New Zealand Institute for Advanced Studies (NZIAS)
Massey University Albany Campus Auckland
New Zealand

Design and construction of software for general linear methods

Saghir Ahmad
June 2016

Supervisors: *Professor J. C. Butcher*
Associate Professor W. L. Sweatman

A thesis submitted in fulfilment of the requirements of Doctor of Philosophy in Mathematics

Abstract

The ultimate goal in the study of numerical methods for ODEs is the construction of such methods which can be used to develop efficient and robust solvers. The theoretical study and investigation of stability and accuracy for a particular class of methods is a first step towards the development of practical algorithms.

This thesis is concerned with the use of general linear methods (GLMs) for this purpose. Whereas existing solvers use traditional methods, GLMs are more complex due to their complicated order conditions. A natural approach to achieve practical GLMs, is to first consider the advantages and disadvantages of traditional methods and then compare these with a particular class of GLMs. In this thesis, GLMs with IRKS– and F–properties are considered within the type 1 DIMSIMs class. The freedom of choice of free parameters in IRKS methods is used here to test the sensitivity and capability of the methods.

A complete ODE software package uses many numerical techniques in addition to the methods considered. These include error estimation, interpolation for continuous output, etc.. Existing ODE software is a combination of these techniques and much work has been done in the past to improve the capability of these traditional methods. The approach has been largely heuristic and empirical. These are developed by fitting all these techniques into one algorithm to produce efficient ODE software.

The design of the algorithm is the main interest in the thesis. An efficient solver will be in (h, p)-refinement mode. This design includes many decisions in the whole algorithm. These include selection of stepsize and order for the next step, rejection criteria, and selection of stepsize and order in case of rejection. To design such a robust algorithm, the Lagrange optimisation approach is used here. This approach for the selection of stepsize and order avoids the use of several heuristic choices and gives a new direction for developing reliable ODE software. Experiments with this design have been carried out on non–stiff, mildly–stiff and some discontinuous problems and are reported in this thesis.

Acknowledgments

First and foremost, my deepest thanks to God for giving me strength to complete this study.

From the depths of my heart, I would like to express my gratitude to my supervisor Professor John Butcher. Your deep knowledge of the subject, great experience, enthusiasm and patience have always been a source of motivation for me. During this long period, I faced a few serious personal crises and you encouraged me each time and made it easy for me to concentrate on my studies. I could not have been successful without your support. Your lectures, long discussions and valuable suggestions on this thesis have been very beneficial for me.

I wish to express gratitude to my co-supervisor Associate Professor Winston Sweatman for his continuous support and feedback on my work. You have always been encouraging and ready for discussions whenever I needed. You have given several opinions on my work which always helps me to work consistently.

I am also very grateful to my mentor Dr Helmut Podhaisky for his knowledgable support and hospitality during my study trip to Germany in 2012. Your deep understanding of the work, strength in this field and valuable suggestions on my thesis, have always been inspiring myself.

I am also grateful to the New Zealand Institute for Advanced Studies (NZIAS) especially Professor Gaven Martin for making it possible for me to continue my research work with John. Here, I am also thankful to the INMS management at Massey University for their support.

A very special thanks to all the other members of the numerical analysis group at the University of Auckland for their friendship and discussions. The weekly workshops have been source of understanding the whole subject and to become up–to–date with the active and recent research areas.

During my study in Auckland, I have found a very nice circle of friends and families, I would like to express my gratitude to all of them for their in–time support and guidance.

Specials thanks to my family members who support me to do this, especially my daughter Saba Ahmad.

I am very grateful to the Higher Education Commission (HEC) of Pakistan and the Marsden Fund of New Zealand for financial support for PhD studies and my study trips respectively.

Dedication

This thesis is dedicated to my beloved late daughter Maha Ahmad who is the only human being I found, just loving me and gave me the meaning of love before she left us to heavens in November of 2013.

Contents

Glossary

p order of the method. 12, 26

q stage order of the method. 12, 26

r number of elements in the data (Nordsieck) vector. 11, 26

s number of stages in the method. 11, 26

ARK almost Runge–Kutta method. 16, 92

DESIRE diagonally extended singly implicit Runge–Kutta effective order method. 22

DIMSIM diagonally implicit multistage integration method. i, 7, 16, 21, 22, 24, 25, 27, 28, 30, 32, 35, 45, 127

DIRK diagonally implicit Runge–Kutta method. 7, 8, 23, 24

FSAL first stage as last. 23, 35

IRKS inherent Runge–Kutta stability. i, xiii, xv, xvi, 9, 15, 17–19, 25, 31–37, 40, 41, 45, 52, 62, 65, 68, 69, 73, 81–91, 93–101, 104, 126–129

SIRK singly implicit Runge–Kutta method. 4, 5, 7, 8, 22, 25, 31, 36

List of Figures

List of Tables

1 Introduction

Natural physical phenomena, from the motion of fluids in each part of the human body to the movement of stars at vast distances, can be translated into mathematics for study; these mathematical models are mostly systems of ordinary differential equations (ODEs). The recent deep study of such phenomena requires classification of ODEs due to their behaviour and characteristics. The analytical solution of ODEs in each case is not always possible or available. In these cases the existing gap in such solutions is filled using approximate solutions. This requires a tool known as an ODE solver and to produce it, the two necessary elements are numerical methods and computers (or other electronic machines).

The algorithm design of ODE solver, using some efficient methods, is the main interest in this thesis. Solvers based on the traditional methods have been constructed and are used efficiently. For example, the MATLAB ODE suite, Maple solvers and some famous codes written in Fortran. Recent research uses the general linear methods (GLMs) for this purpose.

ODE solvers are mostly task related. There exists no single algorithm that can handle every kind of system of ODEs such as initial value problems (IVPs), boundary value problems (BVPs), discontinuous problems, delay differential equations (DDEs) and differential algebraic equations (DAEs). A general purpose ODE software should efficiently solve simple problems and should be capable of handling complicated problems. How can we classify simple and complicated problems? For example, if a discontinuity occurs in the solution of an IVP, it is no longer simple; however, a good solver should handle this. Similarly, DDEs are a kind of discontinuous problem and should be treated in the same way. Stiffness is another characteristic of the ODEs. Stiff problems are not simple problems and need A-stable methods and hence particular solvers. As compared to the stiff problems, a non-stiff problem is considered as a simple problem; although complexity occurs when a problem becomes mildly stiff, it may be just at some point of the solution. The aim here is to test how the GLMs can be efficiently

implemented to solve non–stiff problems as well as mildly stiff problems and discontinuous problems.

How could a general ODE solver handle these problems efficiently? This question is somehow related to the design of algorithms in a solver and the requirement of the ODE due to classification. So, it is better to understand the types of ODEs considered in this thesis, to show how these affect the solver. This may require performing something different at some particular point(s) of the integration. Some characteristics of ODEs are presented in the Section 1.1. In Section 1.2, without going into the history of traditional methods and the modern theory of order and stability of methods, a brief introduction to those traditional and non–traditional but practical classes of methods are mentioned; these can be considered as strong competitors for GLMs. These advantages and disadvantages of the methods and of their corresponding ODE solvers, motivates us to explore the implementation of such a class of GLMs that has all these advantages as well as no such disadvantages; these motivations are given in Section 1.3. The outline of the thesis is presented in Section 1.4.

1.1 Classification of ODEs

The general form of an IVP is

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0, \tag{1.1}$$

where $x \in R$, $y(x) \in R^m$ and $f : R \times R^m \to R^m$ and m is the dimension of the problem. The above form is a non autonomous form because $f(x, y(x))$ depends explicitly on x, but can be converted to an autonomous form without such an explicit dependence by adding the differential equation $x' = 1$ to the system. The existence and uniqueness of the solutions is related to the following Lipschitz condition [32].

Definition 1.1.1 *The function $f : [a, b] \times R^N \to R^N$ is said to satisfy a 'Lipschitz condition in its second variable' if there exist a 'Lipschitz constant' L and a norm $\|\cdot\|$, such that for all $x \in [a, b]$ and all $y, z \in R^N$,*

$$\|f(x, y) - f(x, z)\| \leq L\|y - z\|.$$

This leads to the following existence and uniqueness theorem [32] along with the proof.

Theorem 1.1.1 *Consider the initial value problem (1.1), where $f : [a, b] \times R^N \to R^N$ is continuous in its first variable and satisfies a Lipschitz condition. Then there exists a unique solution to this problem.*

There are many classifications of ODEs. The classifications discussed here will be based on a few of those characteristics of the ODEs which cause issues and need special attention during the numerical computation at some particular point(s) of the integration or maybe for the whole interval.

1.1.1 Classification due to stiffness

For the numerical solution, classification of ODEs could be based on the amount of stiffness. Without relating to the numerical methods, stiffness is very high damping in one or more solutions of a system

of ODEs that occur in some part or the whole time interval. This phenomena can be observed by the magnitude of the Lipschitz constant because stiff problems have very large values of the Lipschitz constant. Consider an inner product on R^N denoted by $\langle y, z\rangle$ and the corresponding norm defined by $\|y\|^2 = \langle y, y\rangle$. A reasonable one–sided Lipschitz constant is defined [32] as:

Definition 1.1.2 *The function $f : [a, b]\times R^N \to R^N$ is said to satisfy a 'one–sided Lipschitz condition' if there exists a 'one–sided Lipschitz constant' ℓ, such that for all $x \in [a, b]$ and all $y, z \in R^N$,*

$$\langle f(x, y) - f(x, z), y - z\rangle \leq \ell\|y - z\|^2.$$

As f could have very large, negative one–sided Lipschitz constant, the following theorem [32] is very useful in this case.

Theorem 1.1.2 *If f satisfies a one–sided Lipschitz condition with one–sided Lipschitz constant l, and y and z are both solutions of $y'(x) = f(x, y(x))$, then for all $x \geq x_0$*

$$\|y(x) - z(x)\| \leq \exp\left(l(x - x_0)\right)\|y(x_0) - z(x_0)\|.$$

This ODE characteristic is very important for numerical computation and classifies ODEs into stiff and non–stiff problems. The latest review about the definition of stiffness is given in [83]. Stiff ODEs can be solved by highly stable numerical methods but at high computational cost. However, in the case of non–stiff problems, such a high stability is not required and hence less expensive methods can be used. Our concern here is not to deal with the highly stiff problems. The question explored here is to test the efficiency of the non–stiff solver based on the explicit GLMs in the case of mildly stiff problems.

1.1.2 Classification due to discontinuities

Problems with discontinuities in the solution(s) and in the higher derivatives also need special attention while solving numerically. The trouble is not with the numerical methods in this case but with the implementation. There are two approaches to handle the crossing of discontinuities. One is to use the general purpose ODE solvers. While crossing the discontinuity event, failures when attempting steps tend to these solvers reducing the stepsize. In this approach, there is no *a priori* information given by the user nor any event analysis in the code. The other approach is to make a special code for the ODEs with discontinuities and design it for such event handling. In this approach, there may or may not be *a priori* information given by the user though using special structures like sensitivity analysis for event handling.

A further subclass of discontinuous problems called DDEs is also very important. Many real life problems are presented by systems of DDEs, for example, spread of infections, immune response models, nervous systems, neural networks and cell kinetics. These are functional differential equations and need special structure of the solver to deal with such problems. DDEs are a kind of discontinuous problem as defined in the following definition.

Definition 1.1.3 *A **DDE** is a differential equation of the form*

$$y'(t) = f(t, y(t), y(t-d_1), y(t-d_2), \ldots, y(t-d_k)),$$

*where $d_i \geq 0$ for $i = 1, 2, \ldots, k$, are called **delays** or **lags**.*

Whereas, the pantograph equation is a special case of DDE and it is of the form

$$y'(t) = f\left(t, y(t), y(\lambda t)\right),$$

where $0 < \lambda < 1$. DDEs can be considered as ODEs with some consecutive open intervals of integration. DDE solvers need to manage data of the past steps to make it available for the existing step. Such approaches have been considered by Butcher [24] within the code STRIDE having singly implicit Runge–Kutta (SIRK) methods in the kernel.

In this classification of ODEs, problems which are not discontinuous are called neutral problems. A very interesting example of discontinuity, square–path problem, is used in this thesis. This problem is very challenging and helps to check the robustness of the code.

Before going into the detail of GLMs and existing ODE software, it is necessary to discuss some of the traditional methods. In the next Section, we only discuss those classes of methods that could be close competitors of the GLMs considered in the thesis.

1.2 Numerical methods and ODE software

The starting point for the numerical methods for ODEs is the Euler method, which is the most simple form of traditional method as well as general linear method (GLM). In the Euler method, the solution depends upon the two quantities; the solution and its derivative at the previous point x_{n-1}. Extensions of this idea gave birth to two main traditional classes of numerical methods for ODEs. The first approach, to extending the Euler method, is to use a linear combination of solutions at the end of more than one previous step to approximate the solution. These are the multistep methods with the general form

$$y_n = \sum_{i=1}^{k} \alpha_i y_{n-i} + h \sum_{i=0}^{k} \beta_i f(y_{n-i}), \tag{1.2}$$

called a k–step method. These methods are characterised by the polynomials:

$$\alpha(z) = 1 - \alpha_1 z - \alpha_2 z^2 - \cdots - \alpha_k z^k,$$
$$\beta(z) = \beta_0 z^k + \beta_1 z^{k-1} + \cdots + \beta_k.$$

Adams methods, obtained by taking all α's zero except $\alpha_1 = 1$ in the α–polynomial, are one of the best competitors. In these methods, the zeros of the β–polynomial are related to the eigenvalues of

the matrix V of GLMs (see Chapters 2 and 3). This makes an interesting competition as both the zeros and the eigenvalues are all zero except one. Thus both the competitors have the same property. Another reason that makes these methods more competitive is due to variable stepsize implementation. The adaptation of stepsize is necessary for efficient implementation and the scale and modify technique is possible in this case which makes this class very practical.

In this thesis, Adam-Moulton pairs are taken from the Adams family and GLMs with exactly same error constants are constructed and then compared in Chapter 6. The comparison criterion at this starting stage is to measure the **error per function evaluation**. This is the intrinsic accuracy of a method. These Adams pairs are also known as predictor–corrector pairs and are pre-Nordsieck forms of multistep methods. Whereas the Nordsieck methods have a special input/output vector, consisting of r scaled higher derivatives as given below, where h_n is the stepsize in the n^{th} step.

$$y^{[n]} = \begin{bmatrix} y_n \\ h_n y'_n \\ h_n^2 y''_n \\ \vdots \\ h_n^r y_n^{(r)} \end{bmatrix}. \tag{1.3}$$

The second approach, to extending the Euler method, is to restrict the use of previous steps to the leading one only and to use the linear combination of derivatives at some internal points of the existing interval which are called *stages*. These are multistage methods better known as a large family of Runge–Kutta methods. The general form is

$$Y_i = y_{n-1} + h \sum_{j=1}^{s} a_{ij} f(x_{n-1} + c_j h, Y_j), \quad i = 1, 2, \dots s, \tag{1.4a}$$

$$y_n = y_{n-1} + h \sum_{i=1}^{s} b_i f(x_{n-1} + c_i h, Y_i), \tag{1.4b}$$

where Y_i are the stage values and y_n is the final output after each step, c_i's are called abscissae, s is the number of stages, and b_i's are called weights. With the discovery of modern theory of the Runge-Kutta methods [32], much work has been done to find the most efficient methods. Many famous classes of methods have been efficiently used to construct robust ODE solvers. SIRK methods are good examples of practical multistage methods that are used in variable stepsize variable order mode in the code STRIDE [12] and then extended [24] for DDEs as well. For variable stepsize implementation of Runge–Kutta methods, there is no extra work required unlike the scale and modify technique required in the case of Nordsieck methods.

In spite of these two traditional families of methods, there are some non–traditional methods as well that were efficiently used at their time of discovery. It is said that extrapolation methods are good only for easy or simple problems. The extrapolation technique for numerical quadrature was successfully used by Romberg [81] in 1955 using the trapezoidal rule. Later on, Bulirsch and Stoer [9] developed a code ODEX based on this idea. This approach was further extended in [3] for stiff

problems. The vector rational interpolation algorithm, based on the Bulirsch–Stoer formula (scalar approach) in [84], was developed by Sweatman in [85]. A recent variable stepsize implementation of extrapolation of Runge–Kutta methods is numerically tested and reported in [70] by Gorgey.

Other non-traditional methods are based on Taylor series. A Taylor series method of order p is simply the expression of the first $(p+2)$ terms in the Taylor expansion given by

$$y_n = y_{n-1} + hf(x_{n-1}, y_{n-1}) + \frac{h^2}{2!} f_2(x_{n-1}, y_{n-1}) + \cdots + \frac{h^p}{p!} f_p(x_{n-1}, y_{n-1}),$$

where $f_2, \ldots, f_p$ are derivatives of the given function. The computation of these derivatives is not always possible or easy to achieve. These methods are expensive though the Lagrange theory for (h, p)–control, discussed in Chapter 5, can be used in this case. This is due to the availability of good approximations to the higher derivatives at each step on which error estimators are mainly based. So it is possible to compare the GLMs with Taylor series methods in the same way as mentioned for the above classes of methods. The Euler method is also the Taylor expansion up to the first two terms. So this idea can be naturally extended with the possibility of algebraically evaluating the higher derivatives. Some of the early famous programs based on this approach were by Gibbons [68] and Barton et al. [4, 5, 6].

Many generalisations of these methods can be found in the literature with different advantages and disadvantages. The most general and natural form known as general linear methods were first introduced in 1966 by Butcher [15]. These have the flavour of both the traditional classes. So it is possible to obtain such a class of GLMs that have all the advantages of the traditional classes and at the same time, avoid issues that exist in the traditional methods. Such a class of GLMs is considered in this thesis and discussed in the later chapters.

The ultimate goal in the study of numerical methods for ODEs is to produce such methods which can be used to develop an efficient ODE solver that optimises implementation cost, accuracy and stability. Theoretical study and investigation for obtaining stability and accuracy in a particular class of methods is a prior stage. After all, a search for practical methods is necessary for developing a good solver.

The invention of modern computers has made the area of numerical analysis very active, attractive and practical. Various software have been developed to solve a wide range of differential equations, though it is still a challenge to develop a general purpose ODE software up to the level of linear algebra software. A brief discussion of this level–comparison is beautifully reported in [33]. For example, when stiffness occurs in the systems of ODEs, it requires more care and in fact additional tools to solve the problems. Nowadays, general purpose codes are used that have the adaptability property to switch between stiff and non-stiff sections.

One famous set of codes named DIFSUB [67] was based on linear multistep methods. There are many implementation issues, in general, for multistep methods like starting procedure, error estimation and change of stepsize. The best feature in these methods is the vector passed from step to step which is in Nordsieck form and which is discussed in later sections. Due to this it is possible to use the *scale*

and modify technique. This technique is used to change the stepsize at a very low implementation cost. Following the DIFSUB code, a number of scientists used Nordsieck form of methods in their codes. These include LSODE [76] and VODE [8]. It is necessary to discuss here the work of Dahlquist [63, 64] who gives the limitations of multistep methods. He proved that multistep methods are A–stable (a characteristic of a method to solve stiff problems [1, 32]) only up to order 2. This restricts the use of multistep methods in stiff solvers. Due to order barriers of multistep methods some other approaches were also used. A famous example here is predictor–corrector pairs like $P(EC)^{\mathsf{N}}E$ and $P(EC)^{\mathsf{N}}$ which are variations of the predict–evaluate–correct scheme (PEC), originally developed by Milne [79].

Multistage methods have high stability properties and due to this, they have been successfully used for stiff and non–stiff solvers. Runge [82] first developed a method with more than one stage. After many attempts by various mathematicians, multistage methods have got the name of Runge–Kutta methods. And Butcher [16] developed the algebraic theory of the methods. There exist embedded pairs of methods in the big family of Runge–Kutta methods used for error estimation. The difference between each in the pair is used for error estimation. Although this is obtained at a very low implementation cost, it is not asymptotically correct. Stiff codes based on Runge–Kutta methods are much more successful compared to multistep methods but overall their implementation cost is very high. For example, the implicit methods by Butcher [14] based on Gauss, Radau and Lobatto quadrature formulae. Later on, Ehle [66] constructed A–stable Radau IA, Radau IIA, Lobatto IIIA, Lobatto IIIB and Lobatto IIIC methods. These methods are then used efficiently by Hairer [75] in his famous implementation of Radau IIA methods. Although the code works well, there are still many difficult design questions, a few of these questions are reported in [1]. Much effort has been done to deal with the implicit nature of these methods for stiff problems. SIRK [17] and DIRK [2] structures were also considered as an alternative to expensive fully implicit methods. Another achievement is the STRIDE code by Burrage, Butcher and Chipman [12] . These have some drawbacks including additional transformation cost and misplacement of some abscissae. Butcher and Chartier [39] discussed these drawbacks using the idea of effective order and then implemented new algorithms [42]. The famous MATLAB ODE suite is also based on Runge–Kutta methods.

Butcher [15] unified the two traditional approaches for the first time. There are many generalisations of the traditional methods in the literature which are now special cases of the GLMs. A wider approach to the proposed methods are DIMSIMs by Butcher [25]. Instead of generalising, a better approach is to gather all the fruitful properties from each class and then look for such design of methods in GLMs. One of the leading aims in the project is to construct GLMs with a special design; these are discussed in later chapters.

1.3 Motivations

The process of developing an ODE software starts from a numerical method and needs a specific design to construct it. The research on numerical methods for ODEs can be divided into three different areas: construction, analysis and implementation. It is rare to work deeply in all three areas at the same time. However, to produce an ODE software in a good form we need this to happen. This gives us a

notion towards the project.

The two traditional classes have their own pros and cons. To get the advantages in unified form, a natural desire is to combine the two extensions of the Euler design. This mainly motivates and leads a person to explore the most general form of numerical methods for ODEs. Further, to see the importance of the design of the GLMs considered here, we first look at the limitations of the traditional methods. Our focus here is mainly on the implementation questions.

To understand this, we first take multistep methods. Along with the theoretical constraints of *stability barriers* [63, 64], which particularly restricts their use in stiff solvers, there are many implementation constraints too. For example, it is very difficult, in general, to implement these methods in variable stepsize and variable order mode. Although it is possible in a sense, it requires a significant increase in cost. It is also necessary to mention here the importance of the *scale and modify* technique that is very reliable and efficient for a particular type of multistep method known as the *Nordsieck methods* [80]. This leads to one of the features, considered in the design of the GLMs, used in this thesis. Furthermore, multistep methods need a starting method as well.

Contrary to multistep methods, multistage methods have high stability properties. This makes the Runge–Kutta methods more in demand for developing software for stiff ODEs. These methods include big families of highly stable methods such as Radau and Lobatto methods. The difficulties in the use of these methods are again some implementation challenges. The implicit nature of the internal stage systems makes it much more expensive when the Jacobian of the matrix appears in its nonlinear part. This is especially true when a code rejects a stepsize or order; the expense of calculating that Jacobian again requires additional function evaluations and hence makes the rejections unbearable in terms of cost. Many approaches have been used to decrease the cost of these methods by keeping the stability requirements up to an acceptable level. This struggle has produced somehow reliable approaches to cover the gap. These include DIRK and SIRK structures of the coefficient matrix A in the Runge–Kutta methods. It proposed the optimal structure of the matrix A in the GLMs as well. It is expected that this will possibly preserve an acceptable stability along with much less computational cost. Also, the Runge–Kutta methods lack a reliable error estimate in general which plays an important role for changing the stepsize as well as the order. Interestingly, the MATLAB ODE suite based on the Runge–Kutta methods and some non–theoretical approaches of error estimations are successfully used although they are not in variable order mode. All these pros and cons motivate the design of GLMs considered in the thesis.

In spite of the necessities impelling us from the disadvantages of the traditional methods, there are some requirements of the ODE software design as well. In the existing ODE solvers, mainly the variable stepsize variable order codes, there are some heuristic approaches based on an empirical basis. For example, the commonly used standard variable stepsize scheme is based on error estimation. However, when it is implemented, a certain limit on the stepsize ratio r is implemented, usually bound to lie within $[0.5, 2]$. This is a non–theoretical approach. This limit restricts the stepsize to not change too quickly in any case and to avoid potentially damaging the stability of the solution.

Another design question, in (h, p)–refinement mode, is related to the switching between methods of consecutive order which is still not deeply studied for ODE solvers. The RADAU code by Hairer [75]

is a successful variable order code though the methods used in that code have very fluctuating order variation. The GLMs used in this thesis are of consistent and consecutive order. But the problem is not completely solved yet, because there is still an important question. How it can be decided to change the method to one order higher or to jump to a method with one order smaller than the order of the method used in the existing step?

In order to decide which method should be selected for the next step, some reliable information is required. A reasonable choice of the method for the next step could be the one that minimises both the cost and the error produced in that step. In other words, such a method should be chosen that allows the use of a bigger stepsize and produces a smaller error at the same time as compared to the values (of stepsize and error) for the other choices of methods. These two quantities, so–called reliable information, are difficult to analyse separately, although a design was proposed by Butcher [20, 23] in which these two quantities are efficiently combined in one frame using the optimisation approach.

The reliable information is simply the error estimation which is efficiently possible for many other traditional methods as well. For variable order code, error estimation is required not only for the method used in the existing step, but it is also necessary to know what the errors would have been if the methods, with one order higher and one order lower than the order of the method used in the existing step, were used for the next step. Such reliable error estimators are possible and available for the GLMs considered here. So the general idea given in [20] for ODE software is used in this thesis which is possible using the GLMs with IRKS and F–properties as a kernel.

1.4 Thesis outline

The work of this thesis can be considered as a step towards the development of ODE software using GLMs. So it is necessary to first discuss GLMs in general, with construction of practical methods from the general form and then their implementation. GLMs are briefly discussed in Chapter 2 along with some existing classes of practical methods as examples.

Chapter 3 gives a detailed discussion about the design of methods. Along with this, the advantages required from the GLMs are discussed and then described in the form of simplifying assumptions. These conditions are then imposed on the most general form of the methods to obtain the desired practical GLMs. At the end, an algorithm for constructing these methods is presented.

Chapter 4 is a deep discussion about the implementation of the GLMs which is always challenging and not straight forward. Before the big goal of design of ODE software in (h, p)–refinement mode, it is better to discuss each implementation procedure separately. These include automatic starting procedure, stepsize control with updating the data vector, and error estimators.

In Chapter 5, the idea of the Lagrange multipliers is analysed to use for stepsize selection and for stepsize and order selection. Using this approach, algorithms for the scale and modify technique for variable order codes and for selection of stepsize and order are constructed and presented.

Finally in Chapter 6, the results of numerical experiments are reported which confirm the importance of the methods as well as the design based on the Lagrange multipliers.

2
General linear methods

The whole journey starting from the birth of the Euler method and its two classical generalisations and through further generalisations leads to the general form of numerical methods for ODEs, known as GLMs. All this struggle is to develop practical methods that can be the kernel of a good ODE solver. In the last half a century, many theoretical and practical approaches have been used to achieve this goal. Without going into the depth of every theoretical aspect of numerical analysis, this chapter only discusses the necessary parts of the GLMs which are required to understand the rest of the thesis. These include the representation of GLMs used in the thesis and a few preliminary properties of numerical methods that are also essential for GLMs. Along with these, the stability of the methods is also very important; this is presented in Section 2.3 including the plots of the stability regions of the methods used in this thesis. The construction of the practical GLMs used in this thesis is presented in Chapter 3.

2.1 Representation of GLMs

A general linear method is a multistage and multivalue method which is used to approximate the numerical solutions of a system of ODEs of the autonomous form

$$y'(x) = f(y(x)), \quad y(x) \in R^m, \quad f : R^m \to R^m. \tag{2.1}$$

Let $Y_i^{[n-1]}$, $i = 1, 2, \ldots$,s, be the s internal stages and $y_j^{[n-1]}$, $j = 1, 2, \ldots$,r, be the r outcoming quantities computed at step number $(n-1)$ or input of step number n. Let h be the stepsize at the

n^{th} step, then the general linear method for approximating these quantities at the n^{th} step is

$$Y_i^{[n]} = h\sum_{j=1}^{s} a_{ij}F_j + \sum_{j=1}^{r} u_{ij}y_j^{[n-1]}, \quad i = 1, 2, ...s,$$
$$y_i^{[n]} = h\sum_{j=1}^{s} b_{ij}F_j + \sum_{j=1}^{r} v_{ij}y_j^{[n-1]}, \quad i = 1, 2, ...r,$$

(where $F_j = F(Y^{[n]})_j = f(Y_j^{[n]})$) or in compact form

$$Y^{[n]} = h(A \otimes I_m)F(Y^{[n]}) + (U \otimes I_m)y^{[n-1]}, \tag{2.2a}$$
$$y^{[n]} = h(B \otimes I_m)F(Y^{[n]}) + (V \otimes I_m)y^{[n-1]}, \tag{2.2b}$$

where

$$Y^{[n]} = \begin{bmatrix} Y_1^{[n]} \\ Y_2^{[n]} \\ \vdots \\ Y_s^{[n]} \end{bmatrix}, F(Y^{[n]}) = \begin{bmatrix} f(Y_1^{[n]}) \\ f(Y_2^{[n]}) \\ \vdots \\ f(Y_s^{[n]}) \end{bmatrix}, \text{and} \quad y^{[n-1]} = \begin{bmatrix} y_1^{[n-1]} \\ y_2^{[n-1]} \\ \vdots \\ y_r^{[n-1]} \end{bmatrix}.$$

For convenience, coefficient matrices (A, U, B, V) in (2.2) can be combined in the form of a partitioned matrix of order $(s + r)$ as

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

Here, the stage values Y_i are not passed from step to step and therefore the abscissae c_i, $i = 1, 2, \ldots, s$ are posed to be within the interval $[0, 1]$ such that

$$Y_i = y(x_{n-1} + c_i h) + O(h^{q+1}), \tag{2.3}$$

where q is called *stage-order* and it is desirable to have stage-order q close to (or greater than) the order of the method p.

2.2 Preliminaries

2.2.1 Consistency

The solutions at the n^{th} step of a GLM depend upon the approximations at the internal stage values and the quantities passed from step to step in general. These quantities can be solutions at the end of previous steps, or their derivatives or higher derivatives or any combination of all these. To obtain just the solutions of a problem, the minimum requirement is that each quantity in $y^{[n-1]}$ represents

$$y^{[n-1]} = uy(x_{n-1}) + O(h), \tag{2.4a}$$
$$y^{[n]} = uy(x_n) + O(h), \tag{2.4b}$$

where u is called the pre-consistency vector. To solve the trivial one-dimensional problem $y'(x) = 0$ with initial condition $y(0) = 1$, (2.2) becomes

$$Y^{[n]} = Uy^{[n-1]},$$
$$y^{[n]} = Vy^{[n-1]}.$$

And therefore the least requirements (2.4) become pre-consistent conditions and leads to the following definition[32]:

Definition 2.2.1 *A general linear method (A, U, B, V) is **preconsistent** if there exists a vector u such that*

$$Vu = u, \tag{2.5a}$$
$$Uu = \mathbf{1}, \tag{2.5b}$$

where $\mathbf{1}$ is a vector of all ones. For consistency conditions, considering the one-dimensional problem $y'(x) = 1$ with $y(0) = 0$, the output of (2.2) becomes

$$Y^{[n]} = A\mathbf{1}h + Uy^{[n-1]},$$
$$y^{[n]} = B\mathbf{1}h + Vy^{[n-1]},$$

and consistency vector v can be determined if

$$y^{[n-1]} = uy(x_{n-1}) + vhy'(x_{n-1}) + O(h^2), \tag{2.6a}$$
$$y^{[n]} = uy(x_n) + vhy'(x_n) + O(h^2). \tag{2.6b}$$

Therefore, the following definition is required [32]:

Definition 2.2.2 *A general linear method (A, U, B, V) is **consistent** if it is preconsistent with pre-consistency vector u and there exists a vector v such that*

$$B\mathbf{1} + Vv = u + v.$$

2.2.2 Stability

As the dual nature of general linear methods includes multistep behaviour, it is important to study stability. This is directly related to the coefficient matrix V; the stability of a matrix is defined as [32]:

Definition 2.2.3 *A matrix V is stable if there exists a constant k such that $\|V^n\| \leq k$, for all $n = 1, 2, \dots$.*

For a stable general linear method, the solution of $y'(x) = 0$ must be bounded. For this, the problem (2.2) becomes

$$\begin{aligned} y^{[n-1]} &= Vy^{[n-1]} = V.Vy^{[n-2]}, \\ &= V^n y^{[0]}. \end{aligned}$$

So the stability of methods depends upon the stability of V. This gives the following definition [32]:

Definition 2.2.4 *A general linear method (A, U, B, V) is **stable** if there exists a constant C such that, for all $n = 1, 2, \ldots, \|V^n\| \leq C$.*

2.2.3 Convergence

For practical computation, a necessary element which a method needs in the theory of numerical analysis is convergence. This idea mainly arises in the theory of multistep methods. As general linear methods are of a multistep nature too, it is necessary for a general linear method to be convergent. This needs the following definition [32]:

Definition 2.2.5 *A general linear method (A, U, B, V) is **convergent** if for any initial value problem (2.1) subject to the Lipschitz condition $\|f(y) - f(z)\| \leq L\|y - z\|$, there exist a nonzero vector $u \in R^r$ and a starting procedure $\phi : (0, \infty) \to R^r$, such that for all $i = 1, 2, \ldots, r$, $lim_{h\to 0}\ \phi_i(h) = u_i y(x_0)$, and such that for any $\bar{x} > x_0$, the sequence of vectors $y^{[n]}$, computed using n steps with stepsize $h = (\bar{x} - x_0)/n$ and using $y^{[0]} = \phi(h)$ in each case, converges to $uy(\bar{x})$.*

In the construction of numerical methods, it is assumed that the stage values and solutions are at least of order 1. This implies consistency and the remaining thing to be considered is stability as with this, convergence will be automatically achieved according to the following theorem [32]:

Theorem 2.2.1 *A general linear method is stable and consistent if and only if it is convergent.*

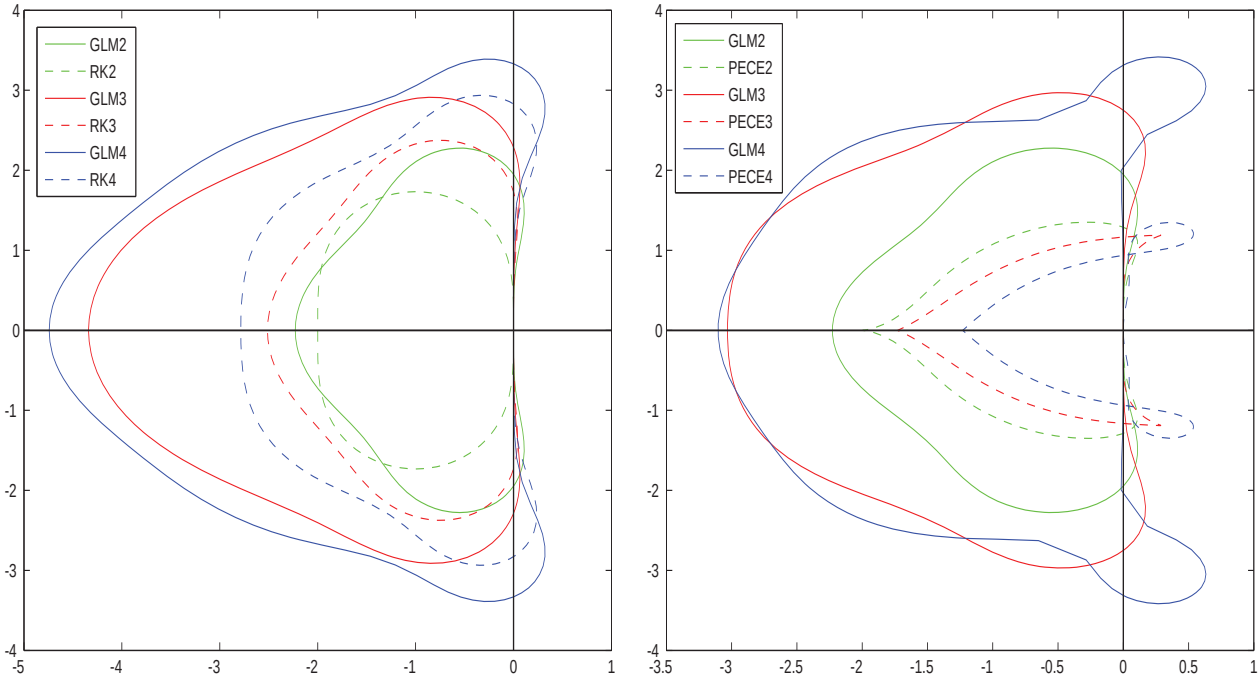
2.3 Stability matrix of general linear methods

The scalar test problem of Dahlquist [64]

$$y'(x) = qy(x), \qquad \text{where} \quad q \in C,$$

is a standard way to judge the stability of numerical methods for ODEs. The purpose of doing so is to study the performance of a single step of the method on the existing approximations. Applying this test to the general linear method (2.2a) and taking $z = hq$, it can be written as

$$\begin{aligned} Y^{[n]} &= zAY^{[n]} + Uy^{[n-1]}, \\ (I - zA)Y^{[n]} &= Uy^{[n-1]}, \end{aligned}$$



Stability regions of GLMs vs RKs

Stability regions of GLMs vs PECEs

Figure 2.1: Stability regions

and (2.2b) becomes

$$y^{[n]} = zBY^{[n]} + Vy^{[n-1]},$$
$$y^{[n]} = (V + zB(I - zA)^{-1}U)y^{[n-1]}.$$

Thus,

$$M(z) = V + zB(I - zA)^{-1}U, \tag{2.7}$$

is the stability matrix of a general linear method. The stability regions of the GLMs and their competitive methods used in this thesis are presented in Figure 2.1.

2.4 Methods

The simplest example would be to choose $p = q = r = s = 1$ which is the explicit Euler method. Any further choice of the number $pqrs$ may not be a unique method in general. Thus, these free parameters are not the best presentation of GLMs. However, in a particular class of GLMs, it can be used as a generic name. One way to classify the traditional families of numerical methods for ODEs could be

- Multistage methods known as Runge–Kutta methods; these are a very big family of GLMs with $r \leq 2$ and $s \geq 1$. A few Runge–Kutta methods in GLM form, used in the experiments, are given in Table 2.1.

- The multistep methods are GLMs with $r \geq 2$ and $s = 1$. The best competitive methods for the

IRKS methods are Adam–Bashforth, Adam–Moulton predictor–corrector methods used in the form of pairs. The general form of PECE pairs is

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{cc|ccccccc} 0 & 0 & 1 & \alpha_1^* & \alpha_2^* & \dots & \alpha_{k-1}^* & \alpha_k^* \\ \alpha_0 & 0 & 1 & \alpha_1 & \alpha_2 & \dots & \alpha_{k-1} & \alpha_k \\ \hline \alpha_0 & 0 & 0 & \alpha_1 & \alpha_2 & \dots & \alpha_{k-1} & \alpha_k \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{array}\right].$$

These methods are given in Table 2.2.

2.5 Some practical GLMs

As mentioned before, GLMs are the most general form of methods. To get practical methods, some simplifying assumptions are needed. To discuss the implementation issues, we consider those classes of GLMs which have been of interest. Most of the attention nowadays is towards getting practical methods. The approach was different in the case of the development of traditional methods. All the traditional methods are GLMs as well, though only those classes are mentioned which are derived from the most general form. There are several different aspects to explore within GLMs. A few practical classes of GLMs along with their purposes, characteristics and implementation issues are discussed below.

Diagonally implicit multistage integration methods (DIMSIMs). These methods are a big family of practical GLMs. These methods are mainly based on some assumptions that assure the stability and implementation of methods *a priori*. The set of free parameters for the construction of these methods allows us to obtain DIMSIMs of the desired features. The structure of these methods is explained in the next chapter. The methods used in this thesis are a sub-class of these methods. DIMSIMs might be implemented in either parallel or in sequential arrangements. Almost all the big challenges that are found during the implementation of the traditional methods can be overcome using these methods.

Almost Runge–Kutta (ARK) methods. ARK methods were introduced by Butcher in [30]. These are GLMs with $r = 3$ which retain many properties of the Runge–Kutta methods with some extra implementation advantages. These methods need a starting method although this is not a major issue for implementation. Initially, ARK methods were used to solve non–stiff problems though the idea was extended to solve stiff problems [57] by adding a constant diagonal element to the coefficient matrix A of the method.

G-symplectic methods. In the recent development of integrators, GLMs are also used to solve symplectic problems i.e. a problem with a property of preserving quadratic invariants. In general, GLMs are not symplectic and for this a condition given by Butcher [32], known as G–norm, is necessary. Construction of G-symplectic GLMs and parasitism control are a few recent developments by Butcher et al. [43]. However, these methods need further work to achieve the efficiency of the IRKS methods with the F–property (see further Section 3.3.6). For example, these methods still need a starting method.

p	RKs	GLMs
1	$\left[\begin{array}{c\|c} 0 & 1 \\ \hline 1 & 1 \end{array}\right]$	$\left[\begin{array}{cc\|cc} 0 & 0 & 1 & \frac{1}{3} \\ \frac{9}{8} & 0 & 1 & \frac{1}{8} \\ \hline \frac{9}{8} & 0 & 1 & \frac{1}{8} \\ 0 & 1 & 0 & 0 \end{array}\right]$
2	$\left[\begin{array}{cc\|c} 0 & 0 & 1 \\ 1 & 0 & 1 \\ \hline \frac{1}{2} & \frac{1}{2} & 1 \end{array}\right]$	$\left[\begin{array}{ccc\|cc} 0 & 0 & 0 & 1 & \frac{1}{3} & \frac{1}{18} \\ \frac{32}{81} & 0 & 0 & 1 & \frac{22}{81} & \frac{22}{243} \\ \frac{37}{96} & \frac{27}{64} & 0 & 1 & \frac{37}{192} & \frac{13}{144} \\ \hline \frac{37}{96} & \frac{27}{64} & 0 & 1 & \frac{37}{192} & \frac{13}{144} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{9}{2} & -\frac{27}{4} & 4 & 0 & -\frac{7}{4} & 0 \end{array}\right]$
3	$\left[\begin{array}{ccc\|c} 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 1 \\ -1 & 2 & 0 & 1 \\ \hline \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 1 \end{array}\right]$	$\left[\begin{array}{cccc\|ccc} 0 & 0 & 0 & 0 & 1 & \frac{1}{4} & \frac{1}{32} & \frac{1}{384} \\ -\frac{111}{560} & 0 & 0 & 0 & 1 & \frac{391}{560} & \frac{391}{2240} & \frac{1453}{53760} \\ -\frac{7877}{6720} & \frac{5}{48} & 0 & 0 & 1 & \frac{12217}{6720} & \frac{4679}{8960} & \frac{20197}{215040} \\ -\frac{1839}{112} & \frac{61}{4} & -4 & 0 & 1 & \frac{691}{112} & -\frac{9}{448} & -\frac{1091}{10752} \\ \hline -\frac{1839}{112} & \frac{61}{4} & -4 & 0 & 1 & \frac{691}{112} & -\frac{9}{448} & -\frac{1091}{10752} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{184}{3} & \frac{146}{3} & -12 & 1 & 0 & \frac{71}{3} & 0 & -\frac{7}{24} \\ 48 & -56 & 16 & 4 & 0 & -12 & 0 & 0 \end{array}\right]$
4	$\left[\begin{array}{cccc\|c} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{4} & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & 1 \\ 1 & -2 & 2 & 0 & 1 \\ \hline \frac{1}{6} & 0 & \frac{2}{3} & \frac{1}{6} & 1 \end{array}\right]$	$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -\frac{95448}{13801525} & 0 & 0 & 0 & 0 \\ -\frac{304475092179}{1945200735025} & \frac{65769}{1127528} & 0 & 0 & 0 \\ \frac{10560520570469048}{4472016489822475} & -\frac{1632164837}{648046718} & \frac{5812}{6897} & 0 & 0 \\ \frac{86257049}{537380} & -\frac{957565}{3104} & 240 & -\frac{1045}{16} & 0 \end{bmatrix}$
		$U = \begin{bmatrix} 1 & \frac{1}{5} & \frac{1}{50} & \frac{1}{750} & \frac{1}{15000} \\ 1 & \frac{5616058}{13801525} & \frac{5616058}{69007625} & \frac{11184392}{1035114375} & \frac{5568334}{5175571875} \\ 1 & \frac{108650517767827}{15561605880200} & \frac{75390830073}{401072316500} & \frac{6703960332723}{1945200735025 00} & \frac{969767428407}{19452007350250 00} \\ 1 & \frac{3070633790332267}{268320989389 34850} & \frac{7324801258911}{209560285371 25} & \frac{2680403278295063}{304910215215 16875} & \frac{175185928297383 16}{167700618368342 8125} \\ 1 & -\frac{110518377}{4299040} & \frac{4957}{110800} & -\frac{107062919}{161214000} & \frac{6929867}{134345000} \end{bmatrix}$
		$B = \begin{bmatrix} \frac{86257049}{537380} & -\frac{957565}{3104} & 240 & -\frac{1045}{16} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -\frac{2648605}{4268} & \frac{595340}{1067} & -\frac{7265}{44} & 0 & 1 \\ \frac{26795}{22} & -\frac{143425}{88} & \frac{31025}{33} & -\frac{3135}{16} & 1 \\ -730 & \frac{2545}{2} & -960 & \frac{1045}{4} & 4 \end{bmatrix}$
		$V = \begin{bmatrix} 1 & -\frac{110518377}{4299040} & \frac{4957}{110800} & -\frac{107062919}{161214000} & \frac{6929867}{134345000} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{483841}{2134} & 0 & -\frac{3889}{1940} & \frac{92294}{80025} \\ 0 & -\frac{176003}{528} & 0 & 0 & -\frac{1753}{3300} \\ 0 & \frac{609}{4} & 0 & 0 & 0 \end{bmatrix}$

Table 2.1: Classical Runge–Kutta methods and their competitive GLMs in the IRKS family with the F–property.

p	PECEs	GLMs
1	$\left[\begin{array}{cc\|cc} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right]$	same
2	$\left[\begin{array}{cc\|ccc} 0 & 0 & 1 & \frac{3}{2} & -\frac{1}{2} \\ \frac{1}{2} & 0 & 1 & \frac{1}{2} & 0 \\ \hline \frac{1}{2} & 0 & 1 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right]$	$\left[\begin{array}{ccc\|ccc} 0 & 0 & 0 & 1 & \frac{1}{3} & \frac{1}{18} \\ \frac{352}{405} & 0 & 0 & 1 & -\frac{82}{405} & -\frac{82}{1215} \\ \frac{569}{480} & \frac{27}{64} & 0 & 1 & -\frac{583}{960} & -\frac{127}{720} \\ \hline \frac{569}{480} & \frac{27}{64} & 0 & 1 & -\frac{583}{960} & -\frac{127}{720} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{9}{2} & -\frac{27}{4} & 4 & 0 & -7/4 & 0 \end{array}\right]$
3	$\left[\begin{array}{cc\|ccc} 0 & 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ \frac{5}{12} & 0 & 1 & \frac{2}{3} & -\frac{1}{12} & 0 \\ \hline \frac{5}{12} & 0 & 1 & \frac{2}{3} & -\frac{1}{12} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$	$\left[\begin{array}{cccc\|cccc} 0 & 0 & 0 & 0 & 1 & \frac{1}{4} & \frac{1}{32} & \frac{1}{384} \\ -\frac{3}{10} & 0 & 0 & 0 & 1 & \frac{4}{5} & \frac{1}{5} & \frac{29}{960} \\ -\frac{377}{240} & \frac{5}{48} & 0 & 0 & 1 & \frac{133}{60} & \frac{199}{320} & \frac{817}{7680} \\ -\frac{65}{4} & \frac{61}{4} & -4 & 0 & 1 & 6 & -\frac{1}{16} & -\frac{41}{384} \\ \hline -\frac{65}{4} & \frac{61}{4} & -4 & 0 & 1 & 6 & -\frac{1}{16} & -\frac{41}{384} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{184}{3} & \frac{146}{3} & -12 & 1 & 0 & \frac{71}{3} & 0 & -\frac{7}{24} \\ 48 & -56 & 16 & 4 & 0 & -12 & 0 & 0 \end{array}\right]$
4	$\left[\begin{array}{cc\|ccccc} 0 & 0 & 1 & \frac{55}{24} & -\frac{59}{24} & \frac{37}{24} & -\frac{3}{8} \\ \frac{3}{8} & 0 & 1 & \frac{19}{24} & -\frac{5}{24} & \frac{1}{24} & 0 \\ \hline \frac{3}{8} & 0 & 1 & \frac{19}{24} & -\frac{5}{24} & \frac{1}{24} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$	$A = \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ -\frac{1572952}{124213725} & 0 & 0 & 0 & 0 \\ -\frac{10241567672 57}{5835602205075} & \frac{65769}{1127528} & 0 & 0 & 0 \\ \frac{93319573112607352}{402481484084022 75} & -\frac{1632164837}{648046718} & \frac{5812}{6897} & 0 & 0 \\ \frac{776670401}{4836420} & -\frac{957565}{3104} & 240 & -\frac{1045}{16} & 0 \end{array}\right]$

$$U = \begin{bmatrix} 1 & \frac{1}{5} & \frac{1}{50} & \frac{1}{750} & \frac{1}{15000} \\ 1 & \frac{51258442}{124213725} & \frac{51258442}{621068625} & \frac{33910136}{3105343125} & \frac{50471966}{46580146875} \\ 1 & \frac{33481007229241}{46684817640600} & \frac{230738737259}{1203216949500} & \frac{20333343979609}{583560220507500} & \frac{87740832818543}{175068066152250 00} \\ 1 & \frac{126621254142249 61}{804962968168045 50} & \frac{67539998604439}{188604256834125} & \frac{738147637971439}{831573314223187 5} & \frac{158529891478451884}{150930556531508 53125} \\ 1 & -\frac{997521073}{38691360} & \frac{29893}{997200} & -\frac{321902677}{483642000} & \frac{186749449}{3627315000} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{776670401}{4836420} & -\frac{957565}{3104} & 240 & -\frac{1045}{16} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -\frac{2648605}{4268} & \frac{595340}{1067} & -\frac{7265}{44} & 0 & 1 \\ \frac{26795}{22} & -\frac{143425}{88} & \frac{31025}{33} & -\frac{3135}{16} & 1 \\ -730 & \frac{2545}{2} & -960 & \frac{1045}{4} & 4 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & -\frac{997521073}{38691360} & \frac{29893}{997200} & -\frac{321902677}{483642000} & \frac{186749449}{3627315000} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{483841}{2134} & 0 & -\frac{3889}{1940} & \frac{92294}{80025} \\ 0 & -\frac{176003}{528} & 0 & 0 & -\frac{1753}{3300} \\ 0 & \frac{609}{4} & 0 & 0 & 0 \end{bmatrix}$$

Table 2.2: PECE pairs and their competitive GLMs in the IRKS family with the F–property.

