we set $\theta_{min} = 0.25$ for first 5 steps.

We can see that the number of rejected steps has been reduced to 3. Therefore, for the case of having too big an initial stepsize, the value of θ_{min} can be chosen to be smaller than 0.5 for the first several steps. For the case of having too small an

step	h	θ
1	0.001	0.25
2	0.00025	0.25
3	0.0000625	0.25
4	0.000015625	1.571
5	0.00002455	0.999
$\vdots$	$\vdots$	$\vdots$

Table 4.8: The stepsize changing for $\theta_{min} = 0.25$ with $Tolerance = 10^{-10}$ and $h_0 = 0.001$.

initial stepsize, we do not want to set a different θ_{max} here since this θ_{max} affects stability of the method as we discussed in Chapter 3. If the initial stepsize is too small, the θ_{max} could be set to larger for some methods and some problems. The values of θ_{min} and θ_{max} are quite problem dependent. For a different problem, one needs to do some preliminary tests to find the optimal values of θ_{min} and θ_{max}.

4.2.2 Effect of ϵ and $Tolerance$

In Chapter 3, we introduced an error tolerance, ϵ, for the Newton procedure. This tolerance depends on the tolerance of the method, $Tolerance$, which is used in the stepsize controller. To increase the efficiency of the solver we try to find an optimal ϵ. Tables 4.9, 4.10 and 4.11 show the numerical results with various rates between ϵ and $Tolerance$ for methods with different orders.

For the Robertson problem, the results show that for the order 2 method and the order 3 method, there is almost no difference between $\epsilon = \frac{Tolerance}{10}$ and $\epsilon = \frac{Tolerance}{100}$; for order 4 method, $\epsilon = \frac{Tolerance}{100}$ is slightly better than $\epsilon = \frac{Tolerance}{10}$ in terms of rejected steps. Therefore, for the Robertson problem, we choose $\epsilon = \frac{Tolerance}{10}$, $\epsilon = \frac{Tolerance}{100}$ and $\epsilon = \frac{Tolerance}{100}$ for the methods with order of 2, 3 and 4 respectively.

ϵ	$Tolerance$	Total steps	Rejected steps
10^{-7}	10^{-6}	328	0
10^{-9}	10^{-8}	1428	1
10^{-11}	10^{-10}	6544	4
10^{-8}	10^{-6}	361	0
10^{-10}	10^{-8}	1428	1
10^{-12}	10^{-10}	6544	4

Table 4.9: Numerical results of $\epsilon = \frac{Tolerance}{10}$ and $\epsilon = \frac{Tolerance}{100}$ for an order 2 method.

ϵ	$Tolerance$	Total steps	Rejected steps
10^{-7}	10^{-6}	154	4
10^{-9}	10^{-8}	382	3
10^{-11}	10^{-10}	1134	1
10^{-8}	10^{-6}	153	3
10^{-10}	10^{-8}	382	3
10^{-12}	10^{-10}	1134	1

Table 4.10: Numerical results of $\epsilon = \frac{Tolerance}{10}$ and $\epsilon = \frac{Tolerance}{100}$ for an order 3 method.

ϵ	$Tolerance$	Total steps	Rejected steps
10^{-7}	10^{-6}	224	7
10^{-9}	10^{-8}	383	63
10^{-11}	10^{-10}	673	27
10^{-8}	10^{-6}	221	4
10^{-10}	10^{-8}	323	52
10^{-12}	10^{-10}	675	24

Table 4.11: Numerical results of $\epsilon = \frac{Tolerance}{10}$ and $\epsilon = \frac{Tolerance}{100}$ for an order 4 method.

4.2.3 Testing methods with the Robertson problem

There are always some limitations on a numerical method solver. Typical limitations include truncation errors from the method and rounding errors from the computer computation. For the Robertson problem, it is difficult to prevent the numerical results becoming negative when the end point of integration x_{end} becomes very large. We use this problem to test how large the x_{end} value can be before the values of y_{end} become negative with these solvers. Full Newton iteration is used in the experiments. The results are presented in Tables 4.12, 4.13 and 4.14. "x_{final}" represents the final x value where at least one element of y_{end} values turns negative; "steps" represents how many steps used to reach x_{final}.

It is seen that these new methods are very accurate and the stepsize controller works very well.

$Tolerance$	Steps	x_{final}
10^{-6}	358	4.3×10^{11}
10^{-8}	1530	5.1×10^{13}
10^{-10}	6958	4.3×10^{15}
10^{-12}	32131	1.9×10^{18}

Table 4.12: Solving Robertson problem using an order 2 method where $h_0 = 10^{-4}$ and $\epsilon = \frac{Tolerance}{10}$.

$Tolerance$	Steps	x_{final}
10^{-6}	179	1.3×10^{9}
10^{-8}	448	1.4×10^{12}
10^{-10}	1275	8.7×10^{13}
10^{-12}	3961	1.9×10^{16}

Table 4.13: Solving Robertson problem using an order 3 method where $h_0 = 10^{-4}$ and $\epsilon = \frac{Tolerance}{100}$.

$Tolerance$	Steps	x_{final}
10^{-6}	225	2.9×10^{9}
10^{-8}	337	2.9×10^{11}
10^{-10}	676	1.8×10^{13}
10^{-12}	1510	4.0×10^{15}

Table 4.14: Solving Robertson problem using an order 4 method where $h_0 = 10^{-4}$ and $\epsilon = \frac{Tolerance}{100}$.

4.3 Performance of the iteration scheme

In Chapter 3, we designed an iteration scheme for the Newton process to reduce the computational cost. Some experiments have been done to test this iteration scheme. The problem we use here is a more demanding problem, namely the Hires problem. We have introduced the Hires problem in Chapter 3.

In the following subsections, we will test the effect of the initial stepsize and the effect of ϵ and $Tolerance$ for this problem. Furthermore, we will test different predictors and interpolators.

To characterize a solver, we list the following details in the tables:

- h_0: the initial stepsize.
- steps: the total number of steps taken by the solver to reach x_{end}.
- rejected steps: the total number of rejected steps.
- #LU: the number of LU-decompositions.
- #J: the number of evaluations of the Jacobian.
- #f: the number of evaluations of the derivative functions.
- *scd*: the *scd* values which represent the minimum number of significant correct digits in the numerical solution at x_{end}, i.e.

$$scd = -log_{10}(||\text{relative error at } x_{end}||_\infty).$$

4.3.1 Effect of the initial stepsize

In the previous sections when we investigated the effect of the initial stepsize on the Robertson problem we used full Newton iteration. In this subsection, we apply the iteration scheme to the solvers, and then test the effect of the initial stepsize on the Hires problem. The numerical results are shown in Tables 4.15, 4.16 and 4.17.

Tolerance	h_0	total steps	Rejected steps	#LU	#J	#f	scd
10^{-4}	10^{-2}	66	6	38	8	718	1.56
	10^{-3}	69	6	37	11	741	1.63
	10^{-4}	72	6	39	11	727	1.57
10^{-7}	10^{-2}	488	5	57	5	3726	3.41
	10^{-3}	490	2	43	5	3647	3.41
	10^{-4}	493	2	47	5	3683	3.40
	10^{-5}	496	2	49	5	3702	3.41
10^{-10}	10^{-2}	4766	8	51	5	31090	5.47
	10^{-3}	4799	4	56	3	31348	5.46
	10^{-4}	4799	1	54	4	31616	5.46
	10^{-6}	4807	2	32	4	30798	5.46

Table 4.15: The effects of initial stepsize, h_0, for an order 2 method on the Hires problem.

Tolerance	h_0	total steps	Rejected steps	#LU	#J	#f	scd
	10^{-2}	259	82	118	46	3747	2.4
10^{-4}	10^{-3}	86	23	58	32	1310	2.89
	10^{-4}	64	5	45	20	924	1.77
	10^{-3}	244	54	91	45	3498	4.60
10^{-7}	10^{-4}	230	39	81	27	3291	5.10
	10^{-5}	238	46	92	41	3304	5.00
	10^{-3}	1004	122	222	78	12675	6.86
10^{-10}	10^{-4}	1055	139	233	86	13342	6.90
	10^{-6}	1043	134	230	81	13238	6.90

Table 4.16: The effects of initial stepsize, h_0, for an order 3 method on the Hires problem.

Tolerance	h_0	total steps	Rejected steps	#LU	#J	#f	scd
10^{-4}	10^{-2}	171	45	109	58	3429	1.80
	10^{-3}	86	14	71	35	1784	1.16
	10^{-4}	107	21	77	38	2205	2.59
10^{-7}	10^{-3}	189	44	122	63	3796	5.60
	10^{-4}	238	65	137	76	4748	6.07
	10^{-5}	246	69	141	79	4784	5.24
10^{-10}	10^{-3}	423	40	304	55	8946	7.42
	10^{-4}	472	65	261	78	9527	7.60
	10^{-6}	430	39	248	52	8714	7.84

Table 4.17: The effects of initial stepsize, h_0, for an order 4 method on the Hires problem.

It is seen that the smaller the tolerance is, the smaller the initial stepsize should be with less rejected steps and less LU decompositions. Therefore, we choose a slightly larger initial stepsize for bigger tolerance. We also note that, for the Hires problem, we need to make the initial stepsize smaller for higher order methods. In this case, we set $h_0 = 10^{-3}$ for tolerance 10^{-4} and 10^{-7}, $h_0 = 10^{-6}$ for tolerance 10^{-10} with order 2 method; $h_0 = 10^{-4}$ for tolerance 10^{-4} and 10^{-7}, $h_0 = 10^{-6}$ for tolerance 10^{-10} with order 3 and order 4 methods for the Hires problem.

4.3.2 Effect of ϵ and $Tolerance$

In this subsection we test the effect of ϵ and $Tolerance$ on the Hires problem. Results of numerical experiments are shown in Tables 4.18, 4.19 and 4.20.

The results show that for the order 2 method, it uses fewer LU factorizations and J evaluations with $\frac{Tolerance}{10}$ than with $\frac{Tolerance}{100}$. For the order 3 method, it has fewer total steps and rejected steps with $\frac{Tolerance}{100}$ than with $\frac{Tolerance}{10}$, however, for smaller tolerance (10^{-10}), it has less LU factorizations and J evaluations with $\frac{Tolerance}{10}$ than with $\frac{Tolerance}{100}$. For the order 4 method, only with $\frac{Tolerance}{100}$ it has slightly fewer total steps, rejected steps, LU factorizations and J evaluations, with other tolerance, $\frac{Tolerance}{1000}$ gives smaller numbers in terms of all the characters. We then set ϵ to be $\frac{1}{10}$, $\frac{1}{100}$ and $\frac{1}{1000}$ of $Tolerance$ for order 2, 3 and 4 methods respectively for Hires problem.

ϵ	$Tolerance$	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-5}	10^{-4}	73	6	55	24	635	1.61
10^{-8}	10^{-7}	530	0	76	8	3479	3.44
10^{-11}	10^{-10}	5232	3	192	5	32805	5.48
10^{-6}	10^{-4}	73	7	71	36	716	1.61
10^{-9}	10^{-7}	531	0	125	34	4407	3.46
10^{-12}	10^{-10}	5239	3	372	15	34427	5.49

Table 4.18: Results of $\epsilon = \frac{Tolerance}{10}$ versus $\epsilon = \frac{Tolerance}{100}$ for an order 2 method.

ϵ	$Tolerance$	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-5}	10^{-4}	197	49	58	16	2225	2.59
10^{-8}	10^{-7}	550	158	159	46	6479	5.02
10^{-11}	10^{-10}	1196	172	210	81	12425	6.93
10^{-6}	10^{-4}	99	31	67	30	1530	2.32
10^{-9}	10^{-7}	239	40	89	34	3316	5.09
10^{-12}	10^{-10}	1142	155	257	106	13754	6.96

Table 4.19: Results of $\epsilon = \frac{Tolerance}{10}$ versus $\epsilon = \frac{Tolerance}{100}$ for an order 3 method.

ϵ	$Tolerance$	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-6}	10^{-4}	1245	409	254	65	19045	3.15
10^{-9}	10^{-7}	267	72	121	57	4927	5.42
10^{-12}	10^{-10}	968	239	426	191	17250	7.58
10^{-7}	10^{-4}	106	22	73	33	2158	2.61
10^{-10}	10^{-7}	278	78	186	101	5696	5.49
10^{-13}	10^{-10}	472	56	359	155	10397	7.49

Table 4.20: Results of $\epsilon = \frac{Tolerance}{100}$ versus $\epsilon = \frac{Tolerance}{1000}$ for an order 4 method.

4.4 Performance of stage predictors

In Chapter 3, we proposed three predictors. To reduce the effect from the Newton iteration scheme, we test these predictors with full Newton iterations. For an order 2 method, we only have three stages, Taylor expansion predictor is good enough. For the order 3 and order 4 methods, we try all three predictors. The numerical results are reported in Tables 4.21, 4.22, 4.23, 4.24, 4.25 and 4.26. The details in the tables include: the total number of steps to reach x_{end}, the number of rejected steps, the *scd* values and the number of evaluations of the derivative functions. Since we use full Newton iteration, the number of LU-decompositions and the number of evaluations of the Jacobian are the same as the number of evaluations of the derivative functions.

It is seen that when the $Tolerance$ is bigger, the second predictor performs slightly better than the other two predictors in terms of the number of function evaluations for the order 3 and order 4 method. When the $Tolerance$ is small, the third predictor performs better with less rejected steps and fewer function evaluations compared to the other two predictors for both methods. By considering the number of function evaluations, rejected steps and *scd* values we pick the Hermite interpolation predictor for our solver.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	123	30	974	1.99
Newton interpolation	64	6	500	1.96
Hermite interpolation	65	8	515	2.03

Table 4.21: Results with $Tolerance = 10^{-4}$ for an order 3 method.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	260	58	2104	5.22
Newton interpolation	248	51	2006	5.09
Hermite interpolation	250	51	2011	5.28

Table 4.22: Results with $Tolerance = 10^{-7}$ for an order 3 method.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	1106	154	8910	6.80
Newton interpolation	1084	148	8749	6.81
Hermite interpolation	1071	142	8622	6.78

Table 4.23: Results with $Tolerance = 10^{-10}$ for an order 3 method.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	75	10	824	2.28
Newton interpolation	75	10	785	2.28
Hermite interpolation	75	10	793	2.28

Table 4.24: Results with $Tolerance = 10^{-4}$ for an order 4 method.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	224	43	2493	5.40
Newton interpolation	209	51	2163	5.51
Hermite interpolation	197	44	2036	5.25

Table 4.25: Results with $Tolerance = 10^{-7}$ for an order 4 method.

Predictor	Total steps	Reject steps	#f	scd
Taylor expansion	431	41	5243	7.52
Newton interpolation	434	45	4459	7.56
Hermite interpolation	423	35	4332	7.72

Table 4.26: Results with $Tolerance = 10^{-10}$ for an order 4 method.

4.5 Performance of interpolations

Two interpolation approaches for an output point, x, have been introduced in Chapter 3. In this section, we test these two approaches with the order 4 method. The *scd* values are listed in Table 4.27. The scd1 represents the result from the first approach and scd2 represents the result from the second approach.

$Tolerance$	scd1	scd2
10^{-4}	1.16	0.96
10^{-7}	5.60	5.33
10^{-10}	7.42	7.49
10^{-12}	8.89	9.19

Table 4.27: Numerical result: Interpolations for the output point with the order 4 method. The scd1 represents the results from the first approach and scd2 represents the results from the second approach.

Numerical experiments show that both approaches are very similar. It can be seen that for tight tolerances, the first approach gives a better approximation; for loose tolerance, the second approach gives a better approximation. For the order 4 method, the second approach has higher accuracy than the first approach.

4.6 Performance of certain methods with Hires problem

Comparison of the performance of alternative numerical methods is complicated. There are many factors which can affect the numerical performance of a method. These factors include the choices of initial stepsize, the stepsize controller, the predictor for the Newton iteration procedure. Tables 4.28, 4.29 and 4.30 list the numerical results found by applying three selected methods to the HIRES problem.

Note that these new general linear methods can solve the Hires problem accurately and efficiently in a reasonable way.

Tolerance	h_0	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-3}	10^{-3}	44	1	31	12	480	0.98
10^{-4}	10^{-3}	69	6	37	11	741	1.63
10^{-5}	10^{-3}	130	14	43	11	1215	2.13
10^{-6}	10^{-3}	242	10	45	6	2006	2.72
10^{-7}	10^{-4}	493	2	47	5	3683	3.40
10^{-8}	10^{-4}	1045	2	46	3	7264	4.11
10^{-9}	10^{-5}	2236	1	46	3	15077	4.78
10^{-10}	10^{-6}	4807	2	32	4	30798	5.46

Table 4.28: The numerical results for the order 2 method.

$Tolerance$	h_0	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-3}	10^{-3}	82	14	59	27	1215	2.34
10^{-4}	10^{-3}	86	23	58	32	1310	2.89
10^{-5}	10^{-4}	120	27	68	36	1730	3.04
10^{-6}	10^{-4}	150	30	74	34	2210	3.87
10^{-7}	10^{-4}	230	39	81	27	3291	5.10
10^{-8}	10^{-4}	386	63	120	41	5317	5.35
10^{-9}	10^{-5}	592	71	126	44	7605	6.10
10^{-10}	10^{-6}	1043	134	230	81	13238	6.90

Table 4.29: The numerical results for the order 3 method.

$Tolerance$	h_0	Total steps	Rejected steps	#LU	#J	#f	scd
10^{-3}	10^{-3}	82	18	59	31	1630	1.24
10^{-4}	10^{-3}	86	14	71	35	1784	1.16
10^{-5}	10^{-3}	122	20	75	39	2545	3.64
10^{-6}	10^{-3}	171	45	105	62	3404	4.50
10^{-7}	10^{-3}	189	44	122	63	3796	5.60
10^{-8}	10^{-4}	232	42	133	55	4611	5.73
10^{-9}	10^{-5}	306	43	162	51	6166	6.88
10^{-10}	10^{-6}	430	39	248	52	8714	7.84

Table 4.30: The numerical results for the order 4 method.