3
Construction of efficient GLMs

3.1 Introduction

The most general form of numerical methods for ODEs, known as general linear methods, were first introduced by J. C. Butcher, almost 50 years ago [15]. There is no straightforward algorithm yet to construct methods of the form of systems of (2.2) or eqs. (3.3) and (3.4). The reasons include complicated order conditions, though low order methods can be constructed with certain conditions. Once, such a method is constructed, the next task is to deal with the implementation issues. The implementation issues of the existing GLMs addressed in the past have scope for further improvement to be more robust and efficient.

On the other hand, the two main classes of numerical methods for ODEs, namely multistep methods and multistage methods, have been the centre of interest for computational scientists; the available ODE solvers are also based on these classes. Many generalisations of these classes have been explored to enhance the theory and practice. Each of these well–studied classes have their own pros and cons. Thus it can be said that enhancement in the field of numerical methods for ODEs is equivalent to achieving the same goal by one of two different trajectories. One way is to start with the classical methods, overcome their difficulties and generalise or improve the results. The other way is to efficiently screen out the advantages and disadvantages of the two families of classical methods and gather these advantages and at the same time, the strategies of overcoming the known disadvantages, into one frame. Then we need to look for some practical methods in the general class to explore such GLMs that fit into that frame. One such frame (with practical methods) has been considered by J. C. Butcher [28]. This big family of implementable GLMs is known as *diagonally implicit multi-stage integration methods* (DIMSIMs).

3.2 DIMSIMs

These methods were introduced by Butcher [28] as a unified approach of the practical methods within the two traditional classes of numerical methods for ODEs. DIMSIMs are one of the biggest families of GLMs that are efficiently implementable. Why is the unified approach possible and how can the methods be unified? This will be discussed in the rest of this section.

3.2.1 Motivation for DIMSIMs

Before the discovery of these methods, much research was directed towards obtaining highly accurate, (highly) stable and cost efficient methods. These qualities are achievable in the two traditional classes (multistep methods and Runge–Kutta methods) but the challenge is to obtain all of these on one platform. Here are a few of the challenges with multistep methods.

- Practical multistep methods are limited by Dahlquist's first barrier [63] which is the relation between the number of steps and the order of the method.
- The A-stable multistep methods are restricted by Dahlquist's second barrier [64] which states that A-stable methods cannot be of order greater than one.
- It is not possible to implement multistep methods in variable stepsize mode without additional cost.

It is possible to overcome the Dahlquist first barrier by cyclic composition of multistep methods [65] and the further second barrier can be overcome by the implicit case of these methods [7].

On the other hand, multistage methods have their own pros and cons. A few challenges for the Runge-Kutta methods are as follows:

- It is not possible to construct an explicit Runge-Kutta method of order $p > 4$ with the number of stages equal to the order of the method. This makes high–order explicit Runge-Kutta methods expensive in terms of cost.
- For high–order explicit Runge–Kutta methods, error estimation is not possible.
- For implicit methods, high–order methods with high stage order are needed. The collocation methods are fully implicit methods including Gauss, Radau and Lobatto families. These are highly stable and acceptable in terms of order, though reduction of order causes their failure. Furthermore, none of these methods have been implemented efficiently e.g. these methods lack reliable error estimation techniques.
- Other methods with high stage order are known as SIRK methods. But these methods have some disadvantages as the abscissae are proportional to the zeros of the Laguerre polynomials outside the interval $[0, 1]$ (or beyond the step). This was overcome in [62, 42] by the methods known as ESIRK or DESIRE methods.

- There exist cost efficient methods known as diagonally implicit Runge–Kutta (DIRK) methods. Though it is hard to find methods of this type with order greater than 4. They suffer from order reduction due to low stage order.

There exist methods in the literature that originate as multistage methods but are of multivalue nature as well. For example, consider the following Runge–Kutta method of order 4 with 5 stages in the form of the Butcher tableau

$$\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{3} & \frac{1}{3} & & & & \\
\frac{2}{3} & -\frac{1}{3} & 1 & & & \\
1 & 1 & -1 & 1 & & \\
1 & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} & 0 \\
\hline
b & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} & 0 \\
\hline
\widehat{b} & 0 & \frac{3}{4} & 0 & -\frac{1}{4} & \frac{1}{2}
\end{array}$$

which is

$$\begin{aligned}
Y_1 &= y_{n-1}, \\
Y_2 &= y_{n-1} + \frac{1}{3}hf(Y_1), \\
Y_3 &= y_{n-1} - \frac{1}{3}hf(Y_1) + hf(Y_2), \\
Y_4 &= y_{n-1} + hf(Y_1) - hf(Y_2) + hf(Y_3), \\
Y_5 &= y_{n-1} + \frac{1}{8}hf(Y_1) + \frac{3}{8}hf(Y_2) + \frac{3}{8}hf(Y_3) + \frac{1}{8}hf(Y_4), \\
y_n &= y_{n-1} + \frac{1}{8}hf(Y_1) + \frac{3}{8}hf(Y_2) + \frac{3}{8}hf(Y_3) + \frac{1}{8}hf(Y_4), \\
\widehat{y_n} &= y_{n-1} \qquad\qquad + \frac{3}{4}hf(Y_2) \qquad\qquad - \frac{1}{4}hf(Y_4) + \frac{1}{2}hf(Y_5).
\end{aligned} \tag{3.1}$$

When this method is implemented, the input to step number n would be simply y_{n-1}. As $c_1 = 0$ and $c_4 = 1$, the first stage derivative is exactly the same as the last stage derivative of the previous step. This is known as the FSAL property (*first same as last*) of Runge–Kutta methods. So it would be efficient to use that value, calculated in the $(n-1)^{\text{th}}$ step, again during the calculation of the n^{th} step, instead of calculating it again. And thus the input to the n^{th} step would be a vector of two quantities $[y_{n-1}, hf(Y_5)]$. This means that from an implementation point of view, this 5-stage single value method is actually a 4-stage 2-values method and its GLM form is

$$c^{\mathsf{T}} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & 1 & 1 \end{bmatrix}$$

$$\left[\begin{array}{cccc|cc} 0 & 0 & 0 & 0 & 1 & \frac{1}{3} \\ 1 & 0 & 0 & 0 & 1 & -\frac{1}{3} \\ -1 & 1 & 0 & 0 & 1 & 1 \\ \frac{3}{8} & \frac{3}{8} & \frac{1}{8} & 0 & 1 & \frac{1}{8} \\ \hline \frac{3}{8} & \frac{3}{8} & \frac{1}{8} & 0 & 1 & \frac{1}{8} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right],$$

with error estimator of order 3 given by

$$\|y_n - \widehat{y_n}\| = -\frac{3}{8}hf(Y_1) + \frac{3}{8}hf(Y_2) + \frac{3}{8}hf(Y_3) - \frac{1}{2}hf(Y_4) + \frac{1}{8}y_2^{[n-1]}.$$

Such kinds of natural generalisation motivates the exploration of such multistage-multivalue methods that are analogous to the existing multistage-singlevalue methods and one-stage-multivalue methods with rich theory in literature and practice.

Thus, in each of the multistage and multivalue methods, there are limits upon how well we can address all of the above issues at the same time. In other words, the disadvantages of one traditional class are advantages of the other and vice versa. This motivates us to look for a class of methods that is above the traditional classes. Such a class should have multistage and multivalue behaviour. In that class the available knowledge of the traditional classes could be implementable. All this was made possible by Butcher [28], theoretically and practically, in the most general form of numerical methods for ODEs.

3.2.2 Formulation of DIMSIMs

As GLMs in general are very complex and difficult to construct, we impose some assumptions to achieve a reasonably simple form of order conditions. Conditions are applied to the coefficient matrices (A, B, U, V) to obtain the desired results. The design of DIMSIMs proposed in [28] can be summarised as follows.

- The four quantities *pqrs* are nearly equal. A fifth quantity t could be considered here presenting the type of DIMSIMs. These types are discussed further in this section.

- To guarantee the stability, the spectrum of the matrix V should be $\{0, \delta_i\}$ where δ_i lies in an open unit disk.

- High stage order is required, at least close to the order of the method.

- The very first condition is on the matrix A; the matrix of the coefficients of the stage derivatives. The complexities (nonlinearity) of the internal stage-derivatives system occurs in the implementation of the Runge–Kutta methods with full matrix A. To avoid this in DIMSIMs, it would be necessary to have the matrix A of diagonal structure with zero entries at least in the upper triangular part and constant diagonal element $\lambda \geq 0$ which is the eigenvalue of A (DIRK structure

[2]). Thus the general structure of matrix A for DIMSIMs is

$$\begin{bmatrix} \lambda & 0 & 0 & \dots & 0 \\ a_{21} & \lambda & 0 & \dots & 0 \\ a_{31} & a_{32} & \lambda & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & a_{s3} & \dots & \lambda \end{bmatrix} \tag{3.2}$$

Further, DIMSIMs are divided into 4 types depending upon the structure of the matrix A. This division is mainly to cover two targets:

First. This division is due to a characteristic of ODEs which differentiates between stiff and non–stiff problems, as given in Subsection 1.1.1. The methods are constructed for both kind of problems, separately.

Second. The second target is related to large scale computing techniques. The existing theory on SIRK methods concludes that the efficient implementation of multistage methods could be either by sequential or parallel programming. So both cases will be considered here.

Thus to cover these two goals, there will be 4 possible acceptable combinations and hence 4 types of the structure of matrix A which are as follows:

Type1. For solving non-stiff problems, explicit methods are constructed by taking $\lambda = 0$ in (3.2). This will produce an explicit type of DIMSIMs for solving non-stiff problems by sequential computing of the internal stage systems.

Type2. Take matrix A as it is with $\lambda \neq 0$ in (3.2), for solving stiff problems via a sequential process of computing the internal stage system.

Type3. For parallel computing, in the case of non-stiff problems, it would be best to consider all the lower triangular entries of the matrix in (3.2) equal to zero along with $\lambda = 0$ i.e. $A = O$.

Type4. Take the matrix A to be strictly diagonal with constant diagonal element λ on the main diagonal. With this form of matrix A, the method will remain implicit, necessary for solving stiff problems, and the internal stage system can be solved using parallel computing.

The DIMSIMs considered in the rest of the chapter will be of type 1. A more specific design is briefly mentioned in Section 3.3. One of the primary aims of considering such a design of GLMs is to simplify the order conditions which, in general, are very complex to handle. To make this easier, the matrix form of order conditions are presented in Subsection 3.3.2. To deal with such a matrix form and design, doubly companion matrices were introduced by Butcher and Chartier in [39]. This matrix is discussed in Subsection 3.3.3; it helps to connect the eigenvalue problem of the matrices A and V. Due to the condition of IRKS, there are conditions on the eigenvalues of the matrix V; this is discussed in

Subsection 3.3.4. Along with IRKS, we impose another property in the design that efficiently reduces the cost; this property is called property–F for GLMs [34] and is discussed in Subsection 3.3.6. Also the use of doubly companion matrices enables us to transform the matrices (A, U, B, V). Why we need this transformation and how it works, is discussed in Subsection 3.4.2.

3.3 Construction of type 1 DIMSIMs

When applying traditional methods, two sorts of information are used to approximate the solution of an IVP from point x_{n-1} to x_n. One sort is a linear combination of function evaluations of F at some points within the step i.e. $[x_{n-1}, x_n]$; these points are called stages and the methods are called multistage methods or Runge–Kutta methods. The other sort of information is based on the solutions at some previous steps for multistep methods or a combination of values at those points which is in the form of a vector which is passed from step to step; these are known as multivalue methods.

If r is the number of elements in the input/output vector passed from step to step and s is the number of stages, then the most general form of method is

$$Y_i^{[n]} = h\sum_{j=1}^{s} a_{ij}F_j + \sum_{j=1}^{r} u_{ij}y_j^{[n-1]}, \quad i = 1, 2, ..., s, \tag{3.3}$$

$$y_i^{[n]} = h\sum_{j=1}^{s} b_{ij}F_j + \sum_{j=1}^{r} v_{ij}y_j^{[n-1]}, \quad i = 1, 2, ..., r, \tag{3.4}$$

where $F_j = F(Y^{[n]})_j = f(Y_j^{[n]})$. The compact form of eqs. (3.3) and (3.4) is

$$\begin{bmatrix} Y^{[n]} \\ y^{[n]} \end{bmatrix} = \begin{bmatrix} A & U \\ B & V \end{bmatrix} \begin{bmatrix} hF(Y^{[n]}) \\ y^{[n-1]} \end{bmatrix}.$$

In matrix notation a GLM is

$$M = \begin{bmatrix} A & U \\ B & V \end{bmatrix}.$$

This notation was introduced by Butcher and Burrage in 1980 [11]. Along with this, another important notation in the literature is in the form of a quadruple $(pqrs)$. In this quadruple, p represents the order of the method, q represents stage order of the method, r represents the number of elements in the Nordsieck vector, and s represents the number of stages.

Construction of a GLM will be complete if it is possible to determine coefficients of the matrices (A, U, B, V) satisfying the order conditions. The search for coefficients depends upon the design of the method and that design should be practical and efficient. One such frame of design is considered here.

3.3.1 Design of GLMs with the IRKS property

In order to obtain practical GLMs, some assumptions are required to be imposed on the general form that ease the construction of methods. The advantages which we look for in this general class must

not be less than what we achieved using classical methods. Along with this, the disadvantages of the classical methods should be overcome here. Keeping these two goals in mind, the design, which consists of the following assumptions on GLMs, will be used in the rest of the work.

1. In GLMs, the data vector passed from step to step is either a combination of approximate solutions $y(x)$ calculated in the previous steps or the values other than that. It can be a combination of both as well. The vector considered here, in a method of order p, consists of p scaled derivatives of the solution known as the Nordsieck vector. This was introduced by Nordsieck [80] for multivalue methods and is given by

$$y^{[n-1]} = \begin{bmatrix} y(x_{n-1}) \\ hy'(x_{n-1}) \\ \vdots \\ h^p y^{(p)}(x_{n-1}) \end{bmatrix} + O(h^{p+1}).$$

   This gives additional information relating the length of Nordsieck vector r and the order p, i.e. $r = p + 1$. Not all the multivalue methods can be used in variable stepsize mode. This special Nordsieck form has the advantage of efficiently adjusting the step size. This is further discussed in the section on the *scale and modify* technique (see Section 4.3.1) which shows how efficiently the vector $D(\mathsf{r})y^{[n]}$ can be used as an input vector to step $(n + 1)$ with stepsize $\mathsf{r}h$ where the diagonal matrix $D(\mathsf{r}) = (1, \mathsf{r}, \mathsf{r}^2, \ldots, \mathsf{r}^p)$.

2. In the unifying design of existing practical GLMs, there are two choices for stage order q, either p or $p - 1$. We select $q = p$ to obtain high stage order. There are two advantages for efficient implementation of the methods. This allows us to use the last abscissa $c_s = 1$ which is cost efficient, because in that case the last stage-derivative value, in practice, can be used as the second component of the data vector. This further guarantees the use of property–F given in Section 3.3.6. Another advantage is due to the error estimators used here. These are based on stage derivatives and higher stage order assures the reliability of the estimator.

3. The number of stages and the length of Nordsieck vector are equal $(s = r)$. The advantage of this relation is that it helps to construct a suitable error estimator. The error term in a method of order p consists of the $(p+1)^{\text{st}}$ derivative of y and this maximum value of r equal to s enables us to use $(p+1)$ stage-derivatives in the existing step, to estimate the error using the estimators in Section 4.5.

4. The matrix A in GLMs represents the coefficients of the stage derivatives in the method. Here, A is lower triangular with constant diagonal element in the case of implicit methods whereas in the case of explicit methods, as we use here, it is strictly lower triangular. Thus, we are considering the DIMSIMs of type 1. The full matrix A produces implicit methods that are very costly and need additional algorithms to solve the nonlinear system of internal stage derivatives.

5. The stability matrix of a GLM is

$$M(z) = V + zB(I - zA)^{-1}U.$$

To assure the stability, *a priori* to construction, the eigenvalues of $M(z)$ are assigned to be all zero except one which is $R(z)$ where

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

This is similar to the stability of well known Runge–Kutta methods with the structure of matrix A used here. This condition is easy to impose during construction of methods.

6. The method has the F– property given in Subsection 3.3.6. There is again a practical advantage of using this property. This enables us to choose $c_s = 1$ as it produces the same coefficients for the last stage and the first Nordsieck element. Thus the advantage occurs during the implementation by taking the last stage value as the first component of the Nordsieck vector. Similarly the second component in the Nordsieck vector is an approximation to $hy'(x)$ which is equivalent to $hf(y)$. Instead of approximating this quantity, the last scaled stage derivative can be used as the second component in the Nordsieck vector. As this component is obtained without using any coefficients of the method, so the second row of matrix B must be all zero except the last element b_{2s} which must be exactly 1 in this case.

3.3.2 Stage and order conditions

The very first requirement for construction of GLMs is to find the order conditions. Here, we are working on DIMSIMs of type 1 and type 2 and the stage order must be equal to the order of the method. Thus the stages Y and the output vector $y^{[n]}$ must follow,

$$Y = Cy^{[n-1]} + O(h^{p+1}),$$
$$y^{[n]} = Ey^{[n-1]} + O(h^{p+1}),$$

where

$$C = \begin{bmatrix} 1 & c_1 & \frac{c_1^2}{2!} & \cdots & \frac{c_1^{(p-1)}}{(p-1)!} & \frac{c_1^p}{p!} \\ 1 & c_2 & \frac{c_2^2}{2!} & \cdots & \frac{c_2^{(p-1)}}{(p-1)!} & \frac{c_2^p}{p!} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & c_p & \frac{c_p^2}{2!} & \cdots & \frac{c_p^{(p-1)}}{(p-1)!} & \frac{c_p^p}{p!} \\ 1 & c_{p+1} & \frac{c_{p+1}^2}{2!} & \cdots & \frac{c_{p+1}^{(p-1)}}{(p-1)!} & \frac{c_{p+1}^p}{p!} \end{bmatrix}, \quad E = \exp(K) = \begin{bmatrix} 1 & \frac{1}{1!} & \frac{1}{2!} & \cdots & \frac{1}{(p-1)!} & \frac{1}{p!} \\ 0 & 1 & \frac{1}{1!} & \cdots & \frac{1}{(p-2)!} & \frac{1}{(p-1)!} \\ 0 & 0 & 1 & \cdots & \frac{1}{(p-3)!} & \frac{1}{(p-2)!} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & & \frac{1}{1!} & \frac{1}{2!} \\ 0 & 0 & 0 & \cdots & 1 & \frac{1}{1!} \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix},$$

$$J = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix} = K^{\mathsf{T}}.$$

Using these conditions in the method (2.2) and using functions of complex variables, the following very useful result [26] is obtained before imposing any other design criteria.

Theorem 3.3.1 *A GLM in Nordsieck form has order and stage order p if and only if*

$$\exp(cz) = zA\exp(cz) + UZ + O(z^{p+1}), \tag{3.5}$$

$$\exp(z)Z = zB\exp(cz) + VZ + O(z^{p+1}). \tag{3.6}$$

where the $\exp(cz)$ *in* (3.5)*, is the vector with components given by* $\exp(c_i z)$*,* $i = 1, 2, \ldots, s$ *and*

$$Z = \begin{bmatrix} 1 & z & z^2 & \cdots & z^{p-1} & z^p \end{bmatrix}^{\mathsf{T}}.$$

It can be easily derived from the above results that

$$\left.\begin{aligned} U &= C - ACK \\ V &= E - BCK \end{aligned}\right\} \tag{3.7}$$

These relations (3.7) between matrices (A, U, B, V) are linear and restrict the search to two matrices to construct a method. Here, matrices A and B are constructed first, using the requirements mentioned in the design. Then it is easy to obtain matrices U and V from (3.7). For example, taking an explicit case, for which $\lambda = 0$, with $p = 1$ then $s = p+1 = 2$ and hence $\mathbf{c}^{\mathsf{T}} = [c, 1]$. Using the design mentioned in the previous section, matrix A is

$$\begin{bmatrix} 0 & 0 \\ a & 0 \end{bmatrix},$$

and using eqs. (3.23) and (3.24) in property-F, matrix B would be

$$B = \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix},$$

where

$$C = \begin{bmatrix} 1 & c \\ 1 & 1 \end{bmatrix}, \qquad K = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad E = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Thus using (3.7),

$$U = C - ACK = \begin{bmatrix} 1 & c \\ 1 & 1-a \end{bmatrix},$$
$$V = E - BCK = \begin{bmatrix} 1 & 1-a \\ 0 & 0 \end{bmatrix}.$$

Therefore the method is

$$M = \left[\begin{array}{cc|cc} 0 & 0 & 1 & c \\ a & 0 & 1 & 1-a \\ \hline a & 0 & 1 & 1-a \\ 0 & 1 & 0 & 0 \end{array}\right].$$

Up to here, we have used all the conditions except the sixth one which is related to an *a priori* stability condition. So the above method would be a DIMSIM of type 1 if a and c are properly chosen. With larger p, the situation would be more complex. So further (linear) relations between the coefficients are required. To guarantee the stability we need further special matrices to construct such a method, as discussed in the following section.

3.3.3 Doubly companion matrices

Doubly companion matrices were first used in the construction of some extensions of classical methods, known as ESIRK methods, by J. C. Butcher and P. Chartier [38, 39]. A (singly) companion matrix is an analogous form for one polynomial. A doubly companion matrix is made using two polynomials.

Definition 3.3.1 *A doubly companion matrix $X(\alpha, \beta)$ has the general form,*

$$X = \begin{bmatrix} -\alpha_1 & -\alpha_2 & -\alpha_3 & \cdots & -\alpha_p & -\alpha_{p+1} - \beta_{p+1} \\ 1 & 0 & 0 & \cdots & 0 & -\beta_p \\ 0 & 1 & 0 & \cdots & 0 & -\beta_{p-1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -\beta_3 \\ 0 & 0 & 0 & \cdots & 0 & -\beta_2 \\ 0 & 0 & 0 & \cdots & 1 & -\beta_1 \end{bmatrix}. \tag{3.8}$$

The two polynomials associated with the matrix X are

$$\alpha(\omega) = 1 + \alpha_1\omega + \alpha_2\omega^2 + \cdots + \alpha_{p+1}\omega^{p+1}, \tag{3.9}$$
$$\beta(\omega) = 1 + \beta_1\omega + \beta_2\omega^2 + \cdots + \beta_{p+1}\omega^{p+1}. \tag{3.10}$$

If all $\alpha_i = 0$ or alternatively all $\beta_i = 0$ then this matrix is simply a companion matrix. Doubly companion matrices are very useful for construction of methods and one important reason is the following theorem [32].

Lemma 3.3.1 *The coefficients in the characteristic polynomial of a doubly companion matrix X, $P(\omega) = \det(\omega I - X) = \omega^{p+1} + \gamma_1\omega^p + \gamma_2\omega^{p-1} + \cdots + \gamma_{p+1}$ are given by*

$$1 + \gamma_1 z + \gamma_2 z^2 + \cdots + \gamma_{p+1} z^{p+1} = \det(I - zX) = \alpha(z)\beta(z) + O(z^{p+2}). \tag{3.11}$$

The zeros of $P(\omega)$ are the eigenvalues of X and these are free parameters in the construction of methods. We select values borrowed from the stability of SIRK methods, as discussed earlier. It is the requirement of the design of the GLMs to construct matrix A with a one point spectrum $\{\lambda\}$. The matrix X would be easier to achieve if X has the same spectrum. Thus from the design, (from def. of IRKS) $P(\omega) = (\omega - \lambda)^{p+1}$. As the β's are also free parameters, the α's can be calculated using (3.11) i.e. $\alpha(\omega) = (\omega - \lambda)^{p+1}\beta^{-1}(\omega)$. This is possible if a similarity transformation can be applied to the matrix A. In this case it is possible to decompose X into Jordan canonical form using generalised eigenvectors. The relation is given in the following theorem given in [39].

Lemma 3.3.2 *Given the column eigenvector $\phi(\lambda)$ and row eigenvector $\chi(\lambda)^{\mathsf{T}}$, the doubly companion matrix X has characteristic polynomial*

$$P(\omega) = (\omega - \lambda)^{p+1}$$

if X can be decomposed as follows

$$X = \Psi(J + \lambda I)\Psi^{-1} \tag{3.12}$$

where Ψ and Ψ^{-1} are the unit upper triangular matrices

$$\Psi = \begin{bmatrix} \frac{1}{p!}\phi^{(p)}(\lambda) & \frac{1}{(p-1)!}\phi^{(p-1)}(\lambda) & \cdots & \phi'(\lambda) & \phi(\lambda) \end{bmatrix} = \beta(K)\exp(\lambda K^-),$$

$$\Psi^{-1} = \begin{bmatrix} \chi(\lambda)^{\mathsf{T}} \\ \chi'(\lambda)^{\mathsf{T}} \\ \vdots \\ \frac{1}{(p-1)!}\chi^{(p-1)}(\lambda)^{\mathsf{T}} \\ \frac{1}{p!}\chi^{(p)}(\lambda)^{\mathsf{T}} \end{bmatrix} = \exp(-\lambda K^-)\beta(K)^{-1},$$

where

$$K^- = \begin{bmatrix} 0 & p & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & p-1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 2 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

Thus doubly companion matrices have a strong connection with the matrix A. It is necessary to mention here that the transformation described above, is to achieve the structure of A. Another trans-

formation for achieving a particular spectrum of a sub-matrix of V is discussed in Section 3.3.5. First it is convenient to see how doubly companion matrices can be used to obtain IRKS along with other design features.

3.3.4 IRKS property for GLMs

The name of this property was first suggested by W. Wright [86] as IRKS property, *inherent Runge–Kutta stability*. The conditions of IRKS, in fact, formulate relations between matrices of the GLMs to assure that the stability of a method is identical to the stability of a known Runge-Kutta method. Here, stability of SDIRK methods is considered for this purpose, *a priori* to the construction. So, we use the existing knowledge of classical methods in this investigation. In the case of GLMs, we have a stability matrix rather than stability function as in the Runge–Kutta methods. Also the eigenvalues of the stability matrix play the role of stability functions. To obtain GLMs that inherit the stability of the Runge–Kutta methods, all the eigenvalues of $M(z) = V + zB(I - zA)^{-1}U$ must be zero except one. Thus, the goal is to achieve such GLMs that follow the theorem given below [86].

Theorem 3.3.2 *If a method has IRKS then its stability matrix satisfies*

$$\rho(V + zB(I - zA)^{-1}U) = \{R(z), 0\},$$

where $R(z)$ is the stability of known Runge–Kutta methods.

To achieve this, much work has been done by Butcher and Wright [59, 60] and reported in the thesis by Wright as well [86]. Here, all this is summarised and we will start from the order conditions. Multiplying (3.5) by zB, and (3.6) by $(I - zX)$, and then adding we get

$$\begin{aligned} zB\exp(cz) + \exp(cz)(I - zX)Z &= z^2BA\exp(cz) + zBUZ \\ &\quad + z(I - zX)B\exp(cz) + (I - zX)VZ + O(z^{p+1}), \\ \Rightarrow \exp(cz)(I - zX)Z &= z^2(BA - XB)\exp(cz) \\ &\quad + z(BU + VX - XV)Z + V(I - zX)Z + O(z^{p+1}), \\ \Rightarrow (\exp(cz)I - V)(I - zX)Z &= z^2(BA - XB)\exp(cz) + z(BU + VX - XV)Z + O(z^{p+1}). \end{aligned} \tag{3.13}$$

This relation gives a lot of information. First the left hand side contains the matrix V which is, in turn, related to M by $M(0) = V$. So, to impose stability conditions on the method, the matrix V will be considered. This can be achieved if the spectrum of V lies inside a unit disc or more specifically $\sigma(V) = \{1, 0\}$ for DIMSIMs of types 1 and 2. Thus to achieve this, it is necessary to consider the matrices on the right hand side of (3.13) as well. In addition, all the elements of the matrices $BA - XB$ and $BU - XV + VX$ must be zero except their first rows. For this, the following definition is necessary for the basis of IRKS.

Definition 3.3.2 *A GLM satisfying $Ve_1 = e_1$ has IRKS if*

$$BA \equiv XB, \tag{3.14}$$

$$BU = XV - VX + e_1\xi^{\mathsf{T}}, \tag{3.15}$$

for some matrix X and s–dimensional row vector ξ^{T} and

$$\det(\omega I - V) = \omega^p(\omega - 1)$$

where $e_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathsf{T}}$.

Here, $\equiv$ sign means that the corresponding elements of the matrices on both sides are equal except their first rows. Further, it was proved [86] that X has a special form related to polynomial α and β, which is given in the following theorem.

Theorem 3.3.3 *Given a GLM in Nordsieck form with $q = p$, then the most general matrix X satisfying*

$$BA \equiv XB,$$
$$BU \equiv XV - VX,$$

is of doubly companion form (3.8).

Without loss of generality, we assume that $\beta_{p+1} = 0$. The first row of $BA - XB$ is not nonzero as stated in the following result [86]:

Lemma 3.3.3 *Given the coefficients β_1, β_2,...,β_p, the coefficients α_1, α_2,..., α_{p+1}, are chosen so that the characteristic polynomial of X is of the form $det(\omega I - X) = (\omega - \lambda)^{p+1}$. Then $BA \equiv XB$ implies*

$$BA = XB. \tag{3.16}$$

Now, which matrix should be constructed first or what procedure should be used to obtain linear conditions? It is necessary to mention again that the aim is to first build matrices A and B as (3.7) provides a straightforward way of obtaining the other two matrices U and V. Also from (3.16), $A = B^{-1}XB$, so A is also easy to obtain, thus the only matrix required is B. It is convenient to start from the first IRKS condition (3.16) and using the transformation (3.12), we get

$$\Psi^{-1}BA = (J + \lambda I)\Psi^{-1}B, \tag{3.17}$$

Let $B = \Psi\widetilde{B}$, thus (3.17) becomes,

$$\widetilde{B}A = (J + \lambda I)\widetilde{B}, \tag{3.18}$$

where $\widetilde{B}$ is lower triangular as $(J+\lambda I)$ and A are lower triangular matrices in (3.17). Further, it would be convenient to obtain the IRKS condition in terms of matrix B only. So, using (3.16) and (3.7) in (3.15), it is easy to obtain

$$BC(I - KX) = XE - EX + e_1\xi^{\mathsf{T}}, \tag{3.19}$$

and it is straightforward to observe that all the entries of the resultant matrices on both sides are zero except for their last columns. Thus, p conditions are obtained from the IRKS conditions. A further condition on the matrix $\ddot{V}$ obtained by removing the first two rows and the first two columns of V is due to the definition of IRKS. The reason for cancelling the two rows and columns is explained in the later Section 3.3.6.

3.3.5 Condition for spectral radius of $\ddot{V}$

As explained before, a GLM would be zero-stable if all the eigenvalues of $\ddot{V}$ are zero. For this purpose, the Schur transform is used in the construction of methods i.e.

$$T^{-1}\ddot{V}T = Q,$$

where Q is an upper triangular matrix and T is a unitary matrix. Now, T can be decomposed as $PL\breve{U}$ ($\breve{U}$ being different from the matrix U of the GLM) so

$$\begin{aligned}(PL\breve{U})^{-1}\ddot{V}PL\breve{U} &= Q,\\ \breve{U}^{-1}L^{-1}P^{-1}\ddot{V}PL\breve{U} &= Q.\end{aligned} \tag{3.20}$$

As the right hand side matrix Q is upper triangular and the matrix T transforms matrix $\ddot{V}$ into upper triangular form, we have to deal with the lower triangular part of (3.20) so the $\breve{U}$ factor has no effect on the structure. Therefore we can consider $T = PL$ without loss of generality. So, (3.20) becomes

$$L^{-1}P^{-1}\ddot{V}PL = Q. \tag{3.21}$$

Hence, during practical construction of methods, T should be a permuted lower triangular matrix PL (L being lower triangular with 1's on the diagonal). For example, if $p = 4$ then possible choices for T can be any one of the following

$$\begin{bmatrix} 1 & 0 & 0\\ * & 1 & 0\\ * & * & 1 \end{bmatrix}, \begin{bmatrix} * & 1 & 0\\ 1 & 0 & 0\\ * & * & 1 \end{bmatrix}, \begin{bmatrix} * & * & 1\\ 1 & 0 & 0\\ * & 1 & 0 \end{bmatrix}, \begin{bmatrix} * & 1 & 0\\ * & * & 1\\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} * & * & 1\\ * & 1 & 0\\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0\\ * & * & 1\\ * & 1 & 0 \end{bmatrix}.$$

where $*$ is a nonzero element in general. When $*$ are zero, $T = I$ which is also a possible choice.

3.3.6 Property-F for GLMs

In the case of Runge–Kutta methods, the FSAL property is very useful from an implementation point of view. It can be extended in the case of GLMs as

Definition 3.3.3 *An IRKS method has property–F if*

$$c_s = 1, \tag{3.22}$$
$$e_1^\intercal B = e_{p+1}^\intercal A, \tag{3.23}$$
$$e_2^\intercal B = e_{p+1}^\intercal. \tag{3.24}$$

where e_i is the unit column vector with all elements zero except its i^{th} element which is equal to one.

The conclusion of this definition gives $\beta_p = 0$. Further, use of (3.23) in (3.7) yields,

$$e_{p+1}^\intercal U = e_{p+1}^\intercal C - e_{p+1}^\intercal ACK,$$
$$e_1^\intercal V = e_1^\intercal E - e_1^\intercal BCK.$$

As, $c_s = 1$ so $e_{p+1}^\intercal C = e_1^\intercal E$ and thus

$$e_{p+1}^\intercal U = e_1^\intercal V.$$

Similarly, (3.24) yields $e_2^\intercal V = O$. Now due to this, the second diagonal element of V is automatically zero. Along with this, from the definition of IRKS, $Ve_1 = e_1$. Thus, the first two eigenvalues are somehow defined by the design and it is required that all the other eigenvalues must be zero. For this, it would be enough to apply the transformation in Section 3.3.5 to the matrix $\ddot{V}$ only, obtained by cancelling the first two rows and columns of V.

3.4 Practical derivation of methods

The ultimate aim of developing the theory of DIMSIMs and then its subclass, the GLMs with IRKS, is to practically produce the implementable methods. The conditions discussed so far are implementable and exactly fit into the desired frame. However, a specific procedure is needed to completely find a method. The first step in this procedure is to select the values of some free parameters. A few of these could be restricted by the design of methods and the rest might be still *free* to choose. But, what should be chosen in the existing case? This will be discussed in the following subsection.

3.4.1 Design choices

The following is the list of free parameters:

- p–the order of the method
- λ–the eigenvalue of the matrix A, to achieve A–stability
- ϵ–the error constant of the method, to obtain small error constants (in the case of explicit methods) and L–stability (in the case of implicit methods)
- c–the abscissae vector such that each abscissa lies in the interval $[0, 1]$
- β–the last column of doubly companion matrix X

- T–the $(p-1) \times (p-1)$ matrix

Order p, eigenvalue λ and error constants ϵ. Depending upon the accuracy requirements, the value of p is the very first choice to construct an IRKS method. This is important as the number of stages $s = p+1$ and length of Nordsieck vector $r = p+1$ are then defined in terms of it. Next is to choose the stability and accuracy parameters, λ and ϵ. The analysis of the λ values is available in the literature for very highly stable methods which are usually required for stiff problems. The error constant ϵ is chosen here for a fair comparison with competitive methods. Though it is not necessary to always choose the same values.

Abscissae vector c. From the practical implementation of DESI methods (a generalisation of SIRK methods) it is desirable that the abscissae should be inside the interval $[0, 1]$. For efficient implementation, the F-property of GLMs restricts the last abscissa c_s to be exactly equal to 1. For the rest of the abscissae, we consider two possible choices:

- $\frac{i}{s}$ for $i = 1, 2, \ldots, s-1$,
- $\frac{i}{s-1}$ for $i = 1, 2, \ldots, s-1$.

We restrict ourselves to only the first option, in the construction of GLMs, used in this thesis.

Vector β. The β-vector is also very important as some of the elements of matrix B directly depend upon the β-terms, e.g. $b_{ss} = 1/\beta_1$ always. However, in the existing algorithm, the β-terms are selected here on the criterion of a fair comparison with the existing competitive methods.

Transformation matrix T. This matrix is still a free parameter and there is no particular choice considered yet except to find such a matrix T that produces small coefficients in the method.

Focusing on the conditions described in the previous section, a reliable and practical algorithm is required to construct the classes of GLMs under the desired conditions. The aim here is to construct the coefficient of the matrices (A, U, B, V).

3.4.2 Conditions on $\widetilde{B}$

After selection of free parameters, it is straight forward to construct the matrices which depend only upon the free parameters, these include J, K, E, C, Ψ, X. Next we need to construct the elements of the method. Up to now, the conditions are the following:

$$\begin{aligned} e_2^{\mathsf{T}} B &= e_s^{\mathsf{T}}, && \text{(F2)} \\ BC(I - KX) &= XE - EX + e_1 \xi^{\mathsf{T}}, && \text{(IRKS)} \\ T^{-1} \ddot{V} T &= Q. && \text{(zero stability)} \end{aligned}$$

The conditions (F2) and (IRKS) are already relating to matrix B. It is easy to convert the third one to relate to B, using $V = E - BCK$. Therefore, all the matrices (A, U, B, V) can be written in terms

order	stages	n(F2)	n(IRKS)	n(zero stability)	$n(\widetilde{B})$	difference
p	s	s	$(s-1)$	$\frac{(s-1)(s-2)}{2}$	$\frac{s(s+1)}{2}$	0
1	2	2	1	–	3	0
2	3	3	2	1	6	0
3	4	4	3	3	10	0
4	5	5	4	6	15	0
5	6	6	5	10	21	0
6	7	7	6	15	28	0

Table 3.1: Number of conditions n required for each p to obtain the elements of the matrix $\widetilde{B}$

of B. Then the transformation Ψ is applied to all the three conditions to obtain linear relations in terms of elements of $\widetilde{B}$. As compared to Runge–Kutta methods, GLMs are in the form of matrices. In addition, conditions are applied in the form of matrices, although these are then simplified and reduced to a comparatively smaller number. In the present case, the size of a GLM is $2s \times 2s$. Though the required number of linear conditions are equal to the number of elements in $\widetilde{B}$ i.e. $s(s+1)/2$ which is somewhat less than $2s \times 2s = 4s^2$. After simplifications, the above 3 conditions (in matrix or vector form) give exactly the same number of linear relations in terms of the elements of $\widetilde{B}$. Table 3.1 shows the number of conditions n for each value of p. Thus,

$$n(\text{F2}) + n(\text{IRKS}) + n(\text{zero stability}) = n(\widetilde{B}).$$

Once the values of $\widetilde{B}$ are obtained, it is easy to find all the numbers in (A, U, B, V). Thus this procedure can be written in the form of an algorithm as follows:

3.4.3 Algorithm for method computation

Step 1. Set $p, \lambda, \epsilon, c, \beta, T$

Step 2. Set $\beta_p = 0$ and make $\beta(z) := 1 + \beta_1 z + \beta_2 z^2 + \cdots + \beta_{p-1} z^{p-1}$

Step 3. Set $\alpha(z) := \frac{(1-\lambda z)^{p+1}}{\beta(z)} + O(z^{p+1})$

Step 4. Calculate $\xi(z) := \alpha(z)(\exp(z) - 1) - \epsilon z^{p+1} + O(z^{p+2})$

Step 5. Construct C, J, K and the following:

$$\begin{aligned} \Psi &= \beta(K), \\ X &= \Psi(\lambda I + J)\Psi^{-1}, \\ E &= \exp(K). \end{aligned}$$

Step 6. Construct lower triangular $\widetilde{B}$ symbolically and then find matrices (A, U, B, V) in terms of elements of $\widetilde{B}$.

Step 7. Solve the conditions on $\widetilde{B}$.

Step 8. Calculate the matrices in the following order

$$\begin{aligned} B &= \Psi\widetilde{B}, \\ A &= B^{-1}XB, \\ U &= C - ACK, \\ V &= E - BCK. \end{aligned}$$

3.4.4 Example

For example,

Step 1. Take the explicit case with $p = 2$ then $\lambda = 0$, $r = s = 3$ and $q = 2$ is obvious. As we are considering equally spaced abscissae so here $c^{\mathsf{T}} = \left[\frac{1}{3}, \frac{2}{3}, 1\right]$. The transformation matrix T is of dimension one and has no effect on the construction of method of order two.