4.7 Conclusions

The primary goal of this thesis is to implement a new type of general linear method and to carry out numerical experiments on the designed implementation strategies with three standard test problems. In the thesis, we select three new general linear methods with order 2, 3 and 4 respectively for the numerical experiments. The implementation of the new general linear methods has been addressed in detail. The implementation questions include: iteration scheme, prediction, error estimation, stepsize control, starting methods and interpolation. The implementation strategies have been designed carefully. We then carry out numerical experiments with these methods using the designed implementation strategies. Numerical results show that these methods are very promising and the implementation is reasonably efficient for these methods. We interpret the numerical experiments as follows:

- Prothero Robinson problem. This problem is used to test the order reduction of a method. The numerical results show no sign of order reduction occuring for the new methods.
- Robertson problem. The numerical results show the positive influence of Inherent Runge-Kutta stability. The stepsize controller works well and x_{final} can go as high as 10^{18} if the tolerance is low enough.
- Hires problem. This is a more demanding problem, but the numerical results show the new methods are very good in terms of accuracy. Furthermore, the iteration scheme works well in terms of reducing computational cost.

Overall, our numerical results show that the new type of general linear method is capable of solving the standard stiff problems accurately and the designed implementation strategies make the methods optimal in a reasonable way for these standard test problems. We have performed some experiments with other solvers. However, in the thesis, we avoid strict comparison with other well known solvers since our experiments are only done with methods of order up to 4. Some experiments have indicated that there is excellent scaling for large problems for our method. In the thesis, we have produced Fortran code based on the techniques described in Chapter 3 for the fixed order implementation. These code can be modified to test other methods in this special group of general linear methods and other stiff test problems. For comparison with these results, where only fixed order is used, note that a variable order implementation of the closely related DIMSIM methods is presented in [28].

Future development

Our work has shown that the new type of general linear method is feasible for production code. The numerical results encourage further investigations such as variable order [28], other stepsize controllers and other more demanding differential equations.

For variable order, one can remove the starting method and start with an order 1 method. In this case, the error estimator needs to be modified to fit a higher order requirement and rescaling the Nordsieck vector will be more complicated. For the stepsize controller, the proportional integral (PI) controller [25] is another option. For the iteration process, to improve the efficiency of the iteration scheme, it is possible to consider using relaxation [3]. Note that in the iteration scheme, we use the following formula to do the Newton process.

$$(I - \lambda hJ)(\eta^{[j]} - \eta^{[j+1]}) = \phi(\eta^{[j]}), \tag{4.1}$$

with

$$\phi(\eta^{[j]}) = \eta^{[j]} - \lambda\bar{h}f(\eta^{[j]}) - \text{rhs},$$

where $\bar{h}$ is the present stepsize and h is the stepsize used when the most recent LU factorization was carried out. To correct the error from the different stepsizes, a possible way is using the following formula,

$$(I - \lambda hJ)(\eta^{[j]} - \eta^{[j+1]}) = w\phi(\eta^{[j]}), \tag{4.2}$$

where w is called the relaxation factor. Let $e^{[j]} = \eta^{[j]} - Y$ be the error in the ith iteration where Y is the final solution. First we assume the approximation

$$f(\eta^{[j]}) - f(Y) \approx J(\eta^{[j]} - Y). \tag{4.3}$$

Then we substitute $\eta^{[j]} = e^{[j]} + Y$ and equation (4.3) into equation (4.2), we find that

$$e^{[j+1]} = Me^{[j]}$$

where

$$M = I - w(I - \lambda hJ)^{-1}(I - \lambda\bar{h}J)$$

We want this to converge as rapidly as possible for nonstiff components (eigenvalues close to zero) and also for stiff components (eigenvalues very large). For small eigenvalues, the corresponding eigenvalue of M is

$$1 - w,$$

and for large eigenvalues

$$1 - w\frac{\bar{h}}{h}.$$

For example, if $w = 1$, which means no relaxation, we always get perfect convergence at zero and worst possible convergence (eigenvalue $1 - \bar{h}/h$) at infinity. To get the best compromise overall, we want to choose w so that the eigenvalues at zero and infinity are equal and opposite. That is,

$$1 - w = -(1 - w\frac{\bar{h}}{h}),$$

which gives

$$w = \frac{2h}{h + \bar{h}} \tag{4.4}$$

Note that if we choose w too small or too large, we can get faster convergence at either 0 or ∞ but the other becomes worse (slower convergence). However, the factor given by (4.4) can often lead to acceptable convergence overall. With this relaxation factor, one can design a different iteration scheme for the Newton procedure.

These methods can also be extended to other differential equations, such as differential algebraic equations (DAEs). Furthermore, one can develop a similar implementation for non-stiff problems.

Chapter 5

Method coefficients and Fortran programs

In this Chapter, we list the coefficients of the methods and corresponding starting methods as well as the Fortran code used to carry out the numerical experiments. The program code is designed in such a way that a user can input his (her) own methods or problems to do experiments. In the thesis, only the code for order 4 method is presented. The code for order 2 and order 3 are very similar to the order 4 and can be easily modified.

5.1 Coefficients of the methods

We list the coefficients of the methods that we have performed the numerical experiments with as follows.

Order 2 main method with 3 stages: $c = [0, \frac{1}{2}, 1]^T$,

$$\left[\begin{array}{ccc|ccc}
\frac{1}{4} & 0 & 0 & 1 & -\frac{1}{4} & 0 \\
\frac{1}{4} & \frac{1}{4} & 0 & 1 & 0 & 0 \\
\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 1 & 0 & \frac{1}{8} \\
\hline
\frac{1}{2} & -\frac{1}{8} & \frac{1}{2} & 1 & \frac{1}{8} & \frac{1}{16} \\
\frac{1}{2} & -\frac{1}{2} & 1 & 0 & 0 & \frac{1}{4} \\
0 & -2 & 2 & 0 & 0 & 0
\end{array}\right]$$

Starting method for order 2 method: $\hat{c} = [\frac{1}{4}, 1]^T$,

$$\left[\begin{array}{cc|c}
\frac{1}{4} & 0 & 1 \\
\frac{3}{4} & \frac{1}{4} & 1 \\
\hline
\frac{2}{3} & \frac{1}{3} & 1 \\
0 & 1 & 0 \\
-\frac{4}{3} & \frac{4}{3} & 0
\end{array}\right]$$

Order 3 main method: $c = [0, \frac{1}{3}, \frac{2}{3}, 1]^T$,

$$\left[\begin{array}{cccc|cccc}
\frac{1}{4} & 0 & 0 & 0 & 1 & -\frac{1}{4} & 0 & 0 \\
\frac{5}{6} & \frac{1}{4} & 0 & 0 & 1 & -\frac{3}{4} & -\frac{1}{36} & -\frac{5}{648} \\
\frac{109057}{33000} & \frac{1701}{2200} & \frac{1}{4} & 0 & 1 & -\frac{20137}{5500} & -\frac{4003}{19800} & -\frac{17509}{356400} \\
\frac{368999}{154000} & \frac{21071}{30800} & \frac{11}{56} & \frac{1}{4} & 1 & -\frac{24319}{9625} & -\frac{3357}{30800} & -\frac{22171}{554400} \\
\hline
\frac{11419277}{5832000} & -\frac{824833}{1166400} & \frac{5303}{23328} & \frac{827}{3888} & 1 & -\frac{341047}{162000} & -\frac{116611}{1166400} & -\frac{619133}{20995200} \\
\frac{529}{1620} & -\frac{17}{162} & -\frac{35}{162} & \frac{41}{36} & 0 & -\frac{13}{90} & \frac{13}{324} & -\frac{91}{5832} \\
\frac{677}{225} & -\frac{197}{45} & -\frac{23}{18} & \frac{19}{6} & 0 & -\frac{13}{25} & \frac{13}{90} & -\frac{91}{1620} \\
6 & -9 & 0 & 3 & 0 & 0 & 0 & 0
\end{array}\right]$$

Starting method for order 3 method: $\hat{c} = [\frac{1}{4}, \frac{1}{2} - \frac{\sqrt{2}}{4}, \frac{1}{3}, 1]^T$,

$$\left[\begin{array}{cccc|c}
\frac{1}{4} & 0 & 0 & 0 & 1 \\
\frac{1}{4} - \frac{\sqrt{2}}{4} & \frac{1}{4} & 0 & 0 & 1 \\
-\frac{4+7\sqrt{2}}{36} & \frac{7+7\sqrt{2}}{36} & \frac{1}{4} & 0 & 1 \\
0 & 0 & \frac{3}{4} & \frac{1}{4} & 1 \\
\hline
0 & 0 & \frac{3}{4} & \frac{1}{4} & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & \frac{-16}{7} + \frac{32\sqrt{2}}{7} & -\frac{45}{14} - \frac{18\sqrt{2}}{7} & \frac{11}{2} - 2\sqrt{2} & 0 \\
0 & \frac{-48}{7} + \frac{96\sqrt{2}}{7} & -\frac{36}{7} - \frac{54\sqrt{2}}{7} & 12 - 6\sqrt{2} & 0
\end{array}\right].$$

Order 4 main method: $c = [0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1]^T$,

$$A = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{47}{64} & \frac{1}{4} & 0 & 0 & 0 \\ \frac{24197}{14476} & \frac{678}{3619} & \frac{1}{4} & 0 & 0 \\ \frac{7102302807}{1544183872} & \frac{987465}{24127873} & \frac{10395}{26668} & \frac{1}{4} & 0 \\ -\frac{117251104}{55207845} & -\frac{27818059}{55207845} & \frac{7255}{6102} & -\frac{59}{135} & \frac{1}{4} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{825449}{430191} & -\frac{1889207}{860382} & \frac{19919}{9153} & -\frac{59}{162} & \frac{1}{6} \\ \frac{1422203}{1433970} & \frac{528694}{716985} & -\frac{4249}{3051} & \frac{118}{135} & \frac{5}{6} \\ -\frac{37397426}{716985} & \frac{61340224}{716985} & -\frac{199780}{3051} & \frac{1888}{135} & 4 \\ -\frac{584578572}{2150955} & \frac{880353408}{2150955} & -\frac{2670168}{9153} & \frac{22656}{405} & 12 \\ -\frac{332267376}{716985} & \frac{477708864}{716985} & -\frac{1397184}{3051} & \frac{11328}{135} & 16 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & -\frac{1}{4} & 0 & 0 & 0 \\ 1 & -\frac{47}{64} & -\frac{1}{32} & -\frac{1}{192} & -\frac{3}{6144} \\ 1 & -\frac{11645}{7238} & -\frac{339}{7238} & -\frac{5653}{347424} & -\frac{4297}{1389696} \\ 1 & -\frac{6995320711}{1544183872} & -\frac{85994121}{772091936} & -\frac{57910455}{1158137904} & -\frac{1871076171}{148241651712} \\ 1 & \frac{579853229}{220831380} & \frac{12065149}{110415690} & \frac{9336821}{294441840} & \frac{15415373}{2119981248} \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & -\frac{603461}{860382} & \frac{116111}{1720764} & -\frac{40393}{2294352} & \frac{19249}{165193344} \\ 0 & -\frac{748481}{716985} & \frac{33116}{1433970} & -\frac{21913}{1911960} & -\frac{90679}{13766112} \\ 0 & \frac{5055197}{1433970} & -\frac{1532237}{716985} & \frac{276353}{477990} & -\frac{14840}{1720764} \\ 0 & \frac{185577168}{2150955} & -\frac{22399584}{2150955} & \frac{703186}{238995} & \frac{134986}{1720764} \\ 0 & \frac{111261984}{716985} & -\frac{11852112}{716985} & \frac{384128}{79665} & \frac{34232}{143397} \end{bmatrix}$$

Starting method for order 4 method: $\hat{c} = [\frac{1}{4}, \frac{1}{2} - \frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4} - \frac{1}{6}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1]^T$, $r = 1$,

$$\left[\begin{array}{ccccccc|c} \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{4} - \frac{\sqrt{2}}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{31}{36} + \frac{17\sqrt{2}}{36} & \frac{4}{9} - \frac{2\sqrt{2}}{9} & \frac{1}{4} & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{3}{8} + \frac{9\sqrt{2}}{32} & -\frac{3}{8} - \frac{9\sqrt{2}}{32} & \frac{1}{4} & 0 & 0 & 0 & 1 \\ 0 & -\frac{9}{8} - \frac{3\sqrt{2}}{4} & \frac{129}{56} + \frac{45\sqrt{2}}{28} & -\frac{13}{14} - \frac{6\sqrt{2}}{7} & \frac{1}{4} & 0 & 0 & 1 \\ 0 & 0 & -\frac{261}{1288} - \frac{351\sqrt{2}}{2576} & \frac{25}{28} + \frac{9\sqrt{2}}{56} & -\frac{35}{184} - \frac{9\sqrt{2}}{368} & \frac{1}{4} & 0 & 1 \\ 0 & 0 & 0 & \frac{5}{12} & \frac{5}{12} & -\frac{1}{12} & \frac{1}{4} & 1 \\ \hline 0 & 0 & 0 & \frac{1}{2} & -\frac{7}{24} & \frac{1}{3} & -\frac{1}{24} & 1 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{16}{3} & 10 & -8 & \frac{10}{3} & 0 \\ 0 & 0 & 0 & 16 & -32 & 16 & 0 & 0 \end{array}\right]$$

5.2 Fortran code

The computer code is programmed in the Fortran language. To solve an IVP, program files include main (such as mainhires.f90), problem function (such as hires.f90), and several subroutines. In this section, we present the following programs:

- main program and problem function. These code can be modified to fit the requirements.
- method. The code is for the order 4 method and includes the coefficients of the method, starting method, calculating stage values, error estimator, rescaling Nordsieck vector and interpolation.
- Newton iteration. This code does the Newton iterations using the iteration scheme in Chapter 3.

5.2.1 Main program and problem function

```
!this is a main code for Hires problem
     program mainhires

      implicit none
      external f, jex
      integer :: dim
      double precision::tol, tolerance,h0,xend

      dim=8
      write(*,*)'tolerance='
      read(*,*)tolerance
      write(*,*)'h0='
      read(*,*)h0
      xend=321.8122d0
      call nglm(dim,tolerance,h0,xend)

     end program mainhires

!Hires problem. can be replaced by other problems

     subroutine f(dim,x,y,ydot)

      implicit none
      integer ::dim
      double precision ::x
      double precision, dimension(dim)::y,ydot
      ydot(1) = -1.71d0*y(1)+0.43d0*y(2)+8.32d0*y(3)+0.0007d0
      ydot(2) = 1.71d0*y(1)-8.75d0*y(2)
      ydot(3) = -10.03d0*y(3)+0.43d0*y(4)+0.035d0*y(5)
      ydot(4) = 8.32d0*y(2)+1.71d0*y(3)-1.12d0*y(4)
      ydot(5) = -1.745d0*y(5)+0.43d0*(y(6)+y(7))
      ydot(6) = -280.d0*y(6)*y(8)+0.69d0*y(4)+1.71d0*y(5)-0.43d0*y(6)+&
                0.69d0*y(7)
      ydot(7) = 280.d0*y(6)*y(8)-1.81d0*y(7)
      ydot(8) = -ydot(7)

    end subroutine f

!the Jacbian matrix
    subroutine jex(dim,x,y,pd)

      implicit none
      integer ::dim
      double precision :: x
      double precision, dimension(dim)::y
      double precision, dimension(dim,dim)::pd

      pd=0.d0
      pd(1,1) = -1.71d0
      pd(1,2) = 0.43d0
      pd(1,3) = 8.32d0
      pd(2,1) = 1.71d0
      pd(2,2) = -8.75d0
```

```
       pd(3,3) = -10.03d0
       pd(3,4) = 0.43d0
       pd(3,5) = 0.035d0
       pd(4,2) = 8.32d0
       pd(4,3) = 1.71d0
       pd(4,4) = -1.12d0
       pd(5,5) = -1.745d0
       pd(5,6) = 0.43d0
       pd(5,7) = 0.43d0
       pd(6,4) = 0.69d0
       pd(6,5) = 1.71d0
       pd(6,6) = -280.d0*y(8)-0.43d0
       pd(6,7) = 0.69d0
       pd(6,8) = -280.d0*y(6)
       pd(7,6) = 280.d0*y(8)
       pd(7,7) = -1.81d0
       pd(7,8) = 280.d0*y(6)
       pd(8,6) = -280.d0*y(8)
       pd(8,7) = 1.81d0
       pd(8,8) = -280.d0*y(6)

      end subroutine jex

!the initial values
      subroutine init(dim,x,y0)

       implicit none
       integer ::dim
       double precision ::x
       double precision, dimension(dim)::y0

       y0(1) = 1.d0
       y0(2) = 0.d0
       y0(3) = 0.d0
       y0(4) = 0.d0
       y0(5) = 0.d0
       y0(6) = 0.d0
       y0(7) = 0.d0
       y0(8) = 0.0057d0
```

```
     end subroutine init

!reference solution at endpoint
     subroutine endpoint(dim,xend,yend)

      implicit none
      integer ::dim
      double precision ::xend
      double precision, dimension(dim)::yend

      xend=321.8122d0
      yend(1)=7.371312573325668d-4
      yend(2)=1.442485726316185d-4
      yend(3)=5.888729740967575d-5
      yend(4)=1.175651343283149d-3
      yend(5)=2.386356198831331d-3
      yend(6)=6.238968252742796d-3
      yend(7)=2.849998395185769d-3
      yend(8)=2.850001604814231d-3

     end subroutine endpoint
```