Steps 2-4. As, $\beta_p = 0$ by design, so α- and β- polynomials are

$$\begin{aligned} \beta(z) &= 1 + z\beta_1, \\ \alpha(z) &= \frac{1}{\beta(z)} + O(z^{p+1}), \\ &= 1 - z\beta_1 + z^2{\beta_1}^2 + O(z^3), \\ \xi(z) &= \alpha(z)(\exp(z) - 1) - \epsilon z^{p+1} + O(z^{p+2}), \\ &= 1 + (\frac{1}{2} - \beta_1)z + (\frac{1}{6} - \frac{1}{2}\beta_1 + {\beta_1}^2 - \epsilon)z^2 \end{aligned}$$

Step 5. So,

$$C = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{18} \\ 1 & \frac{2}{3} & \frac{2}{9} \\ 1 & 1 & \frac{1}{2} \end{bmatrix}, J = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, K = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$E = \begin{bmatrix} 1 & 1 & \frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \Psi = \beta(K) = \begin{bmatrix} 1 & \beta_1 & 0 \\ 0 & 1 & \beta_1 \\ 0 & 0 & 1 \end{bmatrix}, X = \begin{bmatrix} \beta_1 & -{\beta_1}^2 & {\beta_1}^3 \\ 1 & 0 & 0 \\ 0 & 1 & -\beta_1 \end{bmatrix}.$$

Step 6. Let

$$\widetilde{B} = \begin{bmatrix} \widetilde{b}_{11} & 0 & 0 \\ \widetilde{b}_{21} & \widetilde{b}_{22} & 0 \\ \widetilde{b}_{31} & \widetilde{b}_{32} & \widetilde{b}_{33} \end{bmatrix}$$

then

$$B = \Psi\widetilde{B} = \begin{bmatrix} \beta_1\widetilde{b}_{21} + \widetilde{b}_{11} & \beta_1\widetilde{b}_{22} & 0 \\ \beta_1\widetilde{b}_{31} + \widetilde{b}_{21} & \beta_1 b_{32} + \widetilde{b}_{22} & \beta_1\widetilde{b}_{33} \\ \widetilde{b}_{31} & \widetilde{b}_{32} & \widetilde{b}_{33} \end{bmatrix}, \quad A = B^{-1}XB = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\widetilde{b}_{11}}{\widetilde{b}_{22}} & 0 & 0 \\ -\frac{\widetilde{b}_{11}\widetilde{b}_{32} - \widetilde{b}_{21}\widetilde{b}_{22}}{\widetilde{b}_{22}\widetilde{b}_{33}} & \frac{\widetilde{b}_{22}}{\widetilde{b}_{33}} & 0 \end{bmatrix},$$

$$U = C - ACK = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{18} \\ 1 & -\frac{1}{3}\frac{-2\widetilde{b}_{22}+3\widetilde{b}_{11}}{\widetilde{b}_{22}} & -\frac{1}{9}\frac{-2\widetilde{b}_{22}+3\widetilde{b}_{11}}{\widetilde{b}_{22}} \\ 1 & \frac{\widetilde{b}_{11}\widetilde{b}_{32} - \widetilde{b}_{21}\widetilde{b}_{22} - \widetilde{b}_{22}^2 + \widetilde{b}_{22}\widetilde{b}_{33}}{\widetilde{b}_{22}\widetilde{b}_{33}} & \frac{1}{6}\frac{2\widetilde{b}_{11}\widetilde{b}_{32} - 2\widetilde{b}_{21}\widetilde{b}_{22} - 4\widetilde{b}_{22}^2 + 3\widetilde{b}_{22}\widetilde{b}_{33}}{\widetilde{b}_{22}\widetilde{b}_{33}} \end{bmatrix},$$

$$V = \begin{bmatrix} 1 & -\beta_1\widetilde{b}_{21} - \beta_1\widetilde{b}_{22} - \widetilde{b}_{11} + 1 & \frac{1}{2} - \frac{1}{3}\beta_1\widetilde{b}_{21} - \frac{1}{3}\widetilde{b}_{11} - \frac{2}{3}\beta_1\widetilde{b}_{22} \\ 0 & -\beta_1\widetilde{b}_{31} - \beta_1\widetilde{b}_{32} - \beta_1\widetilde{b}_{33} - \widetilde{b}_{21} - \widetilde{b}_{22} + 1 & 1 - \frac{1}{3}\beta_1\widetilde{b}_{31} - 1/3\,\widetilde{b}_{21} - \frac{2}{3}\beta_1\widetilde{b}_{32} - \frac{2}{3}\widetilde{b}_{22} - \beta_1\widetilde{b}_{33} \\ 0 & -\widetilde{b}_{31} - \widetilde{b}_{32} - \widetilde{b}_{33} & 1 - \frac{1}{3}\widetilde{b}_{31} - \frac{2}{3}\widetilde{b}_{32} - \widetilde{b}_{33} \end{bmatrix}.$$

Step 7. There are 6 unknown elements in $\widetilde{B}$. Thus, 6 linear relations are required to complete the method. First, the zero stability condition will be applied. In this example the matrix $\ddot{V}$ is a single element. This must be zero, i.e.

$$1 - \frac{1}{3}\widetilde{b}_{31} - \frac{2}{3}\widetilde{b}_{32} - \widetilde{b}_{33} = 0 \tag{3.25}$$

From the F2 condition we have,

$$\begin{aligned} \beta_1\widetilde{b}_{31} + \widetilde{b}_{21} &= 0, \\ \beta_1\widetilde{b}_{32} + \widetilde{b}_{22} &= 0, \\ \beta_1\widetilde{b}_{33} - 1 &= 0. \end{aligned}$$

From the modified form of the IRKS condition (3.19),

$$\begin{aligned} \frac{1}{3}\beta_1\widetilde{b}_{11} + \frac{1}{18}\widetilde{b}_{11} - \frac{1}{6} + \epsilon &= 0, \\ \frac{1}{3}\beta_1\widetilde{b}_{21} + \frac{2}{3}\beta_1\widetilde{b}_{22} + \frac{1}{18}\widetilde{b}_{21} + \frac{2}{9}\widetilde{b}_{22} - \frac{1}{2} &= 0. \end{aligned}$$

Step 8. Solving these 6 equations for $\widetilde{b}_{ij}$'s and using those values in the 4 matrices, the following form of the GLM is obtained.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ \frac{-2(6\epsilon-1)}{(6\beta_1{}^2-5\beta_1+2)(6\beta_1+1)} & 0 & 0 \\ \frac{-3(36\beta_1{}^4-18\beta_1{}^3+2\beta_1{}^2+6\epsilon+\beta_1-1)}{6\beta_1+1} & \frac{3\beta_1(6\beta_1{}^2-5\beta_1+2)}{2} & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} \frac{-3(36\beta_1{}^4-18\beta_1{}^3+2\beta_1{}^2+6\epsilon+\beta_1-1)}{6\beta_1+1} & \frac{3\beta_1(6\beta_1{}^2-5\beta_1+2)}{2} & 0 \\ 0 & 0 & 1 \\ \frac{3(6\beta_1{}^2-4\beta_1+1)}{\beta_1} & \frac{-3(6\beta_1{}^2-5\beta_1+2)}{2\beta_1} & \frac{1}{\beta_1} \end{bmatrix},$$

$$U = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{18} \\ 1 & \frac{2}{3}\,\frac{36\,\beta_1{}^3-24\,\beta_1{}^2+18\,\epsilon+7\,\beta_1-1}{\left(6\,\beta_1{}^2-5\,\beta_1+2\right)(6\,\beta_1+1)} & \frac{2(36\,\beta_1{}^3-24\,\beta_1{}^2+18\,\epsilon+7\,\beta_1-1)}{9\left(6\,\beta_1{}^2-5\,\beta_1+2\right)(6\,\beta_1+1)} \\ 1 & \frac{108\,\beta_1{}^4-36\,\beta_1{}^3-9\,\beta_1{}^2+36\,\epsilon+12\,\beta_1-4}{2(6\,\beta_1+1)} & \frac{12\,\beta_1{}^3-10\,\beta_1{}^2+12\,\epsilon+4\,\beta_1-1}{2(6\,\beta_1+1)} \end{bmatrix},$$

$$V = \begin{bmatrix} 1 & \frac{108\,\beta_1{}^4-36\,\beta_1{}^3-9\,\beta_1{}^2+36\,\epsilon+12\,\beta_1-4}{2(6\,\beta_1+1)} & \frac{12\,\beta_1{}^3-10\,\beta_1{}^2+12\,\epsilon+4\,\beta_1-1}{2(6\,\beta_1+1)} \\ 0 & 0 & 0 \\ 0 & -\frac{1}{2}\,\frac{18\,\beta_1{}^2-9\,\beta_1+2}{\beta_1} & 0 \end{bmatrix}.$$

The method obtained for $\beta_1 = \frac{1}{4}$, is used in this thesis. This value is taken from the theory of PECE pairs (for $p = 2$) so that a fair comparison with the IRKS methods can be obtained. In the same way, methods of different orders can be constructed. However, it is not feasible yet to write the matrix in a general form or even for each value of p.

4
Implementation of GLMs

The challenges of implementation arise in almost every class of numerical methods for ODEs. These include starting procedures, storage of data at the right time of integration and many other design questions in the coding. In the literature, most of the techniques used for the implementation of GLMs are the standard techniques that exist for the implementation of the traditional methods. Although the features of the IRKS methods with the F–property give advantages to look for a better implementation. The aim of this thesis is to study the case of variable stepsize, variable order implementation of the GLMs. To understand the deep design of the (h, p)–algorithm, it is necessary to discuss each essential aspect that is required for the implementation. For this, it is essential to first present the standard techniques. Along with this, in each section, it is discussed how the GLMs can be used for reliable implementation as compared to standard techniques.

The starting procedure of an (h, p)–controller is not a straightforward process. It includes initial stepsize, initial input vector, starting vector, starting method and maybe even the starting order as well. Although, in the implementation of the GLMs, not all these inputs are necessary. For example, starting method can be avoided for variable order implementation; this is discussed in detail in Section 4.2.

A reliable ODE solver consisting of a method or a set of methods, needs a reliable procedure for adjusting the stepsize. However, this is not enough for adjusting the stepsize. A change in stepsize may cause instability due to that method and is not useful without the scale and modify technique. For this, the Nordsieck vector is rescaled by powers of $\mathsf{r} = (h_{\text{new}}/h_{\text{old}})$, for adjusting the stepsize. These adjustments are required due to β–terms in (4.8). These β's are free parameters chosen during the construction of methods, used in the doubly companion matrix (3.8). Each one of these is separately

presented in Section 4.3.

A reliable ODE solver also needs an asymptotically correct error estimator. Usually, in the existing ODE solvers, error estimation is computed at some additional cost. Here, due to high stage order and use of the Nordsieck form of a data vector, it is possible to construct reliable error estimators, not only for the method of order p but also for the method of order $p+1$. On the basis of the estimators, we can adjust the stepsize. All these essentials are discussed in Section 4.5. Before this, it is necessary to understand how the errors are produced; this is presented in Section 4.4.

The order control is mainly explored in the next chapter in which the main goal is to make the heuristic numbers more logical and rational. Here, in Section 4.6, the need for a variable order code, along with some existing order controllers is presented. However, before all this, it is appropriate to see some of the software issues for ODEs.

4.1 ODE software issues

Here are a few of the current challenges faced in the field of numerical analysis for methods of ODEs and software.

- The ultimate aim in the development of various classes of numerical methods for ODEs is to develop the ODE software. Once a new class is developed, the next step required is to focus on implementable methods. In other words, the implementation issues of the numerical methods for ODEs change the requirements and hence the direction of research in developing the theory. This used to happen from time to time in the past and it is still occurring.
- Much theory has been developed in this field but the ODE software is still far from producing theoretical behaviour. One of the reasons is the wide variety of characteristics of ODEs. This, in fact, divides or somehow produces gaps in the theory to stay on a common point. Thus, depending upon the characteristics of ODE, software is built because each characteristic has its own requirements, such as accuracy, etc.
- There are many algorithmic decisions that we make on heuristic data in the ODE software. For example changing the stepsize with extreme changes, changing of order, rejection criteria, etc.. These will all be discussed in this chapter and also in the next chapter on a theoretical basis.
- There is no benchmark for available ODE software. For a standard in the ODE software, we need standard numerical methods. This requires us to first identify what is the most general form of numerical methods for ODEs. Then we need to repeatedly mould it until it fulfils almost all the implementation requirements.
- One of the main issues in developing ODE software is the design of algorithms.
- In a user–friendly ODE solver, there should be options for the users to select some of the parameters. For example, the tolerance scaling method, initial stepsize etc.. What is the default value for the tolerance scaling method? It depends on the problem. Though it is necessary to categorise which possible tolerance definition can be used for each class of problems. This might make it

easier to conclude which value should be the default one. Naturally, it would be the one with the highest efficiency output so is expensive. Although it could be of the user's choice, the one that suits the problem to obtain accuracy requirements. Another example of the options arises from a question; how to select the initial stepsize h_0? It is not bad to consider the user–supplied value as one of the practical approaches. Though a good solver should automatically select the initial stepsize as well.

Considering the issues mentioned above in the rest of the chapter, the implementation techniques are discussed that can be efficiently used in the case of existing GLMs. The starting procedure of a good solver should first give the option to the user to either choose some of the parameters or to decide to use the default values. How to set these values, is discussed in the following section. Only those elements of the starting procedure that are essential for the design of code considered in this thesis, are presented here.

4.2 Starting procedure

For the GLMs considered here, a modified approach of Gladwell et al. [69] is used for automatically calculating the initial stepsize. Usually in the implementation of GLMs, we require a starting method as well and this requires additional work. In our case this can be easily avoided by using an initial order equal to one. How can the starting method can be avoided? This will be discussed later in this section; first let us discuss the scheme for initial stepsize.

4.2.1 Initial stepsize

Initial stepsize and order are very important information required at the start of our code. Initial stepsize is required in the variable stepsize implementation of every type of method and design of ODE solver. It is required in the case of GLMs as well. In a user–friendly and flexible ODE solver, there must be open options given to the users. One of the options should be the selection of the initial stepsize. The two possibilities for selecting the initial stepsize should be either (i) a user–supplied value or (ii) an automatic selection of h_0 by a separate module of the code.

The later approach does not simply mean the use of a default fixed value of h_0 but it depends upon a specified procedure for calculating its value. In this case the only information we have is the initial conditions and we do not want to go beyond one or two function evaluations. Many approaches have been made to select h_0 automatically. A reliable automatic code should be able to analyse the problem and select the initial stepsize automatically. If the initial stepsize is unacceptable or not properly supplied by the user, the code should have the capability of quickly adjusting the stepsize instead of resulting in too many rejections or too many accepted steps before achieving the correct stepsize sequence.

On the other hand, a user–supplied value is based on the experience and knowledge of the user whereas this knowledge includes the same initial value which is supplied to the solver at the start. Using a user–defined value, there is a chance of too many rejections before the stepsize-curve reaches

the ideal and smooth trajectory. The main reason would be poor knowledge or experience, particularly about ODE solvers. On the other hand, a good stepsize controller could be effective in quickly achieving the ideal trajectory. Thus there should be options so that either users supply h_0 based on their knowledge; or otherwise the solver has a reliable approach for selecting h_0 in the form of a separate module.

Because our code starts with a method of order one, so the initial stepsize h_1 can be selected by modifying the approach used by Gladwell et al. [69]. In this approach the fact that the initial order is one ($p_0 = 1$), is used. For this method, the error term, using the Taylor series formula, is $\frac{1}{2}h_1^2 y''(x_0)$. To obtain the maximum possible stepsize for which the norm of the error term is close to the prescribed tolerance, it is assumed that

$$\left\| \frac{1}{2} h_1^2 y''(x_0) \right\| = \text{tol},$$

where $y''(x_0)$ is a vector quantity. So, practically, we can estimate the value of h_1 by *properly* scaling this quantity and using a *suitable* norm. Here the norm $\|.\|_{sc}$ is defined in subsection 4.3.1. Then,

$$h_1 = \sqrt{\frac{2}{\|y''(x_0)\|_{sc}}}, \tag{4.1}$$

For this, we need an approximation to $y''(x_0)$ which can be obtained from

$$y''(x_0) \approx \frac{f\left(x_0 + h_0, y_0 + h_0 f(x_0, y_0)\right) - f(x_0, y_0)}{h_0},$$

and this requires h_0 which corresponds to the stepsize for a method of order zero (note that the usual notation for initial stepsize is h_0, but in this Subsection, it is h_1). For this order, the error term would be $h_0 y'(x_0)$. This value is simply a function evaluation at x_0. So,

$$h_0 = \|f(x_0, y_0)\|_{sc}^{-1},$$

Thus, two function evaluations are required to calculate the initial stepsize from (4.1). This is only possible if the initial order is one. Other advantages of taking $p_0 = 1$, are discussed in the following subsection.

4.2.2 Initial order

Regardless of the order varying behaviour of a code based on GLMs, if the solver is starting with a GLM of higher order, than 1, then it is necessary to use a starting method as well, and in particular for the fixed order codes based on GLMs. The reason is very straightforward. GLMs are in general multivalue methods. The initial conditions of a problem supplied by the user are not enough to take the first step. For example, the GLMs considered here are in Nordsieck form and if a method of order 4 is used in the first step, it is necessary to calculate the approximate values of $hy'(x_0)$, $h^2 y''(x_0)$ and $h^3 y'''(x_0)$ in order to take the first step. For this purpose a starting method is used before using the main integration

algorithm. One of the advantages of the design we are considering for the implementation of GLMs is that we do not need a separate starting method, as we start the code with a method of order 1. In this case the input vector to step number 1 consists of initial value $y(x_0)$ and its first scaled derivative, which is equal to $hf(x_0)$, and this is available without any additional cost. Thus all we need to start a code is a good scheme for the initial stepsize.

4.2.3 Initial input vector

The initial conditions for an ODE are very important information for starting numerical integration because they provide an exact solution at the initial point. The classical Runge–Kutta methods need just this initial value to start the numerical code. This is true with Runge–Kutta method of any order and even in variable order design settings, for example, the RADAU code in [75]. This is possible only when the quantity we are approximating with the method, depends upon information which is calculated locally (like the internal stages in the case of classical Runge–Kutta methods) to produce the numerical solution at the end of a step. The starting case is not that simple for every type of method and in particular when the input required by the method consists of more than one data value i.e. $r > 1$. For example, take a multistep method with k steps where $k \geq 2$. The very first step to be taken by this k-step method needs the solutions at k previous steps (in other words, it needs the initial input vector consisting of solutions at k previous steps). So an additional starting method or procedure is required to obtain this initial input vector before the use of the main method. Many extensions and generalisations of Runge–Kutta methods also need a starting method for this reason.

The starting method produces the initial (Nordsieck) vector for the main method. The order of the starting method must be at least equal to the order of the main method. The main method should not be built on results of lower order. This low accuracy would not be able to achieve consistency.

In the design of DIMSIMs, r and p are almost equal. The implementation of higher–order DIMSIMs also needs starting methods in the fixed–order environment. The existing GLMs with IRKS, a subclass of DIMSIMs, are in Nordsieck form and their fixed order implementation also requires a starting method or procedure. The initial Nordsieck vector consists of the first p–scaled derivatives of the solution $y(x)$, where p is the order of the method. For higher order GLMs, it is not possible to construct the full Nordsieck vector at the initial point. However, the first and second element of the Nordsieck vector are possible to find without any additional procedure. The first value of the Nordsieck vector is simply the initial value and the second value $h_0 y'(x_0)$ is the product of the initial stepsize and one function evaluation of the given ODE at x_0. Thus, in the case of $p_0 = 1$, no starting method is required and the input vector is available at the cost of single function evaluation.

As the GLMs with IRKS will be implemented in a variable stepsize variable order environment, so it will be efficient to start with method of order 1, as discussed above. Further the methods of consecutive order are possible and are used here as well. It is possible to start the code with a method of order 1 and then increase the order gradually, without disruptive fluctuation in order, according to the rational design presented later in the thesis. This dynamical starting of the code is analogous to driving an automatic gear–controlled car which starts in first gear and then automatically adjusts the gear according to the requirements.

4.3 Stepsize control

In ODE software, variable stepsize is a characteristic feature. Adaptivity of stepsize plays one of the important roles in obtaining minimal computational cost of the methods, although it requires computational stability as well. The question is why do we need adaptive strategies in an ODE solver? A simple answer would be to avoid fixed stepsize. In general, the behaviour of the solution of a DE may not be same throughout the integration interval. So if we are bound to use a fixed stepsize then the value of that stepsize would be the minimal one which can work for any subinterval where accuracy is impossible to achieve with bigger step sizes. Contrary to this, in a variable stepsize scheme, the controller automatically adjusts the stepsize to keep the local error bounded by a prescribed tolerance. In other words, using a variable stepsize controller, an optimal stepsize for the next step would be the one for which the norm of the local error estimation is equal to the prescribed tolerance.

The existing codes impose limits on the stepsize ratio $\mathsf{r} = h_{\text{new}}/h_{\text{old}}$ e.g. $\mathsf{r}_{\min} = 0.5$ and $\mathsf{r}_{\max} = 2$. These values are imposed for two important reasons. Firstly, an abrupt change in stepsize can cause problems in producing asymptotic error estimations. These are favourably multipurpose and thus are very important. Secondly, over large values of r will certainly affect the solution of problems with discontinuities which sometimes occur at unknown points. These limits are discussed in Section 4.3.1; the first thing to understand at this point is the standard scheme for stepsize control.

4.3.1 Standard stepsize control scheme

The standard stepsize controller is used for adjusting the stepsize automatically. Almost all other controllers are modifications and generalisations of it. It is based on the error estimate. Thus reliability of error estimation is very important for this purpose. Here, in the case of GLMs, an analytical approach to error estimation is possible, rather than heuristic techniques used in the past such as the embedded technique. An approximation to the local truncation error of a p^{th} order method is

$$\text{err}_{n-1} = C_{n-1} h_{n-1}^{p+1} y_{n-1}^{(p+1)}. \tag{4.2}$$

Given tolerances can be used to find the maximum possible step size for the next step h_n i.e. $C_{n-1} h_n^{p+1} y_n^{(p+1)}$ should be chosen so that its norm is approximately equal to $\text{Atol} + \|y^{[n]}\|\text{Rtol} = sc$, where Atol and Rtol are vectors of dimension N, usually defined by users. And

$$\left(\frac{h_{n-1}}{h_n}\right)^{p+1} = \|\text{err}_{n-1}\|_{sc}, \tag{4.3}$$

must be satisfied, where the norm is defined as

$$\|\text{err}\|_{sc} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\text{err}_i}{sc_i}\right)^2},$$

From (4.3),

$$h_n = \left(\frac{1}{\|\text{err}_{n-1}\|_{sc}}\right)^{\frac{1}{p+1}} h_{n-1}. \tag{4.4}$$

We are taking a risk in an assumption that $y_{n-1}^{(p+1)} = y_n^{(p+1)}$ and that can cause rejections. To avoid unnecessary rejections of steps a safety factor γ is included and (4.4) becomes

$$h_n = \gamma \left(\frac{1}{\|\text{err}_{n-1}\|_{sc}}\right)^{\frac{1}{p+1}} h_{n-1}, \tag{4.5}$$

where γ is the safety factor used (practically) for avoiding unnecessary rejection of steps. Later in this chapter, we will discuss how we can avoid γ. Here, it is necessary to discuss the limitations of the change in stepsize ratio $\mathsf{r} = h_{\text{new}}/h_{\text{old}}$ used (practically) in the standard control scheme. An abrupt change in a stepsize can cause rejection. So limits are used to avoid too small or too large a change in step size. There are two cases. For example, we may take $\mathsf{r}_{\min} = 0.5$ and $\mathsf{r}_{\max} = 2$ with the standard stepsize control scheme. Using this combination, after a successful attempt of a step, the standard stepsize scheme will estimate the stepsize h_{new} for the next step. If $h_{\text{new}} < h_{\text{old}}$ then ($\mathsf{r} < 1$). So, $\mathsf{r}_{\min} = 0.5$ limits r to a decrease of less than a factor of a half so that there is no big reduction in the stepsize. Similarly, $\mathsf{r}_{\max} = 2$ limits r to a increase of less than a factor of a two so that there is no big increase in the stepsize. Hence, these limits help to produce smooth stepsize trajectories. This stepsize scheme is not useful to implement without the following technique.

4.3.2 *Scale and modify* technique

This technique is essential for using variable stepsize schemes for GLMs in Nordsieck form. Adaptive time-stepping is not the only ingredient required when avoiding constant stepsize. Before going into the detail of *Scale and modify*, it is better to explain why the nature of the Nordsieck vector requires *Scale and modify*.

In a variable stepsize integrator, the output vector, in Nordsieck form, at the end of the n^{th} step, with stepsize $h_n = x_n - x_{n-1}$, is

$$y^{[n]} = \begin{bmatrix} y_1^{[n]} \\ y_2^{[n]} \\ y_3^{[n]} \\ \vdots \\ y_{p+1}^{[n]} \end{bmatrix} = \begin{bmatrix} y_n \\ h_n y_n' \\ h_n^2 y_n'' \\ \vdots \\ h_n^p y_n^{(p)} \end{bmatrix}. \tag{4.6}$$

It is not possible to use the same vector as an input to the $(n+1)^{\text{th}}$ step. This is due to the variation in stepsize so that the stepsize in the next step is $h_{n+1} = \mathsf{r}h_n$ which is different from h_n (in general).

So the input to the $(n+1)^{\text{th}}$ step must be

$$y^{[n]} = \begin{bmatrix} y_n \\ rh_n y'_n \\ rh_n^2 y''_n \\ \vdots \\ rh_n^p y_n^{(p)} \end{bmatrix} \tag{4.7}$$

But (4.6) is an approximation to

$$y^{[n]} \approx \begin{bmatrix} y(x_n) \\ h_n y'(x_n) \\ h_n^2 y''(x_n) - \beta_{p-1} h_n^{p+1} y^{(p+1)}(x_n) \\ \vdots \\ h_n^p y^{(p)}(x_n) - \beta_1 h_n^{p+1} y^{(p+1)}(x_n) \end{bmatrix} = y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n). \tag{4.8}$$

In other words, the Nordsieck vector on the right hand side of (4.6) obtained from the numerical method is in fact an approximation to the above vector (4.8). The β's occurring as coefficients of the $(p+1)^{\text{st}}$–derivative terms are the same as those we used in the construction of that particular method. This result is given by Wright [86] which is stated as follows.

Theorem 4.3.1 *The vector β which satisfies*

$$y^{[n]} = y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n) + O(h^{p+2}),$$

is given by

$$\beta = \begin{bmatrix} \beta_p & \beta_{p-1} & \beta_{p-2} & \cdots & \beta_1 \end{bmatrix}^{\mathsf{T}},$$

where the β_i are the free parameters from the doubly companion matrix X.

It is important to mention here that according to the design of the existing GLMs, $\beta_p = \beta_{p+1} = 0$, so the first two elements of the vector in (4.8) do not contain $(p+1)^{\text{st}}$–derivative terms. Hence these two elements need only scaling. Thus,

$$y^{[n]} = \begin{bmatrix} y(x_n) \\ hy'(x_n) \\ h_n^2 y''(x_n) - \beta_{p-1} h_n^{p+1} y^{(p+1)}(x_n) + O(h_n^{p+2}) \\ h_n^3 y^{(3)}(x_n) - \beta_{p-2} h_n^{p+1} y^{(p+1)}(x_n) + O(h_n^{p+2}) \\ \vdots \\ h_n^p y^{(p)}(x_n) - \beta_1 h_n^{p+1} y^{(p+1)}(x_n) + O(h_n^{p+2}) \end{bmatrix},$$

$$= y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n) + O(h_n^{p+2}). \tag{4.9}$$

Recalling to mind that the value of approximation to the above vector is obtained using (2.2b). This is an output vector from the n^{th} step. Before starting the next step, the new stepsize $h_{n+1} = \mathsf{r}h_h$ is obtained from the stepsize controller. Due to variable stepsize, the stepsize ratio r might not be exactly 1. Thus, the input vector to the $(n+1)^{\text{th}}$ step must be scaled correctly. This is only possible if suitable approximations to $h^{p+1}y^{(p+1)}(x_n)$ are available. Thus, for the adjustment of stepsize i.e. when $h_n \to \mathsf{r}h_n$, we rescale the Nordsieck vector and it becomes

$$y^{[n]} = \begin{bmatrix} y(x_n) \\ rh_n y'(x_n) \\ (\mathsf{r}h_n)^2 y''(x_n) - \beta_{p-1}\mathsf{r}^2 h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ (\mathsf{r}h_n)^3 y^{(3)}(x_n) - \beta_{p-2}\mathsf{r}^3 h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ \vdots \\ (\mathsf{r}h_n)^p y^{(p)}(x_n) - \beta_1 \mathsf{r}^p h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \end{bmatrix}, \tag{4.10}$$

$$= D(\mathsf{r}) \left(y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \right), \tag{4.11}$$

where

$$D(\mathsf{r}) = \text{diag}(1, \mathsf{r}, \mathsf{r}^2, \ldots, \mathsf{r}^p).$$

But this is not exactly what we need. The β–terms are not scaled properly and instead of (4.11) we require

$$y^{[n]} = D(\mathsf{r}) \left(y^{[n]}(x_n) \right) - \beta \mathsf{r}^{p+1} h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}). \tag{4.12}$$

So, rescaling on its own is not enough for adjusting the stepsize. A proper adjustment of stepsize will be completed in two steps: scaling as in (4.11), and modifying. Modifying consists of two parts; (i) we have to somehow remove the wrongly scaled β–terms and then (ii) we need to add properly scaled β–terms. Thus, in removing the wrong terms, (4.10) or (4.11) becomes,

$$y^{[n]} = \begin{bmatrix} y(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ \mathsf{r}h_n y'(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ (\mathsf{r}h_n)^2 y''(x_n) - \beta_{p-1}\mathsf{r}^2 h_n^{p+1} y^{(p+1)}(x_n) + \beta_{p-1}\mathsf{r}^2 h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ (\mathsf{r}h_n)^3 y'''(x_n) - \beta_{p-2}\mathsf{r}^3 h_n^{p+1} y^{(p+1)}(x_n) + \beta_{p-2}\mathsf{r}^3 h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \\ \vdots \\ (\mathsf{r}h_n)^p y^{(p)}(x_n) - \beta_1 \mathsf{r}^p h_n^{p+1} y^{(p+1)}(x_n) + \beta_1 \mathsf{r}^p h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}) \end{bmatrix}, \tag{4.13}$$

$$= D(\mathsf{r}) \left(y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n) \right) + D(\mathsf{r}) \beta h_n^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}). \tag{4.14}$$

And by adding the correct terms,

$$y^{[n]} = \begin{bmatrix} y(x_n) \\ \mathsf{r}h_n y'(x_n) \\ (\mathsf{r}h_n)^2 y''(x_n) - \beta_{p-1}\mathsf{r}^2 h_n^{p+1} y^{(p+1)}(x_n) + \beta_{p-1}\mathsf{r}^2 h_n^{p+1} y^{(p+1)}(x_n) - \beta_{p-1}(\mathsf{r}h_n)^{p+1} y^{(p+1)}(x_n) \\ (\mathsf{r}h_n)^3 y'''(x_n) - \beta_{p-2}\mathsf{r}^3 h_n^{p+1} y^{(p+1)}(x_n) + \beta_{p-2}\mathsf{r}^3 h_n^{p+1} y^{(p+1)}(x_n) - \beta_{p-2}(\mathsf{r}h_n)^{p+1} y^{(p+1)}(x_n) \\ \vdots \\ (\mathsf{r}h_n)^p y^{(p)}(x_n) - \beta_1 \mathsf{r}^p h_n^{p+1} y^{(p+1)}(x_n) + \beta_1 \mathsf{r}^p h_n^{p+1} y^{(p+1)}(x_n) - \beta_1(\mathsf{r}h_n)^{p+1} y^{(p+1)}(x_n) \end{bmatrix}$$
$$+ O((\mathsf{r}h_n)^{p+2}), \tag{4.15}$$

$$= D(\mathsf{r})\left(y^{[n]}(x_n) - \beta h_n^{p+1} y^{(p+1)}(x_n)\right) + D(\mathsf{r})\beta h_n^{p+1} y^{(p+1)}(x_n) - \beta(\mathsf{r}h_n)^{p+1} y^{(p+1)}(x_n) + O((\mathsf{r}h_n)^{p+2}), \tag{4.16}$$

$$= D(\mathsf{r})y^{[n]}(x_n) + (D(\mathsf{r}) - \mathsf{r}^{p+1}I)\beta h_n^{p+1} y^{(p+1)}(x_n) + O(h_n^{p+2}). \tag{4.17}$$

The *scale and modify* technique is necessary from many aspects. From the implementation point of view, it is very important to approximate $h^{p+1}y^{(p+1)}(x_n)$ asymptotically correctly, as it is used for multi-purposes including stepsize control, and error estimation of method of order p as well as $p+1$. It is discussed further in Section 4.5 how the approximations to $h^{p+1}y^{(p+1)}(x_n)$ depend on the stage derivatives and the elements of the output Nordsieck vector at the end of a step. Wrongly scaled β-terms in (4.8) would definitely affect the reliability of the estimators. Another important aspect is due to stability which is explained in the following section.

4.3.3 *Scale and modify* technique effect on stability

The scale and modify technique is necessary for producing stable results. A GLM will be stable if $\|V^n\|$ lies in a unit circle. Here, the same thing can be viewed from a different angle. Let us say that when a Nordsieck vector, obtained from (2.2b), is scaled by $D(\mathsf{r})$, the following two changes happen:

$$\begin{aligned} B &\longrightarrow D(\mathsf{r})B + \Delta B \qquad \text{some modification to } \ B, \\ V &\longrightarrow D(\mathsf{r})V + \Delta V \qquad \text{some modification to } \ V. \end{aligned}$$

For stability, we need to settle these modifications. This can be done by choosing ΔV so that $D(\mathsf{r})V + \Delta V = V$.

For simplicity, consider the one-dimensional problem $y' = 1$ and let $y^{[0]}$ be the initial Nordsieck vector; then the first output vector is

$$y^{[1]} = Vy^{[0]},$$

and, as explained in the previous section, due to variable stepsize implementation, the input vector to

the second step would be

$$y^{[1]} = D(\mathsf{r}_1)Vy^{[0]}.$$

Similarly the input to the third step is

$$y^{[2]} = D(\mathsf{r}_2)V.D(\mathsf{r}_1)Vy^{[0]}.$$

In the same way the input to n^{th} step is

$$y^{[n-1]} = D(\mathsf{r}_{n-1})V.D(\mathsf{r}_{n-2})V \dots D(\mathsf{r}_2)V.D(\mathsf{r}_1)Vy^{[0]},$$

and for stable results

$$\left\| \prod_{i=1}^{n} D(\mathsf{r}_i)V \right\| \leq constant.$$

As discussed in the previous section, the first two elements of the Nordsieck vector need neither scaling nor modification, due to $\beta_p = \beta_{p+1} = 0$. So instead of the whole matrix V, the stability effect is related to only $\ddot{V}$ and so

$$\left\| \prod_{i=1}^{n} \ddot{D}(\mathsf{r}_i)\ddot{V} \right\| \leq constant,$$

where double dot means the matrix obtained by removing the first two rows and the first two columns of that matrix. So, if this product for each step is defined as a function of r, i.e.

$$\ddot{V}(\mathsf{r}) = \ddot{D}(\mathsf{r})\ddot{V} = \begin{bmatrix} \mathsf{r}^2 v_{31} & \mathsf{r}^2 v_{32} & \dots & \mathsf{r}^2 v_{3(s-1)} & \mathsf{r}^2 v_{3s} \\ \mathsf{r}^3 v_{41} & \mathsf{r}^3 v_{42} \dots & & \mathsf{r}^3 v_{4(s-1)} & \mathsf{r}^3 v_{4s} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathsf{r}^{p-1} v_{(s-1)1} & \mathsf{r}^{p-1} v_{(s-1)2} & \dots & \mathsf{r}^{p-1} v_{(s-1)(s-1)} & \mathsf{r}^{p-1} v_{(s-1)s} \\ \mathsf{r}^p v_{s1} & \mathsf{r}^p v_{s2} & \dots & \mathsf{r}^p v_{s(s-1)} & \mathsf{r}^p v_{ss} \end{bmatrix},$$

this product must be power bounded for stability. For this the eigenvalues of $\ddot{V}$ must lie in $[-1, 1]$. Therefore they satisfy

$$|\det(\omega I - \ddot{V}(\mathsf{r}))| \leq 1.$$

This will certainly give some restrictions on the stepsize ratio r, so that $\mathsf{r} \in [\mathsf{r}_{\min}, \mathsf{r}_{\max}]$ which may vary from one method to another. These restrictions on r must be imposed for numerical stability and this is not a practical approach. Fortunately these restrictions can be avoided if proper scaling and modification of the Nordsieck vector is performed after the first step and before the second step.

Furthermore, it is not necessary that only $h^{p+1}y^{(p+1)}(x_n)$-terms are to be considered for the scale and modify process. It might be a combination of $h^{p+1}y^{(p+1)}(x_n)$ with other quantities like $h^{p+2}y^{(p+2)}(x_n)$ and $h^{p+2}f'(y_n)y^{(p+1)}(x_n)$. This is discussed in [56]. Thus further terms can be considered for more efficient scale and modify to further refine the estimators.

In order to test the potential loss of zero stability for the methods due to scale and modify, the analysis for the methods of order 2, 3 and 4 are presented in the Figure 4.1. For each of these methods,

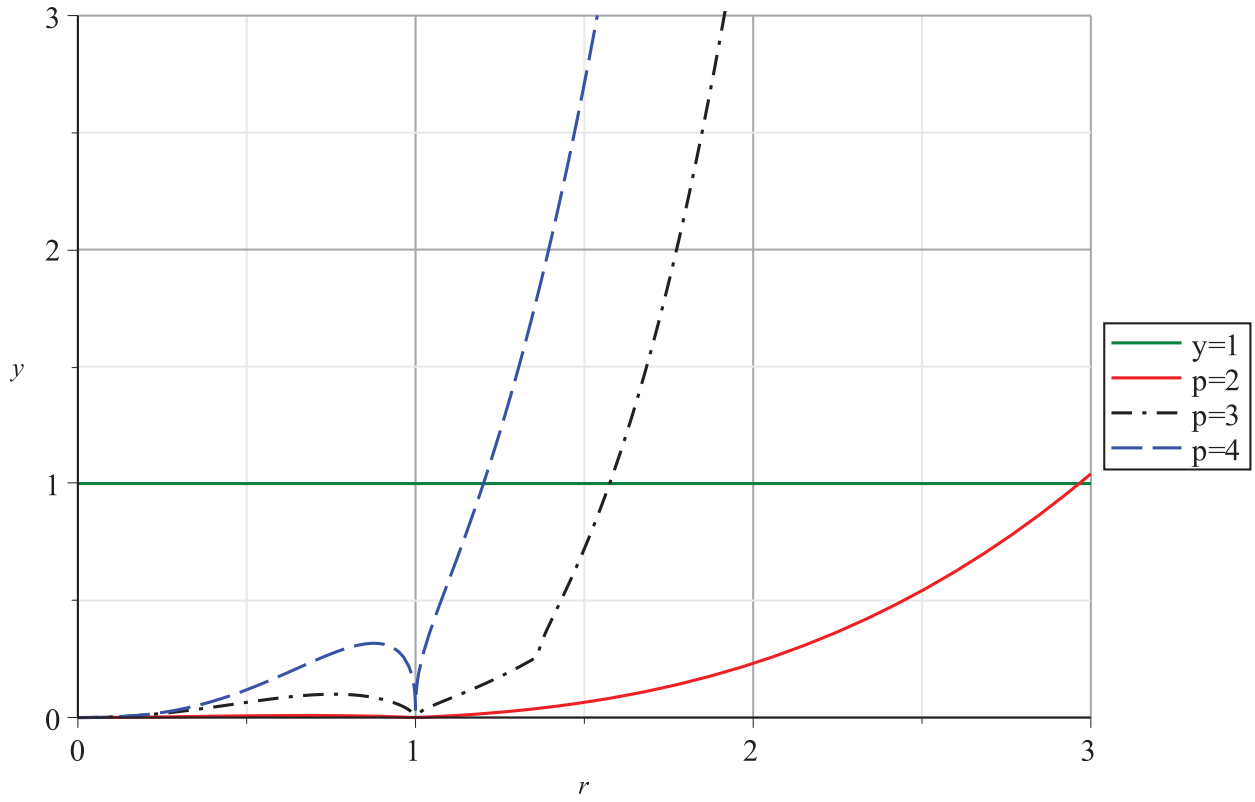


Figure 4.1: Possible stepsize ratio r for each of the methods of order two to four.

the eigenvalue with maximum absolute value of $\ddot{V}(\mathsf{r})$ is plotted on the y axis. Whereas $y = 1$ indicates the stable boundary. For example, the graph shows that the method of order 4 can produce unstable computations in the case where the step size is increased by a factor of more than 1.2 over many consecutive steps.

4.3.4 PI (proportional-integral) controller

The purpose of a stepsize controller is to minimise the computational cost by maintaining the accuracy requirements. The standard stepsize controller gives a stepsize for the next step which is based on the stepsize and the error estimate for the existing step. This means that proper values of these two items are necessary and particularly the value of the error estimate. In other words, this controller is in fact an error controller. Without discussing the definition of norm used in a code, we will first think about error estimation. In the case of IRKS methods, we have very promising estimators. Thus the goal is to keep the error in a step close to the specified tolerance. In the mechanism of this controller, the reason for rejection of a stepsize is not straightforward; it is too big a stepsize. It might be too small a prediction of the stepsize which may be due to too big an estimation of error. If the error produced, and hence correctly estimated, is too much smaller than the tolerance, then it may produce too big a stepsize for the next step. Ultimately there will be a chance of unnecessary rejection of steps. Another reason would be an abrupt change in the value of stepsize ratio r. This can also cause an abrupt change

in the stepsize and hence the error produced in the next step. Usually to avoid these disadvantages or chances of failure of the scheme, the stepsize ratio r is bounded by an upper and lower limit. These two values are another example of heuristic approaches to software design.

In the work by Gustafsson et al. [71], a stepsize control algorithm is analysed from a control theory point of view this is known as a PI controller. According to these authors, the standard stepsize control needs analysis and improvements. The selection of stepsize can be treated as an automatic control problem so that we can study it in the control theory. In terms of control theory, the standard stepsize control is simply I-control (integral) which depends on the error estimation of the most recent step. On the other hand, in PI-control, the error estimation of the two most recent steps are involved i.e.

$$\mathsf{r}_n = \left(\frac{\text{tol}}{est_{n-1}}\right)^{\alpha/(p+1)} \cdot \left(\frac{\text{tol}}{est_{n-2}}\right)^{\beta/(p+1)}$$

where $\alpha + \beta = 1$. PI control is a sophisticated stepsize control used in modern solvers. We are going to implement PI-control along with a Lagrange controller. The changes we need to make are not that complex. We only need to change the formula with restrictions given below. In the PI-controller, we take $\beta = 0$ in the following cases:

⋆ At the start of the integration.

⋆ When a step is rejected.

⋆ When the order changes.

In order to study the local discretisation errors in a step with method of order p and one order higher $(p + 1)$, it is necessary to first discuss the error growth of the general linear methods. Then these results are applied to the estimation of local discretisation errors. The following section contains some of these results.

4.4 Error propagation

One of the aims in this chapter is to discuss the error estimators and for this it is necessary to first see how the error propagates when the GLM is used for numerical integration. Another reason for this study is for the maintenance of the Nordsieck vectors due to variable stepsize implementation. The GLMs considered here are with $s = p + 1$ stages and $r = s$ values in the Nordsieck vector, which is up to the p^{th} derivative (scaled) of $y(x)$. A combined need of these two reasons is to analyse the quantities passed from step to step to within $O(h^{p+2})$. To investigate the error propagation in variable stepsize mode, it is very straight forward in the case of multistage methods. However, in the case of the GLMs, it is necessary to amend the solution vector $y^{[n]}$. The input Nordsieck vector from x_{n-1} to

x_n is

$$\mathbf{y^{[n-1]}} = \begin{bmatrix} y(x_{n-1}) + O(h^{p+2}) \\ hy'(x_{n-1}) - \delta_1 h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}) \\ h^2 y''(x_{n-1}) - \delta_2 h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}) \\ \vdots \\ h^p y^{(p)}(x_{n-1}) - \delta_p h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}) \end{bmatrix}, \tag{4.18}$$

where these δ–terms must be considered for discussing error estimation (see (4.8)). It can be easily observed that there is no error term of order $(p+1)$ in the first component of the Nordsieck vector. This is due to the design choice $\beta_{p+1} = 0$. Therefore, the scale and modify process is not required to touch the first component. For this reason it is convenient to partition the Nordsieck vector as

$$\mathbf{y^{[n]}} = \begin{bmatrix} y_n \\ y^{[n]} \end{bmatrix}.$$

Using (4.18) and the result in [86] that $\delta_i = \beta_{p+1-i}$ for $i = 1, 2, \ldots, p$, we have

$$y^{[n-1]} = z(x_{n-1}, h) - (\beta \otimes I) h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}) \tag{4.19}$$

where

$$z(x, h) = \begin{bmatrix} hy'(x) \\ h^2 y''(x) \\ \vdots \\ h^p y^{(p)}(x) \end{bmatrix}.$$

And with this, we can partition the coefficient matrix as

$$\left[\begin{array}{c|c} A & \mathbf{U} \\ \hline \mathbf{B} & \mathbf{V} \end{array}\right] = \left[\begin{array}{c|cc} A & \mathbf{1} & U \\ \hline b^{\intercal} & 1 & v^{\intercal} \\ B & 0 & V \end{array}\right],$$

and therefore,

$$\begin{aligned} Y &= y_{n-1}\mathbf{1} + AhF + Uy^{[n-1]}, \\ y_n &= y_{n-1} + b^{\intercal} hF + v^{\intercal} y^{[n-1]}, \\ y^{[n]} &= BhF + Vy^{[n-1]}, \end{aligned}$$

where Y and F are vectors of length s consisting of approximations to $y(x_{n-1} + c_i h)$ and $y'(x_{n-1} + c_i h)$ for $i = 1, 2, \ldots, s$. And $y^{[n-1]}$ and $y^{[n]}$ are approximations of order p to the exact Nordsieck vectors $z(x_{n-1}, h)$ and $z(x_n, h)$ respectively. As the stage order q is equal to the order of the method p in the

GLMs, it is necessary to consider the vector ξ of stage errors such that

$$\begin{aligned} Y &= y(x_{n-1}\mathbf{1} + ch) - \xi h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}), \\ hF &= hy'(x_{n-1}\mathbf{1} + ch) - \xi h^{p+2} f'(y_{n-1}) y^{(p+1)}(x_{n-1}) + O(h^{p+3}), \end{aligned}$$

where $f'(y_{n-1}) = \frac{\partial f(y(x_{n-1}))}{\partial y}$ denotes the matrix of partial derivatives $\frac{\partial f}{\partial y}$ evaluated at y_{n-1}. Now these stage derivatives, with error terms of h^{p+2}, are used to calculate the Nordsieck vector at the end of step number n, so it is necessary to consider a few more terms in (4.19) as the solution propagates, which leads to the following [56]:

Theorem 4.4.1 *Suppose the input to step number n consists of*

$$\begin{aligned} y_{n-1} &= y(x_{n-1}), \\ y^{[n-1]} &= z(x_{n-1}, h) - \alpha_{n-1} h^{p+1} y^{(p+1)}(x_{n-1}) - \beta_{n-1} h^{p+2} y^{(p+2)}(x_{n-1}) \\ &\quad - \gamma_{n-1} h^{p+2} f'(y_{n-1}) y^{(p+1)}(x_{n-1}) + O(h^{p+3}). \end{aligned}$$

Then the stage values, scaled stage derivatives and the output values are given by

$$\begin{aligned} Y &= y(x_{n-1}\mathbf{1} + ch) - \xi h^{p+1} y^{(p+1)}(x_{n-1}) + O(h^{p+2}), \\ hF &= hy'(x_{n-1}\mathbf{1} + ch) - \xi h^{p+2} f'(y_{n-1}) y^{(p+1)}(x_{n-1}) + O(h^{p+3}), \\ y_n &= y_{n-1}(x_n) - \epsilon h^{p+1} y^{(p+1)}(x_n) - \rho h^{p+2} y^{(p+2)}(x_n) - \sigma h^{p+2} f'(y_n) y^{(p+1)}(x_n) + O(h^{p+3}), \\ y^{[n]} &= z(x_n, h) - \alpha_n h^{p+1} y^{(p+1)}(x_n) - \beta_n h^{p+2} y^{(p+2)}(x_n) - \gamma_n h^{p+2} f'(y_n) y^{(p+1)}(x_n) + O(h^{p+3}), \end{aligned}$$

where

$$\begin{aligned} \xi &= \frac{c^{p+1}}{(p+1)!} - \frac{Ac^p}{p!} + U\alpha_{n-1}, \\ \epsilon &= \frac{1}{(p+1)!} - \frac{b^{\intercal} c^p}{p!} + v^{\intercal} \alpha_{n-1}, \\ \rho &= \frac{1}{(p+2)!} - -\frac{b^{\intercal} c^{p+1}}{(p+1)!} + v^{\intercal} \beta_{n-1} - \epsilon, \\ \sigma &= b^{\intercal} \xi + v^{\intercal} \gamma_{n-1}, \\ \alpha_n &= E_p - B\frac{c^p}{p!} + V\alpha_{n-1}, \\ \beta_n &= E_{p+1} - B\frac{c^{p+1}}{(p+1)!} + V\beta_{n-1} - \alpha_n, \\ \gamma_n &= B\xi - \epsilon e_1 + V\gamma_{n-1}. \end{aligned}$$

Note that the coefficients $(\alpha_n, \beta_n, \gamma_n)$ of the perturbed quantities introduced in the n^{th} step, are different from α's and β's used in the construction of GLMs. These coefficients converge to fixed values $(\widetilde{\alpha}, \widetilde{\beta}, \widetilde{\gamma})$ in the limiting case as used in the following theorem. This theorem gives these relations and describes those terms which should be used for the scale and modify procedure in the existing case.

Theorem 4.4.2 *Assume that the input to step n, for which the stepsize will be $h_n = x_n - x_{n-1}$, consists of y_{n-1} and $y^{[n-1]}$ given by*

$$\begin{aligned} y^{[n-1]} &= \widehat{z}(x_{n-1}, h_n), \\ &= z(x_{n-1}, h_n) - \widetilde{\alpha} h_n^{p+1} y^{(p+1)}(x_{n-1}) - \widetilde{\beta} h_n^{p+2} y^{(p+2)}(x_{n-1}) \\ &- \widetilde{\gamma} h_n^{p+2} f'(y_{m-1}) y^{(p+1)}(x_{n-1}) + O(h_n^{p+3}), \end{aligned}$$

where

$$\begin{aligned} \widetilde{\alpha} &= (1 - V)^{-1} \left(E_p - B \frac{c^p}{p!} \right), \\ \widetilde{\beta} &= (1 - V)^{-1} \left(E_{p+1} - B \frac{c^{p+1}}{(p+1)!} - \widetilde{\alpha} \right), \\ \widetilde{\gamma} &= (1 - V)^{-1} \left(B\xi - \epsilon e_1 \right). \end{aligned}$$

Then the result of applying the scaling $D(\mathsf{r})$ to $h_n BF + V y^{[n-1]}$ is

$$\begin{aligned} &\widehat{z}(x_{n-1}, h_{n+1}) - \theta_1(\mathsf{r}) h_n^{p+1} y^{(p+1)}(x_n) - \theta_2(\mathsf{r}) h_n^{p+2} y^{(p+2)}(x_n) \\ &- \theta_3(\mathsf{r}) h_n^{p+2} f'(y_n) y^{(p+1)}(x_n) + O(h_n^{p+3}), \end{aligned}$$

where

$$\begin{aligned} \theta_1(\mathsf{r}) &= \left(D(\mathsf{r}) - \mathsf{r}^{p+1} I \right) \widetilde{\alpha}, \\ \theta_2(\mathsf{r}) &= \left(D(\mathsf{r}) - \mathsf{r}^{p+2} I \right) \widetilde{\beta}, \\ \theta_3(\mathsf{r}) &= \left(D(\mathsf{r}) - \mathsf{r}^{p+2} I \right) \widetilde{\gamma}. \end{aligned}$$

4.5 Error estimation

In general, it is not easy or reliable to estimate the error in the case of Runge–Kutta methods. In the case of Runge–Kutta methods it is much more difficult to estimate the error for a method other than the one used in the existing step. This is because when we want to change the order, we need a reliable error estimation for that method. That is, we want to predict what the error would be if a different order method has been used in the existing step. This helps us not only to decide to change the order but also to select a suitable stepsize for the next step with a method of new order. For the methods considered here, error estimation plays one of the key roles to practically and efficiently implement these methods in (h, p)-refinement mode.

In a design test of an ODE solver, the main focus is towards achieving or testing the stability, accuracy and computational efficiency of the method. However, to measure accuracy, it is required to measure the errors properly. Here we have to rely on the error estimations which are analytically reliable in the case of GLMs. A point to be noted is how to measure a proper/suitable norm and scaling of the error.

4.5.1 Estimation of higher order terms $h^{p+1}y^{(p+1)}(x)$ and $h^{p+2}y^{(p+2)}(x)$

The Nordsieck vector in our design of methods is of length $(p+1)$, consisting of the first p (scaled) derivatives of $y(x)$. In order to implement GLMs in variable step size variable order mode, one more higher derivative term $h^{p+1}y^{(p+1)}(x)$ is to be estimated for the following purposes:

- ⋆ to estimate error
- ⋆ for selection of step size in the next step using standard step size control
- ⋆ to estimate $h^{p+2}y^{(p+2)}(x)$

Use of the F-property gives the advantage of using $c_0 = 0$ and $hF_0 = y_0^{[n-1]}$ and the result in [58, 35] can be expressed as follows.

Theorem 4.5.1 *The error estimates of the scaled-higher derivatives are:*

$$\delta^{\mathsf{T}} hF + \psi y_2^{[n-1]} = h^{p+1}y^{(p+1)}(x_n) + O(h^{p+3}), \tag{4.20}$$

$$\frac{\mathsf{r}}{(\mathsf{r}+\theta-\mathsf{r}\theta)}\left(y^{(p+1)}(x_n) - \mathsf{r}^{p+1}y^{(p+1)}(x_{n-1})\right) = h^{p+2}y^{(p+2)}(x_n) + O(h^{p+3}), \tag{4.21}$$

where $\psi = -\sum_{i=1}^{s}\delta_i$ *and* $\mathsf{r} = h_n/h_{n-1}$. *The coefficients* δ *and* ψ *in* (4.20) *satisfy*

$$\delta^{\mathsf{T}} C + \psi e_1 = 0, \qquad \delta^{\mathsf{T}}\varepsilon = 0, \qquad \delta^{\mathsf{T}}\frac{c^p}{p!} = 1, \tag{4.22}$$

C *is the Vandermonde matrix (see Section 3.3.2), and* θ *in* (4.21) *satisfies*

$$\delta^{\mathsf{T}}\frac{c^{p+1}}{(p+1)!} = \theta. \tag{4.23}$$

Proof.

$$\begin{aligned}\delta^{\mathsf{T}} hF + \psi y_2^{[n-1]} &= \delta^{\mathsf{T}}\left(hy'(x_{n-1}+hc_i) - \varepsilon h^{p+2}f'(y(x_{n-1}))y^{(p+1)}(x_{n-1}) + O(h^{p+3})\right)\\ &\quad + \psi\left(y_2^{[n-1]}(x_{n-1}) + O(h^{p+3})\right).\end{aligned}$$

Using the Taylor series,

$$\begin{aligned}&= \delta^{\mathsf{T}}\left(Cy^{[n-1]}(x_{n-1}) + \frac{c^p}{p!}h^{p+1}y^{(p+1)}(x_{n-1}) + \frac{c^{p+1}}{(p+1)!}h^{p+2}y^{(p+2)}(x_{n-1})\right.\\ &\left.-\varepsilon h^{p+2}f'(y(x_{n-1}))y^{(p+1)}(x_{n-1}) + O(h^{p+3})\right) + \psi\left(y_2^{[n-1]}(x_{n-1}) + O(h^{p+3})\right),\\ &= \left(\delta^{\mathsf{T}}C + \psi e_1\right)y^{[n-1]}(x_{n-1}) + \left(\delta^{\mathsf{T}}\frac{c^p}{p!}\right)h^{p+1}y^{(p+1)}(x_{n-1})\\ &+ \delta^{\mathsf{T}}\varepsilon h^{p+2}f'(y(x_{n-1}))y^{(p+1)}(x_{n-1}) + \left(\delta^{\mathsf{T}}\frac{c^{p+1}}{(p+1)!}\right)h^{p+2}y^{(p+2)}(x_{n-1}) + O(h^{p+3}).\end{aligned} \tag{4.24}$$

In order to obtain the quantity on the right hand side in (4.20), the above result induces that

$$\delta^{\mathsf{T}} C + \psi e_1 = 0,$$
$$\delta^{\mathsf{T}} \varepsilon = 0.^{1}$$

Now to obtain the result in (4.21), we consider

$$\delta^{\mathsf{T}} hF + \psi y_2^{[n-1]} = h^{p+1} y^{(p+1)}(x_n) + \theta h^{p+2} y^{(p+2)}(x_n) + O(h^{p+3}),$$

and comparing this with (4.24) gives

$$\delta^{\mathsf{T}} \frac{c^{p+1}}{(p+1)!} = \theta.$$

For this, we need to consider ($h_n = \mathsf{r}h$)

$$\begin{aligned} est_n^{p+1} &= h_n^{p+1} y_n^{(p+1)}, \\ &= (\mathsf{r}h)^{p+1} \left(y^{(p+1)}(x_{n-1}) + (1-\theta)(\mathsf{r}h) y^{(p+2)}(x_{n-1}) \right) + O(h^{p+3}), \qquad (4.25) \\ &= (\mathsf{r}h)^{p+1} y^{(p+1)}(x_n - \theta \mathsf{r}h) + O(h^{p+3}). \qquad (4.26) \end{aligned}$$

Now to estimate $h_n^{p+2} y^{(p+2)}(x_n)$, the difference of est_n^{p+1} and est_{n-1}^{p+1} at the end of steps $n-1$ and n can be used where $h_n = \mathsf{r}h$. Then at the end of step $n-1$ (4.26) would be

$$\begin{aligned} est_{n-1}^{p+1} &= h_{n-1}^{p+1} y_{n-1}^{(p+1)}, \\ &= h^{p+1} \left(y^{(p+1)}(x_{n-2}) + (1-\theta) h y^{(p+2)}(x_{n-2}) \right) + O(h^{p+3}), \qquad (4.27) \\ &= h^{p+1} y^{(p+1)}(x_{n-1} - \theta h) + O(h^{p+3}), \\ &= h^{p+1} y^{(p+1)}(x_n - \mathsf{r}h - \theta h) + O(h^{p+3}). \qquad (4.28) \end{aligned}$$

Therefore using (4.26) and (4.28),

$$\begin{aligned} est_n^{p+1} - \mathsf{r}^{p+1} est_{n-1}^{p+1} &= (\mathsf{r}h)^{p+1} y^{(p+1)}(x_n - \theta \mathsf{r}h) - \mathsf{r}^{p+1} \left(h^{p+1} y^{(p+1)}(x_n - \mathsf{r}h - \theta h) \right) + O(h^{p+3}), \\ &= (\mathsf{r}h)^{p+1} \left(y^{(p+1)}(x_n - \theta \mathsf{r}h) - y^{(p+1)}(x_n - \mathsf{r}h - \theta h) \right) + O(h^{p+3}), \\ &= (\mathsf{r}h)^{p+1} \left(y^{(p+1)}(x_n) - \theta \mathsf{r} h y^{(p+2)}(x_n) - y^{(p+1)}(x_n) + (\mathsf{r}+\theta) h y^{(p+2)}(x_n) \right) + O(h^{p+3}), \\ &= (\mathsf{r}h)^{p+1} \left(\mathsf{r} + \theta - r\theta \right) h y^{(p+2)}(x_n) + O(h^{p+3}), \\ \frac{r(est_n^{p+1} - \mathsf{r}^{p+1} est_{n-1}^{p+1})}{(\mathsf{r} + \theta - \mathsf{r}\theta)} &= (\mathsf{r}h)^{p+2} y^{(p+2)}(x_n) + O(h^{p+3}), \qquad (4.29) \\ \text{or} est_n^{p+2} &= \frac{\mathsf{r}(est_n^{p+1} - \mathsf{r}^{p+1} est_{n-1}^{p+1})}{(\mathsf{r} + \theta - \mathsf{r}\theta)}. \end{aligned}$$

[1] The case in which it is non–zero naturally, is not discussed here.

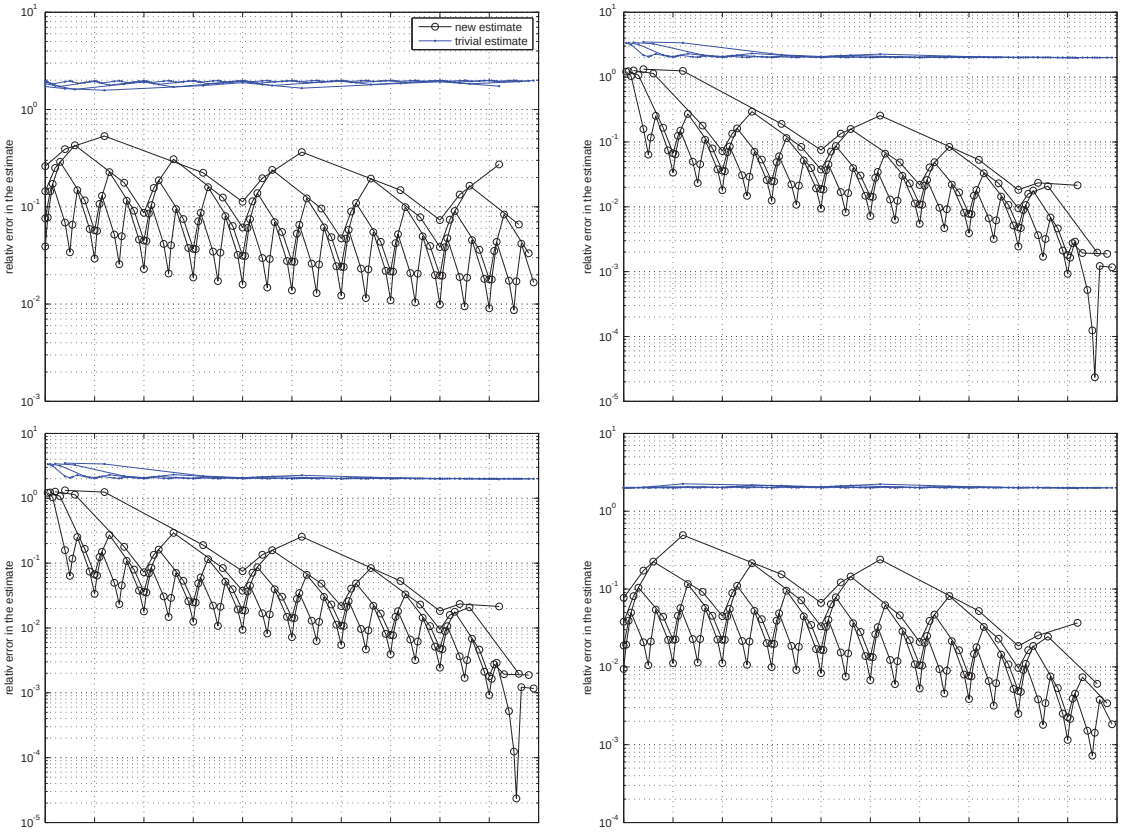


Figure 4.2: Relative errors using method of order 2 with problems A2(top–left), A3(top–right), A3–2D(bottom–left) and A3–3D(bottom–right).

The practical approach of calculating the coefficients $\delta_i, i = 0, 1, \ldots, p+1$ and θ leads to the matrix L given in [35]. This matrix is obtained by using the conditions given in (4.22) and (4.23)

$$L = \begin{bmatrix} \epsilon_0 & 1 & c_0 & \frac{1}{2}c_0^2 & \cdots & \frac{1}{p!}c_0^p & \frac{1}{(p+1)!}c_0^{p+1} \\ \epsilon_1 & 1 & c_1 & \frac{1}{2}c_1^2 & \cdots & \frac{1}{p!}c_1^p & \frac{1}{(p+1)!}c_1^{p+1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \epsilon_{p+1} & 1 & c_{p+1} & \frac{1}{2}c_{p+1}^2 & \cdots & \frac{1}{p!}c_{p+1}^p & \frac{1}{(p+1)!}c_{p+1}^{p+1} \end{bmatrix}.$$

For example this matrix L for $p = 3$ is

$$\begin{bmatrix} \frac{53}{6144} & 1 & 0 & 0 & 0 & 0 \\ \frac{39}{640} & 1 & \frac{1}{4} & \frac{1}{32} & \frac{1}{384} & \frac{1}{6144} \\ \frac{2019}{10240} & 1 & \frac{1}{2} & \frac{1}{8} & \frac{1}{48} & \frac{1}{384} \\ \frac{1}{192} & 1 & \frac{3}{4} & \frac{9}{32} & \frac{9}{128} & \frac{27}{2048} \end{bmatrix}$$

In order to check the quality of the error estimators, the linear problems A2, A3 and the autonomous forms of A3 (see Appendix) are used for testing. These problems are tested for the interval $[0, 1]$. The

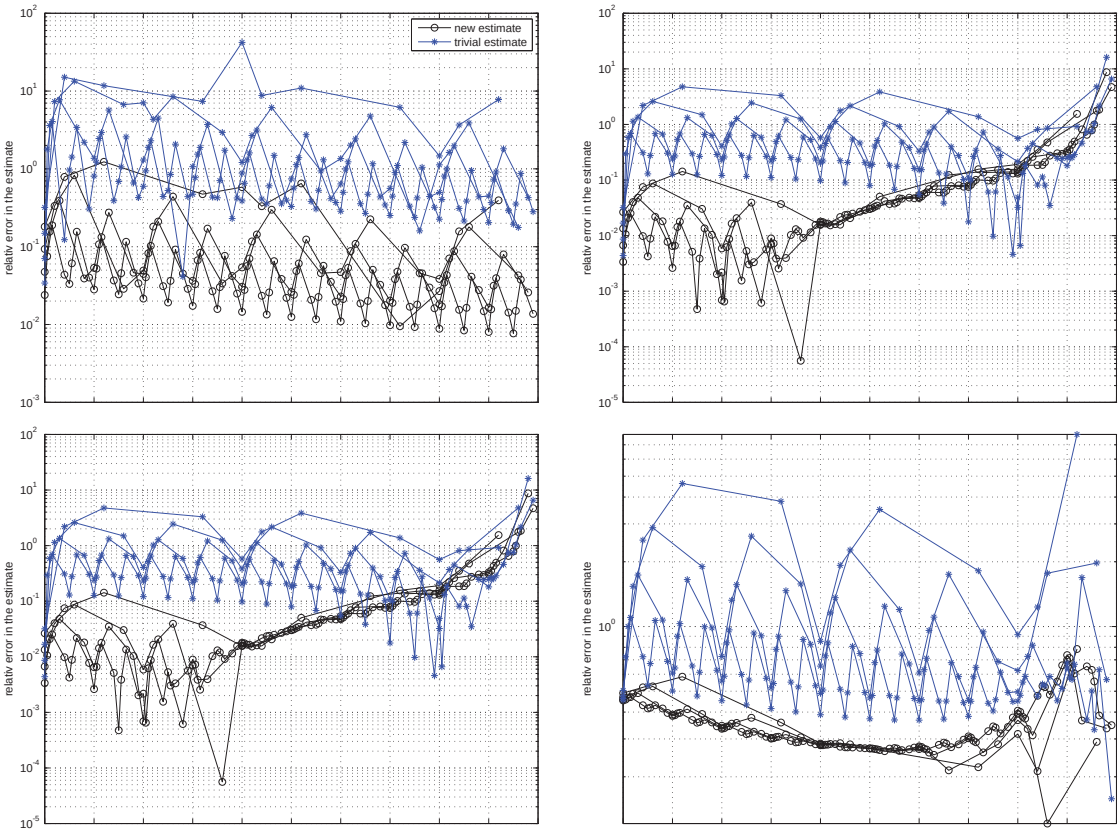


Figure 4.3: Relative errors using method of order 3 with problems A2(top–left), A3(top–right), A3–2D(bottom–left) and A3–3D(bottom–right).

relative error at the end of the interval is calculated using the error estimator mentioned in this section and trivial error estimator. The relative errors (along vertical axis on logarithmic scale) for the time interval $[0, 1]$ (along horizontal axis) are shown in Figures 4.2, 4.3 and 4.4 for methods of order two, three and four respectively. For this, a pseudo variable stepsize scheme is used with a stepsize pattern of five consecutive step sizes, selected using the step sizes ratios $\mathsf{r} = [1, 2, 5, 2]$. This is repeated for 25, 50, 100 and 200 steps.

4.6 Stepsize and order control

An (h, p)-controller in an ODE solver decides when it should switch between methods of different order and how to adjust the stepsize accordingly. This is a part of the main question addressed in [20]. Before going ahead, a few questions that could naturally arise are narrated here.

Why variable order? Instead of only using adaptive time stepping (for optimising cost), another practical approach is to change the order of the method (or in other words, change the method) during the integration. This is because not all the ODE problems can be efficiently solved using a fixed order method. Sometimes it is required to achieve a very high accuracy, and in this case, low order methods use very small step sizes that can greatly affect the cost. On the other hand, if the tolerance is quite

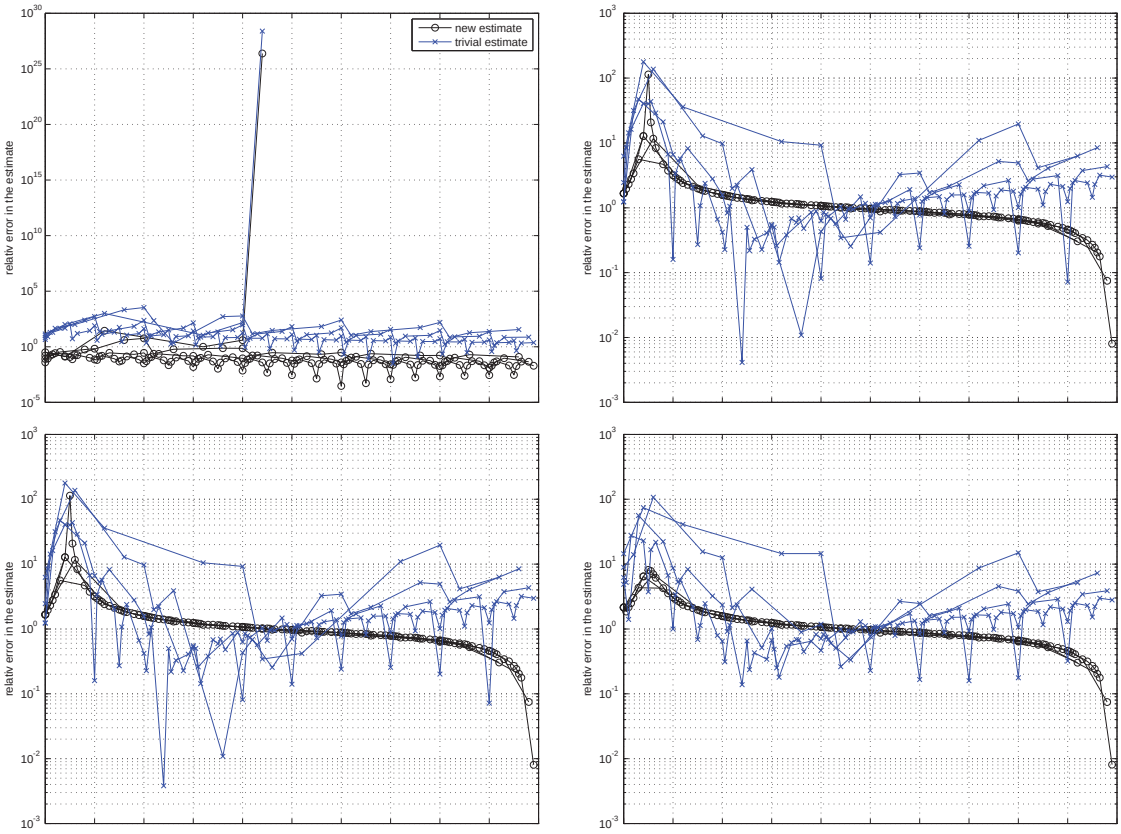


Figure 4.4: Relative errors using method of order 4 with problems A2(top–left), A3(top–right), A3–2D(bottom–left) and A3–3D(bottom–right).

low and higher order methods (in fixed order mode) are used, then the error estimates will be less accurate. This will result in bigger step sizes and thus rejections will occur. Hence, a robust ODE solver should be flexible enough to switch between a set of methods varying from lower order to higher orders. It should have the ability to choose a proper method that suits the best for that particular problem or particular interval of a problem. Now the question is how to decide whether low order methods or higher order methods can solve a particular problem efficiently. But before this, questions can be raised regarding the disadvantages of lower order methods and higher order methods.

Why not higher order methods always? This is to avoid use of higher order methods in a particular subinterval of the integration where not such a high accuracy required. To understand this, it is better to consider the case of fixed order first. Suppose we are solving an ODE using a method of order p. This problem can be solved using the methods of order less than p. The reason that we prefer order p is simply to achieve the required accuracy. However, it may be possible that in a particular problem, and in a particular sub interval of the integration, the required accuracy can be achieved by lower order methods. Now the question is, what is the benefit of using lower order methods in such sub intervals? A simple answer would be 'to decrease the number of function evaluations and thus to save time and computational effort'.

Why not lower order methods always? In contrast, if very high accuracy is required in some sub intervals of the problem, low order methods take very small step sizes and might even not work. Also higher order methods could possibly take larger step sizes where accuracy is required.

Why not fixed order? This could be considered as a combined answer to the above two questions. The variation of the order is also dependent upon the stepsize. In a variable stepsize code with fixed order, the desire is to keep the h-graph smooth. Definitely, there is an optimal set of step sizes for an ODE problem, although it may not be possible to follow it due to the fixed order method. Though, variable order implementation can help to widen the stepsize choices and thus achieve what is optimal in terms of step sizes. Also, the step sizes are dependent upon the error estimation so the use of a variable order scheme can produce better asymptotic error estimations. Thus all we need is a variable order scheme that can optimise what is the best choice of the pair (h, p) for the next step.

How to implement variable order? The next question is how to change the order. The answer is not that straightforward. For example, if a step is accepted then we have three options about selecting the order for the next step. These are either to increase the order, decrease the order or to keep the order the same. Also a criterion is needed to select the correct one. In the existing codes, many heuristic approaches have been used which are interestingly successful. There is a large gap between practice and analysis in this area. This question can be considered as an optimisation problem, though a suitable optimisation method is still required for this problem.

Is an optimisation method possible for selection of order? For optimising this, we need to take account of what we want to achieve. We want to achieve accuracy and efficiency (cheap cost). For accuracy, we need to bound the local error to a minimum called tolerance, and for efficiency we need to minimise function evaluations. To achieve this, the next thing is to see what is required and what is available to achieve this. In general, we have an error estimation at the end of a successful step. Depending upon this estimation, the new stepsize can be predicted using a standard stepsize controller. At this stage if this predicted stepsize is not the optimal one or close to it, there are options to switch between the methods with orders higher or lower. These may not be sophisticated enough to switch the methods with orders more than one order higher or lower as for the RADAU code [75]. In order to switch the method to a method of different order, all we need is to predict the stepsize for the new method. This requires reliable information as to what the error would have been in the completed step if the same stepsize had been used, with this new method.

Is it possible in the case of IRKS methods? For the GLMs under consideration, it is very easy to get this information reliably due to the Nordsieck form in the methods. For the error estimation for higher order methods, the work has been done by J. C. Butcher and H. Podhaisky [58] and in [56]. Using this information, we can find the predicted step sizes for the next step, not only for the method used in the completed step but also for the methods of one order higher or lower.

How these questions are dealt with in the existing codes, is presented in the following section.

4.6.1 Some existing controllers

Before discussing the new order control, a few (h, p)-controllers of some ODE solvers will be discussed that are effectively used although they require a sophisticated approach. The aim here is not to find the drawbacks of the codes. However, to emphasise the claim made in the thesis, it is useful to signify the design of a few existing codes and a few reasons of setting the design parameters. For this purpose, the Radau code by Hairer based on the Runge–Kutta methods and some existing codes based on GLMs are presented as follows.

Controller in the Radau code

This code is thoroughly analysed in [1] from a software design and (h, p)-control point of view. Here are a few of the features.

1. The Radau methods used in this code are of order $1, 3, 5, 9$ and 13. The orders are fluctuating and varying unequally. So this code may incur very high cost due to abrupt changes in the order.

2. Selection of order depends upon the convergence rate θ of the Newton method which is bounded by another number 0.8. There is no theoretical rationale behind this condition.

3. Further heuristic numbers include the restriction on stepsize not to change when the stepsize ratio `r` lies between 0.8 and 1.2.

4. For error estimation, the embedded technique is used using a lower order method although there are methods of fluctuating the order used in the code.

In spite of all these design questions, this variable stepsize variable order code works and is used for stiff problems. However, the design of this software is made on an empirical basis and demands more sophisticated approaches.

Controller in the DIM18 code

In this code, the variable order scheme depends upon the error estimation in the existing step with method of order p as well as the estimation of the error if the method of order $(p-1)$ were to be used in the same step. This is possible due to the Nordsieck form of the method. So the ratio is

$$ratio = \frac{\|est_p(x_n)\|}{\|est_{p-1}(x_n)\|}, \tag{4.30}$$

and the order changes according to the following algorithm.

Algorithm 1 p–control of DIM18

if $(ratio > \mathsf{r}_{\min})$ **and** $(p < p_{\max})$ **and** (previous step is not rejected) **then**
  increase the order by 1
else if $(ratio < \mathsf{r}_{\max})$ **and** $(p > 1)$ **then**
  decrease the order by 1
else
  do not change the order
end if

where $\mathsf{r}_{\min} = 0.9$ and $\mathsf{r}_{\max} = 1.1$ are the limits on (4.30). Also the rejection criteria is as follows:

Algorithm 2 Rejection criteria of DIM18

if $(est > 1)$ **and** previous step has been accepted **then**
  $nrej = 1$ **and** do not change the order
else
  $p = p - (nrej - 1)$
  $nrej = nrej + 1$
end if
$h_{\text{new}} = h_n * \min(\text{fac}_{\min}, fac * \mathsf{r}_{\text{opt}})$

The heuristic numbers in this case are $\text{fac}_{\min}$, fac and r_{opt} (fac is a safety factor as in Subsection 4.3.1). Again this design of order control is not supported by any theoretical background.

Controller in the thesis [86] by William Wright.

In this thesis, the order changing criteria is based on maximising the stepsize per unit of computational cost. This cost is equal to the number of stages in the case of the GLMs. So it is in fact a measure of the distance covered per stage. Thus, if the method of order p is used in the existing step then the order for the next step would be an element of $\{p-1, p, p+1\}$. The ratios (distance covered per stage) for each of the methods would be

$$\frac{h_{n+1,p-1}}{p}, \frac{h_{n+1,p}}{p+1}, \frac{h_{n+1,p+1}}{p+2},$$

respectively. A value from $\{p-1, p, p+1\}$ is chosen to be the order for the next step for which the corresponding value of the ratio is maximum. Along with this a safety factor is also used to avoid too rapid change in the order. Further restriction is applied on increasing the order. In addition, the order is not increased unless the stepsize remains the same in three consecutive successful steps.

4.6.2 An order control paradigm

From all the explanations above, it is easy to conclude that switching of order in fact depends upon the optimal selection of stepsize. Thus we can say that order control is in fact optimisation of stepsize.

To further explain this consider an example. Let the $(n-1)^{\text{th}}$ step be accepted with method of order p and with stepsize $h_{(n-1,p)}$. In variable stepsize fixed order mode, it is easy to propose the stepsize for the next step using standard schemes. Here we are discussing variable stepsize variable order mode. So we have the option to either decrease the order, increase the order or keep the order the same. Thus a variety of pairs (h, p) are available. We restrict ourselves here to choose the same method of order p or change it by at most order 1. Thus the possible choices of order for the next step are $\{p-1, p, p+1\}$. Now we need to predict three corresponding step sizes for the next step i.e. $\{h_{(n,p-1)}, h_{(n,p)}, h_{(n,p+1)}\}$. Out of these three, the middle one is the easy one and can be calculated the same way as we did in the fixed order case. The other two require some further information. We need a reliable estimator that can guide us as to what the error estimate would be at the end of the $(n-1)^{\text{th}}$ step if the same stepsize $h_{(n-1,p)}$ had been used for the method of order $(p-1)$ or $(p+1)$. For any order, lower than p, the error term is

$$E_m = C_m h^m_{(n-1,p)} y^{(m)}(x_{n-1}), \tag{4.31}$$

where $m = p, p-1, \ldots, 2$ and C_m are the error constants of the corresponding methods available in the code. Also, reliable approximations to the scaled derivatives $h^m_{(n-1,p)} y^{(m)}(x_{n-1})$ are available in the Nordsieck vector. Thus, in this case, there is no need for a separate error estimator formula or any additional function evaluations to get the approximations to the quantities in (4.31). For estimating the error with method of $(p+1)$ (or higher), we need a good estimator for the higher scaled derivative $h^m_{(n-1,p)} y^{(p+2)}(x_{n-1})$. This has been done in [58] and is explained in Section 4.5.1.

4.7 Conclusion

Our aim is to implement the GLMs with IRKS– and F–properties, in variable stepsize and variable order mode. For this we need an efficient and robust control for optimising stepsize and order. How this can be decided using optimisation methods will be addressed in the next chapter. The Lagrange multipliers can be helpful to find the optima. However, this is not possible without reliable error estimators, not only for the method used in the existing step but also (possibly) for the method of one order higher and lower. This is possible in the case of existing GLMs. Thus, along with the automatic starting procedure, variable stepsize stability and reliable error estimators, it is possible to use the IRKS methods as a kernel of a robust algorithm presented in the next chapter.

5

Software design; a multivariable optimisation approach

The ultimate aim of the theory of numerical methods for ODEs is to produce practical algorithms for the solution of initial value problems. Such algorithms should attempt to fulfil all the requirements of stability, accuracy and efficiency. The stability is a built–in property with the existing methods. Thus it remains to balance the other two quantities. There are no practical ODE solvers, that can balance these two quantities at the same time, in a sophisticated way. A control paradigm, based on the Lagrange optimisation approach, that can make a balance between the accuracy and efficiency, accordingly with our desire, is discussed in detail in this chapter. The scope of this chapter is to explain the theoretical parts of the optimisation approach that can help to understand, develop and implement the new design.

A general ODE solver consists of different techniques for different purposes. Here are a few of the techniques and the reason why these are required for developing a complete ODE software.

- For numerical integration of ODEs, a numerical method or a suite of methods is required.
- For obtaining continuous output of the solution, a suitable interpolation technique is required.
- For error estimation, some suitable approximations to the higher derivatives are required. For example, finite differences which are trivial estimators.
- For solving the implicit internal stage systems, a numerical method for nonlinear systems is required, in the case of stiff solvers. For example, the Newton method, the Krylov subspace method etc..
- For variable order mode of an ODE solver, a suitable design of the whole algorithm is required. This design is empirical in the existing solvers and there is no single ODE solver based on a theoretically studied approach.

Thus, in practice, there is no use of the theory of numerical analysis in the last requirement, i.e. the design of the whole algorithm. To overcome this, we need a sophisticated approach that can fill this gap. For this we need to first point out those features of the design of existing practical algorithm which are empirical and specific to the problem or the method. For example, the RADAU code is made for stiff problems and implicit methods are used in it. Many design features are set to deal with the stiff problems. Such features, in the (h, p)–control RADAU code, are reported in [1]. We use an optimisation approach for developing a paradigm of a variable order algorithm. One main goal of this thesis is to explore and implement the Lagrange optimisation approach for designing an algorithm for the IRKS methods with F–properties. However, the paradigm presented in this chapter is not practical for the traditional methods.

One of the serious issues arising in traditional methods is the design of the algorithms. Many existing ODE solvers are based on empirical or heuristic design features. There are many design parameters used for making very important decisions in the code. These include selection of stepsize and order for the next step, accepting or rejecting a step etc.. For example, if a step is accepted, the selection of stepsize and order for the next step can be considered as one case. In the case of rejection, the situation is not exactly the same and it may not be suitable to increase the order at rejections. This case can be considered analogous to driving an automatic gear car in which the gear does not jump to higher gear when brakes are applied. A question is how we can design an algorithm for ODEs on the basis of this hypothesis?

Another feature is the safety factor, used in the standard stepsize controller, considered to avoid too much change in the error estimates or to decrease the risk of rejection. A big change in stepsize may cause too big a change in the error estimation which will further cause too big changes in the next step sizes and hence increase the chance of rejections. For this, we use a safety factor less than but close to one. This safety factor has different values in different codes. For example, its value is 0.9 in the DIM18 code , $0.9 \times (2k_{\max} + 1)/(2k_{\max} + k)$ (where k is the number of Newton iterations) in the RADAU code. This value is 0.85 in the preliminary testing of the Euler–Lagrange theory for stiff problems in [58].

Without going into further details of existing codes, it would be valuable to first address what is required from an automatic code. In general, the tolerance (which may or may not be user supplied) is a very important guide. A stringent tolerance indicates that the user needs high accuracy whereas expenses are, at least, not the first priority. On the other hand, a crude value for tolerance indicates that high accuracy is not so important and cheap results must be obtained. How can an automatic code address these (two needs) using one single guideline? The aim is not to achieve both the goals at the same time but there should be one guideline that can be used to balance the two goals. An automatic code should be able to decide the criteria to be followed.

In an adaptive code, how could the stepsize and order be selected in each step on the basis of accuracy and expense. It was first proposed by Butcher [20] and developed further in [23] that this optimisation is a multivariate problem and can be achieved by optimising the function $E + TW$ where E represents total error and W represents total work or cost. The Lagrange multiplier T has the

role of tolerance. The significance of this function is explained in Section 5.1. The Lagrange stepsize controller is a bit relaxed in the sense that it can select the stepsize without any heuristic bounds or safety factors as are used in the case of a standard stepsize controller. The Lagrange stepsize controller is presented in Section 5.2.

An efficient adaptive controller needs an appropriate criterion for rejecting the steps that are producing too much error. Traditionally, this criterion is based on a user supplied tolerance. It is not always advantageous to reject a step on the basis of this tightened criterion. As rejections are expensive, it is necessary to reconsider the criterion and consider the possibility that it may be advantageous to accept a traditionally rejected step, the idea is presented in Subsection 5.2.1. An important decision after rejection is how to reduce the stepsize. This is presented in Subsection 5.2.2.

Our aim is to implement the GLMs with IRKS– and F–properties, in variable stepsize and variable order mode. This requires a sophisticated procedure for changing the stepsize or/and order. For this we need an efficient and robust control for optimising stepsize and order. So multivariable optimisation is a possible way to attempt this in an adaptive ODE code. How can this be decided using optimisation methods? We have chosen the Lagrange multipliers for this purpose. The Lagrange approach is reliable in switching the order as well, using the existing GLMs of consecutive orders. This is discussed in Section 5.3.

However, due to the (h, p) refinement mode, we need to adjust the stepsize for the method of one order higher or lower than the method used in the existing step. A change of method in the next step causes a change in stepsize as well. This needs a sophisticated procedure for the scale and modify technique. This is due to change of β–terms in the Nordsieck vector. The approach is presented in the form of an algorithm in Subsection 5.3.2.

5.1 Lagrange multiplier controller

In the theory of optimisations, there are two ways to solve constrained optimisation problems based on multiple objective functions. One is a substitution method in which a problem in multiple variables is converted into a single variable problem by a simple substitution method. The other method is to use the idea of Lagrange multipliers. This method is used here not only for variable stepsize implementation but also for designing an automatic algorithm for controlling the stepsize and order in an adaptive ODE solver.

Basically, the Lagrange multiplier approach is used in *constrained optimisation*. In this approach, a constrained optimisation problem is first converted into an unconstrained problem. If the problem, in general, is to

$$\begin{aligned} &\text{minimize,} && y = f(x_1, x_2), \\ &\text{subject to} && g(x_1, x_2) \leq k, \end{aligned}$$

then using a Lagrange multiplier T, the problem would be to

$$\text{optimize,} \qquad z = f(x_1, x_2) + Tg(x_1, x_2), \tag{5.1}$$

by obtaining critical points from $\nabla f(x_1, x_2) = -T\nabla g(x_1, x_2)$. The augmented function (5.1) is known as a Lagrange function. The Lagrange multiplier T gives additional information about the problem. This value is very appropriate for the present situation and is discussed later in this section. Before this it is important to find the objective function which is to be optimised. Also it is equally important to decide about the constraint. A straightforward idea of calculus of variations is used for this purpose. However, it is necessary to mention the practical design of this approach which is presented later in this chapter.

5.1.1 Lagrange function; $(E + TW)$ as cost function

The goal is to optimise stepsize h and order p for each step. The approach considered here, proposed by J. C. Butcher [32], is based on the Lagrange multipliers. For this, consider an initial value problem over the interval $[a, b]$. The method starts from the initial value $y(a)$ and the goal is to approximate the solution of the problem at the end point of the interval i.e. $y(b)$. In a variable stepsize code, the stepsize for the next step is required. This is based on the error estimation in the existing step. It is possible to estimate the truncation error sufficiently accurately when a step is taken from x to $x + h$. If p is the order of the method, the norm of the error can be written as $C(x)h^{p+1}$, where $C(x)$ is the norm of C (the error constant of the method) times the approximate value of $y^{(p+1)}(x)$. So, if $H(x)$ is the function for calculating sufficiently small values of h for each step and $W(H)$ represents the work then an approximation of the truncation error and total work from a to b is

$$E(H) = \int_a^b C(x)H(x)^p \, dx, \tag{5.2}$$

$$W(H) = \int_a^b H(x)^{-1} \, dx. \tag{5.3}$$

An efficient code must produce the lowest values of the above two integrals. So, an ideal case would be to optimise the value of stepsize h after each step that minimises the value of both the integrals. However, it is not possible to minimise both at the same time, but from the existing theory of optimisation, it is possible to minimise one quantity by imposing some constraint on the other quantity. Thus, this problem can be viewed as the following dual problem:

primal problem. Impose limit on the cost W (a constraint) and minimise the error E

dual problem. Impose limit on the error E (a constraint) and minimise the work W

These two problems are somehow diametrically opposed. In the primal problem, minimising E means that small values of h are needed. In the dual problem, minimising cost W means that larger values of h are needed. The central role is played by the parameter tol that determines the balance of focus between these two. So we need to devise a design that can work for both situations. How can these two problems be fitted into one frame? This was considered by Butcher [23] in 1990. This idea is to minimise $(error + T \times work)$ per unit step i.e.

$$\Phi(H) = \frac{E(H) + TW(H)}{H}, \tag{5.4}$$

The function $E(H) + TW(H)$, where T is a Lagrange multiplier. It is shown later in this section that T is a proportion of the tolerance, which is quite reasonable because in an ODE solver

- when T is small in (5.4), then finding the minimum of $\Phi(H)$ means that high accuracy is required from the solver and thus error should be minimised. Or we can say, within limited resources i.e. cost (constraint), best results (least value of global truncation error) should be achieved.

- when T is large in (5.4), then finding the minimum of $\Phi(H)$ means that high efficiency is required from the solver and therefore work should be minimised. In other words, to achieve the required accuracy (constraint), cost should be minimised.

Thus in both cases, the Lagrange objective function is to be minimised. The very next requirement for implementation is to find the value of the Lagrange multiplier T. It is necessary to mention as to how this method would be helpful to design an algorithm for controlling stepsize as well as order. Usually when a real world optimisation problem is solved using any method, some critical values of the objective function are obtained first and then, using those critical values, optima of the objective function are obtained. In the present case, the implementation of the method is slightly different. It is necessary to implement this method after each step so that the optimal value of h (and p) can be obtained for the next step, while the Lagrange multiplier remains the same throughout the interval. For this it is necessary to *a priori* decide what this Lagrange multiplier represents for the existing optimisation problem in an abstract way.

5.1.2 Lagrange multiplier: a proportion of tolerance

As discussed before, in this optimisation method, the values of the Lagrange multiplier and the optimal values can be obtained from partial derivatives of the objective function. The work done is considered to be equal to the number of function evaluations, and, as our methods hold property-F, so $(p+1)$ function evaluations are calculated in each step. Thus, the work done per unit step is constant. Therefore the objective function (5.4) becomes

$$\Phi(H) = C(x)H^p(x) + \frac{T(p+1)}{H(x)}. \tag{5.5}$$

So to calculate the value of T,

$$\begin{aligned} \frac{\partial \Phi}{\partial H} &= 0, \\ pC(x)H^{p-1}(x) - \frac{T(p+1)}{H^2(x)} &= 0, \\ C(x)H^{p+1}(x) &= T\frac{(p+1)}{p}. \end{aligned} \tag{5.6}$$

Thus, the value of truncation error $C(x)H^{p+1}(x)$ should be kept close to some proportion of T. Using traditional methods, this value is kept close to a constant which is simply the user supplied tolerance.

Therefore, in the case of existing GLMs it should be kept close to some proportion of the tolerance i.e. $(p+1)/p$.

As T is playing the role of tolerance, it means that (5.6) can be interpreted as the truncation error should be kept close to the slightly perturbed tolerance. This can give a chance to accept those steps which are unnecessarily rejected by the traditional controllers. However, we can see later in Subsection 5.2.1, that the accept–reject criterion decides this according to what minimises the cost function.

The next step is to find the variable stepsize scheme and the rejection criteria using the Lagrange approach. This is presented in the following section.

5.2 Lagrange Stepsize control

The above explained interpretation of (5.6) can be alternatively used to find the optima. It gives the optimal stepsize formula which is

$$h_n = \left(\frac{T \frac{(p+1)}{p}}{E} \right)^{1/(p+1)} h_{n-1}. \tag{5.7}$$

The above result is the same as the classical controller in Subsection 4.3.1 with tolerance $T(p+1)/p$ so is the norm of the estimated error. As compared to the classical controller, the Lagrange controller (5.7) gives a slightly larger stepsize that can tolerate the required accuracy limit. Also, we use some safety factor in (4.4). However, this heuristic value is not required in (5.7). The difference obtained by equating these two controllers is tolerance factor $(\frac{p+1}{p})^{1/(p+1)}$. This factor obtained by the Lagrange approach is to be used for optimal cost. Along with the optimal value of stepsize for the next step, it is also possible to check which order method is the best for calculating the next step; this is discussed in Section 5.3. Before this it is discussed in the following section when it is optimal to accept or reject a step using the cost function.

5.2.1 Acceptance and rejection of steps

Criteria for accepting and rejecting a step are required in the case of the variable stepsize scheme. The standard stepsize control scheme is used for this purpose. In this scheme the criteria for accepting or rejecting a step are very simple and usually straight forward; a step is accepted if the estimated error in that step is less than or equal to the specified tolerance, otherwise it is rejected. Thus, there are only two options in the standard control.