5.2.2 Method

```
!!this  subroutine solve problems with order 4 method
   subroutine nglm(dim,tolerance,h0,xend)

     implicit none
     external f, jex
     integer :: dim, step,flag,order,count,exitstage,nreject,naccept
     integer :: njumpout,k,nredo,njex,ncallf,nfactorize,i
     integer, dimension(dim)::ipiv
     double precision, dimension(dim) :: y0,y1,y2,y3,y4,y5
     double precision, dimension(dim) :: f1,f2,f3,f4,f5
     double precision, dimension(dim) :: y1_pre,y2_pre,y3_pre
     double precision, dimension(dim) :: y4_pre,y5_pre
     double precision, dimension(dim) :: y1_stage,y2_stage,y3_stage
double precision, dimension(dim) :: y4_stage,y5_stage
double precision, dimension(dim) :: f1_pre,f2_pre,f3_pre
double precision, dimension(dim) :: f4_pre,f5_pre
double precision, dimension(dim) :: ypoint,y_value,error,error_pre
double precision, dimension(dim) :: yend,enderror,y_pre
double precision :: tol,x,h,x_pre,delta,xpoint,tolerance, h0,&
                   h_pre,coefficient,h_lastfactorize,lambda,xend,&
                   errormax,scd
double precision, dimension(dim,dim)::matrix,pd
logical ::jumpout,justfactorize,redo

 order=4
 count=0
 call init(dim,x,y0)
 call endpoint(dim,xend,yend)
 coefficient=1.d0
 step=0
 nreject=0
 naccept=0
 njumpout=0
 nredo=0
 nfactorize=0
 njex=0
 ncallf=0
 exitstage=0
 x=0.d0
 h=h0
 h_pre=h
 tol=tolerance/1000.d0
 jumpout=.false.
 justfactorize=.false.
 redo=.false.
 xpoint=xend
 call nosieckstart4(f,dim,x,y1,y2,y3,y4,y5,y0,y_value, &
                 jex,h,tol,f1,f2,f3,f4,f5,matrix,pd,ipiv)
 call errorestimate(dim,f1,f2,f3,f4,f5,h,error)
 h_lastfactorize=h
 do
   justfactorize=.false.
   y1_pre=y1
```

```
        y2_pre=y2
        y3_pre=y3
        y4_pre=y4
        y5_pre=y5
        x_pre=x
        y_pre=y_value
        f1_pre=f1
        f2_pre=f2
        f3_pre=f3
        f4_pre=f4
        f5_pre=f5
        h_prepre=h_pre
        do
        step=step+1
        jumpout=.false.
        delta=coefficient
!rescale the Nordsieck vector
        call varistep(dim,y1,y2,y3,y4,y5,delta)
        call nordsieck4(f,dim,x,xpoint,y1,y2,y3,y4,y5,y_value,&
            jex,h,tol,f1,f2,f3,f4,f5,exitstage,h_lastfactorize,&
            matrix,jumpout,justfactorize,step,ipiv,pd,njex, &
            ncallf,nfactorize,redo)
        if (jumpout)then
           x=x_pre
           y1=y1_pre
           y2=y2_pre
           y3=y3_pre
           y4=y4_pre
           y5=y5_pre
           h=0.5d0*h
           coefficient=0.5d0*coefficient
           njumpout=njumpout+1
        else
           call errorestimate(dim,f1,f2,f3,f4,f5,h,error)
           call stepsize(dim,error,order,tolerance,h,coefficient,redo)
           if(redo)then
              nreject=nreject+1
              x=x_pre
              y1=y1_pre
              y2=y2_pre
              y3=y3_pre
              y4=y4_pre
              y5=y5_pre
              coefficient=h/h_pre
           else
              naccept=naccept+1
              exit
           end if
        end if

       end do
       h_pre=h/coefficient
!interpolation
      if (xpoint<x.and.xpoint>x_pre)then
       call point(dim,xpoint,ypoint,y_pre,y_value,x_pre,x, &
                  h_pre,f5_pre,f5)
       enderror=abs(ypoint-yend)/ypoint
       errormax=enderror(1)
       do i=2,dim
        if (enderror(i).ge.errormax)then
         errormax=enderror(i)
        end if
       end do
       scd=-(log(errormax))/log(10.d0)
       print *,'nfactorize=', nfactorize, ' njex=',njex,' ncallf=',ncallf
       print *, 'scd=',scd
       exit
      end if
     end do
     print *,'njumpout=',njumpout,' nreject=',nreject, ' totalstep=',step
    end subroutine nglm
```

The following Module provides the coefficients for order 4 method.

```
  MODULE matrix_abuv_order4
    implicit none
    double precision, parameter :: a11=1.d0/4.d0,&
                                 a21=47.d0/64.d0, &
```

```
                                a22=1.d0/4.d0,  &
                                a31=24197.d0/14476.d0, &
                                a32=678.d0/3619.d0, &
                                a33=1.d0/4.d0,  &
                                a41=7102302807.d0/1544183872.d0, &
                                a42=987465.d0/24127873.d0, &
                                a43=10395.d0/26668.d0, &
                                a44=1.d0/4.d0,      &
                                a51=-117251104.d0/55207845.d0, &
                                a52=-27818059.d0/55207845.d0, &
                                a53=7255.d0/6102.d0,  &
                                a54=-59.d0/135.d0,   &
                                a55=1.d0/4.d0,      &
     u11=1.d0,                       &
     u12=-1.d0/4.d0,                 &
     u13=0.d0,                       &
     u14=0.d0,                       &
     u15=0.d0,                       &
     u21=1.d0,                       &
     u22=-47.d0/64.d0,               &
     u23=-1.d0/16.d0,                &
     u24=-1.d0/32.d0,                &
     u25=-3.d0/256.d0,               &
     u31=1.d0,                       &
     u32=-11645.d0/7238.d0,          &
     u33=-339.d0/3619.d0,            &
     u34=-5653.d0/57904.d0,          &
     u35=-4297.d0/57904.d0,          &
     u41=1.d0,                       &
     u42=-6995320711.d0/1544183872.d0, &
     u43=-85994121.d0/386045968.d0,    &
     u44=-57910455.d0/193022984.d0,    &
     u45=-1871076171.d0/6176735488.d0, &
     u51=1.0d0,                      &
     u52=579853229.d0/220831380.d0,    &
     u53=12065149.d0/55207845.d0,      &
     u54=9336821.d0/49073640.d0,       &
     u55=15415373.d0/88332552.d0,      &
     b11=825449.d0/430191.d0,        &
     b12=-1889207.d0/860382.d0,      &
```

```
b13=19916.d0/9153.d0,           &
b14=-59.d0/162.d0,              &
b15=1.d0/6.d0,                  &
b21=1422203.d0/1433970.d0,      &
b22=528694.d0/716985.d0,        &
b23=-4249.d0/3051.d0,           &
b24=118.d0/135.d0,              &
b25=5.d0/6.d0,                  &
b31=-18698713.d0/716985.d0,     &
b32=30670112.d0/716985.d0,      &
b33=-99890.d0/3051.d0,          &
b34=944.d0/135.d0,              &
b35=2.d0,                       &
b41=-97429762.d0/2150955.d0,    &
b42=146725568.d0/2150955.d0,    &
b43=-445028.d0/9153.d0,         &
b44=3776.d0/405.d0,             &
b45=2.d0,                       &
b51=-13844474.d0/716985.d0,     &
b52=19904536.d0/716985.d0,      &
b53=-58216.d0/3051.d0,          &
b54=472.d0/135.d0,              &
b55=2.d0/3.d0,                  &
v11=1.d0,                   &
v12=-603461.d0/860382.d0,   &
v13=116111.d0/860382.d0,    &
v14=-40393.d0/382392.d0,    &
v15=-19249.d0/6883056.d0,   &
v21=0.0d0,                  &
v22=-748481.d0/716985.d0,   &
v23=33116.d0/716985.d0,     &
v24=-21913.d0/318660.d0,    &
v25=-90679.d0/573588.d0,    &
v31=0.0d0,                  &
v32=5055197.d0/716985.d0,   &
v33=-1532237.d0/716985.d0,  &
v34=276353.d0/159330.d0,    &
v35=-14840.d0/143397.d0,    &
v41=0.0d0,                  &
v42=30929528.d0/2150955.d0, &
```

```
        v43=-7466528.d0/2150955.d0, &
        v44=703186.d0/238995.d0,    &
        v45=134986.d0/430191.d0,    &
        v51=0.d0,                   &
        v52=4635916.d0/716985.d0,   &
        v53=-987676.d0/716985.d0,   &
        v54=96032.d0/79665.d0,      &
        v55=34232.d0/143397.d0,     &
        c1=0.d0,     &
        c2=1.d0/4.d0, &
        c3=1.d0/2.d0, &
        c4=3.d0/4.d0, &
        c5=1.d0

 END MODULE matrix_abuv_order4
```