One of the desired goals here is to implement the GLMs in variable stepsize variable order mode. In this mode, when a step is completed, there are a few decisions that the solver has to make, e.g. accept the step, reject the step, increase the order, and decrease the order. Out of these, rejection is a very serious action. When an error occurs or something non-smooth, we should avoid (or decrease) the chance of heading towards rejection. Perhaps, increasing the order is not suitable in such a situation. Furthermore, we do not increase the order unless and until everything is going smoothly and well, only

the other three options should be considered up to that point.

Here, we are discussing the Lagrange controller point of view given by J. C. Butcher [32], in the case of GLMs with IRKS– and F–properties. According to this book, while using Lagrange theory for (h, p)-control, the criteria for accepting or rejecting a step are based on the stepsize ratio r (or on the ratio (E/T)). The aim here is, to avoid too early rejections and look for alternatives. For this, first, remove the safety factor. Accept the step when the ratio of error to the tolerance (E/T) is not 'too great'. We need to decide what is 'too great'. The criterion is based on minimising the rate:

$$\text{rate of increase of obj} = \frac{(E + TW)}{h}. \tag{5.8}$$

Now we need to calculate the value of the above rate in both cases; acceptance and rejection of this step, and then compare both to get the required criteria. Assume that a step with stepsize h has been carried out and $W = (p + 1)$ for the GLMs. Also, from Lagrange theory

$$E = \left(\frac{p+1}{p}\right) T. \tag{5.9}$$

Case I. First, suppose the step is **accepted** and the actual error is slightly different from what is expected i.e.

$$E = K \left(\frac{p+1}{p}\right) T.$$

Then from (5.7) the stepsize ratio would be,

$$\begin{aligned} \mathsf{r} &= \left(\frac{\left(\frac{p+1}{p}\right) T}{E}\right)^{\frac{1}{p+1}}, \\ &= K^{-\frac{1}{p+1}}, \end{aligned}$$

as the right hand side of (5.9) plays the role of tolerance. Also, in this case, $W = 0$ is considered. However, $(p+1)$ function evaluations have been performed here. We are considering the rate of change of the objective function and these $(p + 1)$ function evaluations are performed prior to deciding the acceptance or rejection of the step, hence would be considered constant and so can be ignored. Thus the expenditure (linear combination of error and work) would be increased by

$$\mathsf{r}^{-(p+1)} \left(\frac{p+1}{p}\right) T$$

and

$$\text{rate of increase of obj} = \frac{\mathsf{r}^{-(p+1)}}{h} \left(\frac{p+1}{p}\right) T \tag{5.10}$$

Case II. Let the step be **rejected**. The new proposed stepsize is $\mathsf{r}h$. In covering the distance $\mathsf{r}h$, the

p	$(p+1)^{-1/p}$	$(p+1)^{(p+1)/p}$
1	0.5	4.0
2	0.57735027	5.1961524
3	0.62996052	6.3496042
4	0.6687403	7.4767439
5	0.69882712	8.5858145
6	0.72302003	9.6816129
7	0.74299714	10.767202
8	0.75983569	11.844666
9	0.77426368	12.915497
10	0.78679344	13.980798

Table 5.1: Optimal values for the stepsize ratio and the ratio of estimated error to the tolerance for accepting a step for each order p.

rate of work is increased by $(p+1)/\mathsf{r}$. So,

$$\begin{aligned} \text{obj} &= \left(\frac{p+1}{p}\right) T + (p+1)T, \\ &= \frac{(p+1)^2}{p} T, \end{aligned}$$

and

$$\text{rate of increase of obj} = \frac{(p+1)^2}{p(\mathsf{r}h)} T \tag{5.11}$$

To find the criteria for acceptance or rejection of the step, (5.10) and (5.11) are compared and it is concluded that the step should be accepted if

$$\frac{\mathsf{r}^{-(p+1)}}{h} \left(\frac{p+1}{p}\right) T \leq \frac{(p+1)^2}{p(\mathsf{r}h)} T, \tag{5.12}$$

$$\mathsf{r} \geq (p+1)^{-\frac{1}{p}}. \tag{5.13}$$

which depends on the stepsize ratio r (and the order p). Alternatively, this criteria can be expressed in terms of the ratio of estimated error to the tolerance. For this, consider the above relation again,

$$\begin{aligned} \mathsf{r}^{-1} &\leq (p+1)^{\frac{1}{p}}, \\ \mathsf{r}^{-(p+1)} &\leq (p+1)^{\frac{p+1}{p}}, \\ \frac{E}{\left(\frac{p+1}{p}\right) T} = \mathsf{r}^{-(p+1)} &\leq (p+1)^{\frac{p+1}{p}}. \end{aligned} \tag{5.14}$$

Thus, the step should be accepted if this ratio does not exceed $(p+1)^{(p+1)/p}$. The values of $(p+1)^{-1/p}$ and $(p+1)^{(p+1)/p}$ are given in Table 5.1.

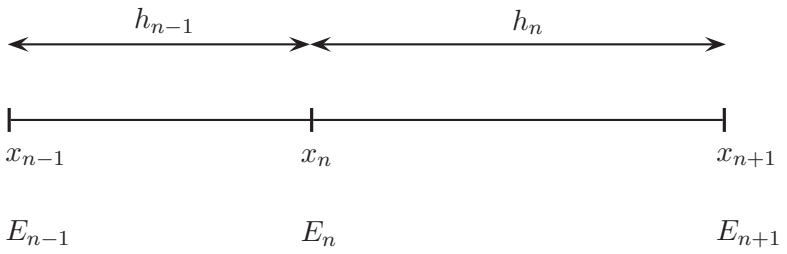


Figure 5.1: (Case when solution at x_n is accepted.)

5.2.2 Criteria for reducing the stepsize after rejection.

One of the natural/classical ways of reducing the stepsize, after rejection, is performed by simply reducing the rejected stepsize by a fixed factor (for example, it is reduced to 50% in the Matlab ODE codes and DIM18, and it is reduced to 10% in the Radau code). In principle, this (new) reduction of stepsize is based on the standard stepsize control but with the most recently available error estimation and stepsize. In order to explain this way of reducing the stepsize after rejection, it is better to see what we do when a step is accepted. Suppose a step is accepted from x_{n-1} to x_n (see Figure 5.1) using stepsize h_{n-1}. Then, using a standard stepsize controller, we estimate the stepsize for the next step using the value of error estimation E_n at x_n i.e.

$$h_n = \left(\frac{E_n}{T}\right)^{-\frac{1}{p+1}} \times h_{n-1}. \tag{5.15}$$

Now consider the case of rejection at x_n (see Figure 5.2). If the step is rejected, it means that the estimated error E_n is too large and we need to somehow reduce h_{n-1} to get a new value of stepsize h^*_{n-1} for the same step. Before this rejected step, h_{n-1} has been calculated depending upon E_{n-1}. This time, we have to again use the same formula (5.15) with the recent values of h_{n-1} and E_n i.e.

$$h^*_{n-1} = \left(\frac{E_n}{T}\right)^{-\frac{1}{p+1}} \times h_{n-1}. \tag{5.16}$$

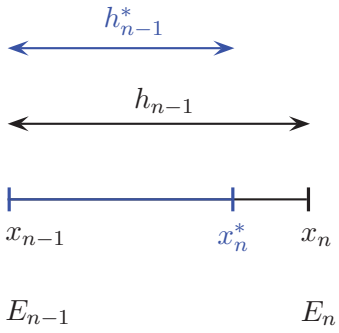


Figure 5.2: (Case when solution at x_n is rejected.)

5.3 Variable order algorithm

An adaptive code must have a reliable controller for efficiently changing the stepsize from step to step. Along with this, a code would become robust by the possible addition of variable order design. Unlike the existing heuristic designs of variable order ODE solvers, it is possible to rationally choose the optimal method (value of p) using the objective function (5.4) (or (5.5)). For this the use of (5.6) in (5.5) gives

$$\begin{aligned}\Phi(H) =& \frac{T\frac{(p+1)}{p}}{H(x)} + \frac{T(p+1)}{H(x)},\\ =& \frac{T(p+1)^2}{pH(x)},\end{aligned} \tag{5.17}$$

which is to be minimised. In (5.17), T is constant for all values of p so can be ignored. The alternative form to that of (5.17), which can be used, is to maximise the reciprocal of it.

5.3.1 Stepsize and order control

In our algorithm, we have to calculate the value of the function (5.4) not only with the order p of the existing step but also for the methods of order $p-1$ and $p+1$, available in the suite of methods of the code. This is only possible if we are able to find what the stepsize(s) would be in the next step if the method of order $p+1$ or $p-1$ is used. If the order is kept the same then the stepsize for the next step can be simply obtained using (5.7). This requires a reliable error estimation which is possible in the case of the existing GLMs. So, if the order is to be changed then the estimates of errors with methods of order $p+1$ and $p-1$ are needed. The error estimate for the method of order $p-1$ is of the order p i.e. $C_p h^p y^{(p)}(x_n)$. This quantity is simply the product of the error coefficient of the method of order $p-1$ and the last component of the Nordsieck vector. Similarly, the error estimator for the method of order $p+1$ is the product of the error coefficient C_{p+1} and $h^{p+2}y^{(p+2)}(x_n)$. A reliable estimator of this higher derivative is discussed in Subsection 4.5.1. Therefore, it is possible to obtain the step sizes for the next step with methods of order $p-1$, p and $p+1$. This is one of the advantages in this implementation due to the design of our methods. So we have to find a pair (h,p) which minimises the function (5.4) for the next step. This is presented in Algorithm 5.3.1.

Let $\mathrm{S} = \{1, 2, \ldots, p_{\max}\}$ be the set of orders of the methods used in the code and p be the order of the method used in the existing step and 'est=[est$_i$]' be the vector of available error estimates (scaled by tolerance) for $i \in \{p-1, p, p+1\} \subset \mathrm{S}$. Then Algorithm 5.3.1 can be used to implement the Lagrange controller to find the optimal stepsize and order for the next step and the accept–reject criterion. In this algorithm, *holdp* is used to hold the order p of the method used in the existing step from increasing to $p+1$ for a certain number of consecutive accepted steps. It somehow measures the strong evidence of increasing the order so that order–oscillations can be avoided. This is further explained in Chapter 6 with experimental evidence.

Algorithm 5.3.1: LAGRANGECONTROLLER (h, p, est, T)

comment: First, select the possible order(s), for the next step, from S.

if $\left(\text{est}_p \leq (p+1)^{\frac{p+1}{p}}\right)$

then
- **if** (psteps $\geq holdp$)
  - **then** pset $\leftarrow \{p-1, p, p+1\} \subset$ S
- **if** (psteps $\leq holdp$)
  - **then** pset $\leftarrow \{p\}$

else pset $\leftarrow \{p-1, p\} \subset$ S

comment: Now, for each possible order, calculate the cost.

for each $i \in$ pset

do
- $\mathsf{r}_i \leftarrow \min\left(2, \max\left(0.5, (\text{est}_i)^{-\frac{1}{i+1}}\right)\right)$
- $h_i \leftarrow \mathsf{r}_i h$
- $\text{obj}_i \leftarrow \left(\frac{T(i+1)^2}{ih_i}\right)$

comment: Now, select the order and corresponding stepsize that minimises the cost.

$j \leftarrow \text{index}\left(\min\left(\text{obj}_i\right)\right)$

$p_{\text{new}} \leftarrow j$

$h_{\text{new}} \leftarrow h_j$

if ($j \neq p$ **or** step is rejected)

then psteps $\leftarrow 0$

else psteps $\leftarrow$ psteps $+ 1$

return $(h_{\text{new}}, p_{\text{new}})$

As this algorithm is variable stepsize as well, it is necessary to use a sophisticated version of the scale and modify technique. This is presented in the following Subsection.

5.3.2 Algorithm for the scale and modify technique in variable order mode

At the end of a successful calculation of a step (acceptable), the next task is to select the stepsize and order. In both cases, stepsize varies on the basis of the stepsize control scheme. Due to the variable stepsize, it requires a scale and modify process as well.

On the basis of Lagrange theory, we have to select the new pair (h, p). A reasoned change of the order depends upon the information available. In our design of methods, it is possible to have enough information to increase or decrease the order by 1. This is a better option than the approach used in the RADAU code. In that code, methods of order $1, 3, 5, 9, 13$ are used and order varies unequally. In our design of code, if p is the order of methods used in the existing step then we have enough information available to increase or decrease the order by 1. This causes a smooth change in the order

and hence produces asymptotically correct solutions and error estimation. Consequently, it helps to produce better interpolation. Thus, the possible choices of order are $p-1, p, p+1$ with corresponding β vectors $\beta_{p-1}, \beta_p, \beta_{p+1}$. If the order of a method changes, the stepsize for the step also changes. So in variable order mode, it is necessary to make a slight change in the scale and modify technique. First we will see a few steps of this technique:

$$\begin{aligned}
\textbf{Step1 (scaling):} \quad & y^{[n]} = D(\mathsf{r})y^{[n]} \\
\textbf{Step2 (removing):} \quad & y^{[n]} = D(\mathsf{r})y^{[n]} + D(\mathsf{r})\beta h^{p+1}y_n^{(p+1)} \\
\textbf{Step3 (adding):} \quad & y^{[n]} = D(\mathsf{r})y^{[n]} + D(\mathsf{r})\beta h^{p+1}y_n^{(p+1)} - \beta(\mathsf{r}h)^{p+1}y_n^{(p+1)}, \\
\textbf{(scale and modify):} \quad & = D(\mathsf{r})y^{[n]} + (D(\mathsf{r}) - \mathsf{r}^{p+1}I)\beta h^{p+1}y_n^{(p+1)}. \qquad (5.18)
\end{aligned}$$

Usually, instead of doing this process in three steps, only (5.18) is used during implementation. However, in variable order mode we need to make some small changes as presented in algorithm 5.4.1.

5.4 Design of software for ODEs using GLMs

Generally in mathematical software there must be options for the user to either use all the default values, or some subset of them, or perhaps none of them. Thus when a solver is initiated, all the options must be initialised and the user should choose either the default ones or provide their own based on available information.

5.4.1 Generic functions

There are many ODE solvers in various programming languages including MATLAB, FORTRAN, C, etc.. It is easy to implement the numerical methods for ODEs in MATLAB. Here, the GLMs are implemented and tested in MATLAB. Most of the ODE solvers consist of a single large file or a couple of files (functions). Another approach is to define each task as a separate function with a generic name. In this way, it would be easy to call a function or functions when required. For example, the scaling of various quantities is required many times, and at many places, during the execution of an ODE code.

5.4.2 (h,p)-algorithm

We try to avoid heuristic numbers as much as we can, although there are a few such numbers in the design of this algorithm which are still unavoidable. For example, the upper and lower limits on the stepsize ratio denoted by $\mathsf{r}_{\min}$ and $\mathsf{r}_{\max}$. These numbers are necessary because these are related to error control. We are using asymptotic error estimators so we cannot bear values of r too different from 1. Also, because of roundoff-errors, we cannot use too small a value for the stepsize. Another reason is that we are aiming to use our methods to track the behaviour of discontinuities in a problem. For example, the square–path problem may not behave well if we use a value other than $\mathsf{r}_{\min} = 0.5$. This square–path problem is the ultimate test of our code. The setting $\mathsf{r}_{\max} = 2$ can be helpful for producing smooth behaviour of the estimators.

Algorithm 5.4.1: ScaleAndModify $(p_{\text{new}}, \mathsf{r}, y^{[n]}, h^{p+1}y_n^{(p+1)})$

$p \leftarrow \text{length}(y^{[n]})$
if $(p_{\text{new}} = p + 1)$
then {
Do scaling and removing
$y^{[n]} \leftarrow D(\mathsf{r})y^{[n]} + D(\mathsf{r})\beta h^{p+1}y_n^{(p+1)}$
Adjust the length of Nordsieck vector
i.e. add the $h^{p+1}y^{(p+1)}$ term as last component of $y^{[n]}$

else if $(p_{\text{new}} = p - 1)$
then {
Do scaling and removing
$y^{[n]} \leftarrow D(\mathsf{r})y^{[n]} + D(\mathsf{r})\beta h^{p+1}y_n^{(p+1)}$
Adjust the length of Nordsieck vector
i.e. truncate the last component of $y^{[n]}$ i.e. $h^p y_n^{(p)}$

else {
Do usual scaling and modifying:
$y^{[n]} \leftarrow D(\mathsf{r})y^{[n]} + (D(\mathsf{r}) - \mathsf{r}^{p+1}I)\beta h^{p+1}y_n^{(p+1)}$
return $(y^{[n]})$

5.5 Conclusion

The purpose of this chapter is to describe and analyse the development of a variable stepsize variable order algorithm for ODE solvers. It is developed with the design that fits all the necessary implementation techniques presented in the previous chapter and avoids several heuristics including variable order control and safety factor used in the traditional stepsize controllers. The order selection criteria are based on the optimisation function instead of the empirical criteria used in the existing controllers, a few of which are presented in Subection 4.6.1. It also gives a new stepsize controller (tested in the next chapter) which gives a better control to balance the work and accuracy of the results. Hence it gives a solid frame to design such a numerical algorithm for ODEs based on IRKS methods which is robust and efficient as well as sophisticated.

6
Numerical experiments

Comparison of numerical results using various numerical methods or/and different techniques requires a set of fair criteria. These different techniques involve many design questions as explained in Chapters 4 and 5. The experiments recorded in this chapter are performed using fairly available competitive techniques. These experiments are performed to verify that variable stepsize variable order implementation of IRKS methods with the F–property in the Lagrange design of optimisation approach is a step towards considering GLMs and the Lagrange design as a kernel and the structure of a competitive code, respectively.

6.1 Framework

The problem set used for these experiments consists of some non–stiff problems from the DETEST set of Hull et al. in [77], a new discontinuous problem named the square path problem and some mildly stiff problems. This set is given at the end of this thesis in the Appendix. This is a preliminary stage and testing of the Lagrange approach. So, all the numerical results are obtained by implementing the IRKS methods in MATLAB only.

The first set of experiments compares the classical methods with equally weighted competitors in the family of IRKS methods. These results verify the order of the methods with fixed order and are presented in Section 6.2.

The second set of experiments includes the verification of the (Lagrange) variable stepsize implementation of the IRKS methods with the F–property using the Lagrange theory. For variable stepsize, an automatic initial stepsize scheme is necessary. A modification of such a scheme given in [69], is also tested and reported. A preliminary comparison of fixed order of the Lagrange stepsize and its rejection criteria, tests these against standard techniques. This is presented at the end of Section 6.3.

p	$\left(\frac{p+1}{p}\right)$	n	$\overline{n} = \left(\frac{p+1}{p}\right) n$
1	$\frac{2}{1}$	60	120
		120	240
		240	480
2	$\frac{3}{2}$	60	90
		120	180
		240	320
3	$\frac{4}{3}$	60	80
		120	160
		240	320
4	$\frac{5}{4}$	60	75
		120	150
		240	300

Table 6.1: Number of steps taken for each of the IRKS and Runge–Kutta methods

The third and main set of experiments comprises of the implementation of the IRKS methods with the whole Lagrange design, i.e. in variable stepsize variable order manner. This set includes the performance of the solution components in each step, along with the step sizes and the order of the methods. A few geometric problems are also tested and their phase portraits are presented in Subsection 6.4.2. The SQUARE problem is also tested to check the robustness of the code. Finally, the behaviour of the Lagrange design is also reported here by comparing the (h, p)–sequences obtained using numerical results, with the ideal and optimal sequences.

6.2 Experiments with fixed stepsize

In the first set of experiments, Runge–Kutta methods and PECE pairs are compared with IRKS methods with the F–property. These methods are given in Chapter 2. The experiments in this section are performed with **fixed stepsize**.

6.2.1 GLMs vs Runge–Kutta methods

For fair comparison between IRKS and Runge–Kutta methods, we need to take care of a few quantities. One quantity is the rate of doing work. This quantity is measured in terms of function evaluations. For example, there are $(p+1)$ function evaluations in each step of a GLM whereas in the case of Runge–Kutta methods, there are p function evaluations in each step, where p is the order of the method. For fair comparison, we need this quantity to be equal for each class with equal intervals of integration. For this, let h be the stepsize used in a GLM of order p then

$$\text{rate of producing error } = \epsilon h^p y^{(p+1)}, \tag{6.1}$$

$$\text{rate of doing work} = \frac{p+1}{h}, \tag{6.2}$$

p	$\overline{\epsilon} = \frac{1}{(p+1)!}$	$\epsilon = \overline{\epsilon}\left(\frac{p}{p+1}\right)^p$
1	$\frac{1}{2}$	$\frac{1}{4}$
2	$\frac{1}{6}$	$\frac{2}{27}$
3	$\frac{1}{24}$	$\frac{9}{512}$
4	$\frac{1}{120}$	$\frac{32}{9375}$

Table 6.2: Error coefficients for each of the IRKS and Runge–Kutta methods

whereas in the case of Runge–Kutta methods,

$$\text{rate of producing error } = \overline{\epsilon}\overline{h}^p y^{(p+1)}, \tag{6.3}$$

$$\text{rate of doing work} = \frac{p}{\overline{h}}. \tag{6.4}$$

Stepsize and number of steps. Equating (6.2) and (6.4),

$$\overline{h} = \left(\frac{p}{p+1}\right) h, \tag{6.5}$$

which is the stepsize to be taken in the case of Runge–Kutta methods. So for the Runge–Kutta methods, we take a smaller stepsize. Thus if n is the number of steps taken with fixed stepsize over the interval of length $l = (x_{end} - x_0)$, in the case of GLMs then

$$n = \frac{l}{h},$$

and the corresponding number of steps $\overline{n}$ in the Runge–Kutta methods of same order is

$$\begin{aligned} \overline{n} &= \frac{l}{\overline{h}}, \\ &= \left(\frac{p+1}{p}\right)\frac{l}{h}, \\ &= \left(\frac{p+1}{p}\right) n. \end{aligned}$$

Table 6.1 gives the number of steps to be taken for each class for a fair comparison.

Error coefficients. Similarly, the rate of producing error must be the same. For this, use (6.5) to combine (6.1) and (6.3), and we obtain

$$\epsilon = \overline{\epsilon}\left(\frac{p}{p+1}\right)^p.$$

As, in the case of GLMs, the error coefficient is a free parameter, and we select a set of GLMs with an error coefficient that satisfies the above relation. For the Runge–Kutta methods described in Chapter 2, we will take $\overline{\epsilon} = \frac{1}{(p+1)!}$. Table 6.2 gives the values of error coefficients for a fair comparison. Thus, using the stepsize in (6.5) for the Runge–Kutta methods and number of steps given in Table 6.1, the

p	$\left(\frac{p+1}{2}\right)$	n	$\widetilde{n} = \left(\frac{p+1}{2}\right) n$
1	$\frac{2}{2}$	60	60
		120	120
		240	240
2	$\frac{3}{2}$	60	90
		120	180
		240	320
3	$\frac{4}{2}$	60	120
		120	240
		240	480
4	$\frac{5}{2}$	60	150
		120	300
		240	600

Table 6.3: Number of steps taken for each of the IRKS and PECE pairs

number of function evaluations are equal for comparing the GLMs with Runge–Kutta methods.

6.2.2 GLMs vs PECE pairs

There are $(p+1)$ function evaluations in each step of a GLM whereas in the case of PECE pairs, there are 2 function evaluations in each step. For a fair comparison, we need this quantity to be equal for each class. Following (6.1) and (6.2), in the case of PECE pairs,

$$\text{rate of producing error } = \widetilde{\epsilon}\widetilde{h^p}y^{(p+1)}, \tag{6.6}$$

$$\text{rate of doing work} = \frac{2}{\widetilde{h}}, \tag{6.7}$$

Stepsize and number of steps. Comparing (6.2) and (6.7),

$$\widetilde{h} = \left(\frac{2}{p+1}\right) h, \tag{6.8}$$

which is the stepsize to be taken in the case of PECE pairs. So in this case, we take a smaller stepsize. So, if n is the number of steps taken with fixed stepsize over the interval of length $l = (x_{end} - x_0)$, in the case of GLMs then

$$n = \frac{l}{h},$$

and then the corresponding number of steps $\widetilde{n}$ in PECE pairs of the same order is

$$\begin{aligned} \widetilde{n} = \frac{l}{\widetilde{h}} &= \left(\frac{p+1}{2}\right) \frac{l}{h}, \\ &= \left(\frac{p+1}{2}\right) n. \end{aligned}$$

p	$\widetilde{\epsilon}$	$\epsilon = \widetilde{\epsilon}\left(\frac{2}{p+1}\right)^p$
1	$\frac{1}{2}$	$\frac{1}{2}$
2	$-\frac{1}{12}$	$-\frac{1}{27}$
3	$\frac{1}{24}$	$\frac{1}{192}$
4	$-\frac{19}{720}$	$-\frac{19}{28125}$

Table 6.4: Error coefficients for each of the IRKS and PECE pairs

Table 6.3 gives the number of steps to be taken for each class for a fair comparison.

Error coefficients. Similarly, the rate of producing error must be same. For this use (6.8) to combine (6.1) and (6.6), and we obtain

$$\epsilon = \widetilde{\epsilon}\left(\frac{2}{p+1}\right)^p.$$

As, in the case of GLMs, the error coefficient is a free parameter, and we select a set of GLMs with an error coefficient that satisfies the above relation. For the PECE pairs, described in Chapter 2, and the competitive GLMs, Table 6.4 gives the values of error coefficients for a fair comparison.

Using the above tables of error coefficients for GLMs ($p = 2, 3, 4$), two new sets of GLMs are constructed. Using stepsize $h = 60 \times 2^{-t}$, where $t = 1, 2, \ldots, 8$, these problems are solved and the exact error at x_{end} are plotted for each value of h, on the log–log scale. Then these values of h are used to obtained competitive step sizes for Runge–Kutta methods and PECE pairs, using (6.5) and (6.8), respectively.

For these two classes, the step sizes are smaller as compared to the step sizes for GLMs. Due to this, there is a shift in the graphs of these two classes when they are drawn against GLMs. To remove this, the plots are shifted by $-\log(\frac{p}{p+1})^p$ in the case of Runge–Kutta methods and $-\log(\frac{2}{p+1})^p$ in the case of PECEs.

For comparison of the IRKS methods and the Runge–Kutta methods, the results are calculated and reported in Figures 6.1 to 6.4 for methods of order 1 to 4 on the problems A1, A3, A3-2D, A3-3D, PR(i), PR(ii), C1 and C2. In Figure 6.1, the four plots in the first column of the plots, show the graphs for methods of order 1 to 4 (top to bottom) on problem A1. Similarly, the other columns in Figures 6.1 to 6.4, present the rest of the above mentioned problems. On the basis of this fair criteria of comparison, we can see in these figures that the IRKS methods perform equally well as compared with the classical Runge–Kutta methods with fixed stepsize.

For comparison of the IRKS methods and the PECE pairs, the results are calculated and reported in Figures 6.5 and 6.6 for methods of order 1 to 4 on the same set of problems. In Figure 6.5, the three plots in the first row of the plots, show the graphs for methods of order 2 to 4 (left to right) on problem A1. Similarly, the other rows in Figures 6.5 and 6.6 present each of the rest of the problems. Again, this comparison shows that the IRKS methods work equally well when compared with the PECE pairs with fixed stepsize.

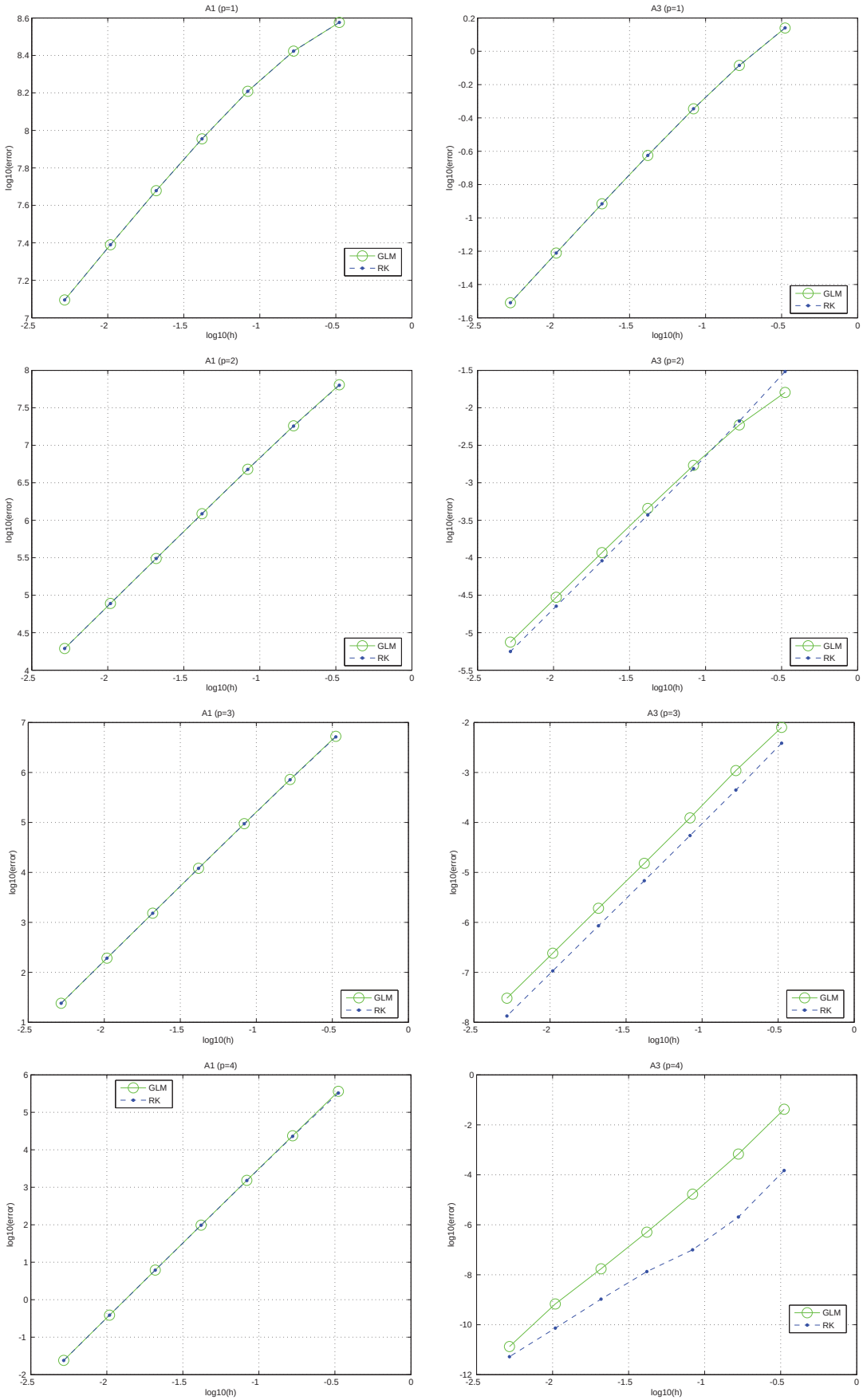


Figure 6.1: Global error (fixed stepsize) for the problems A1 and A3 (left to right) using the IRKS and Runge–Kutta methods of order one to four (top to bottom).

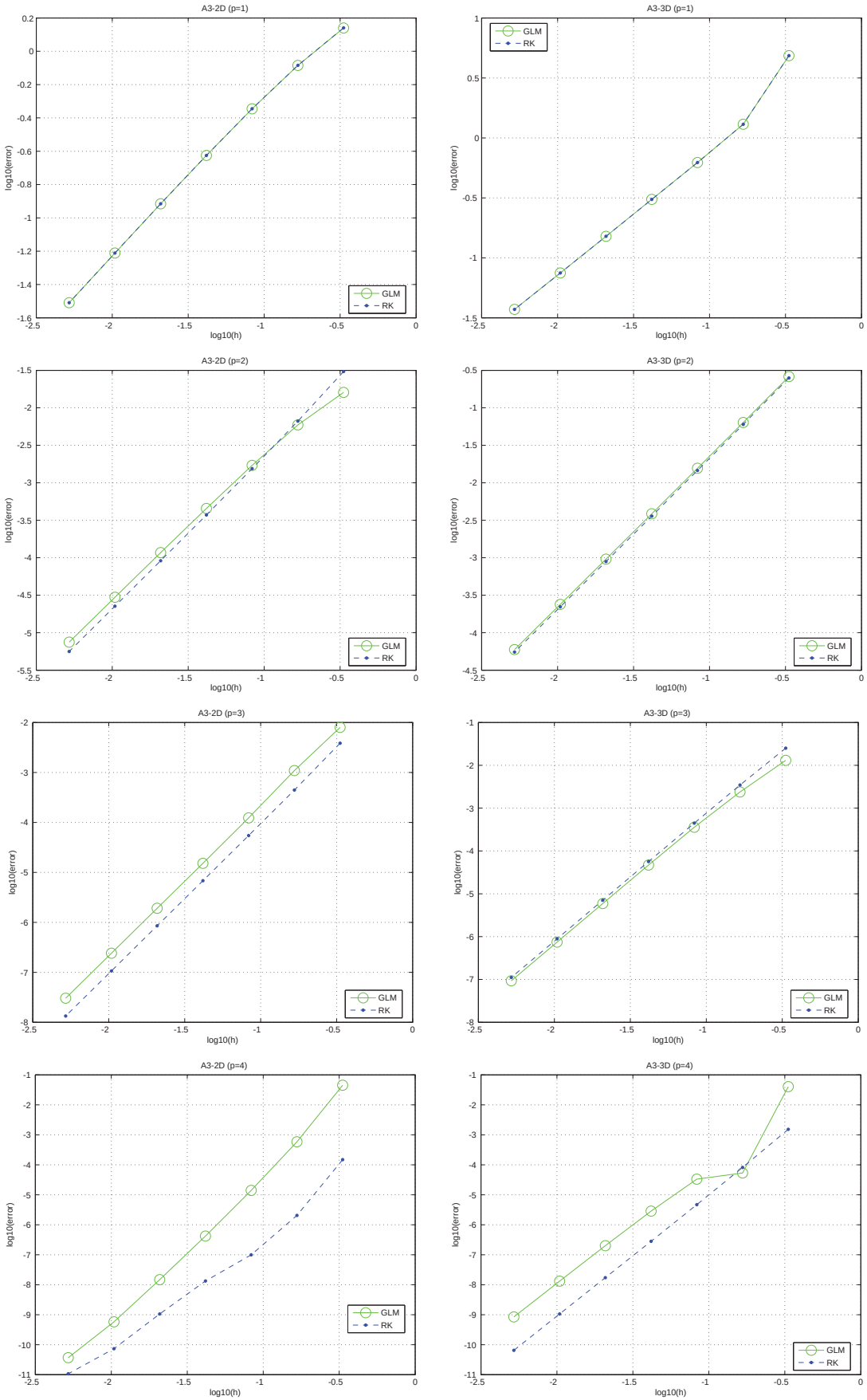


Figure 6.2: Global error (fixed stepsize) for the problems A3–2D and A3–3D (left to right) using the IRKS and Runge–Kutta methods of order one to four (top to bottom).

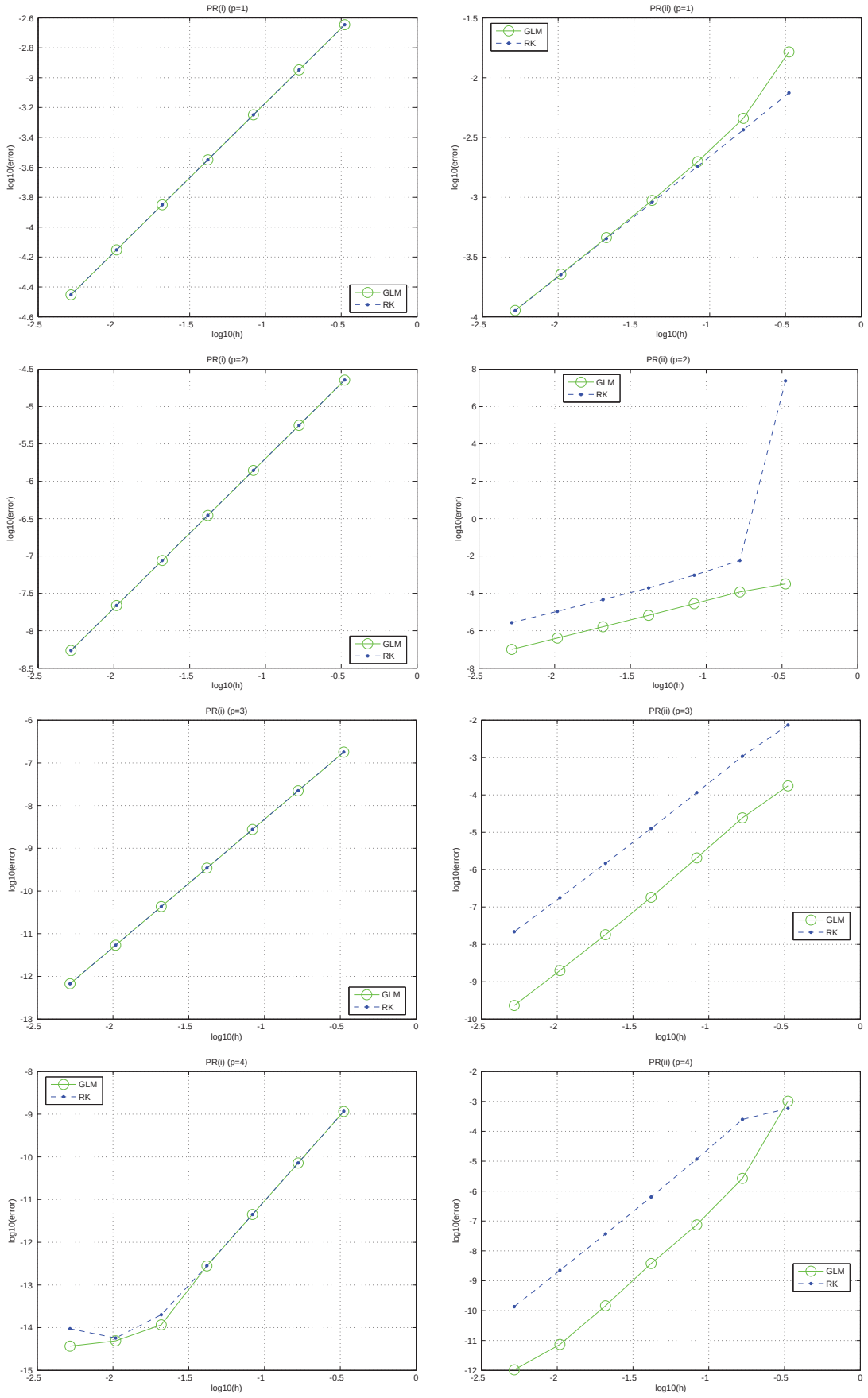


Figure 6.3: Global error (fixed stepsize) for the mildly stiff problems PR(i) and PR(ii) (left to right) using the IRKS and Runge–Kutta methods of order one to four (top to bottom).

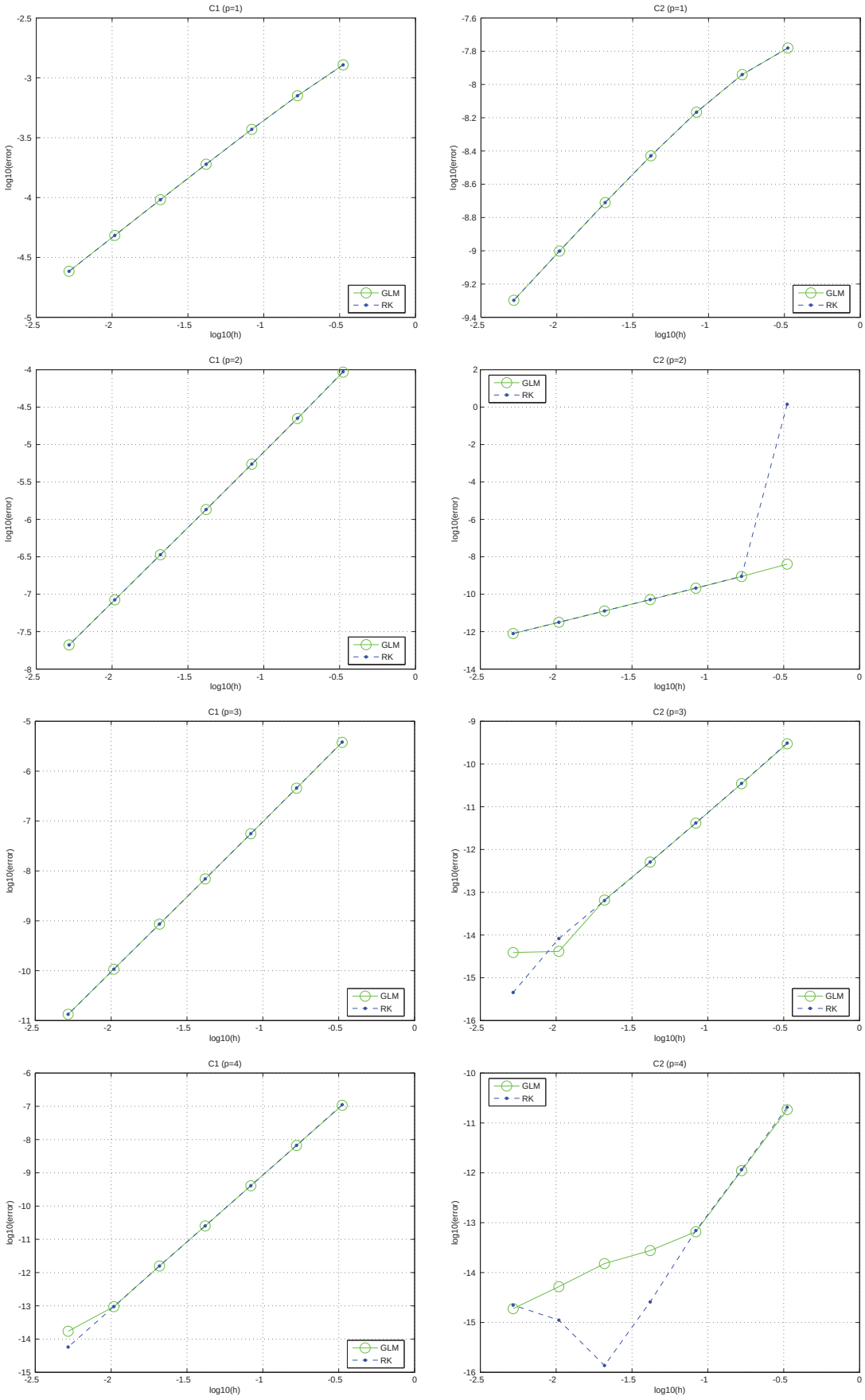


Figure 6.4: Global error (fixed stepsize) for the problems C1 and C2 (left to right) using the IRKS and Runge–Kutta methods of order one to four (top to bottom).

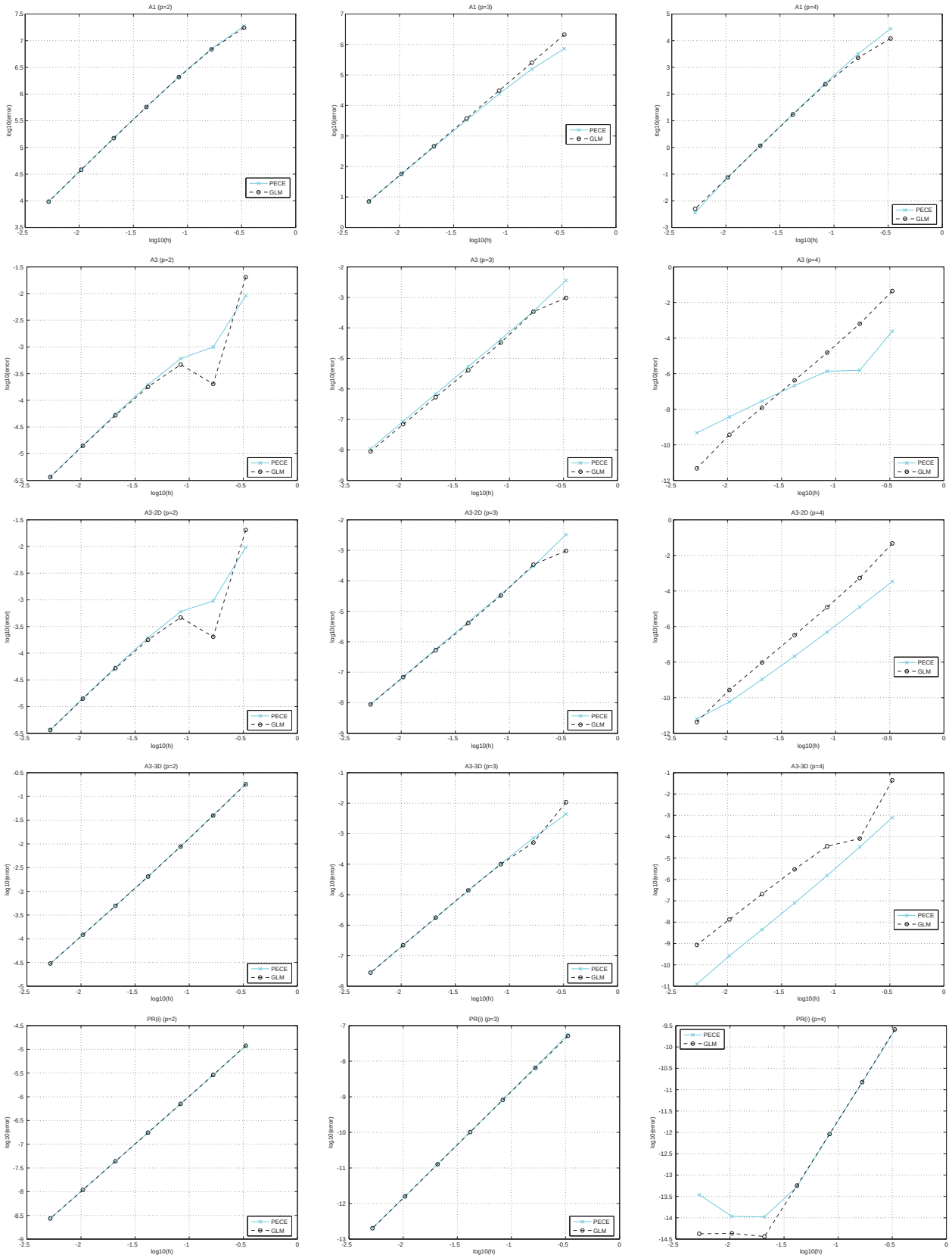


Figure 6.5: Global error (fixed stepsize) for the problems A1, A3, A3–2D, A3–2D and PR(i) (top to bottom) using the IRKS methods and PECE pairs of order two, three and four (left to right).

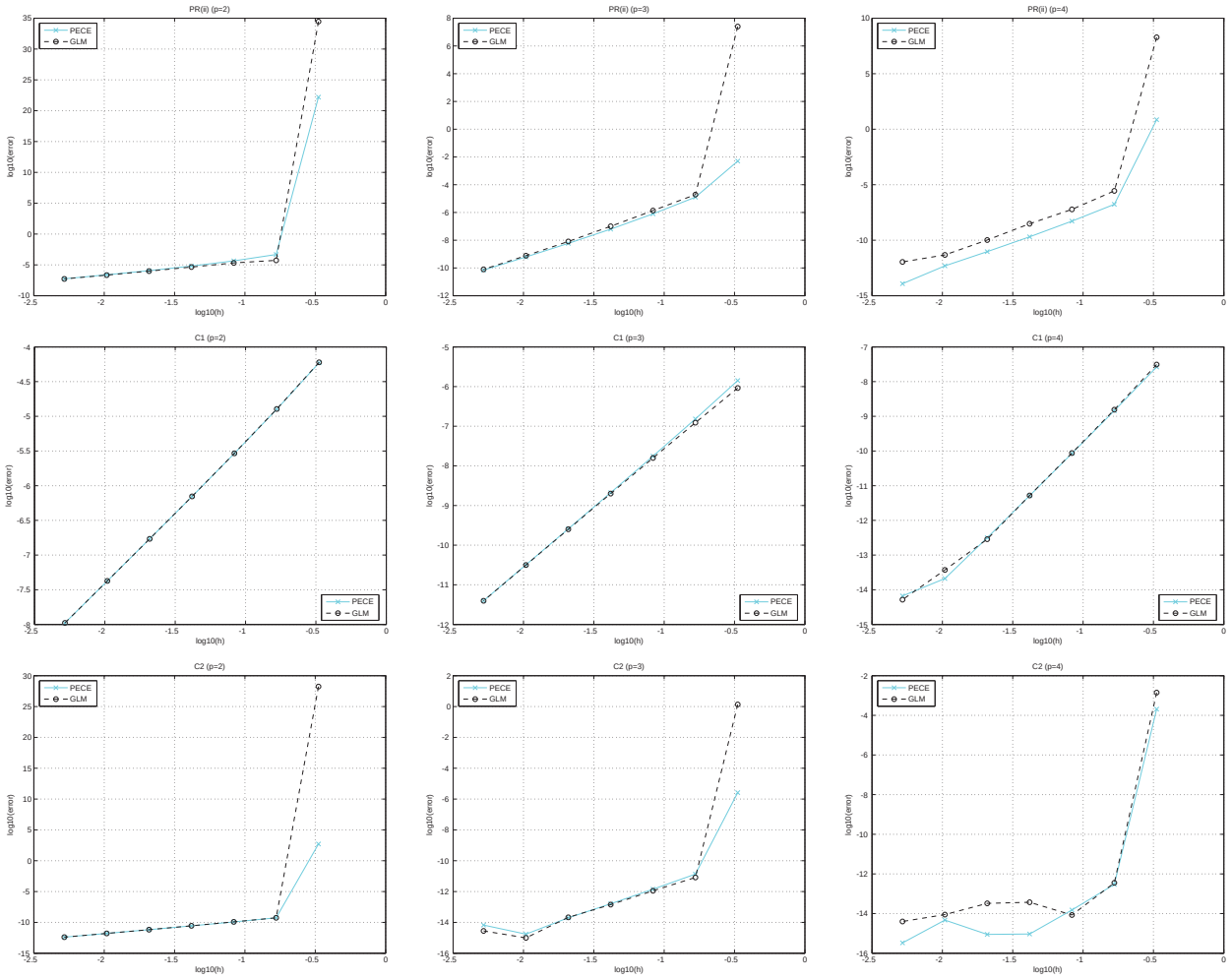


Figure 6.6: Global error (fixed stepsize) for the problems PR(ii), C1 and C2 (top to bottom) using the IRKS methods and PECE pairs of order two, three and four (left to right).

6.3 Experiments with variable stepsize, fixed order

In this set of experiments, variable stepsize controllers for the implementation of the GLMs (in Table 2.2) of (fixed) order 2, 3 and 4 are tested. The Lagrange stepsize controller (5.7) and the acceptance rejection criteria given in Subsection 5.2.1, are used. However, to start any variable stepsize scheme, an initial stepsize is necessary. This is customarily specific to the nature of the method. The automatic scheme for initial stepsize given in Subsection 4.2.1 is used in the rest of this chapter. Before this it is tested on a few problems, as follows.

6.3.1 Initial stepsize

The selection of initial stepsize plays an important role in the ODE solvers. A good choice for the initial stepsize h_0, helps to efficiently obtain the desired trajectory of smooth step sizes without any rejections. For variable stepsize implementation (in this section and in Section 6.4), the modified form of the automatic selection of initial stepsize approach by Gladwell et al. [69], is used (Subsection 4.2.1).

Atol	10^{-8}		10^{-12}	
problem \ approach	trivial	automatic	trivial	automatic
Pend.	3	0	9	0
Bubble.	6	0	12	0
Bruss.	5	0	12	0
Kepler.($\epsilon = 0.1$)	5	0	12	0
Kepler.($\epsilon = 0.3$)	6	0	12	0
Kepler.($\epsilon = 0.5$)	7	0	13	0
Kepler.($\epsilon = 0.7$)	7	0	14	0
Kepler.($\epsilon = 0.9$)	10	0	17	0
VPol.	4	0	11	0
3-Body.	6	0	12	0
HO.	3	0	10	0

Table 6.5: Number of rejected steps occurring only in the start of the integration ($p_0 = 1$), using a trivial and the modified automatic approach for h_0.

To investigate this scheme, some physical problems are used. To check the performance of the initial stepsize, we need to measure the number of rejections occurring during the start of the integration. The GLM code starts with $p = 1$, even with higher fixed order, used in the main part of the code. So any rejections due to initial stepsize, only occur with the method of order one. Thus for any value of $p = 1, 2, 3, 4$, the number of rejections due to initial stepsize would be same.

The modified approach of Gladwell. et al. [69] presented in Subsection 4.2.1 and a trivial choice $h_0 = (x_{end} - x_0)/1000$, are tested. The rejections have been recorded for (absolute tolerance) Atol $= 10^{-8}$ and Atol $= 10^{-12}$ whereas (relative tolerance) Rtol = Atol. Table 6.5 records the number of rejected steps in the start of the integration, for each of the two choices for initial stepsize for a number of different problems. When the trivial value for h_0 is taken, there are a few rejections in the start of integration. Whereas, with the use of the automatic scheme, there are no rejections occurring in the start of integration.

Thus, to achieve our aim, the modified automatic scheme works for avoiding unnecessary rejections. Along with this, the stepsize controller is also very important. For avoiding unnecessary rejections, a slightly relaxed controller should be capable of accepting a step when it is too early to actually reject it. How this is possible using the Lagrange stepsize controller? This is examined in the following section.

6.3.2 Lagrange stepsize controller

For practical methods, it is necessary to verify the order of a method, numerically. A numerical verification of the order of methods is completed as in Section 6.2 but with a fixed stepsize. To verify the order, obtained by a method, in variable stepsize mode, the Lagrange theory can be used. Here it is necessary to mention that the aim is to verify the numerical order using the Lagrange stepsize controller in (5.7), so we need to use the Lagrange theory for its verification as well. Such experiments are presented in [32] for ARK methods.

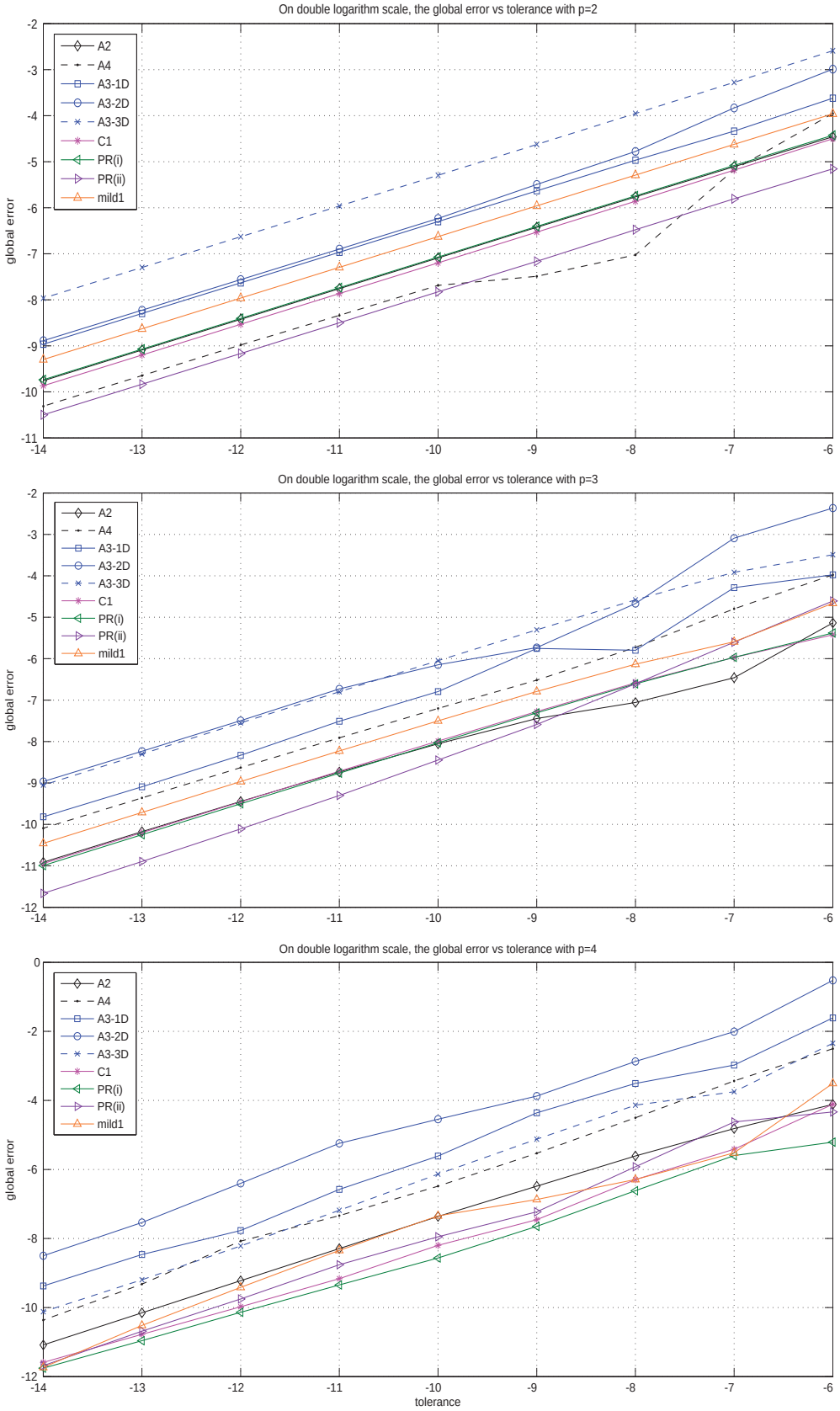


Figure 6.7: IRKS methods of order two, three and four (top to bottom) using the Lagrange stepsize controller and its rejection criteria, on problems A2, A3, A4, C1, PR(i), PR(ii) and mild1.

As, the aim in the variable stepsize implementation is to keep the norm of the error close to the tolerance provided, so, referring to (5.6),

$$C(x)H^{p+1} = T\frac{(p+1)}{p}.$$

Using this relation in (5.2),

$$\begin{aligned} E(H) &= \int_a^b C(x) \left(\frac{T\left(\frac{p+1}{p}\right)}{C(x)} \right)^{\frac{p}{p+1}} dx, \\ &= T^{\left(\frac{p}{p+1}\right)} \int_a^b \left(\frac{p+1}{p}\right)^{\frac{p}{p+1}} (C(x))^{\frac{1}{p+1}}\, dx, \\ E &\lesssim T^{\left(\frac{p}{p+1}\right)} \end{aligned} \tag{6.9}$$

Thus in variable stepsize mode, this relation can be used to verify the *numerical order* of a method. For this, using a particular problem, the exact (global) error at x_{end} is to be recorded for different values of the tolerance. On a double logarithm scale, the slope of the line, obtained by plotting the recorded errors for each of the tolerances, should be approximately $p/(p+1)$.

So using the Lagrange stepsize scheme (5.7) and its rejection criteria in Section 5.2.1 described in terms of the Lagrange theory, the slopes in the log–log plots of the relation in (6.9) must be close to $p/(p+1)$. Various problems with exact solutions are used to verify the result. Initially, some linear problems including A2, A3 and A4 of the DETEST are used. On successful verification of these simple problems, the multi dimensional forms of the problem A3 and problem C1 are tested and the test set is further extended to a few mildly stiff problems including the PR problems and mild1. These results are presented in Figure 6.7 by plotting the exact error at the end of the integration interval for each value of the tolerance from Atol $= 10^{-6}$ to Atol $= 10^{-14}$. The numerical values of the slopes, for the methods of order two, three and four (top to bottom in Figure 6.7), are quite close to the value $p/(p+1)$, where p is the order of the method.

After the verification of order for the new implementation techniques, the next step is to test the advantages of these new techniques over the standard ones, this is presented in the following Subsection.

6.3.3 Comparing the Lagrange and standard approaches

As mentioned in Chapter 5, the Lagrange approach for variable stepsize works as it gives a relaxation to avoid unnecessary rejections of steps and hence gives the optimal step sizes. To check the efficiency (fixed order) of the Lagrange stepsize controller and its rejection criteria (given in the Section 5.1), the first step is to compare it with the standard techniques (given in Section 4.3). Although the use of standard approaches for the implementation of the IRKS methods is itself efficient, due to the reliable error estimators, to get the full advantage of these estimators, the Lagrange optimisation approach must be used for the implementation of IRKS methods.

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	180	0	180	0.226	293	8	301	0.289
A3–2D	127	4	131	0.121	195	21	216	0.163
A3–3D	208	0	208	0.099	337	0	337	0.161
PR(i)	26	0	26	0.016	34	0	34	0.017
PR(ii)	176	5	181	0.097	277	12	289	0.127
mild1	43	1	44	0.027	57	0	57	0.022
B2	79	2	81	0.051	109	0	109	0.058
C1	77	0	77	0.047	119	0	119	0.064
C2	100	2	102	0.053	127	3	130	0.075
Bubble	50	0	50	0.030	73	0	73	0.042
Kepler($e = 0.1$)	208	0	208	0.109	336	0	336	0.149
Kepler($e = 0.3$)	232	0	232	0.117	375	0	375	0.163
Kepler($e = 0.5$)	272	0	272	0.137	443	0	443	0.191
Kepler($e = 0.7$)	334	0	334	0.151	547	3	550	0.248
Kepler($e = 0.9$)	474	3	477	0.231	769	15	784	0.373
VPol	143	0	143	0.075	231	2	233	0.137
3-Body	567	0	567	0.292	936	0	936	0.510
HO	66	0	66	0.032	102	0	102	0.041

Table 6.6: For the IRKS method of order two, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-5}$).

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	387	3	390	0.328	623	10	633	0.426
A3–2D	257	4	261	0.173	407	10	417	0.235
A3–3D	444	0	444	0.212	720	0	720	0.32
PR(i)	42	0	42	0.0181	62	0	62	0.0259
PR(ii)	367	0	367	0.178	595	7	602	0.281
mild1	71	0	71	0.0324	111	0	111	0.0452
B2	140	0	140	0.0811	216	0	216	0.0966
C1	153	0	153	0.0716	241	0	241	0.116
C2	151	2	153	0.0801	216	3	219	0.104
Bubble	92	0	92	0.0389	140	0	140	0.0559
Kepler($e = 0.1$)	443	0	443	0.205	716	0	716	0.31
Kepler($e = 0.3$)	496	0	496	0.232	803	0	803	0.367
Kepler($e = 0.5$)	586	0	586	0.271	950	0	950	0.443
Kepler($e = 0.7$)	723	0	723	0.337	1173	0	1173	0.544
Kepler($e = 0.9$)	1009	0	1009	0.492	1639	0	1639	0.765
VPol	303	0	303	0.145	490	0	490	0.241
3-Body	1244	0	1244	0.62	2028	0	2028	0.983
HO	134	0	134	0.0656	214	0	214	0.0938

Table 6.7: For the IRKS method of order two, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-6}$).

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	827	4	831	0.543	1340	12	1352	0.752
A3–2D	536	2	538	0.332	867	15	882	0.447
A3–3D	952	0	952	0.429	1545	0	1545	0.69
PR(i)	80	0	80	0.0326	125	0	125	0.0529
PR(ii)	791	6	797	0.374	1281	8	1289	0.596
mild1	145	0	145	0.0627	231	0	231	0.093
B2	280	0	280	0.136	441	0	441	0.194
C1	314	0	314	0.16	501	0	501	0.246
C2	271	2	273	0.147	417	5	422	0.199
Bubble	179	0	179	0.081	280	0	280	0.131
Kepler($e = 0.1$)	947	0	947	0.46	1535	0	1535	0.712
Kepler($e = 0.3$)	1062	0	1062	0.504	1723	0	1723	0.767
Kepler($e = 0.5$)	1256	0	1256	0.593	2038	0	2038	0.945
Kepler($e = 0.7$)	1551	0	1551	0.747	2516	0	2516	1.28
Kepler($e = 0.9$)	2168	0	2168	1.07	3518	0	3518	1.78
VPol	647	0	647	0.305	1047	0	1047	0.505
3-Body	2687	0	2687	1.42	4370	0	4370	2.56
HO	281	0	281	0.143	453	0	453	0.221

Table 6.8: For the IRKS method of order two, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-7}$).

In this comparison, we cannot simply conclude that the approach causing fewer rejections works better. A controller might avoid unnecessary rejections but might take more accepted steps using step sizes which are too small. Thus, along with the number of rejected steps, it is essential to count the number of accepted steps and consider the total number of steps.

The safety factor used in the standard controller is 0.9 whereas there is no safety factor used in the Lagrange controller. Without using the safety factor (or taking its value equal to one) in the standard stepsize controller, there are too many rejections. This number of rejections decreases, as the value of the safety factor is decreased from 1 to 0.8. Although a fixed heuristic value of 0.9, works better. The difference between the two controllers is the $(\frac{p+1}{p})^{\frac{1}{p+1}}$, appearing in the Lagrange controller. This factor works well, practically, as a safety factor and is different for each method. The results show its effectiveness.

Tables 6.6 to 6.14 record the number of steps taken by the code using the Lagrange and the standard approaches, for each of the methods of order two, three and four respectively. For each of the two approaches, the number of accepted steps na, number of rejected steps nr and total number of steps $ns = na + nr$, are shown. A column for CPU time is also added for each approach. These timings are not always very reliable because they can change when an experiment is repeated. The tests are run with three different tolerances 10^{-5}, 10^{-6} and 10^{-7} and Rtol=Atol. These are tested on different problems including a few simple problems, a few moderate problems as well as some mildly stiff problems. The Kepler problem is also tested, with eccentricity $0.1, 0.3, 0.5, 0.7$ and 0.9. Further problems include the Van der Pol problem, 3-body problem and harmonic oscillator. Here, $holdp = 0$

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	86	3	89	0.199	129	8	137	0.242
A3–2D	64	1	65	0.107	95	8	103	0.12
A3–3D	88	0	88	0.0431	133	0	133	0.0598
PR(i)	16	0	16	0.0112	19	0	19	0.0146
PR(ii)	69	0	69	0.0386	100	1	101	0.0544
mild1	34	1	35	0.0181	36	10	46	0.0241
B2	49	2	51	0.033	56	0	56	0.0356
C1	41	0	41	0.0231	57	0	57	0.0343
C2	74	5	79	0.0555	81	10	91	0.06
Bubble	32	0	32	0.017	41	0	41	0.028
Kepler($e = 0.1$)	80	0	80	0.035	122	0	122	0.0605
Kepler($e = 0.3$)	98	0	98	0.0516	151	0	151	0.0884
Kepler($e = 0.5$)	127	3	130	0.0658	191	20	211	0.113
Kepler($e = 0.7$)	162	6	168	0.0819	247	41	288	0.137
Kepler($e = 0.9$)	231	15	246	0.134	360	79	439	0.207
VPol	70	3	73	0.0439	100	6	106	0.0734
3-Body	290	10	300	0.148	420	58	478	0.249
HO	29	0	29	0.0178	40	0	40	0.0248

Table 6.9: For the IRKS method of order three, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-5}$).

as we are using fixed order. The results indicate that the Lagrange approach is efficient as compared to the standard approaches.

For the method of order two with Atol= 10^{-5}, the Lagrange controller is cost efficient for the problems in Table 6.6. There are always fewer rejections using the Lagrange controller except for the problems mild1 and B2. The problem mild1 causes one rejection using the Lagrange controller whereas there is no rejection in the case of the standard controller. Similarly, B2 has two rejections. However, the total number of steps are always less in the case of the Lagrange controller and the standard controller takes more steps to complete the integration. The Lagrange controller also takes less CPU time for all the problems except the problem mild1.

For the method of order two with Atol= 10^{-6} and Atol= 10^{-7}, the Lagrange controller is cost efficient for all the problems as in Tables 6.7 and 6.8. In this case, the Lagrange controller takes less steps (accepted, rejected and hence total number of steps) to complete the integration. For these two tolerances, the Lagrange controller is efficient in terms of CPU time.

For the method of order three with Atol= 10^{-5}, the Lagrange controller is cost efficient for the problems in Table 6.9 except problem B2. There are two rejected steps using the Lagrange controller for solving problem B2, though the standard controller takes more accepted steps to complete the integration.

For the method of order three with Atol= 10^{-6}, the Lagrange controller is cost efficient for the problems in Table 6.10 except problems mild1 and 3-Body. For each of these problems, there is one

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	152	4	156	0.223	221	15	236	0.278
A3–2D	111	4	115	0.11	164	7	171	0.144
A3–3D	154	0	154	0.0723	233	0	233	0.122
PR(i)	20	0	20	0.0106	26	0	26	0.0123
PR(ii)	117	0	117	0.063	174	1	175	0.0938
mild1	35	1	36	0.0187	39	0	39	0.019
B2	65	0	65	0.0387	89	0	89	0.0531
C1	65	0	65	0.0383	93	0	93	0.0522
C2	86	5	91	0.0538	108	8	116	0.081
Bubble	48	0	48	0.026	67	0	67	0.0325
Kepler($e = 0.1$)	142	0	142	0.0682	215	0	215	0.121
Kepler($e = 0.3$)	176	0	176	0.0911	268	0	268	0.124
Kepler($e = 0.5$)	216	0	216	0.0972	332	0	332	0.161
Kepler($e = 0.7$)	280	3	283	0.142	420	3	423	0.217
Kepler($e = 0.9$)	410	9	419	0.202	600	15	615	0.308
VPol	113	0	113	0.0695	171	3	174	0.0963
3-Body	472	1	473	0.243	719	0	719	0.371
HO	47	0	47	0.0237	68	0	68	0.0296

Table 6.10: For the IRKS method of order three, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-6}$).

rejection using the Lagrange controller. Whereas, there is no rejections in the case of the standard controller, however, the standard controller takes more accepted steps to complete the integration. Again, CPU time is always less using the Lagrange controller as compared to the standard controller.

For the method of order three with Atol$= 10^{-7}$, the Lagrange controller is cost efficient for all the problems as in Tables 6.11. In this case, the Lagrange controller takes less steps (accepted, rejected and hence total number of steps) and CPU time to complete the integration.

For the method of order four with Atol$= 10^{-5}$, the Lagrange controller takes fewer accepted steps for the problems as in Table 6.12 except for problems C2 and Bubble. When the Lagrange controller is used to solve C2, it took slightly more accepted steps as compared to the standard controller. However, the Lagrange controller causes fewer rejections and overall, performs better. For the Bubble problem, both the controllers take an equal number of steps to complete the integration. The problem A3–3D takes one extra rejection in the case of the Lagrange controller, though again, the standard controller takes, comparatively, more total steps to solve this problem. With this value of tolerance, the Lagrange controller takes less CPU time to solve each problem.

For the method of order four with Atol$= 10^{-6}$, the Lagrange controller is efficient for the problems as in Table 6.13 except for problems A3–3D. For this problem, it causes three rejections when the Lagrange controller is used whereas there is no rejection when the standard controller is used. However, the Lagrange controller has overall better performance due to fewer total steps taken. With this value of tolerance, the Lagrange controller takes less CPU time to solve all the problems.

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	261	0	261	0.272	392	6	398	0.367
A3–2D	189	0	189	0.144	283	8	291	0.199
A3–3D	271	0	271	0.137	409	0	409	0.197
PR(i)	29	0	29	0.0158	40	0	40	0.0183
PR(ii)	203	1	204	0.101	305	5	310	0.159
mild1	45	0	45	0.0235	63	0	63	0.0283
B2	102	0	102	0.0542	145	0	145	0.0803
C1	107	0	107	0.0631	155	0	155	0.0765
C2	116	4	120	0.0796	153	4	157	0.0828
Bubble	76	0	76	0.0414	107	0	107	0.0496
Kepler($e = 0.1$)	250	0	250	0.127	378	0	378	0.188
Kepler($e = 0.3$)	312	0	312	0.156	473	0	473	0.247
Kepler($e = 0.5$)	387	0	387	0.191	588	0	588	0.292
Kepler($e = 0.7$)	488	0	488	0.25	744	0	744	0.375
Kepler($e = 0.9$)	694	0	694	0.348	1059	0	1059	0.527
VPol	198	0	198	0.109	298	0	298	0.152
3-Body	840	0	840	0.431	1285	6	1291	0.673
HO	79	0	79	0.0369	116	0	116	0.0528

Table 6.11: For the IRKS method of order three, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-7}$).

For the method of order four with Atol$= 10^{-7}$, the Lagrange controller is efficient for almost all the problems as in Table 6.14 in terms of number of accepted steps. Though, the Lagrange controller causes a few extra rejections for some of the problems including C2, Kepler with $e = 0.5$ and $e = 0.7$, and 3-Body problems. However, the total number of steps is always less in these problems, except for the problem C2. Also, the Lagrange controller takes comparatively less CPU time for all the problems, except for the problem C2.

For most of the problems, there are no rejections for very small values of tolerances using either controller. Although the Lagrange controller is much more efficient in terms of number of steps or function evaluations. Based upon these experiments, we have been successful in reducing the heuristic safety factor used in the standard controllers with an improved controller (the Lagrange controller).

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	55	3	58	0.199	73	8	81	0.222
A3–2D	48	3	51	0.0892	59	8	67	0.112
A3–3D	56	2	58	0.0315	76	1	77	0.0463
PR(i)	13	0	13	0.00768	14	0	14	0.00804
PR(ii)	69	5	74	0.0335	69	5	74	0.037
mild1	65	3	68	0.0299	67	10	77	0.0396
B2	59	0	59	0.0409	63	3	66	0.0417
C1	32	0	32	0.0241	37	0	37	0.0292
C2	149	2	151	0.0895	144	12	156	0.0906
Bubble	34	0	34	0.0168	33	1	34	0.0226
Kepler($e = 0.1$)	58	0	58	0.0306	70	0	70	0.0423
Kepler($e = 0.3$)	71	3	74	0.0435	93	7	100	0.06
Kepler($e = 0.5$)	78	6	84	0.0419	116	19	135	0.0664
Kepler($e = 0.7$)	89	6	95	0.0504	139	31	170	0.0891
Kepler($e = 0.9$)	94	8	102	0.0475	183	37	220	0.121
VPol	47	2	49	0.0419	63	7	70	0.045
3-Body	94	3	97	0.0546	266	50	316	0.165
HO	18	0	18	0.0106	23	0	23	0.0126

Table 6.12: For the IRKS method of order four, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-5}$).

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	83	4	87	0.202	115	11	126	0.257
A3–2D	64	2	66	0.113	90	12	102	0.127
A3–3D	85	3	88	0.0489	121	0	121	0.0564
PR(i)	14	0	14	0.00767	17	0	17	0.011
PR(ii)	76	4	80	0.0455	85	8	93	0.0576
mild1	66	2	68	0.0351	69	3	72	0.0443
B2	68	1	69	0.0372	76	5	81	0.0528
C1	40	0	40	0.0285	56	0	56	0.0394
C2	146	2	148	0.0986	149	16	165	0.1
Bubble	36	0	36	0.019	45	0	45	0.0227
Kepler($e = 0.1$)	76	0	76	0.0406	111	0	111	0.0586
Kepler($e = 0.3$)	99	0	99	0.0514	145	0	145	0.0794
Kepler($e = 0.5$)	128	5	133	0.0682	184	20	204	0.112
Kepler($e = 0.7$)	163	8	171	0.0879	240	41	281	0.141
Kepler($e = 0.9$)	232	15	247	0.137	354	77	431	0.209
VPol	70	2	72	0.0444	97	4	101	0.0676
3-Body	291	10	301	0.162	408	39	447	0.242
HO	26	0	26	0.0143	37	0	37	0.0179

Table 6.13: For the IRKS method of order four, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-6}$).

	Lagrange				standard			
problem	na	nr	ns	CPU time	na	nr	ns	CPU time
A3	125	2	127	0.243	176	10	186	0.273
A3–2D	100	2	102	0.124	138	10	148	0.141
A3–3D	129	0	129	0.067	189	0	189	0.0956
PR(i)	18	0	18	0.0096	24	0	24	0.0119
PR(ii)	86	0	86	0.0441	129	4	133	0.0888
mild1	68	2	70	0.0335	69	9	78	0.0351
B2	81	1	82	0.0469	95	3	98	0.0523
C1	59	0	59	0.0525	82	0	82	0.0534
C2	150	5	155	0.102	157	3	160	0.0882
Bubble	49	0	49	0.0276	66	0	66	0.034
Kepler($e = 0.1$)	119	0	119	0.0634	174	0	174	0.104
Kepler($e = 0.3$)	155	0	155	0.0823	229	0	229	0.125
Kepler($e = 0.5$)	199	3	202	0.111	286	0	286	0.144
Kepler($e = 0.7$)	259	6	265	0.146	368	4	372	0.198
Kepler($e = 0.9$)	380	12	392	0.201	535	32	567	0.291
VPol	103	0	103	0.0668	151	0	151	0.0869
3-Body	434	4	438	0.239	635	0	635	0.345
HO	39	0	39	0.0229	55	0	55	0.0308

Table 6.14: For the IRKS method of order four, number of accepted steps (na), number of rejected steps (nr), total number of steps (ns) and CPU time, using the Lagrange and standard approaches (Atol $= 10^{-7}$).

To compare the overall efficiency of the Lagrange controller, we need to check how well it can balance the accuracy and the cost without using the safety factor. And to check this effect, we need to compare it again with the standard stepsize controller using the safety factor ($fac = 0.9$).

Figures 6.8 and 6.9 record the number of function evaluations (horizontal axis) versus the global error (vertical axis) on a double logarithmic scale. To check the efficiency of the Lagrange controller, it is compared with the standard controller. These tests are run with Atol $= 10^{-j}$, $j = 5, \ldots, 10$ where Rtol = Atol. The three columns in Figures 6.8 and 6.9, represents the results of methods of order two, three and four respectively. Whereas, each row represent a specific problem as mentioned in the title of each plot and in the figure caption.

Most of these plots shows that the Lagrange controller is efficient in terms of function evaluations. However, this efficiency, due to the Lagrange controller, is gained with some reduction of global error as compared to the standard controller. Here, it is necessary to mention that the aim of the Lagrange controller is not to achieve higher accuracy as compared to the standard approaches. However, the aim is to check how well the Lagrange stepsize controller can balance the accuracy and the cost without using the safety factor. Now the question is how much accuracy should be achieved using the Lagrange controller and whether the code is achieving it or not? This is verified in Subsection 6.3.2.

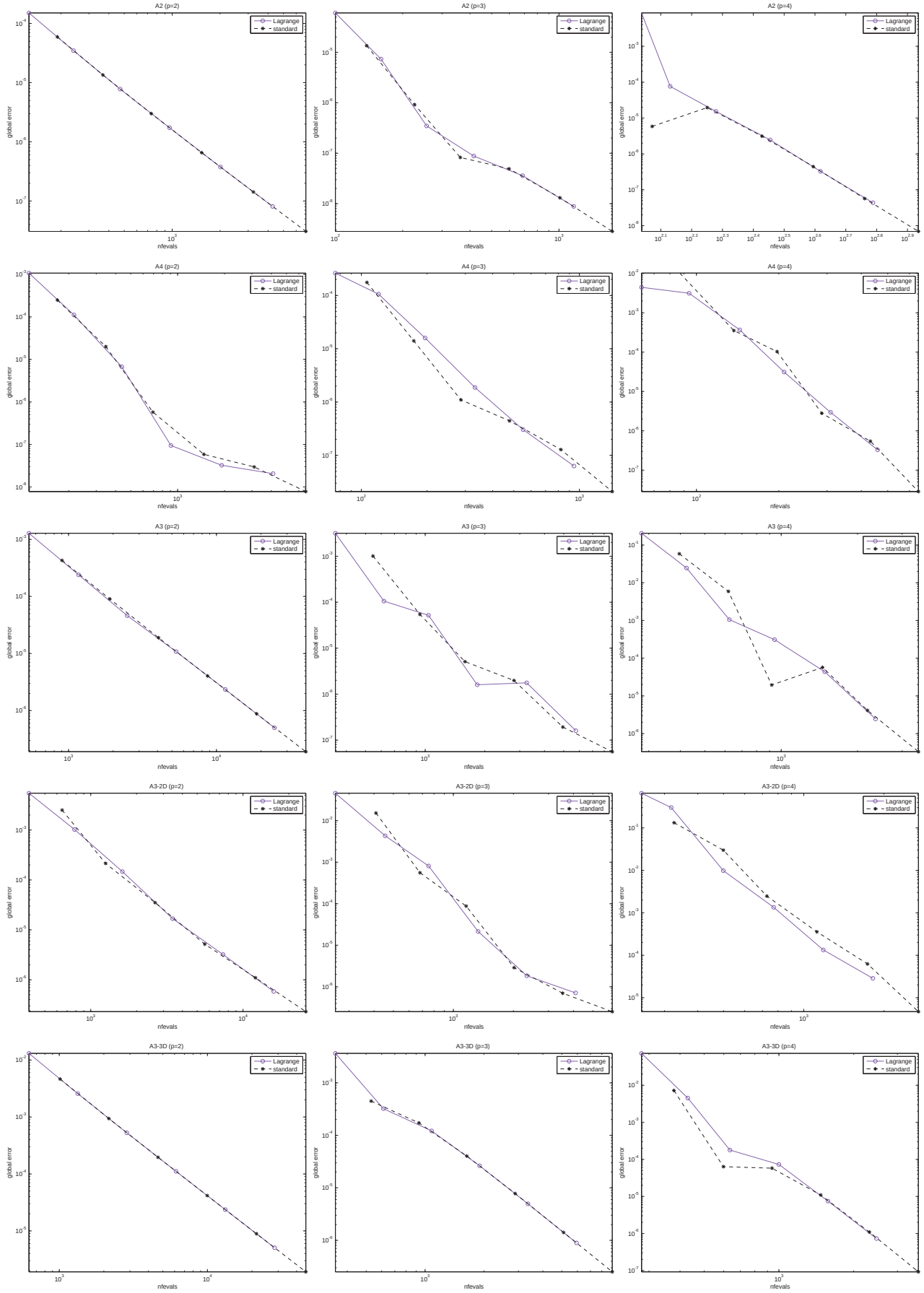


Figure 6.8: Comparison of the Lagrange and standard controllers for methods of order two, three and four (column–wise left to right) on the problems A2, A4, A3, A3–2D and A3–3D (row–wise top to bottom), respectively.

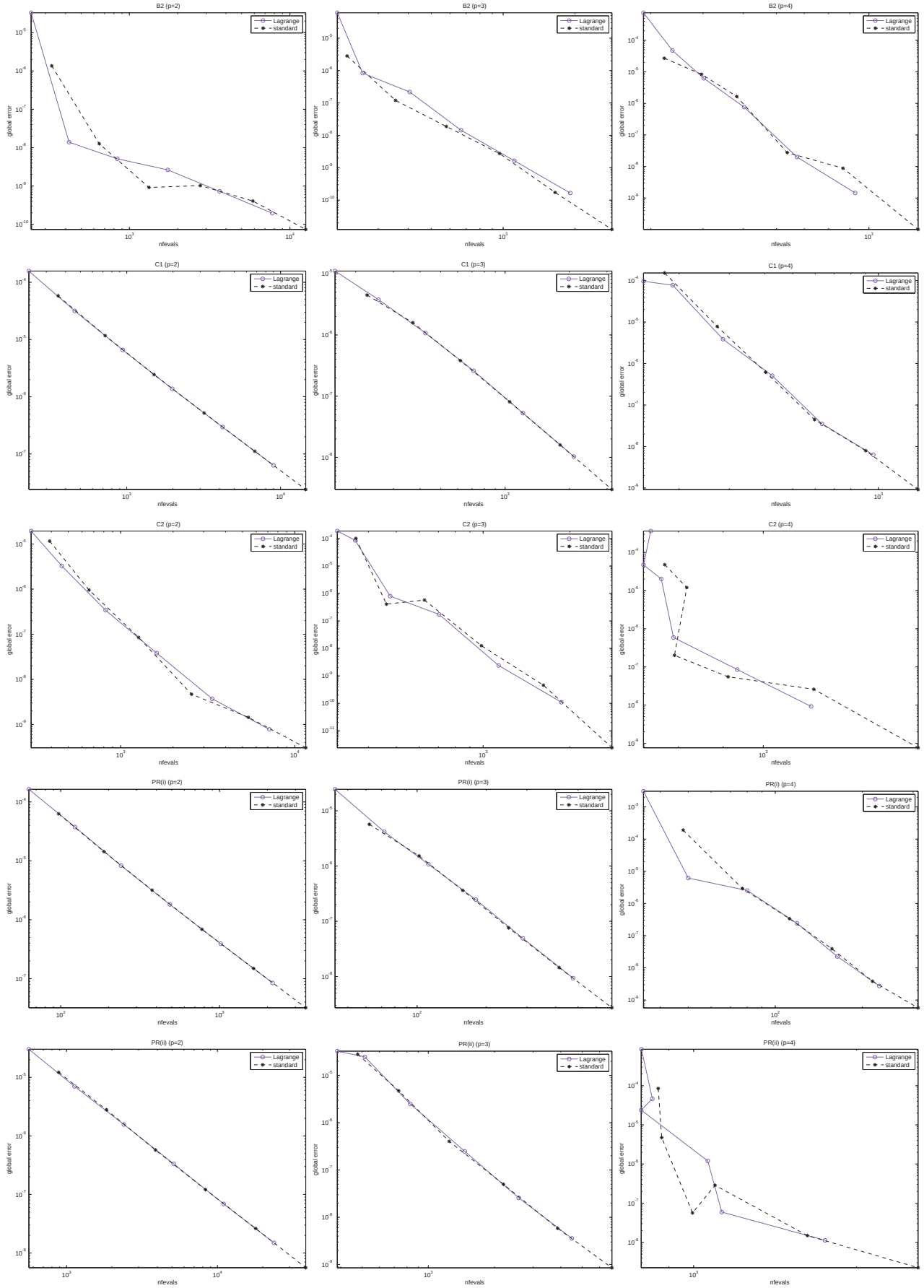


Figure 6.9: Comparison of the Lagrange and standard controllers for methods of order two, three and four (column–wise left to right) on problems B2, C1, C2, PR(i) and PR(ii) (row–wise top to bottom), respectively.

6.4 Experiments with variable stepsize, variable order

Implementation of the (h, p)–controller is not as straight forward as in the case of the fixed order controller. The optimisation approach of the Lagrange controller makes it technical to design. In the case of the fixed–order variable–stepsize solver, it is straight forward to choose an initial stepsize and calculate the next step. If a step is rejected, the standard controller may take some time to again attain the asymptotic stepsize behaviour. Contrary to this, in variable–stepsize variable–order mode, there are at least five natural cases that can arise. Three possible cases can occur when a step is accepted; the controller may choose to increase the order, decrease the order or may keep the order the same. Similarly in the case of rejection, the two possibilities would be to either keep the order the same or to decrease it (by one). There are many implementation issues that are to be considered in each of the cases. Instead of considering all five cases together, it is better to divide them in to the two traditional categories; acceptance and rejection cases.

However, it is necessary to first summarise the implementation approaches used for different tasks in the code, these are presented as follows.

6.4.1 GLM code

The whole code, written in MATLAB, consists of many techniques from Chapters 4 and 5. These are used for the implementation of the GLMs with IRKS– and F–properties presented in Table 2.2. The start of this GLM code is quite unique due to the design of GLMs used here. It is best to summarise the whole code that is used for performing the experiments with the Lagrange (h, p)–controller in this section. A few of the techniques are briefly explained in Chapters 4 and 5, these are just referred to here. The rest are presented in more detail.

Initial stepsize and order. The whole starting procedure is as in Section 4.2. The initial stepsize is chosen using the approach in Section 4.2.1. Naturally, a starting method is not required and the code starts with $p_0 = 1$ and therefore, with the exact initial Nordsieck vector.

Error estimation of the $(p+1)^{\text{st}}$ and $(p+2)^{\text{nd}}$ derivatives. For this, the results in Section 4.5.1 are used. The $(p+1)^{\text{st}}$ derivative is estimated in every step. The $(p+2)^{\text{nd}}$ derivative is estimated only after *a few* consecutive accepted steps when a change of order is allowed.

Acceptance and rejection criterion. This is described in Section 5.2.1 or more precisely accepted when the condition (5.14) is true.

Checking h for the final step. When an adaptive ODE solver is close to the end of the interval of integration, the new stepsize ($h_{\text{new}} = x_{n+1} - x_n$) proposed by the controller may be bigger than the remaining length of integration, i.e. $(x_{end} - x_n)$. In this case, the stepsize for the final step might be slightly outside the interval of integration. To overcome this, we use

```
xfinal=abs(xend-x-h);
hnew=min(hnew,xfinal);
```

Selection of stepsize and order for the next step. This is described in Algorithm 5.3.1.

Adjusting the length of the Nordsieck vector. When the order changes, the length of Nordsieck vector also changes. In the case of decreasing the order, length is simply adjusted by dropping the last element of the Nordsieck vector. When the stepsize increases, the length of Nordsieck vector is to be increased by one. This additional element is obtained, without any additional cost, as a reliable approximation of the $(p+1)^{\text{st}}$ derivative that is available as described in Algorithm 5.4.1.

Scale and modify technique. For this purpose, the algorithm 5.4.1 is used.

Use of *holdp*. This is described with each experiment.

With these implementation techniques, the whole code is tested for various purposes. An initial test for the performance of the (h, p)–controller, is to check the solution at each step and the corresponding stepsize and the order chosen for that step. A few problems are tested in this way and presented as follows.

6.4.2 Performance of the Nordsieck elements

A straight forward initial test of the Lagrange (h, p)–controller, is to check the (h, p) sequences. However, it is also convincing to see the solutions at each step, using these sequences. For this, the first two Nordsieck elements of each solution component of the problem are compared with the exact solution at each step.

First, one–dimensional problems A2, A4 and A3 are tested as shown in Figures 6.10, 6.11 and 6.12 respectively. In each figure, there are four plots. The performance of the first two Nordsieck elements are shown in the third and the fourth plots respectively, and their corresponding step sizes including rejected step sizes and the corresponding order, chosen by the Lagrange controller, are shown in the first and the second plots, respectively. The reason that we require the approximations to just the first two Nordsieck elements i.e. $y(x)$ and $hy'(x)$, is due to variable order mode. The maximum number of Nordsieck elements available at each step must be equal to the size of the Nordsieck vector with method of order one, i.e. $r = p + 1 = 2$.

For the experiment on the problem A2, Atol $= 10^{-6} =$ Rtol. The first two plots, from the top, in Figure 6.10 show the selection of stepsize and order at each step. It can be easily observed that the Lagrange controller goes to the method of maximum order four and stays there till the end of the integration. After setting to the maximum order, the stepsize increases gradually for the rest of the interval without any rejection of stepsize. Overall, it takes 29 steps or 135 function evaluations for this problem. The performance of the first two Nordsieck elements is also shown in the last two plots of Figure 6.10 by comparison with the exact solutions at each step. Similarly, the problem A4 is tested with Atol $= 10^{-6} =$ Rtol as shown in Figure 6.11. For this problem, the order increases gradually from one to the maximum order four at the start and remains at four till the end of the interval. No rejections occur and it takes 21 steps or 94 function–evaluations for this problem.

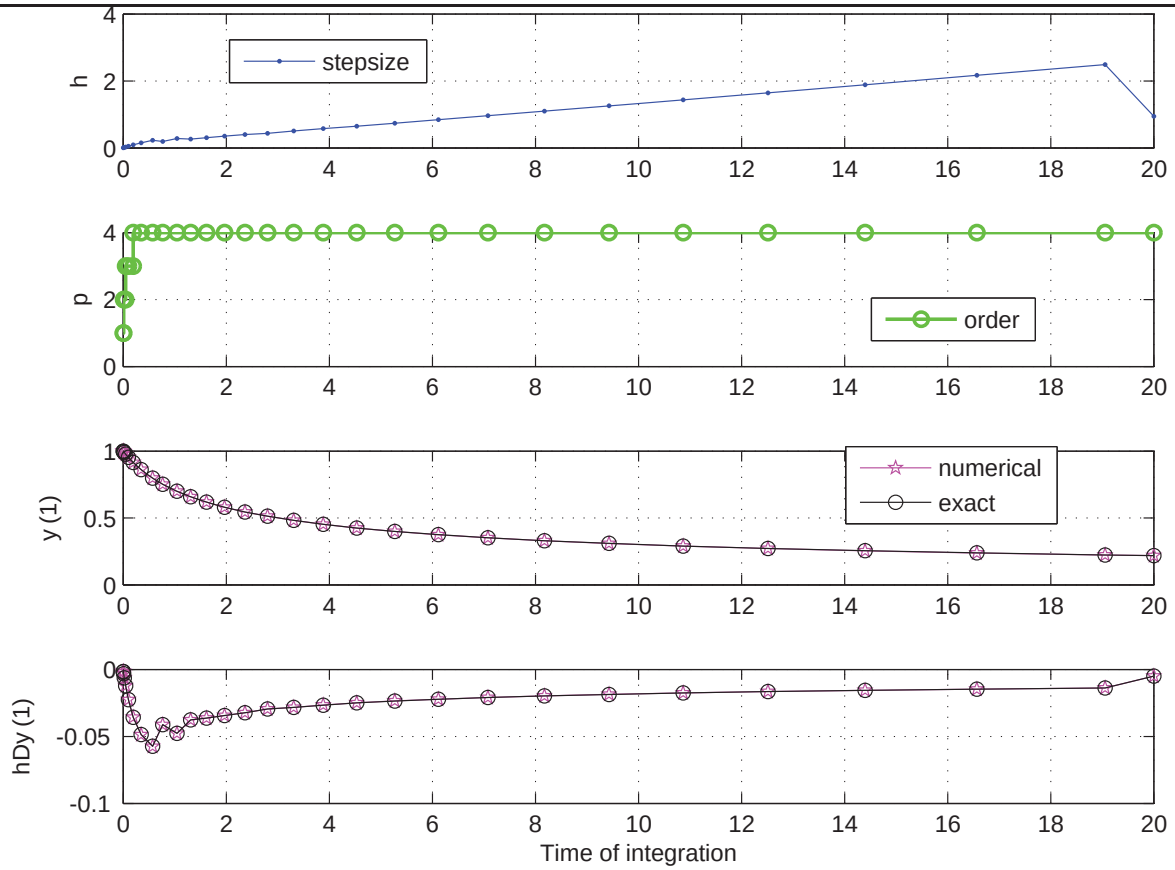


Figure 6.10: Stepsize control, order control and the first two Nordsieck elements for problem A2.