The following subroutine calculates the stage values for the order 4 method.

```
   subroutine nordsieck4(f,dim,x,xpoint,y1,y2,y3,y4,y5,y_value,&
            jex,h,tol,f1,f2,f3,f4,f5,exitstage,h_lastfactorize,&
            matrix,jumpout,justfactorize,step,ipiv,pd,njex, &
            ncallf,nfactorize,redo)

     use matrix_abuv_order4
     implicit none
     external f, jex,iter
     integer ::dim, maxstep,k,exitstage,nfactorize,njex,step,stage
     integer ::ncallf,ii
     integer,dimension(dim):: ipiv
     double precision, dimension(dim)::yend,rerror
     double precision, dimension(dim)::part1,part2,part3,part4,part5
     double precision, dimension(dim)::y1_stage,y2_stage,y3_stage
     double precision, dimension(dim)::y4_stage,y5_stage
     double precision, dimension(dim)::y1,y2,y3,y4,y5
     double precision, dimension(dim)::f1,f2,f3,f4,f5
     double precision, dimension(dim)::y0,y_value,ydot,phi,y_old,&
     double precision, dimension(dim)::y_new,ypoint
     double precision, dimension(dim)::ydot1,ydot2,ydot3,ydot4,ydot5
     double precision, dimension(dim)::y1_new,y2_new,y3_new,y4_new,y5_new
     double precision, dimension(dim,dim)::matrix,pd
     double precision::x,xpoint,x1,x2,x3,x4,x5,x_old,x_new,sumsum
     double precision::hlam,h,lambda,tol,sum,h_lastfactorize,r,scd
     logical ::converge,justfactorize,jumpout,redo

     lambda=1.d0/4.d0
! calculate stage 1
     part1=u11*y1+u12*y2+u13*y3+u14*y4+u15*y5
! predictor for stage 1
     y1_stage=y_value
     x1=x+c1*h
     hlam=h*a11
     stage=1
!Newton iteration
     call iter(dim,x1,y1_stage,part1,f,jex,lambda,f1,&
              tol,r,h,pd,matrix,justfactorize,&
              h_lastfactorize,converge,ipiv,nfactorize,&
              njex,step,stage,ncallf)
     if (converge) then
! calculate stage 2
      part2=a21*h*f1+u21*y1+u22*y2+u23*y3+u24*y4+u25*y5
! predictor for stage 2
      y2_stage=y1+c2*y2+c2*c2/2.d0*y3+c2*c2*c2/6.d0*y4 &
               +c2*c2*c2*c2/24.d0*y5
      x2=x+h*c2
      hlam=a22*h
      stage=2
      call iter(dim,x2,y2_stage,part2,f,jex,lambda,f2,&
                tol,r,h,pd,matrix,justfactorize,&
                h_lastfactorize,converge,ipiv,nfactorize,&
                njex,step,stage,ncallf)
      if(converge) then
! calculate stage 3
         part3=a31*h*f1+a32*h*f2+u31*y1+u32*y2+u33*y3 &
               +u34*y4+u35*y5
! predictor for stage 3
         y3_stage= 5.d0*y1_stage+1.d0/2.d0*h*f1-4.d0*y2_stage+h*f2
         x3=x+c3*h
         hlam=a33*h
         stage=3
         call iter(dim,x3,y3_stage,part3,f,jex,lambda,f3,&
                   tol,r,h,pd,matrix,justfactorize,&
                   h_lastfactorize,converge,ipiv,nfactorize,&
                   njex,step,stage,ncallf)
         if (converge) then
! calculate stage 4
           part4=a41*h*f1+a42*h*f2+a43*h*f3 &
             +u41*y1+u42*y2+u43*y3+u44*y4+u45*y5
! predictor for stage 4
           y4_stage=5.d0*y2_stage+1.d0/2.d0*h*f2-4.d0*y3_stage+h*f3
           x4=x+c4*h
           hlam=a44*h
           stage=4
           call iter(dim,x4,y4_stage,part4,f,jex,lambda,f4,&
                     tol,r,h,pd,matrix,justfactorize,&
                     h_lastfactorize,converge,ipiv,nfactorize,&
                     njex,step,stage,ncallf)
           if(converge) then
! calculate stage 5
             part5=a51*h*f1+a52*h*f2+a53*h*f3+a54*h*f4   &
                   +u51*y1+u52*y2+u53*y3+u54*y4+u55*y5
! predictor for stage 5
             y5_stage= 5.d0*y3_stage+1.d0/2.d0*h*f3-4.d0*y4_stage+h*f4
             x5=x+c5*h
             hlam=a55*h
             stage=5
             call iter(dim,x5,y5_stage,part5,f,jex,lambda,f5,&
                       tol,r,h,pd,matrix,justfactorize,&
                       h_lastfactorize,converge,ipiv,nfactorize,&
                       njex,step,stage,ncallf)
             if (converge)then
               jumpout=.false.
               exitstage=0
               x=x5
               y_value=y5_stage
               y1_new=b11*h*f1+b12*h*f2+b13*h*f3+b14*h*f4      &
                      +b15*h*f5+v11*y1+v12*y2+v13*y3+v14*y4+v15*y5
               y2_new=b21*h*f1+b22*h*f2+b23*h*f3+b24*h*f4      &
                      +b25*h*f5+v21*y1+v22*y2+v23*y3+v24*y4+v25*y5
               y3_new=b31*h*f1+b32*h*f2+b33*h*f3+b34*h*f4      &
```

```
                       +b35*h*f5+v31*y1+v32*y2+v33*y3+v34*y4+v35*y5
                y4_new=b41*h*f1+b42*h*f2+b43*h*f3+b44*h*f4      &
                       +b45*h*f5+v41*y1+v42*y2+v43*y3+v44*y4+v45*y5
                y5_new=b51*h*f1+b52*h*f2+b53*h*f3+b54*h*f4      &
                       +b55*h*f5+v51*y1+v52*y2+v53*y3+v54*y4+v55*y5

                y1=y1_new
                y2=y2_new
                y3=y3_new
                y4=y4_new
                y5=y5_new
              else
                jumpout=.true.
                exitstage=5
              end if
            else
             jumpout=.true.
             exitstage=4
            end if
         else
           jumpout=.true.
           exitstage=3
         end if
       else
         jumpout=.true.
         exitstage=2
       end if
     else
       jumpout=.true.
       exitstage=1
     end if

    end subroutine nordsieck4
```