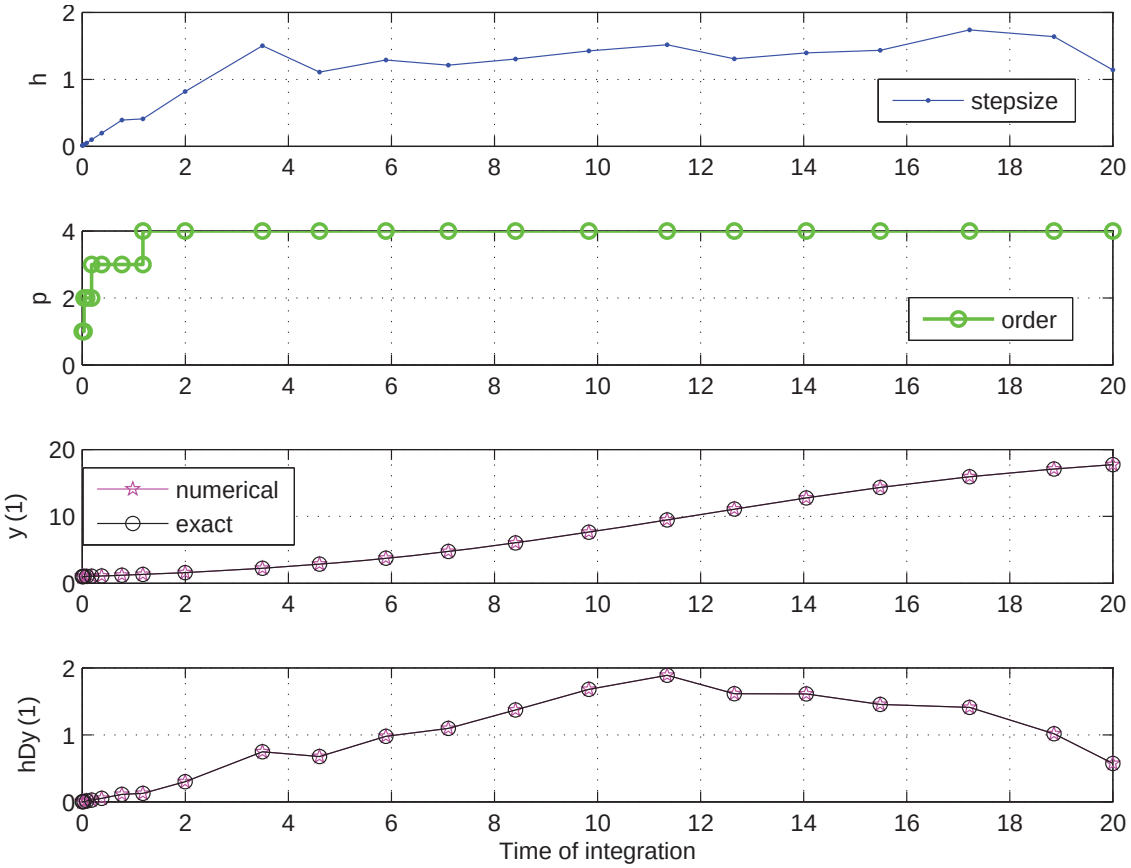


Figure 6.11: Stepsize control, order control and the first two Nordsieck elements for problem A4.

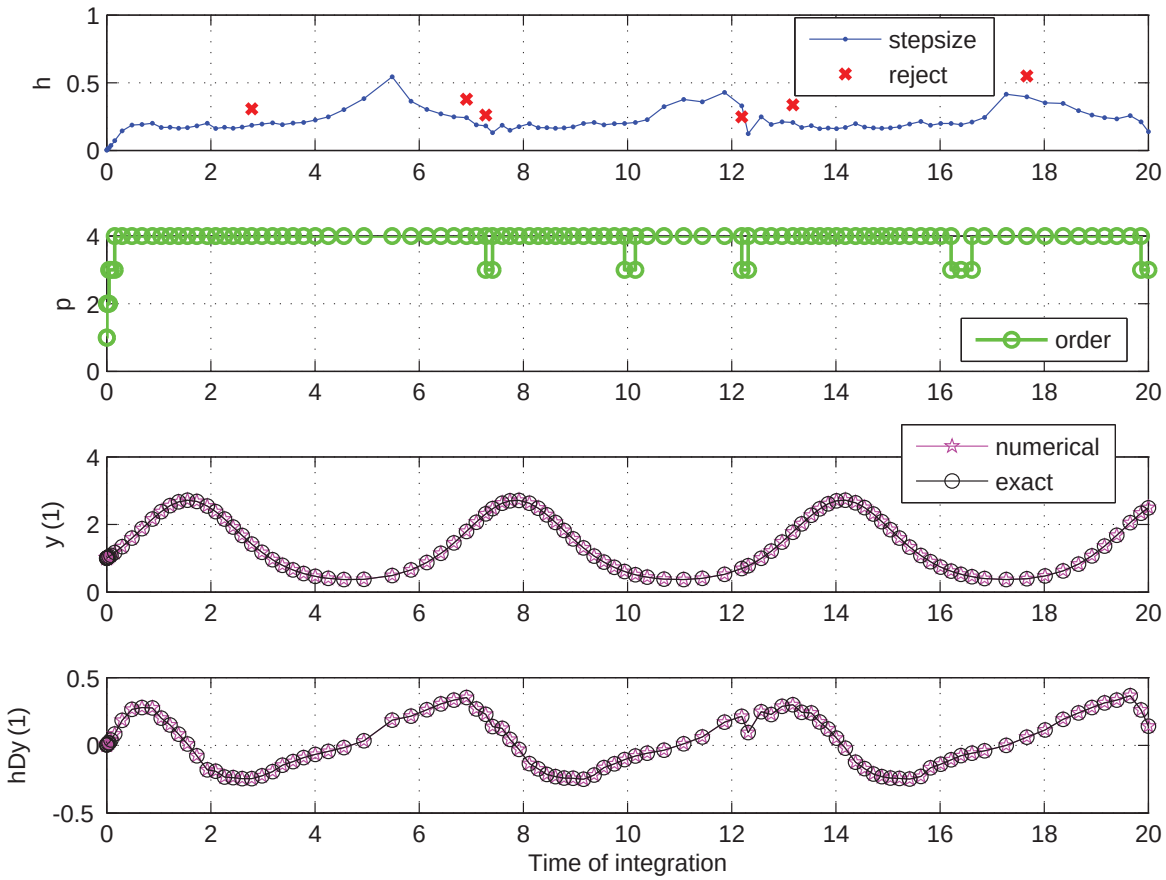


Figure 6.12: Stepsize control, order control and the first two Nordsieck elements for problem A3.

When the problem A3 is solved using Atol $= 10^{-6}$ and Rtol $= 0$, the results are very interesting. There are 6 rejections and it takes 97 steps or 495 function–evaluations for this problem. We can see in Figure 6.12 that unlike the problems A2 and A4, after reaching the maximum order four, there are a few order–oscillations followed by a few rejections. The important question in this situation is whether the Lagrange controller should decrease the order at these points or not? The Lagrange controller depends upon the optimisation function (5.4) which is based on the error estimation. Thus to analyse this, it is necessary to first see the behaviour of the $C_i h^{i+1} y^{(i+1)}(x)$ for $i = 3, 4$ for the problem A3. As h^{i+1} is a very small quantity, we scale by it in each case.

Figure 6.13 shows the comparison of $C_3 y^{(4)}(x)$ and $C_4 y^{(5)}(x)$ for problem A3 where C_3 and C_4 are error constants for the methods of order three and four respectively, and $y^{(4)}(x)$ and $y^{(5)}(x)$ are the exact higher derivatives of the solution. This comparison helps us to find which quantity could possibly minimise the cost function. In other words, this indicates which order of methods could be selected by the controller at each time step. As A3 is a periodic problem with period 2π, Figure 6.13 shows that it happens many times that the third order error term $C_3 y^{(4)}(x)$ is less than the fourth order term $C_4 y^{(5)}(x)$ though for very small intervals of time. Due to this, there are order–oscillations at some of these small intervals when we solve problem A3 using variable order (second plot of Figure 6.12).

For this problem, we also need to check the behaviour of estimations to the quantities $C_3 y^{(4)}(x)$ and

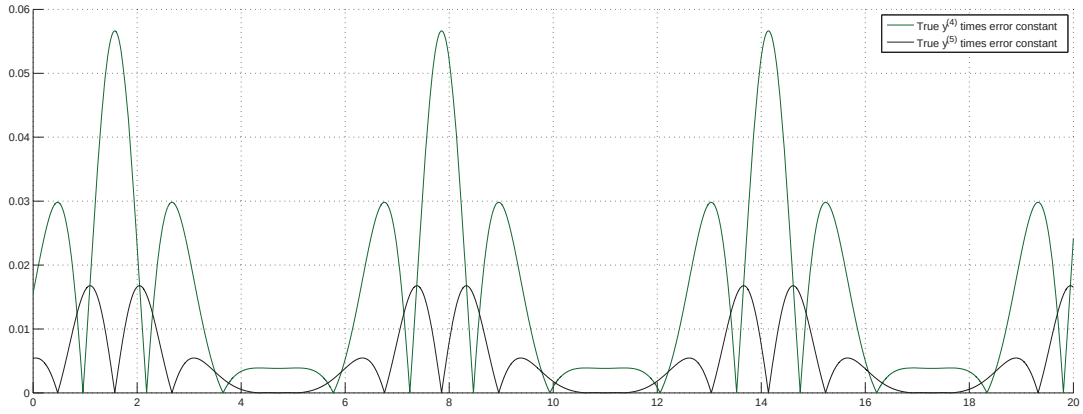


Figure 6.13: For problem A3, error analysis for the methods of order three and four.

$C_4 y^{(5)}(x)$ using error estimators. For example, first take $C_3 y^{(4)}(x)$, the estimation to this quantity is simply the error estimation $E3$ in the method of order three divided by h^4. Due to variable–order, $E3$ is estimated using either the method of order three or the method of order four, depending upon the selection by the Lagrange controller. For this, we first estimate $E3$ using $p = 3$ and $p = 4$ separately. From these quantities we compare the theoretical and the numerical results of $C_3 y^{(4)}(x)$ as shown in the top–left and bottom–left plots of Figure 6.14. These show the quality behaviour of the error estimators. Similarly, the error estimations $E4$ are calculated using $p = 3$ and $p = 4$ separately . The top–right and bottom–right plots of Figure 6.14 present the theoretical and numerical comparison of $C_4 y^{(5)}(x)$ which are also very reliable.

Thus the order–oscillations in the problem A3 are due to the theoretical behaviour of the problem and hence the Lagrange controller is following it. However, these order–oscillations occur only when a step ends inside these small intervals where the method of order three has a low value for its fourth derivative. Moreover, these order–oscillations are not so necessary and the Lagrange controller should only change the order when the order should decrease, theoretically, for long intervals.

These order–oscillations are overcome in [58] by giving some privilege to the existing order in a step to keep the order the same for the next step. For example, if a method of order p is used in a step and in the next step, the controller has three choices (in general) depending upon the values of the optimisation function for three different methods of order $p-1$, p or $p+1$. As we are using the maximisation form of (5.17) so to use the idea for controlling order–oscillations, in [58], maximum of the following should be chosen by the controller.

$$\frac{(p-1)h_{p-1}}{T(p)^2}, \quad \alpha\frac{ph_p}{T(p+1)^2}, \quad \frac{(p+1)h_{p+1}}{T(p+2)^2}$$

where $\alpha = 1.2$ for the methods used by the authors of that paper. The purpose here is to avoid too many or unnecessary order changes. In other words, the order should only be changed when there is a comparatively strong evidence that the change in order now would be overall efficient and it is not just for a few steps. This gives a notion to use another idea of controlling the order–oscillations. This is to keep the order the same for a couple of such consecutive steps in which the Lagrange controller

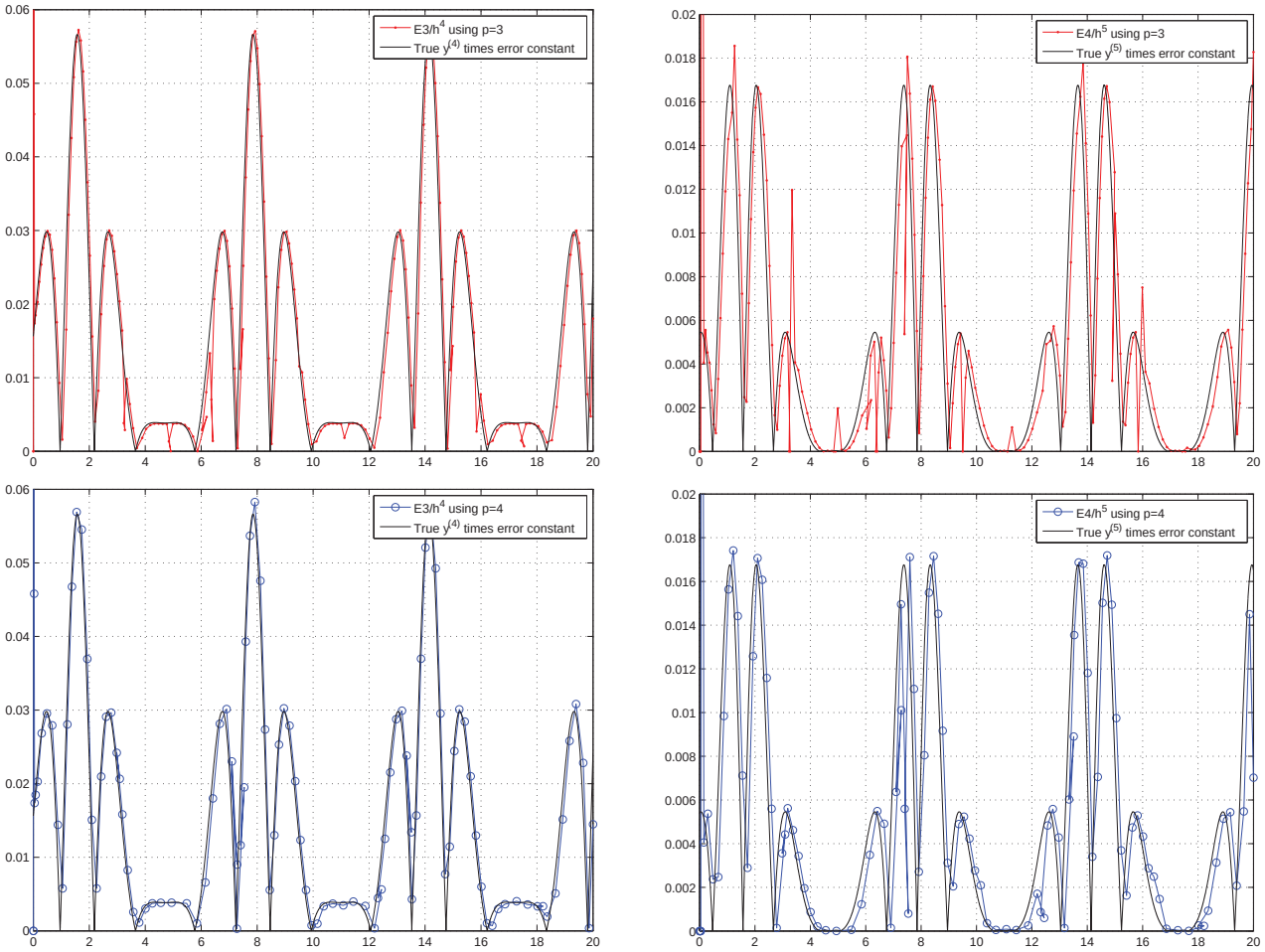


Figure 6.14: For problem A3, error analysis for the methods of order three and four.

decides to change the order in the same direction. This could be stronger evidence for changing the order. To hold the order p for a few consecutive steps (in a direction) we set a new parameter as *holdp*. So if $holdp = 2$, it means that the order will not be changed to $p+1$ or $p-1$, unless the Lagrange controller indicates to do so for 2 consecutive steps in the same direction.

To test this, A3 is repeated with the same value of Atol and Rtol. To control the order–oscillations, in A3, both the above mentioned techniques have been tested separately as well as in combination. It is observed that for $holdp > 2$, for various tolerances, there may be a delay in increasing the order up to maximum order, especially at the start of the integration. This means that we need to use the minimum value of *holdp* as much possible. For $holdp = 1$ and $\alpha = 1.2$, problem A3 shows no order–oscillations and a smooth and almost periodic stepsize pattern is obtained as shown in Figure 6.15. Now, it takes 3 rejections and 98 steps or 488 function–evaluations for this problem.

A few multi–dimensional problems are also used to test the performance of the controller and are reported in the same way. First, the 2D–form of problem A3 is tested as shown in Figure 6.16. In this problem, the first solution component is periodic whereas the second component is equal to the time elapsed. The first two plots of Figure 6.16 show the stepsize and the corresponding order of the

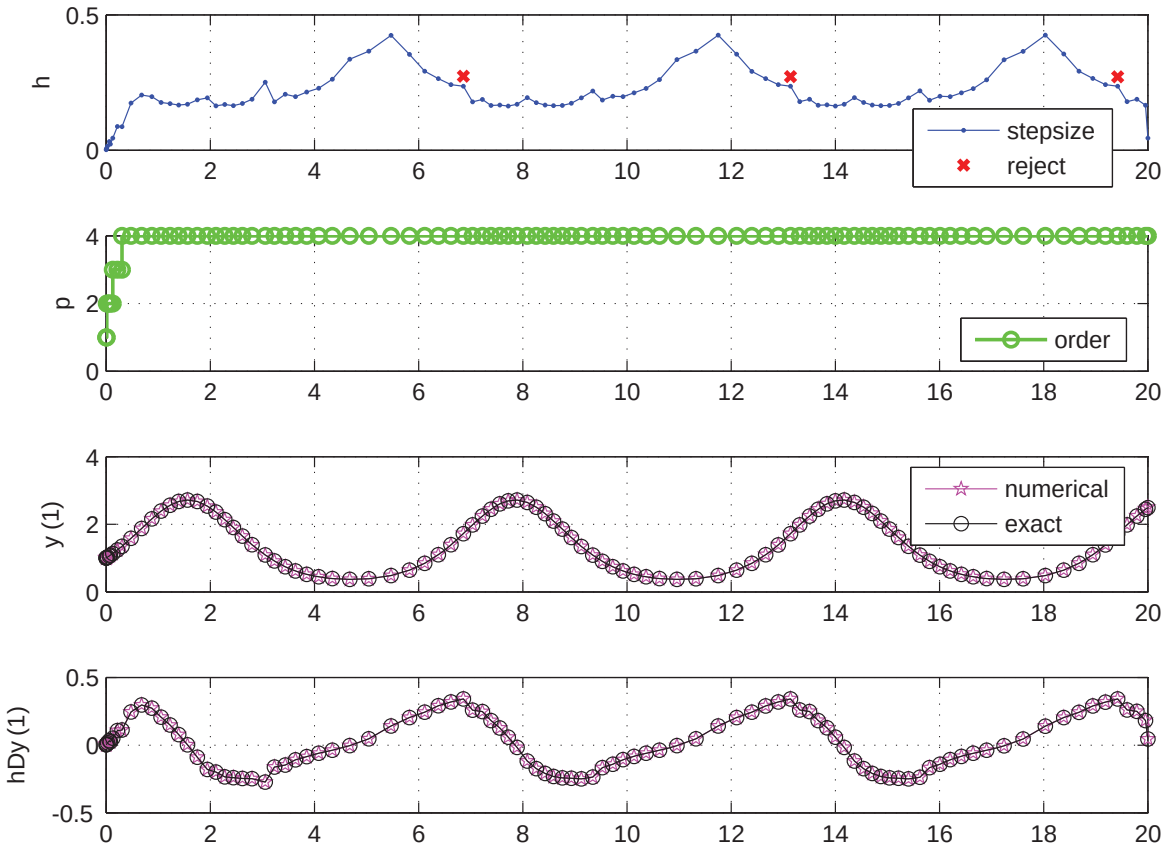


Figure 6.15: Stepsize control, order control and the first two Nordsieck elements for problem A3 (without order–oscillations).

methods used in each step to solve the problem A3–2D. The last two plots compare the exact and the numerical results of the first scaled derivatives of the solution components, respectively. The numerical results for the first two Nordsieck elements of the two components of the problem A3–2D are matching with the exact behaviour of the problem, at each step. For this problem, it is observed that using $holdp = 2$ order–oscillations can be avoided, however it causes order–delays for some crude tolerances. Using the same values $holdp = 1$ and $\alpha = 1.2$, the results are similar to the ones obtained for problem A3 as shown in Figure 6.16.

Similarly, the 3D–form of A3 is examined. For this figure we use the same value of tolerance as used for A3, interestingly, the results are better with $holdp = 0$ and $\alpha = 1$, i.e without any technique as shown in Figure 6.17. The three solution components are periodic but with different periods. The numerical solution of these three components matches at each step with the exact solution. The stepsize sequence is also periodic as shown in the first plot of Figure 6.17, whereas these step sizes are chosen for method of order four throughout. No rejections occur and it takes 100 steps or 490 function–evaluations which is almost the same as for the previous two forms of A3.

Following the same scheme, the three dimensional problem B2, is tested as shown in Figure 6.18. For this problem, Atol= 10^{-12}, Rtol= 10^{-8}, $holdp = 0$ and $\alpha = 1$, i.e without any technique. This is a decay problem in which all the solution components decay to zero very quickly in the interval $[0, 20]$. The GLM code starts with the method of order one and the Lagrange controller quickly increases this

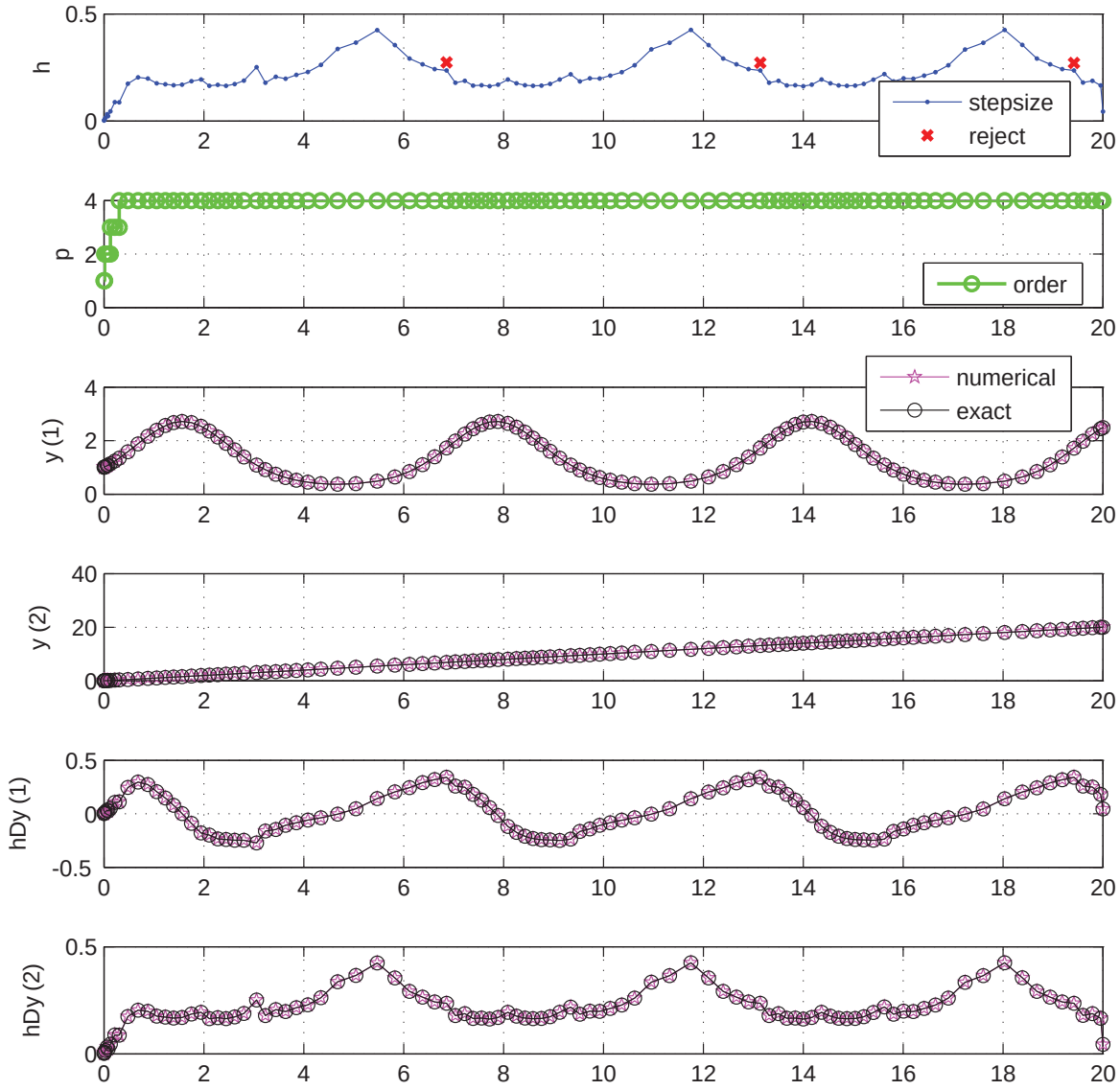


Figure 6.16: Stepsize control, order control and the first two Nordsieck elements for problem A3-2D.

to order four, in the start of integration. The initial stepsize is also very small and increases as the time increases.

As cases of mildly stiff problems, the PR–problems are tested and presented in Figure 6.19 using Atol = Rtol = 10^{-8}. For PR(i), $\lambda = -0.1$ and $\mu = -0.1$ are used and we set no parameter for controlling order–oscillations. There are no rejections and the numerical accuracy is not destroyed at any point. The stepsize increases as the time increases. For PR(ii), $\lambda = -4$ is used. This problem has a periodic solution. And the stepsize sequence obtained by the Lagrange controller is also periodic. For this problem, we take *holdp* = 1 and $\alpha = 1$ and the order remains four throughout the interval. Whereas, without using *holdp*, there are periodic order–oscillations and rejections as well.

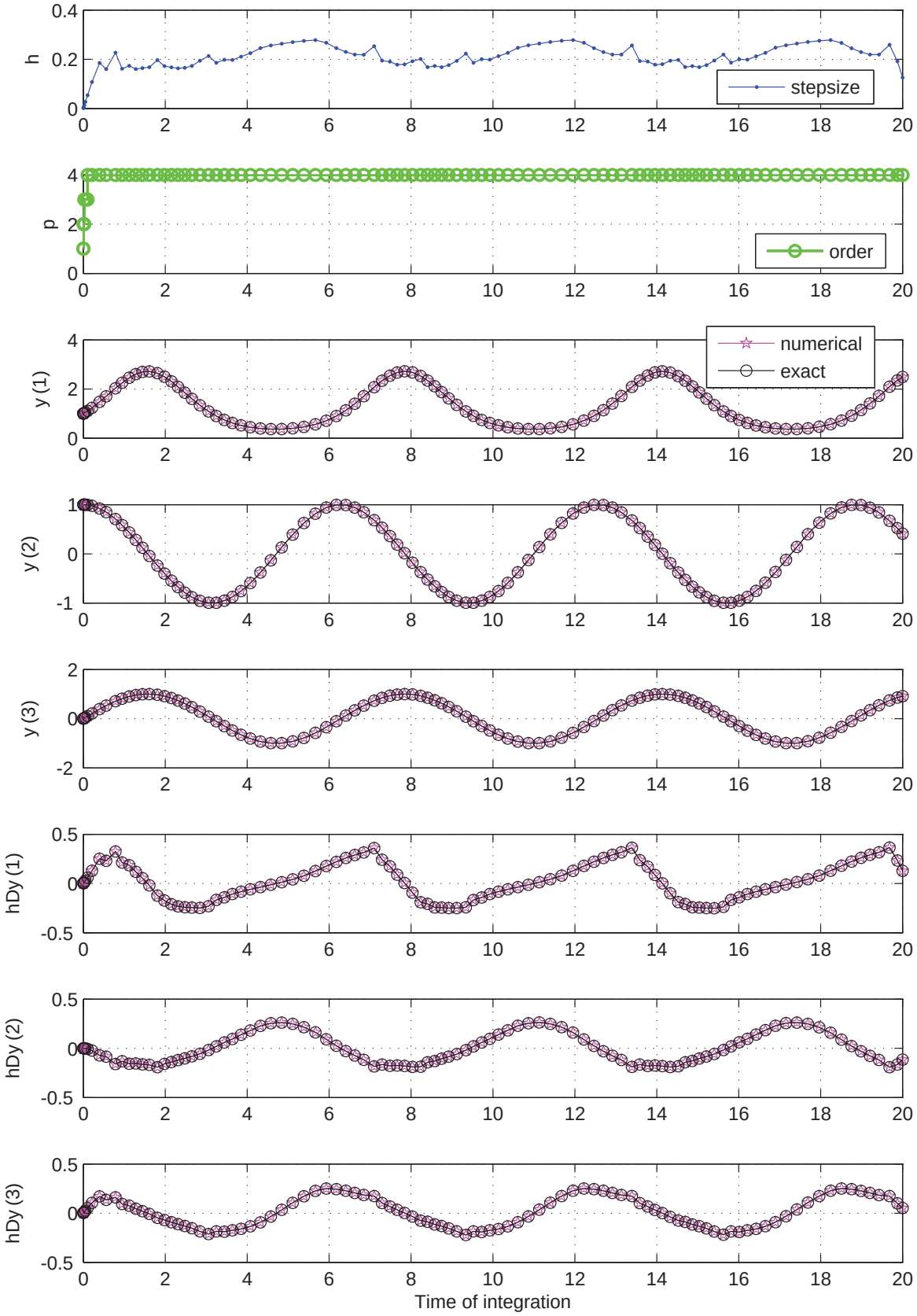


Figure 6.17: Stepsize control, order control and the first two Nordsieck elements for problem A3-3D.

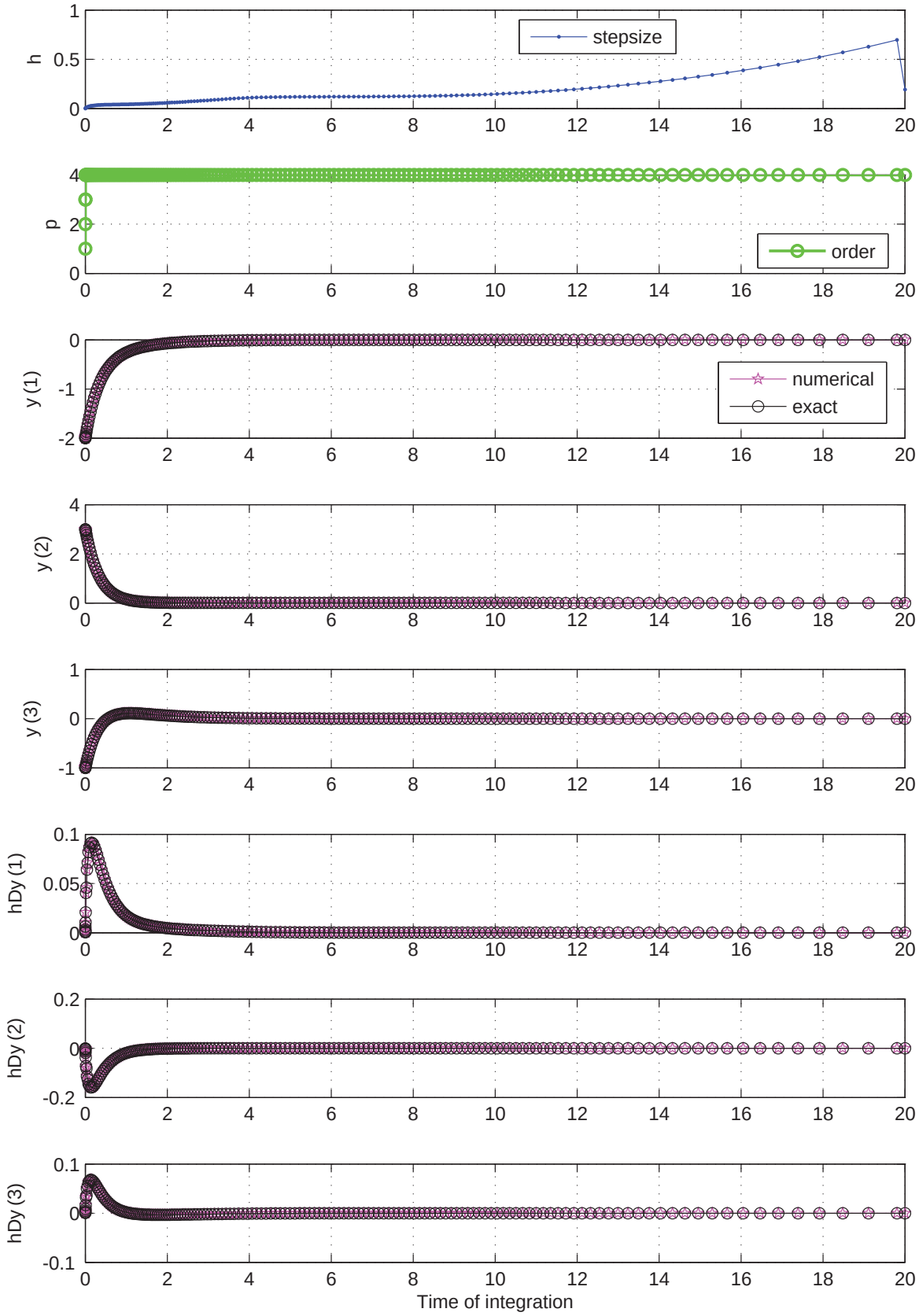


Figure 6.18: Stepsize control, order control and the first two Nordsieck elements for problem B2

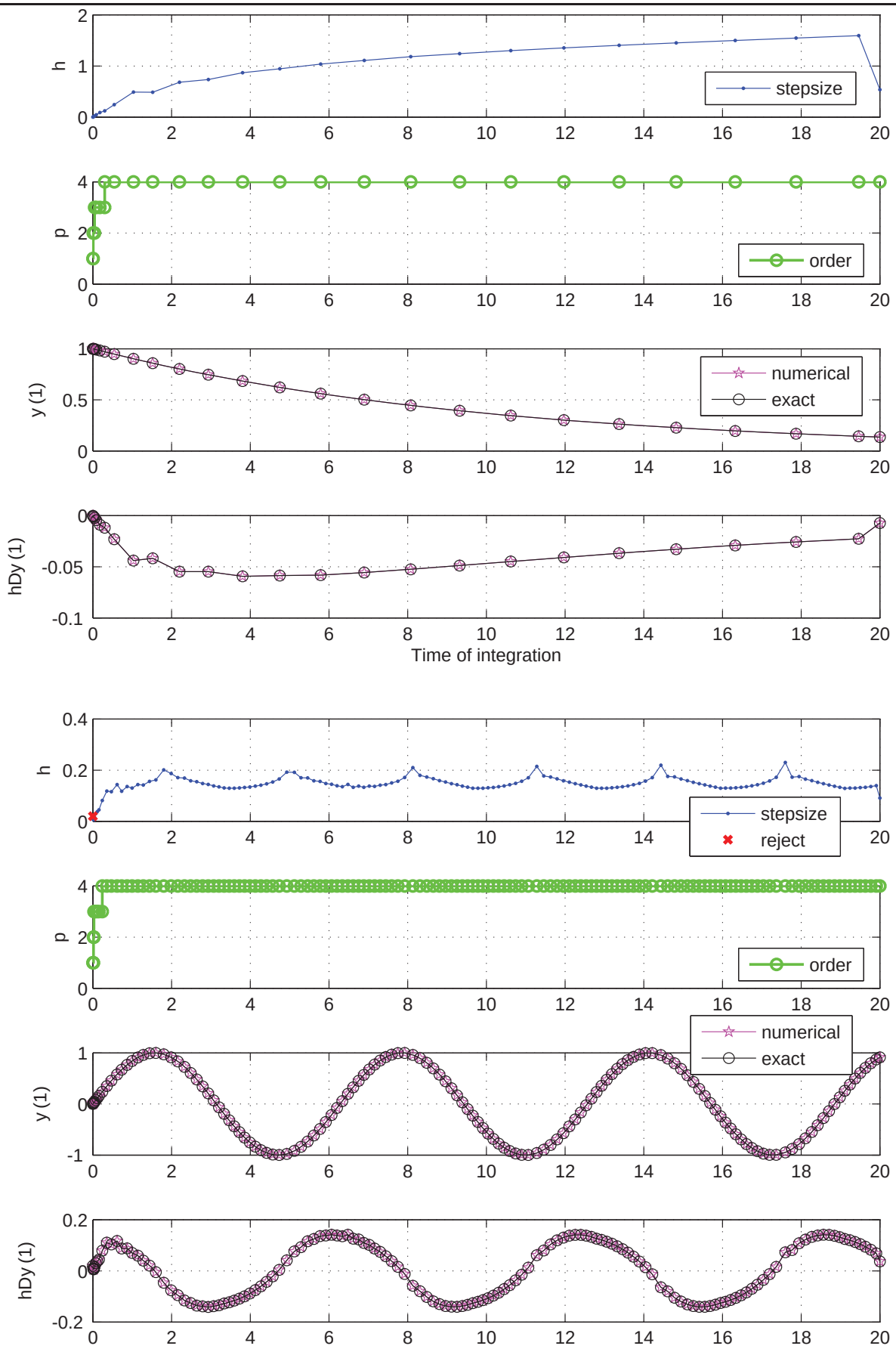


Figure 6.19: Stepsize control, order control and the first two Nordsieck elements for the problems PR(i) (top) and PR(ii) (bottom)

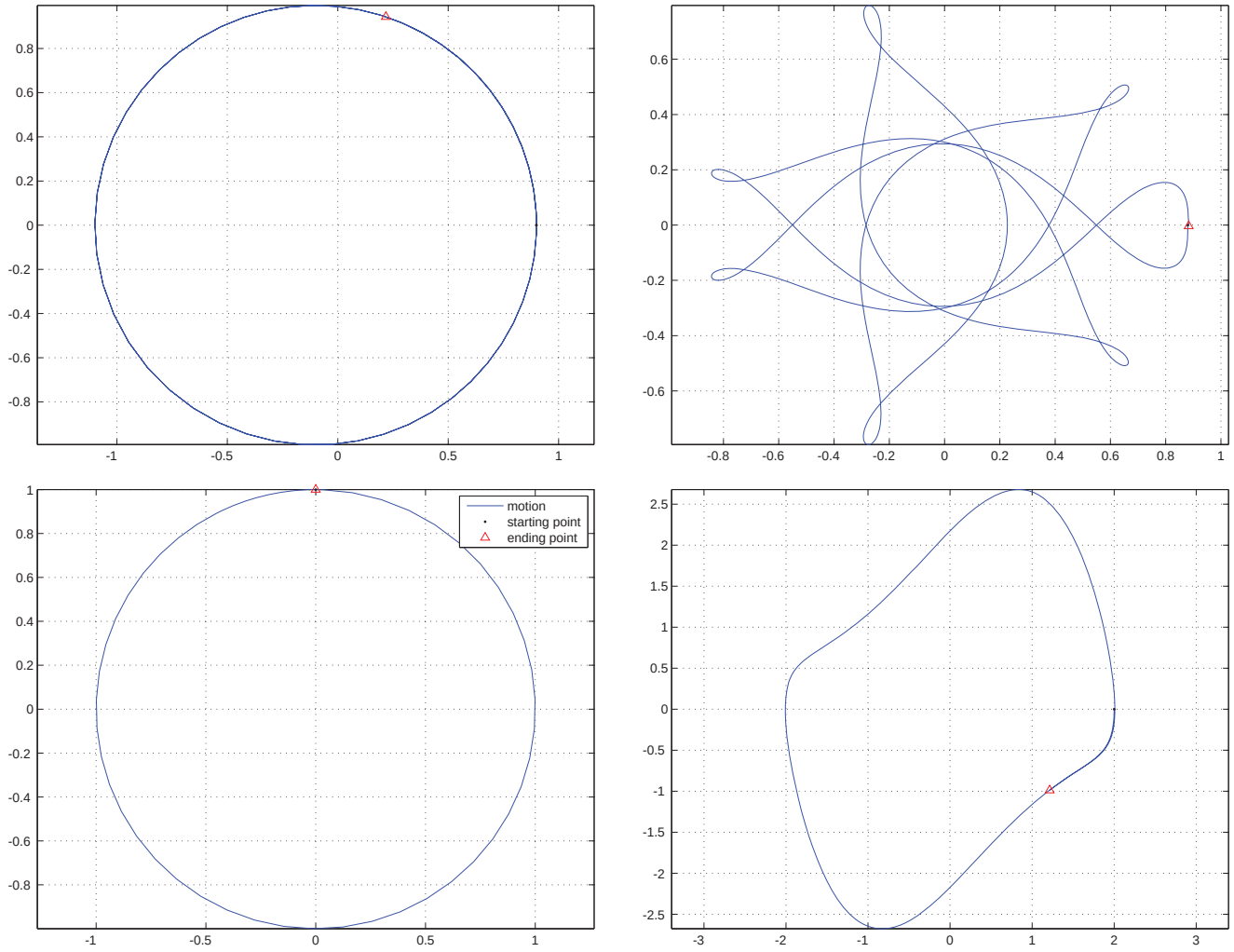


Figure 6.20: Motion in two–dimensions of the 2–body problem (top-left), 3–body problem (top-right), harmonic oscillator (bottom-left) and Van der Pol problem (bottom-right) using the GLM code.

6.4.3 Experiments on physical problems with motion in two dimensions

Some physical problems are tested to observe the performance of the automatic initial stepsize in Subsection 4.2.1. A few of these problems are again tested to see the performance of the GLM code. These include, the 2–body, 3–body, harmonic oscillator and Van der Pol problems. Here, Atol = Rtol = 10^{-8}. In the Van der Pol problem, $\mu = 1$. For these problems, none of the techniques has been used against order–oscillations. The order increases in the start to the maximum order four and remains four with smooth stepsize patterns.

6.4.4 Robustness of the controller

For this problem, *holdp* = 1 and Atol=Rtol= 10^{-6}. To explore the robustness of the Lagrange controller, the square path problem is examined thoroughly which is a two dimensional problem. Geometrically, this path is of square shape with centre at the origin and vertices $(1, 1)$, $(-1, 1)$, $(-1, -1)$

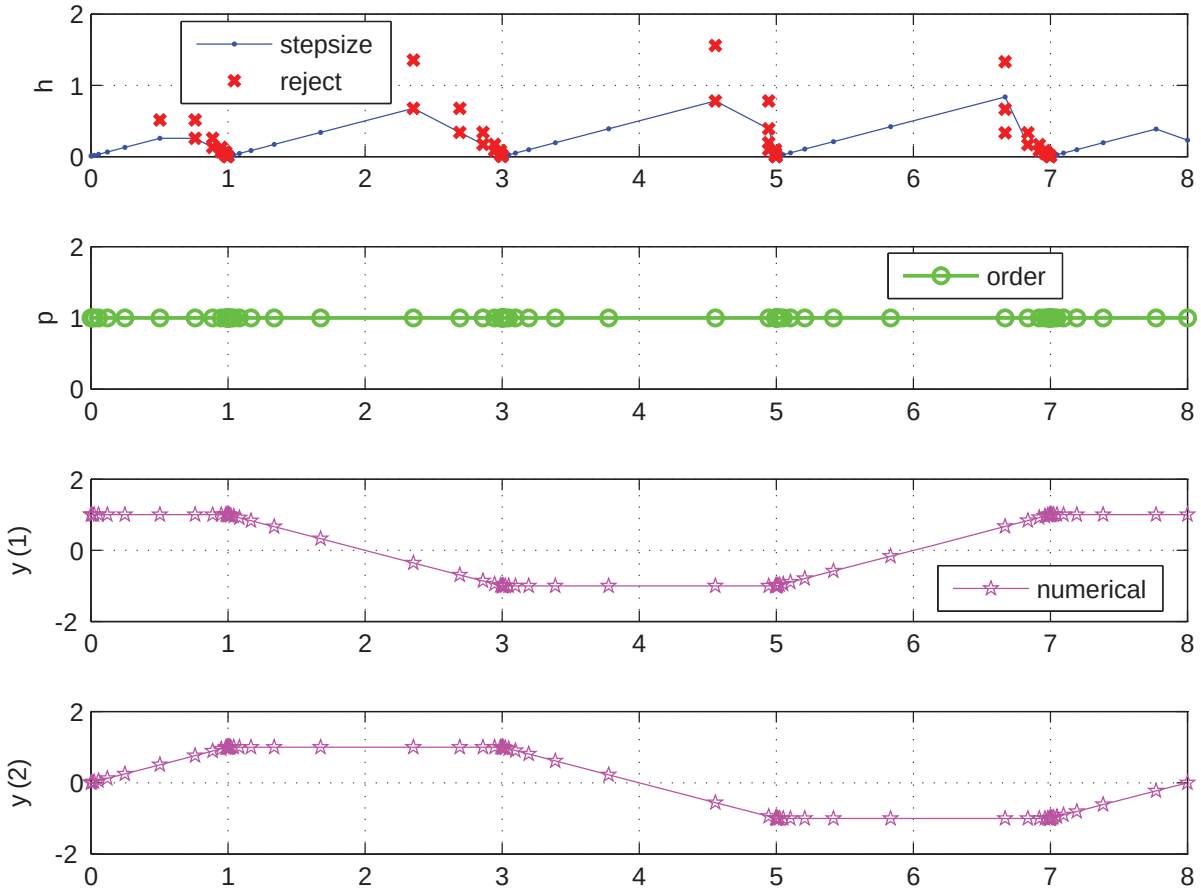


Figure 6.21: Stepsize control, order control and the first two Nordsieck elements for the square path problem.

and $(1, -1)$, respectively (counter clockwise on the xy-plane).