The following codes are the starting method for the order 4 method.

```
     subroutine nosieckstart4(f,dim,x,y1,y2,y3,y4,y5,y0, &
           y_value,jex,h,tol,f1,f2,f3,f4,f5,matrix,pd,ipiv)
      implicit none
```

```
external f,jex,iter
integer ::dim,stage
integer, dimension(dim)::ipiv
double precision, dimension(dim):: y0,y1,y2,y3,y4,y5
double precision, dimension(dim):: y1_stage,y2_stage,y3_stage
double precision, dimension(dim):: y4_stage,y5_stage,y6_stage,y7_stage
double precision, dimension(dim):: ydot1,ydot2,ydot3,ydot4
double precision, dimension(dim):: ydot5,ydot6,ydot7
double precision, dimension(dim):: part1,part2,part3,part4
double precision, dimension(dim):: part5,part6,part7
double precision, dimension(dim):: f1,f2,f3,f4,f5,f6,f7
double precision, dimension(dim):: ydot,phi,y_value
double precision, dimension(dim,dim):: matrix,pd
double precision :: x,x1,x2,x3,x4,x5,x6,x7,hlam,tol,sum,lambda,h
double precision :: aa11,aa21,aa22,aa31,aa32,aa33,aa41,aa42,aa43,&
                    aa44,aa51,aa52,aa53,aa54,aa55,aa61,aa62,aa63,&
                    aa64,aa65,aa66,aa71,aa72,aa73,aa74,aa75,aa76,&
                    aa77,bb11,bb12,bb13,bb14,bb15,bb16,bb17,&
                        bb21,bb22,bb23,bb24,bb25,bb26,bb27,&
                        bb31,bb32,bb33,bb34,bb35,bb36,bb37,&
                        bb41,bb42,bb43,bb44,bb45,bb46,bb47,&
                        bb51,bb52,bb53,bb54,bb55,bb56,bb57,&
                        uu11,uu21,uu31,uu41,uu51,uu61,uu71,&
                        vv11,vv21,vv31,vv41,vv51,&
                        cc1,cc2,cc3,cc4,cc5,cc6,cc7
   aa11=1.d0/4.d0
   aa21=1.d0/4.d0-dsqrt(2.d0)/4.d0
   aa22=1.d0/4.d0
   aa31=-31.d0/36.d0+17.d0*dsqrt(2.d0)/36.d0
   aa32=4.d0/9.d0-2.d0*dsqrt(2.d0)/9.d0
   aa33=1.d0/4.d0
   aa41=0.d0
   aa42=3.d0/8.d0+9.d0*dsqrt(2.d0)/32.d0
   aa43=-3.d0/8.d0-9.d0*dsqrt(2.d0)/32.d0
   aa44=1.d0/4.d0
   aa51=0.d0
   aa52=-9.d0/8.d0-3.d0*dsqrt(2.d0)/4.d0
   aa53=129.d0/56.d0+45.d0*dsqrt(2.d0)/28.d0
   aa54=-13.d0/14.d0-6.d0*dsqrt(2.d0)/7.d0
   aa55=1.d0/4.d0
```

```
aa61=0.d0
aa62=0.d0
aa63=-261.d0/1288.d0-351.d0*dsqrt(2.d0)/2576.d0
aa64=25.d0/28.d0+9.d0*dsqrt(2.d0)/56.d0
aa65=-35.d0/184.d0-9.d0*dsqrt(2.d0)/368.d0
aa66=1.d0/4.d0
aa71=0.d0
aa72=0.d0
aa73=0.d0
aa74=5.d0/12.d0
aa75=5.d0/12.d0
aa76=-1.d0/12.d0
aa77=1.d0/4.d0
uu11=1.0d0
uu21=1.0d0
uu31=1.0d0
uu41=1.0d0
uu51=1.0d0
uu61=1.0d0
uu71=1.0d0
bb11=0.0d0
bb12=0.0d0
bb13=0.0d0
bb14=2.d0/3.d0
bb15=-1.d0/3.d0
bb16=2.d0/3.d0
bb17=0.d0
bb21=0.0d0
bb22=0.0d0
bb23=0.0d0
bb24=0.0d0
bb25=0.0d0
bb26=0.0d0
bb27=1.0d0
bb31=0.0d0
bb32=0.0d0
bb33=0.0d0
bb34=-4.d0/3.d0
bb35=6.d0
bb36=-12.d0
```

```
bb37=22.d0/3.d0
bb41=0.0d0
bb42=0.0d0
bb43=0.0d0
bb44=-16.d0
bb45=64.d0
bb46=-80.d0
bb47=32.d0
bb51=0.d0
bb52=0.d0
bb53=0.d0
bb54=-64.d0
bb55=192.d0
bb56=-192.d0
bb57=64.d0
vv11=1.d0
vv21=0.d0
vv31=0.d0
vv41=0.d0
vv51=0.d0
cc1=1.d0/4.d0
cc2=1.d0/2.d0-dsqrt(2.d0)/4.d0
cc3=dsqrt(2.d0)/4.d0-1.d0/6.d0
cc4=1.d0/4.d0
cc5=1.d0/2.d0
cc6=3.d0/4.d0
cc7=1.d0
part1=uu11*y0
y1_stage=y0
x1=x+cc1*h
hlam=aa11*h
stage=1
call iter1(dim,x1,y1_stage,part1,f,jex,hlam,f1,&
           tol,matrix,ipiv,stage,pd)
part2=uu21*y0+aa21*h*f1
y2_stage=y1_stage+cc2*h*f1
x2=x+cc2*h
hlam=h*aa22
stage=2
call iter1(dim,x2,y2_stage,part2,f,jex,hlam,f2,&
          tol,matrix,ipiv,stage,pd)
part3=y0*uu31+aa31*h*f1+aa32*h*f2
y3_stage=y1_stage+cc3*h*f2
x3=x+cc3*h
hlam=h*aa33
stage=3
call iter1(dim,x3,y3_stage,part3,f,jex,hlam,f3,&
          tol,matrix,ipiv,stage,pd)
part4=y0*uu41+aa41*h*f1+aa42*h*f2+aa43*h*f3
y4_stage=y1_stage+cc4*h*f3
x4=x+cc4*h
hlam=aa44*h
stage=4
call iter1(dim,x4,y4_stage,part4,f,jex,hlam,f4,&
          tol,matrix,ipiv,stage,pd)
part5=y0*uu51+aa51*h*f1+aa52*h*f2+aa53*h*f3   &
      +aa54*h*f4
y5_stage=y1_stage+cc5*h*f4
x5=x+cc5*h
hlam=aa55*h
stage=5
call iter1(dim,x5,y5_stage,part5,f,jex,hlam,f5,&
          tol,matrix,ipiv,stage,pd)
part6=y0*uu61+aa61*h*f1+aa62*h*f2+aa63*h*f3    &
      +aa64*h*f4+aa65*h*f5
y6_stage=y1_stage+cc6*h*f5
x6=x+cc6*h
hlam=aa66*h
stage=6
call iter1(dim,x6,y6_stage,part6,f,jex,hlam,f6,&
          tol,matrix,ipiv,stage,pd)
part7=y0*uu71+aa71*h*f1+aa72*h*f2+aa73*h*f3    &
      +aa74*h*f4+aa75*h*f5+aa76*h*f6
y7_stage=y1_stage+cc7*h*f6
x7=x+cc7*h
hlam=aa77*h
stage=7
call iter1(dim,x7,y7_stage,part7,f,jex,hlam,f7,&
          tol,matrix,ipiv,stage,pd)
x=x7
```

```
        y_value=y7_stage
        y1=bb11*h*f1+bb12*h*f2+bb13*h*f3  &
           +bb14*h*f4+bb15*h*f5+bb16*h*f6  &
           +bb17*h*f7+vv11*y0
        y2=bb21*h*f1+bb22*h*f2+bb23*h*f3  &
           +bb24*h*f4+bb25*h*f5+bb26*h*f6  &
           +bb27*h*f7+vv21*y0
        y3=bb31*h*f1+bb32*h*f2+bb33*h*f3  &
           +bb34*h*f4+bb35*h*f5+bb36*h*f6  &
           +bb37*h*f7+vv31*y0
        y4=bb41*h*f1+bb42*h*f2+bb43*h*f3  &
           +bb44*h*f4+bb45*h*f5+bb46*h*f6  &
           +bb47*h*f7+vv41*y0
        y5=bb51*h*f1+bb52*h*f2+bb53*h*f3  &
           +bb54*h*f4+bb55*h*f5+bb56*h*f6  &
           +bb57*h*f7+vv51*y0
    end subroutine nosieckstart4

!this is a full newton iteration for the starting method
 ! a simple version
    subroutine iter1(dim,x,y0,rhs,f,jex,hlam,fstage,tol,matrix,&
                     ipiv,stage,pd)
     implicit none
     external f, jex
     integer ::dim, maxstep,stepp,info,i,stage
     integer, dimension(dim)::ipiv
     double precision, dimension(dim):: y0,ystage,ydot
     double precision, dimension(dim):: phi,fstage,rhs
     double precision, dimension(dim,dim)::matrix,pd,identy
     double precision ::x,hlam,tol,sum
     maxstep=6
     ystage=y0
     stepp=0
     identy=0.d0
     do i=1,dim
         identy(i,i)=1.0d0
     end do
     do
        sum=0.d0
        stepp=stepp+1
        call jex(dim,x,ystage,pd)
        pd=hlam*pd
        matrix=identy-pd
        call f(dim,x,ystage,ydot)
        phi=ystage-hlam*ydot-rhs
        call DGETRF(dim,dim,matrix,dim,ipiv,info)
        call DGETRS('N',dim,1,matrix,dim,ipiv,phi,dim,info)
        ystage=ystage-phi
        do i=1,dim
           sum=sum+(phi(i))*(phi(i))
        end do
        sum=sqrt(sum)
        if (sum<tol.or.stepp>=maxstep) then
         fstage=(ystage-rhs)/hlam
         y0=ystage
         exit
        end if
     end do

    end subroutine iter1
```

The following subroutines are error estimator, stepsize controller, interpolation and rescaling Nordsieck vector.

```
!calculate the error
    subroutine errorestimate(dim,f1,f2,f3,f4,f5,h,error)
     implicit none
     integer ::dim
     double precision:: h
     double precision, dimension(dim)::f1,f2,f3,f4,f5,error
     error=13.d0/60.d0*(h*f1-4.d0*h*f2+6.d0*h*f3-4.d0*h*f4 &
                        +h*f5)
    end subroutine errorestimate

! stepsize controller
    subroutine stepsize(dim,error,order, &
                        tolerance,h,coefficient,redo)
     implicit none
     integer ::dim,flag,order,i,j
     double precision ::h,sum,coefficient,tolerance,power
     double precision ::safefactor,max
     double precision, dimension(dim):: error,y_value
     logical ::redo
     max=0.d0
     safefactor=0.9d0
     max=abs(error(1))
     do i=2,dim
       if(max<abs(error(i)))then
        max=abs(error(i))
       end if
     end do
     sum=max
     coefficient=(tolerance/sum)**(1.d0/5.d0)
     if (coefficient.le.0.5d0)then
       coefficient=0.5d0
      else if (coefficient.ge.2.d0) then
       coefficient=2.d0
      else
       coefficient=coefficient*0.9d0
     end if
     h=coefficient*h
     if(tolerance.ge.sum) then
          redo=.false.
     else
          redo=.true.
     end if
    end subroutine stepsize

! interpolation
    subroutine point(dim,xpoint,ypoint,y_old,y_new, &
                      x_old,x_new,h,f_old,f_new)
     implicit none
     integer :: dim
     double precision:: xpoint,x_old,x_new,si,h
     double precision,dimension(dim):: ypoint,y_old,y_new
     double precision,dimension(dim):: f_old,f_new
     si=(xpoint-x_old)/(x_new-x_old)
     ypoint=(si-1.d0)**2*(2.d0*si+1.d0)*y_old   &
            +(3.d0-2.d0*si)*si**2*y_new         &
            +si*(1.d0-si)**2*h*f_old            &
            +si**2*(si-1.d0)*h*f_new
    end subroutine point

!rescale the Nordsieck vector
    subroutine varistep(dim,y1,y2,y3,y4,y5,delta)
     implicit none
     integer:: dim
     double precision ::delta
     double precision, dimension(dim):: y1,y2,y3,y4,y5
     y1=y1
     y2=delta*y2
     y3=(delta**2)*y3
     y4=(delta**3)*y4
     y5=(delta**4)*y5
    end subroutine varistep
```

5.2.3 Newton iteration

```
!Newton iteration
    subroutine iter(dim,x,y0,rhs,f,jex,lambda,fstage,&
                   tol,r,h,pd,matrix,justfactorize,&
                   h_lastfactorize,converge,ipiv,  &
                   nfactorize,njex,step,stage,ncallf)

     implicit none
     external f, jex
     integer ::dim, maxstep,stepp,info,i
     integer ::stage,step,ncallf,count,j
     integer, dimension(dim)::ipiv
     double precision, dimension(dim):: y0,ystage,ydot
     double precision, dimension(dim):: phi,fstage,rhs
     double precision, dimension(dim,dim)::pd,identy
     double precision ::x,lambda,tol,r,h
     double precision ::sum, sum_old
     double precision, dimension(dim,dim)::matrix
     logical ::justfactorize,recomputej,converge,notconverge,full
```

```
double precision ::h_lastfactorize
integer ::nfactorize,njex

 maxstep=6
 sum_old=1.d0
 identy=0.d0
 do i=1,dim
    identy(i,i)=1.0d0
 end do
 ystage=y0
 stepp=0
 full=.false.
 if(full)then
   do  i=1,maxstep
     call jex(dim,x,ystage,pd)
     pd=h*lambda*pd
     matrix=identy-pd
     call f(dim,x,ystage,ydot)
     ncallf=ncallf+1
     phi=ystage-h*lambda*ydot-rhs
     call DGETRF(dim,dim,matrix,dim,ipiv,info)
     call DGETRS('N',dim,1,matrix,dim,ipiv,phi,dim,info)
     ystage=ystage-phi
     nfactorize=nfactorize+1
     njex=njex+1
     sum=0.d0
     do j=1,dim
      sum=sum+(phi(j))*(phi(j))
     end do
     sum=dsqrt(sum)
     if(sum<tol)then
      converge=.true.
      exit
     end if
    end do
    fstage=(ystage-rhs)/(h*lambda)
    y0=ystage
 else
  recomputej=.false.
  sum=1.d0
```

```
       converge=.false.
       recomputej=.false.
      do
        stepp=stepp+1
        do i=1,maxstep
         sum_old=sum
         call f(dim,x,ystage,ydot)
         ncallf=ncallf+1
         phi=ystage-h*lambda*ydot-rhs
         call DGETRS('N',dim,1,matrix,dim,ipiv,phi,dim,info)
         ystage=ystage-phi
         sum=0.d0
         do j=1,dim
           sum=sum+(phi(j))*(phi(j))
         end do
         sum=dsqrt(sum)
!check convergence
         if(sum<tol)then
           converge=.true.
           exit
         else
          if(sum>(sum_old*2.d0)) then
            exit
          end if
         end if
        end do
        if (sum<tol) then
         fstage=(ystage-rhs)/(h*lambda)
         y0=ystage
         converge=.true.
         exit
        else
         if(justfactorize.and.recomputej)then
           converge=.false.
           exit
         end if
!if factorization has been done, recompute J
         if(justfactorize)then
           recomputej=.true.
           call jex(dim,x,ystage,pd)
           njex=njex+1
         end if
!do factorization with current h
         matrix=identy-h*lambda*pd
         call DGETRF(dim,dim,matrix,dim,ipiv,info)
         nfactorize=nfactorize+1
         h_lastfactorize=h
         justfactorize=.true.
        end if
       end do
     end if

    end  subroutine iter
```

Bibliography

[1] F. Bashforth & J. C. Adams, *An attempt to test the theories of capillary action by comparing the theoretical and measured forms of drops of fluid, with an explanation of method of integration employed in constructing the tables which give the theoretical forms of such drops*, Cambridge University Press, 1883.

[2] K. Burrage & J. C. Butcher, *Non-linear stability for a general class of differential equation methods*, BIT, 20, 185-203, 1980.

[3] K. Burrage, J. C. Butcher & F. H. Chipman, *An implementation of singly-implicit Runge-Kutta methods*, BIT, 20, 326-340, 1980.

[4] J. C. Butcher, *Numerical methods for ordinary differential equations in the 20th century*, J. Comput. Appl. Math., 125, 1-29, 2000.

[5] J. C. Butcher, *The numerical analysis of ordinary differential equations*, John Wiley & Sons, 1987.

[6] J. C. Butcher, *On the convergence of numerical solutions to ordinary differential equations*, Math. Comp, 20, 1-10, 1966.

[7] J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, 2003.

[8] J. C. Butcher, *Diagonally-implicit multi-stage integration methods*, Appl. Numer. Math. 11, 347-363, 1993.

[9] J. C. Butcher, P. Chartier and Z. Jackiewicz, *Nordsieck representation of DIMSIMs*, Numer. Alg. 16, 209-230, 1997.

[10] J. C. Butcher, P. Chartier and Z. Jackiewicz, *Experiments with a variable-order type* 1 *DIMSIM code*, Numer. Alg. 22, 237-261, 1997.

[11] J. C. Butcher and Z. Jackiewicz, *Construction of diagonally implicit general linear methods of type 1 and 2 for ordinary differential equations*, Appl. Numer. Math. 21 385-415, 1996.

[12] J. C. Butcher and Z. Jackiewicz, *Implementation of diagonally implicit multistage integration methods for ordinary differential equations*, SIAM J. Numer. Anal. 34, 2119-2141, 1997.

[13] J. C. Butcher, *General linear methods for stiff differential equations*, BIT, 41, 240-264, 2001.

[14] J. C. Butcher and W. M. Wright, *The construction of practical general linear methods*, BIT, 43, 695-721, 2003.

[15] C. W. Cryer, *On the instability of high order backward-difference multistep methods*, BIT, 12, 17-25, 1972.

[16] G. Dahlquist, *Convergence and stability in the numerical integration of ordinary differential equations* Math. Scand. 4, 33-53, 1956.

[17] G. Dahlquist, *A special stability problem for linear multistep methods*, BIT, 3, 27-43, 1963.

[18] B. L. Ehle, *On Páde approximations to the exponential function and A-stable methods for the numerical solution of initial value problems*, Report 2010, University of Waterloo, 1969.

[19] L. Euler, *Institutions calculi integralis* Volumen Primum, Opera Omnia XI, G. Teubneri, Lipsiae et Berolini MCMXIII, 1768.

[20] D. Garfinkel and C. B. Marbach, *Stiff differential equations*, Ann. Rev. Biophys. Bioeng, 6, 525-542, 1977.

[21] C. W. Gear, *Hybrid methods for initial value problems in ordinary differential equations*, SIAM J. Numer. Anal., 2, 69-86, 1965.

[22] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, 1971.

[23] C. W. Gear, *DIFSUB for solution of ordinary differential equations*, Comm. ACM. 14, 69-86, 1971.

[24] C. W. Gear *Runge-Kutta starters for multistep methods*, ACM Trans. Math. Software, 6, No. 3, 263-279, 1980.

[25] K. Gustafsson, M. Lundh and G. Söderlind, *A PI stepsize controller for the numerical integration of ordinary differential equations*, BIT, 28, 270-287, 1988.

[26] E. Hairer and G. Wanner, *Solving ordinary differential equations II: Stiff and differential-algebraic problems*, Springer-Verlag, Berlin, 1991.

[27] K. Heun, *Neue Methode zur approximativen Integration der Differentialgleichungen einer unabhängigen Veränderlichen*, Math. Phys. 45, 23-38, 1900.

[28] Z. Jackiewicz, *Implementation of DIMSIMs for stiff differential systems*, Appl. Numer. Math. 42, 251-267, 2002.

[29] W. Kutta, *Beitrag zur näherungswenisen Integration totaler Differentialgleichungen*, Math. Phys. 46, 435-453, 1901.

[30] W. M. Lioen and J. J. B. de Swart, *Test set for initial value problem solvers*, CWI test set, 1999.
http://pitagora.dm.uniba.it/ testset/index.htm

[31] F. R. Moulton, *New methods in exterior ballistics*, University of Chicago Press, 1926.

[32] A. Nordsieck, *On numerical integration of ordinary differential equations*, Math. Comp. 16, 22-49, 1962.

[33] A. Prothero and A. Robinson, *On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations*, Math. Comp., 28, 145-162, 1974.

[34] H. H. Robertson, *The solution of a set of reaction rate equations* In J. Walshed. Numer. Anal., an Introduction, Academic Press, 178-182, 1966.

[35] C. Runge, *Über die numerische Auflösung von Differentialgleichungen*, Math. Ann. 46, 167-178, 1895.

[36] E. Schafer, *A new approach to explain the 'high irradiance responses' of photomorphogenesis on the basis of phytochrome*, J. of Math. Biology, 2, 41-56, 1975.

[37] L. F. Shampine, *Implementation of implicit formulas for the solution of ODEs*, SIAM Journal of Scientific and Statistical Computing, 1, 103-118, 1980.

[38] A. D. Singh, *Parallel diagonally implicit multistage integration methods for stiff ordinary differential equations*, PhD thesis, The University of Auckland, 1998.

[39] W. M. Wright, *Practical general linear methods*, MSc thesis, The University of Auckland, 1998.

[40] W. M. Wright, *General linear methods with inherent Runge-Kutta stability*, PhD thesis, The University of Auckland, 2003.

[41] W. M. Wright, *Explicit general linear methods with inherent Runge-Kutta stability*, Numer. Alg., 31, 381-399, 2002.

[42] http://www.netlib.org/lapack.