For the experiments on the square problem, the initial value is $[1, 0]$. Thus, the trajectory of the square path starts from the point $(1, 0)$ on the x-axis, represented by a black dot in Figure 6.22, and travels upward and then turns to the left at the point $(1, 1)$. Due to this turn, discontinuity in the first derivative of the solution occurs. Naturally, the steps should be rejected at these turns (corners) by an efficient code and it should turn properly. This is the main interest here, to see how the GLM code handles this discontinuity which is unknown to the code *a priori*.

Before the first rejection at the corner, the code should attempt a bigger step and should go outside the square path in the upward direction beyond the turning point point $(1, 1)$. Eventually this step fails to be accepted and a prescribed reduction in the stepsize should occur. This discontinuity should cause failure of a sequence of steps until the required accuracy is achieved. And this phenomena should happen at all the four corners of the square path.

Thus, the step should be rejected when a discontinuity occurs and each time the stepsize should be reduced. This discontinuity is efficiently handled with the GLM code as shown in Figure 6.22. The first plot in Figure 6.21 shows that the step sizes are rejected and reduced each time, near (just before)

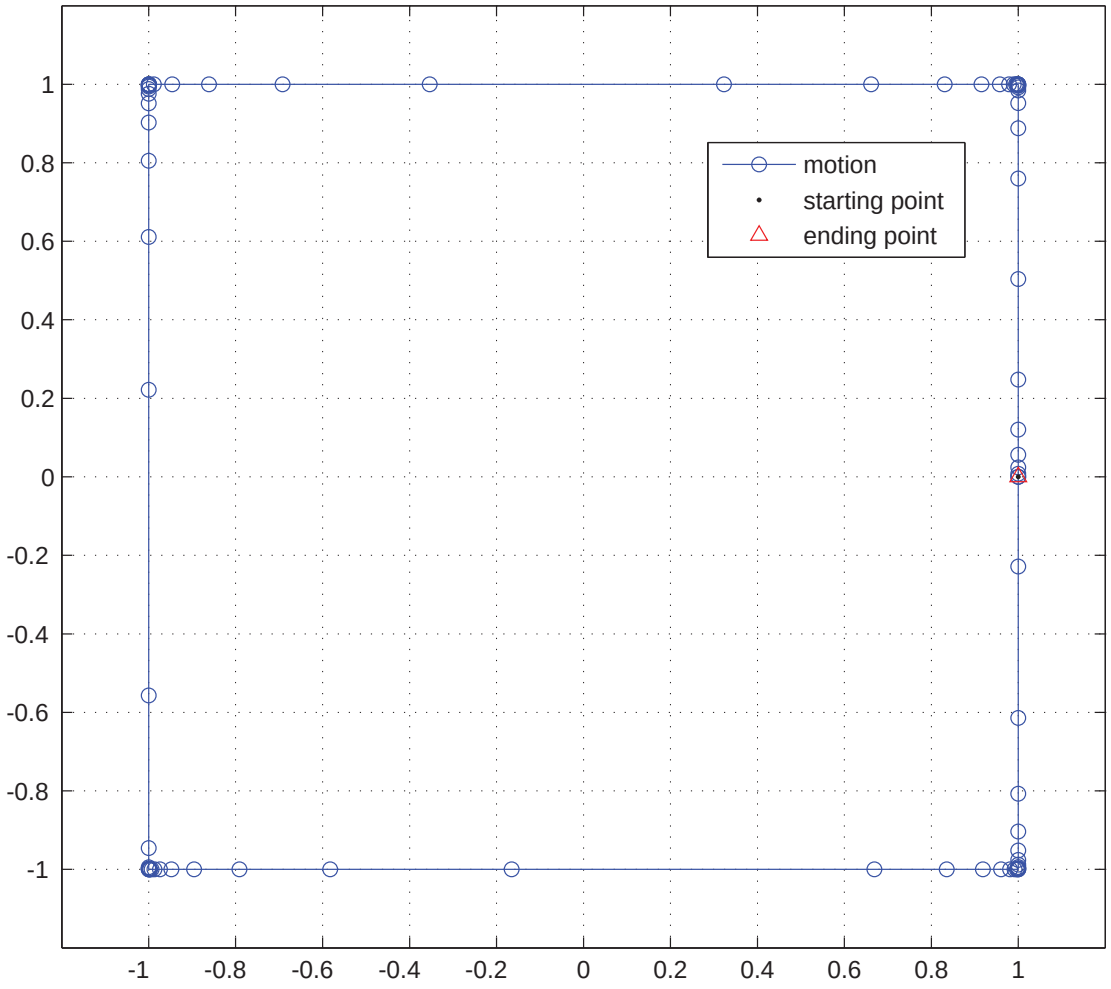


Figure 6.22: Motion in two–dimensions of the square path problem.

the corners.

After the start, and after each turn on the square path problem, the stepsize increases but the order remains 1 throughout. This is also of main interest and we should discuss the reason the order remains 1 along the sides of the square. In this case, when the step sizes increases rapidly, the Lagrange optimisation controller has two choices to either jump to order 2 or to keep the order the same, i.e. 1. The error term of the method of order one is the norm of $\frac{1}{2}h^2y''(x)$. One of the second derivative components is zero along each side of the square and so only one component is contributing to the error. Also, the error term of the second order method is zero. Thus the Lagrange controller, when minimising the rate of $E+T(p+1)$, will find that there is the same contribution from the error term for each p. However, the minimum value of p, will minimise the cost and hence the method of order one will be selected.

While travelling upward from the start, one solution component is constant and hence it has zero derivatives. Here, the order remains 1 throughout which is an expected result. This is because one of the solution components has zero derivatives and thus has no contribution in the error. Because of

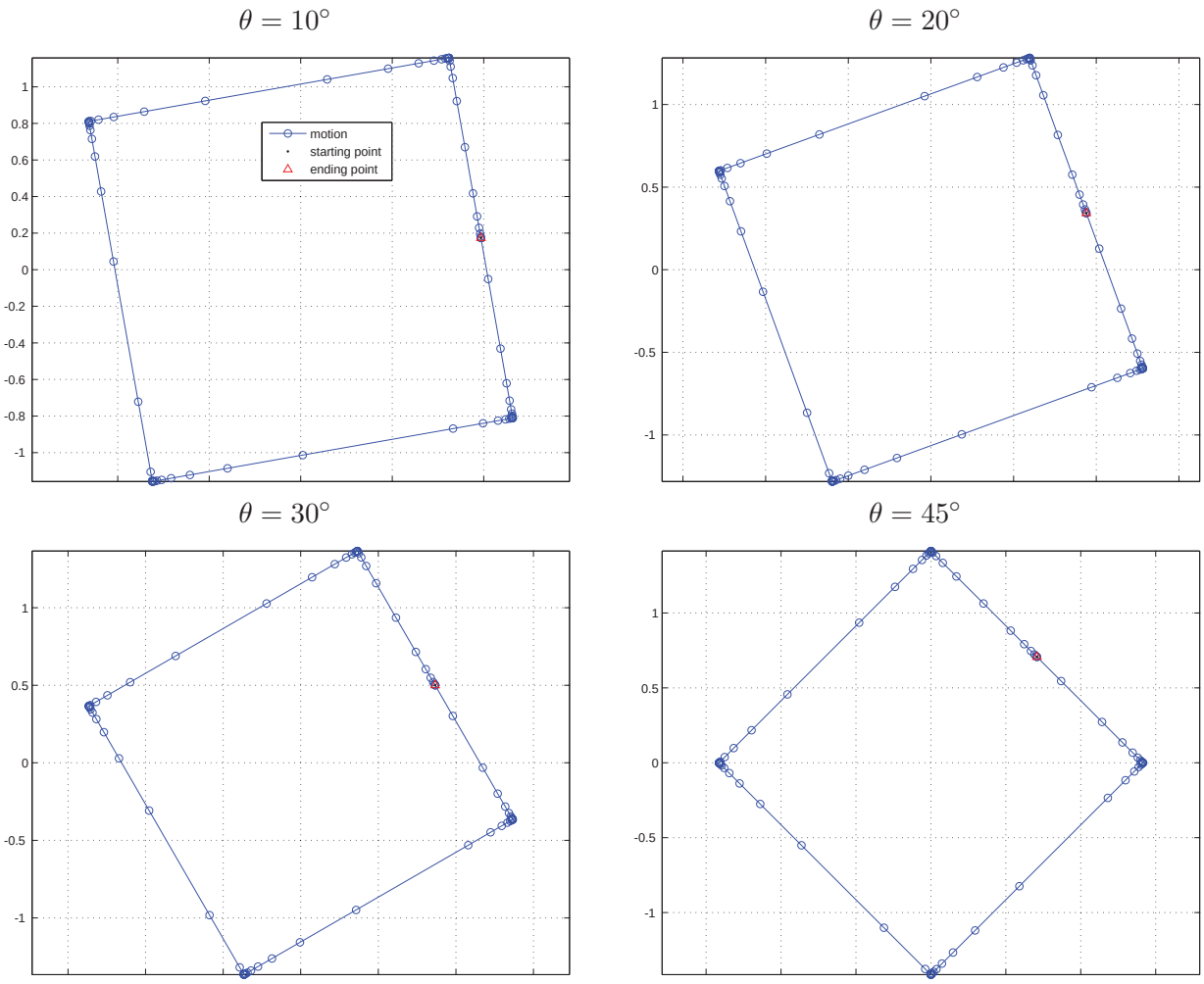


Figure 6.23: Motion in two–dimensions of the square path problem (rotated at $10°, 20°, 30°$ and $45°$, respectively).

the zero derivatives, there is a doubt whether the order is not increasing due to the problem rather than the code or methods. To clear this doubt, we also test a modified form of this square problem by rotating it through an angle $\theta = 10°, 20°, 30°, 45°$ as shown in Figure 6.23. For example, at $\theta = 10°$, the order of the method remains one throughout and similar rejections occur before crossing the corners of the square path. This shows the robustness of the code. Similar results are found when the square is rotated at the other three different angles.

6.4.5 Comparison of optimal sequences with the Lagrange controller

In order to check the performance of the Lagrange controller, it is first better to see what is expected from the controller and then to compare it with the numerical results of the code. So, the first step is to explain what the optimal sequence of the pair (h, p) would be using a set of methods. In order to understand this we need to focus on just the optimisation problem. In simple words we can say that this is a comparison of the results obtained from the Lagrange controller using the true values versus

the numerical values obtained from the implementation of the GLMs.

In the start of the code $p = 1$, to avoid the need for a starting method. There will be two choices of stepsize after the first few steps when the order is allowed to increase and the order will be changed if the controller selects order two. In general, there are three choices of methods after each successful step. Before optimising the order, the three step sizes, of each possible order, are calculated as it is possible in the present case. Using these three predicted step sizes (based on very reliable information), an optimal order is selected to make a pair (h, p) for the next step. The proposed step sizes for the next step are calculated using the formula in (5.7) which are mainly based on error estimators or more precisely on the $(p+1)^{\text{st}}$ derivative(s) for each order p. The optimal controller selects the pair that minimise the cost function in (5.4).

To understand this, it is better to first take a suitable ODE problem (taken here to be A3), fix $p_{\max} = 2$ and analyse the following steps one by one. The main aim is to check the performance of the new controller. Before going ahead, it is important to mention about the four different (h, p) sequence graphs in each of the next results. Each step is presented in each of the four plots.

Step1. (The true h–graphs) For each possible value of p, these graphs are plotted using the exact values of the higher derivatives in (5.7). The graphs plotted in this step are not (h, p)–sequence graphs though. These are just (h)–sequence graphs multiplied by some constant. Instead of the minimisation problem in (5.4), this optimisation problem can be easily converted into a maximisation problem, by multiplying the stepsize (for a method of order p) with a factor $p/(p+1)^2$. Such graphs for methods of order 1 to $p_{\max}$ (calculated with exact solutions) are shown in the first plot of the each of the next sets of figures in this section.

Step2. (Optimal upper cover of the h graphs or the optimal (h, p) sequences) As the cost function is to be maximised, so the ideal selection of step sizes and order is the upper cover of the first plot in step 1 as shown in the second plot.

Step3. (Optimal cover by the Lagrange controller with exact solutions) Before going to a problem using the GLM code, it is best to see what the Lagrange controller will do if the exact values of the step sizes (as in the first plot) of the possible orders are provided to the controller at each point. This would produce the true (h, p) sequence using the Lagrange controller as shown in plot 3. (Whereas the ideal (h, p) sequence in plot 2 is without using the Lagrange controller.)

Step4. (Optimal cover by the Lagrange controller with numerical solutions) Using the GLM code, the numerical values of the the higher derivatives are used in the Lagrange controller to obtain the optimal sequences.

We are looking for a problem that can better explain the switching of methods of various orders. In favourable circumstances, the Lagrange controller will switch the order from p to $(p+1)$ or $(p-1)$ depending upon the objective function in (5.4). This can be explained in a better way with a problem in which the stepsize graphs of different orders fluctuate and cross each other frequently. This will help to analyse the working of the (h, p)-controller.

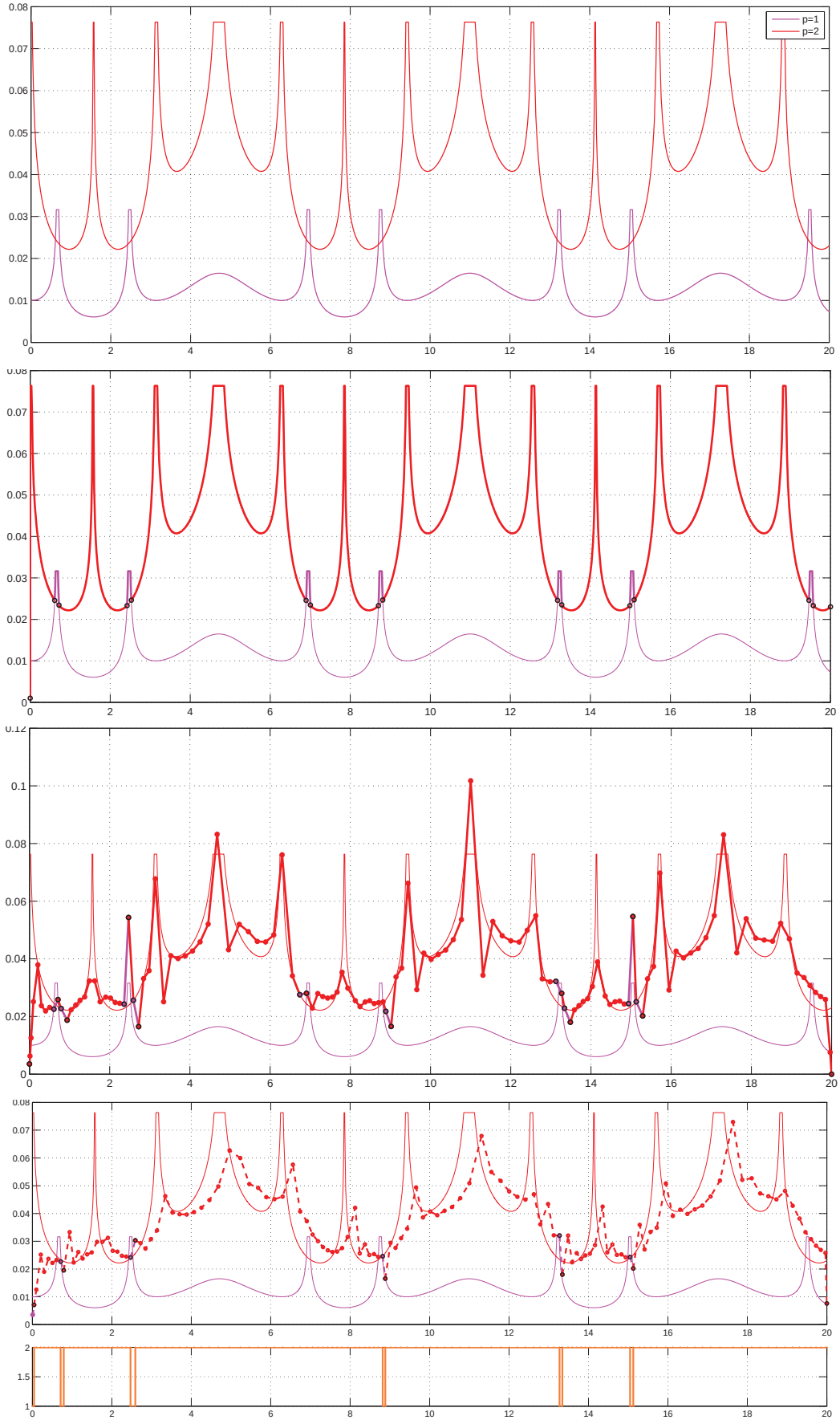


Figure 6.24: Variable–order analysis for problem A3

As the above four plots are for a maximising problem, so it is better to use an upper limit l on the optimisation function, in some of the problems. For these tests, problems A3, PR(ii), B2 and C2, are used. Here, $l = 0.1$ for the A3 problem, and for the PR problems it is $l = 0.05$. For problems B2 and C2, $l = 0$. The experiments in this section are recorded mostly with $tol = 10^{-4}$, unless otherwise mentioned.

Experiments with $p_{\max} = 2$

For example, take the PR(ii) problem. In the first plot of Figure 6.25, the pink graph represents the step sizes for the method of order one times a factor $p/(p+1)^2 = 1/4$. Whereas the red graph represents the same quantity for the method of order two. The next three plots are obtained by plotting the three different graphs in each case so that the difference from the exact one can be observed.

The maximum of the two graphs in plot 1 is the upper cover of the whole of plot 1 which is highlighted in the second plot and the black circles represents the points of integration where the order should be switched. For example, the order switches from one to two in the start and remains two before $x = \pi$ and then quickly drops back to one. It can be easily observed that, in this problem, the order switches six times in the whole interval.

Now we have to check the performance of the Lagrange controller based on the exact values of the higher derivatives in (5.7). The results are plotted in the third plot of Figure 6.25 which is quite close to the ideal upper cover in the second plot. This graph indicates what the Lagrange controller should do in the GLM code.

Finally, when the numerical results are generated using the GLM code and used in the Lagrange controller, the optimal cover is plotted in the fourth plot which is very similar to the third one.The slightly low values of the step sizes are due to some limits on the step sizes ratio r, for example the use of PI-control restricts the stepsize so it does not increase in exactly that ratio which the original stepsize controller predicts. Furthermore, the stepsize ratio is also bounded to avoid abrupt changes.

Similarly the (h, p) sequences for problems A3 and PR(ii) are presented in the Figures 6.24 and 6.25 respectively. In these three examples, $p_{\max}$ is fixed to two so that there are just two choices for p. By doing this, the behaviour of the controller is more easily understood. Now for the next few results, we will directly jump to $p_{\max} = 4$, to observe the performance of the code.

Experiments with $p_{\max} = 4$

For this the problem B2 is tested. As there is no order variation after the time $x = 9$ and above the value of 1 of the objective function, so the plots in the Figure 6.26 are plotted by fixing the axes to these limits. This figure shows that the theoretical and the numerical (h, p) sequences presented in the third and the fourth plot are quite close to each other. Similarly, problem C2 is tested with its own restricted axes as shown in Figure 6.27. This graph also shows almost the same behaviour.

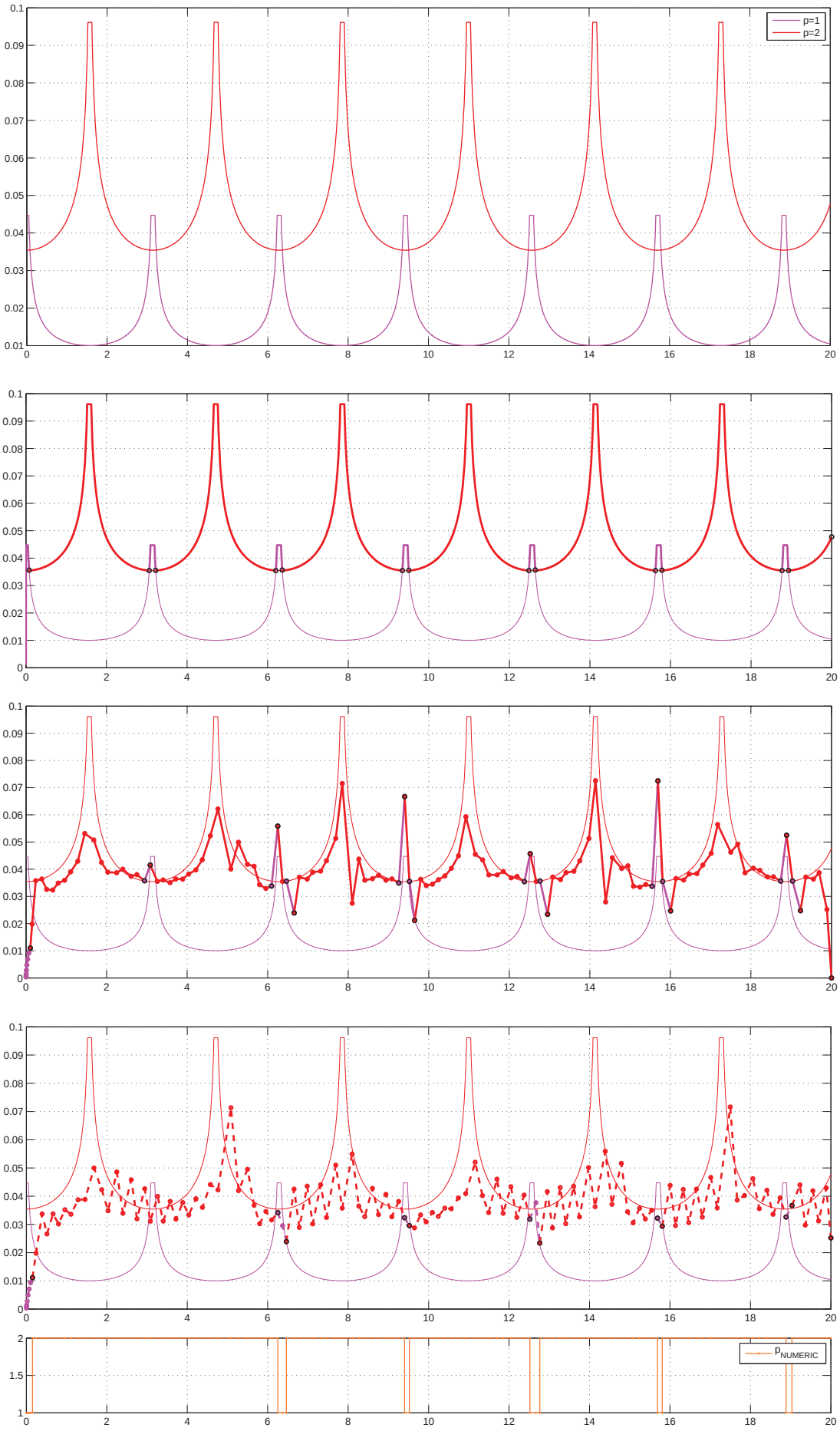


Figure 6.25: Variable–order analysis for problem PR(ii)

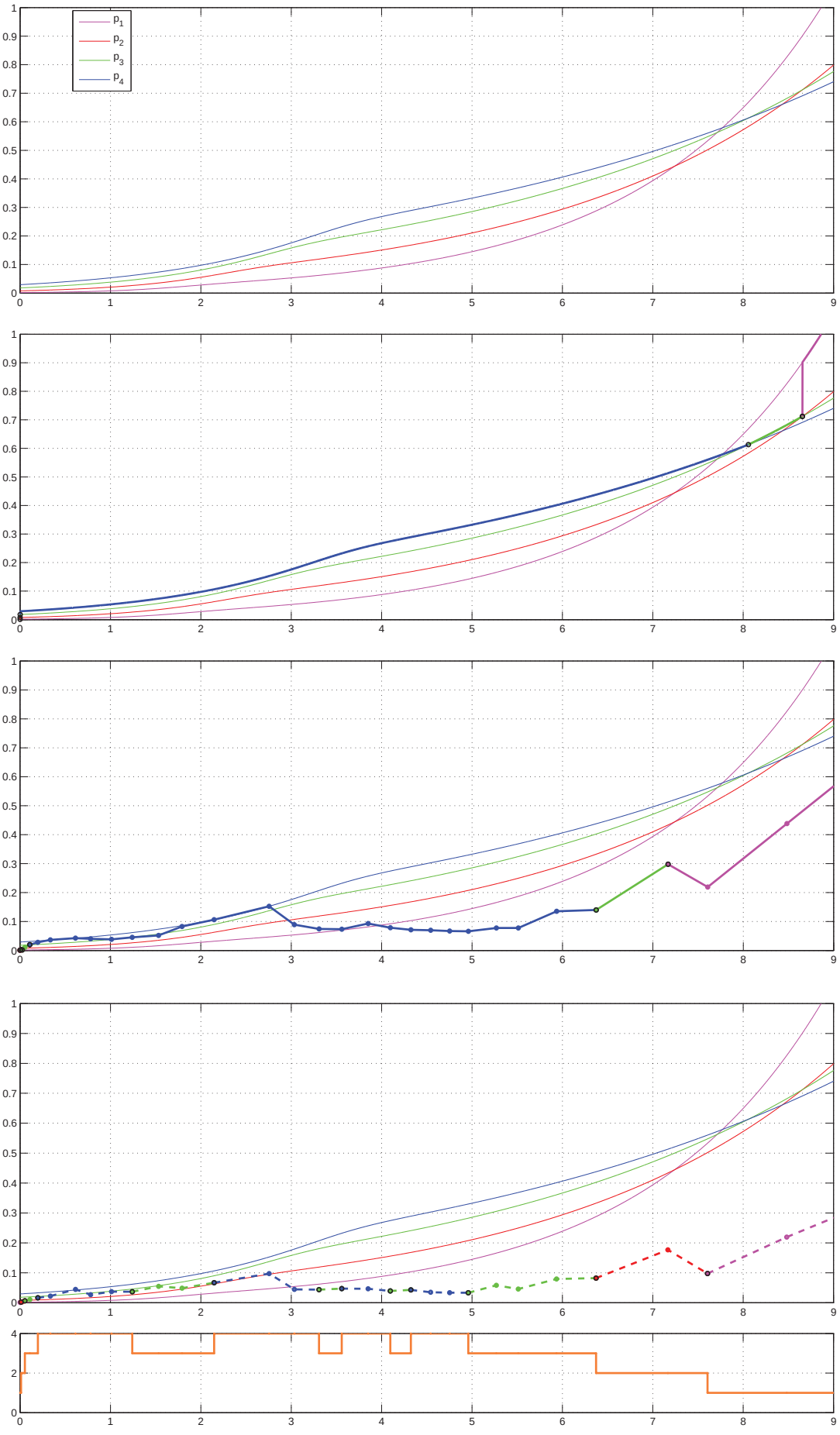


Figure 6.26: Variable–order analysis for problem B2 with $p_{\max} = 4$.

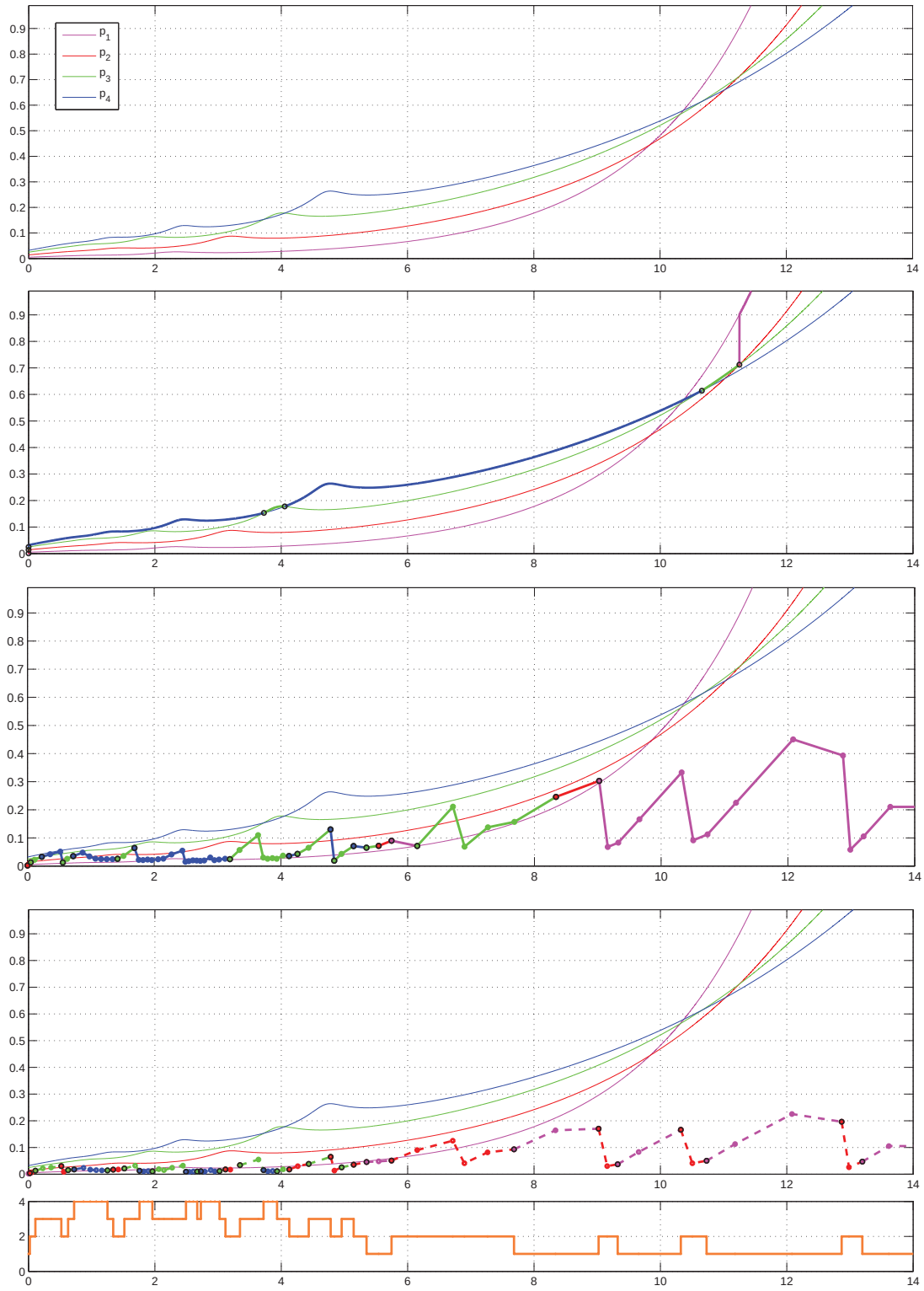


Figure 6.27: Variable–order analysis for problem C2 with $p_{\max} = 4$.

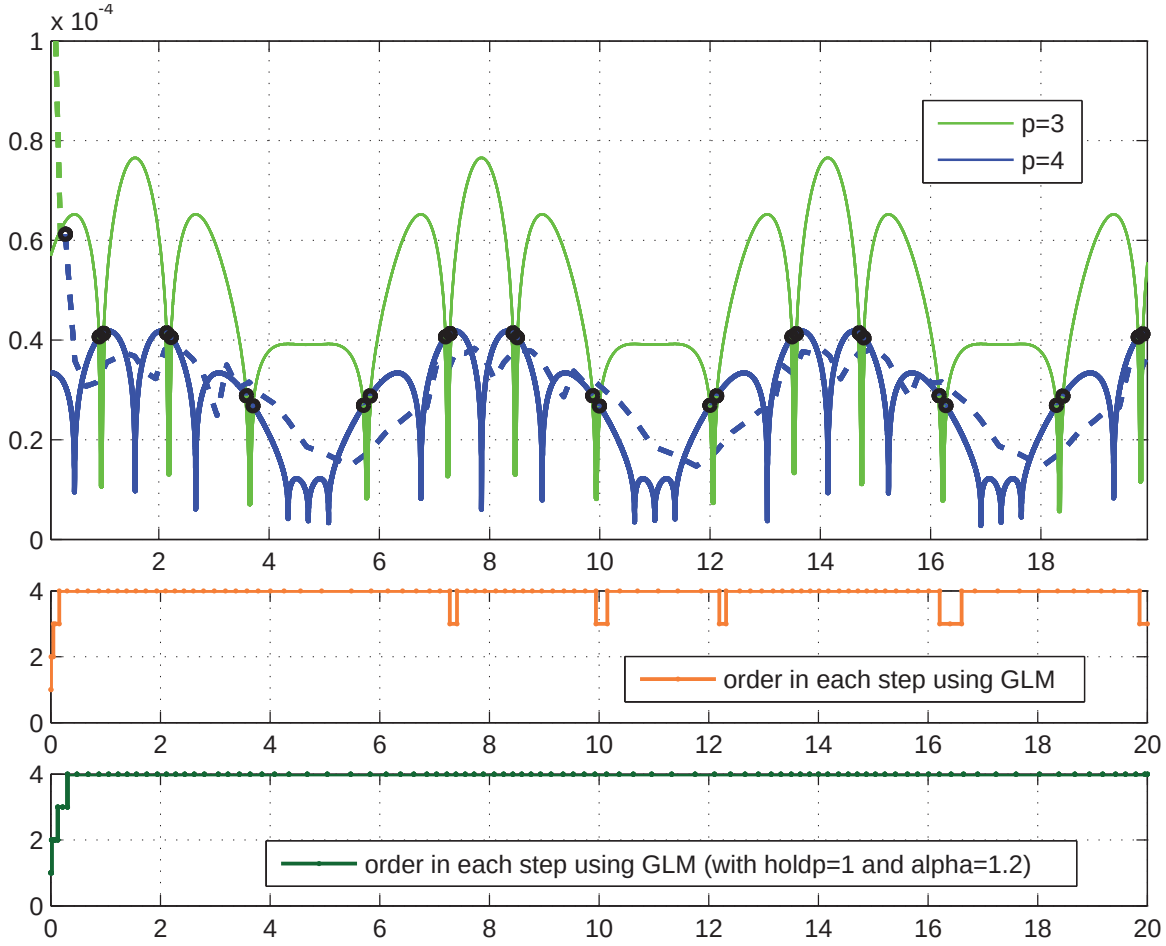


Figure 6.28: Variable–order analysis for problem A3 with $p_{\max} = 4$.

Figure 6.28 examines problem A3 with $p_{\max} = 4$, Atol= 10^{-6} and Rtol= 0, in the same way except, for this, the objective function for the minimisation problem is used. In the top–first plot of Figure 6.28, the true graphs for methods of order three and four only are shown as these two methods are mutually switched during the numerical integration. The optimal lower cover is drawn, with extra width, switching from one order to another at black circled connectors. This cover indicates what order should be followed by the method. When the Lagrange controller is used with the integrators, the path followed is almost the same as shown by the dashed lines of the same colour as defined for each order. However, sometimes, it is not easy to see exactly where the orders are switching. For this, the second plot of Figure 6.28 shows the discrete selection of the order in each step using the numerical methods. We can see that these order–oscillations are occurring when it should actually theoretically change. To overcome this, we use the strategies explained in the initial experiments of this section. For $holdp = 1$ and $\alpha = 1.2$, this experiment is repeated and no order–oscillations are found as shown in the third plot of Figure 6.28.

6.5 Conclusions

Conclusions of this chapter are:

1. The IRKS methods performs equally well (in case of fixed stepsize) as compared to the classical methods including the Runge–Kutta methods and PECE pairs.

2. In the case of variable stepsize, it is possible to use the automatic starting procedure presented in Section 4.2 without any starting method. The automatic initial stepsize controller performs efficiently.

3. The Lagrange stepsize controller and its rejection criteria works for methods of order one to four with fixed order, without the safety factor used in the traditional controllers. This has been tested and found that for each order, the numerical results obtained with fixed order, are following what we theoretically expect.

4. The Lagrange stepsize controller and its rejection criteria outperform when it is compared with the standard controller and the rejection criteria for IRKS methods with fixed order.

5. The Lagrange approach for variable order (presented in Section 5.3) has been tested for non–stiff problems as well as a discontinuous problem, physical problems and a few mildly stiff problems. The IRKS methods have been implemented using the Lagrange variable order algorithm which is efficient and robust for solving the above mentioned problems. This is verified in the numerical results presented.

7
Conclusions and future work

The main aim of this thesis is to construct a practical class of general linear methods that can be implemented and preliminarily tested in variable stepsize variable order manner using the Lagrange optimisation approach. The experiments on the test problems considered in this thesis, provide us grounds to consider the class of GLMs used in this thesis to be suitable as the kernel of a general ODE solver.

Chapter 1 consisted of a brief classification of the ODEs which are to be considered whilst solving the ODEs numerically. This chapter was a review of those traditional methods that are successfully used in the existing numerical solvers for various types of problems. Along with this, the advantages and disadvantages of such methods were discussed. On the basis of these, it was shown that we need to look for a better class in the most general form of numerical method known as general linear methods.

Chapter 2 was a brief review of general linear methods. In spite of repeating the whole theory of GLMs, the only theoretical part that needs some attention before the construction of methods in the next chapter, is to know about the stability matrix in the case of GLMs. This was briefly presented along with a few plots of the stability regions of all the numerical methods used in this thesis including a table of those methods as well.

Chapter 3 was a detailed review of the construction of practical GLMs with IRKS– and F–properties; a subclass of type 1 DIMSIMs. At the start, it was presented how a few of the classical methods behave naturally like special GLMs and this motivates us to look for a big family of practical methods known as DIMSIMs. A design of these methods was also presented here. The order conditions for constructing GLMs are very complex. For ease of these, some simplifying assumptions were also presented including the F–property for GLMs. On the basis of all these conditions, practical derivation

of these GLMs is summarised in the form of an algorithm along with an example for the method of order three.

Chapter 4 was based on many implementation techniques related to GLMs. In this chapter, some implementation issues of the existing solvers were presented. The starting procedure of an ODE solver is not straight forward, it requires many parameters to be set either by the user or automatically by the code. An ideal procedure consisting of a few automatic techniques was presented here that just starts with the initial value as an input and neither a starting vector from the user nor a special starting method are required for it. How this type of procedure is possible to use with the GLMs used in this thesis, was discussed in detail. Stepsize controllers were also presented along with the techniques of balancing the data vector to avoid the loss of stability. To construct the efficient and reliable error estimators, the growth of errors due to the methods were presented here. This was followed by a theorem showing the possible and reliable construction of the error estimators required of each of the GLMs in tables 2.1 and 2.2. These estimators were also preliminarily tested and reported in this chapter. At the end of this chapter, it was discussed in detail why variable stepsize variable order design of the ODE solvers are needed. Along with this, some of the existing variable order designs were presented which are mostly based on heuristic approaches.

Chapter 5 was the main development in which it was discussed how the variable stepsize variable order implementation of the numerical methods for ODEs forms a dual problem of optimisation. Using the theory of multivariate calculus, it was justified how the selection of the pair, stepsize and order, after each acceptable step can be considered as an unconstrained optimisation problem. Further it was discussed and shown how the actual tolerance used for any particular order is related to the Lagrange multiplier in the optimisation problem. Also, a stepsize controller based on the Lagrange approach was constructed and this helps to avoid some heuristic parameters like the safety factor used in almost every standard controllers. Along with this, a rejection criteria was developed which allows us to give some relaxation to the standard acceptance criteria so that too early rejections can be avoided. This rejection criteria is based on the stepsize ratio r rather than the estimated error as in the case of standard techniques. At the end, a variable stepsize variable order algorithm based on the above mentioned design techniques of Lagrange theory was presented. It was followed by another algorithm for the scale and modify technique that is necessary for the implementation of variable stepsize variable order code.

Chapter 6 investigated the new methods and implemented these with the new design based on the Lagrange theory. At the start, the numerical verification of the order(s) of the new methods in the simple case of fixed stepsize were reported. The code was then extended to the variable stepsize option. A modified approach for initial stepsize that is possible and automatic in the existing case was also tested and this worked better than the trivial approach. In the variable stepsize mode, it was verified (numerically) using the Lagrange theory and multivariate calculus that the methods show the correct order behaviour. Also, a preliminarily comparison of the Lagrange approaches for stepsize and rejection with the standard techniques was reported and it was found that the new approaches are very efficient in comparison. The last section reported the main results of the thesis in which the whole design of Lagrange approaches were used to implement the IRKS methods with F–properties.

For this, first some linear single dimensional and then some multidimensional problems including the mildly stiff PR-problems were tested. For each problem, the behaviour of the solution components were compared with the exact solution at each point of integration. Along with this the step sizes, rejections and the order of the methods used in each step were also presented in the form of graphs. After the satisfactory validation with these problems, some physical problems were tested. To check the robustness of the (h, p) controller, an interesting discontinuous problem named the square path problem was examined in depth. It was discussed in detail and thoroughly investigated that the GLM code handled the discontinuities at the corners of the square efficiently. To check the performance of the optimisation controller, the (h, p) sequences obtained from the numerical results were compared with the optimal sequences.

This thesis studies the use of Lagrange optimisation approach for the implementation of IRKS methods for solving a set of non–stiff problems, discontinuous problems, mildly stiff problems and some geometric problems. Here are a few conclusions:

- It is possible to implement GLMs in a variable order mode, in a sophisticated way, which is not possible in the case of Runge–Kutta methods.
- The Lagrange approach for a variable stepsize controller has been successfully used for the efficient implementation of the GLMs. The results shows that the unnecessary rejections due to empirical safety factors in the standard controller, can be successfully avoided using the Lagrange controller.
- The order–oscillations in the variable–order mode of the solver, for example in problem A3, (using the Lagrange controller) are theoretically correct and can be avoided.
- This code not only solves the non–stiff problems efficiently but is also capable of handling mildly stiff problems and discontinuous problems.
- We replace some heuristic values which occur in the design of the ODE software with automatic selections. These include the safety factor used for avoiding too many rejections while selecting the new stepsize. Another important feature are the rejection criteria necessary for variable stepsize implementation (with either fixed or variable order). Now, the new rejection criteria are more logical and comparatively relaxed in a sense that they allow us to accept the step as long as the error is not 'too much' beyond the required accuracy. This 'too much' is precisely defined in terms of Lagrange theory which is a $(p + 1)/p$ proportion of the tolerance for the GLMs.

All these conclusions collectively give a significant notion that the implementation of the IRKS methods with F–properties, in the Lagrange design can be considered a step towards making the GLMs the kernel of a general ODE solver. Here are a few directions for future work, we would like to work on:

- Construct the GLMs with IRKS and F–property of higher order. For this some sophisticated search routines can be used that can help to find suitable free parameters. For example, the methods used in this thesis are of explicit nature and as these are IRKS methods, the spectrum

of the matrix V was set to have just one nonzero eigenvalue. This idea can be extended to more than one non–zero eigenvalue inside the unit disc. This will give further opportunities to explore a wider range of methods in the search for the most efficient methods.

- Review of the Lagrange (h, p)–control techniques for higher order methods.
- Evaluation of this code by comparing with other standard methods on some standard test sets.
- Review of the designs of stiff codes and extending the techniques used in the non–stiff case.
- Construct an efficient and robust implementation of the methods for stiff and non–stiff cases in high-level languages using stiffness switching.
- As the GLM code can handle discontinuities, so it can be reviewed on DDEs these being a sub class of discontinuous problems.

Appendix

Test problems

The problem set can be considered as a set of three types of problems; non-stiff problems, mildly stiff problems and some discontinuous problems. First, the DETEST set is included here consisting of six problem classes. The problems have interval $[0, 20]$ unless otherwise mentioned.

A1. the exponential

$$y' = y, \qquad y(0) = 1,$$

A2. a special case of the Riccatti equation

$$y' = -\frac{y^3}{2}, \qquad y(0) = 1,$$

A3. an oscillatory problem

$$y' = y\cos(x), \qquad y(0) = 1,$$

A3-2D. 2D–version of A3 (autonomous form)

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

A3-3D. 3D–version of A3 (autonomous form)

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

A4. a logistic curve

$$y' = \frac{y}{4}(1 - \frac{y}{20}), \qquad y(0) = 1,$$

B2. Linear chemical reaction

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

C1. a radioactive decay chain

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ \vdots \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -1 & 0 & \dots & \dots & 0 \\ 1 & -1 & \ddots & & \vdots \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 & 0 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{10} \end{bmatrix}, \qquad y(0) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}.$$

C2. another radioactive decay chain

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ \vdots \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -1 & 0 & \dots & \dots & 0 \\ 1 & -2 & \ddots & & \vdots \\ 0 & 2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -9 & 0 \\ 0 & \dots & 0 & 9 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{10} \end{bmatrix}, \qquad y(0) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}.$$

Kepler problem. Kepler's Orbit equations

$$\begin{aligned} y_1^{'} &= y_3, & y_1(0) &= 1-e, \\ y_2^{'} &= y_4, & y_2(0) &= 0, \\ y_3^{'} &= -\tfrac{y_1}{(y_1^2+y_2^2)^{3/2}}, & y_3(0) &= 0, \\ y_4^{'} &= -\tfrac{y_2}{(y_1^2+y_2^2)^{3/2}}, & y_4(0) &= \sqrt{\tfrac{1+e}{1-e}}. \end{aligned}$$

where e is eccentricity.

E2. Van der Pol

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2, \\ y_2' &= (1-y_1^2)y_2 - y_1, & y_2(0) &= 0, \end{aligned}$$

with $x \in [0, 8]$.

3-body problem.

$$\begin{aligned} y_1^{'} &= y_4, & & \\ y_2^{'} &= y_5, & y_1(0) &= 0.994, \\ y_3^{'} &= y_6, & y_2(0) &= 0, \\ y_4^{'} &= 2y_5 + y_1 - \tfrac{\mu(y_1+\mu-1)}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}} - \tfrac{(1-\mu)(y_1+\mu)}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}}, & y_3(0) &= 0, \\ y_5^{'} &= -2y_4 + y_2 - \tfrac{\mu y_2}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}} - \tfrac{(1-\mu)y_2}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}}, & y_4(0) &= 0, \\ y_6^{'} &= -\tfrac{\mu y_3}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}} - \tfrac{(1-\mu)y_3}{(y_2^2+y_3^2+(y_1+\mu-1)^2)^{3/2}}, & y_5(0) &= -2.0015851063790825224, \\ & & y_6(0) &= 0. \end{aligned}$$

where $\mu = \frac{1}{81.45}$ and $x \in [0, 19.14045706162071]$.

Pend. Simple Pendulum

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

HO. Harmonic Oscillator

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

Brusselator.

$$\begin{aligned} y_1' &= 1 + y_1^2 y_2 - 4y_1, & y_1(0) &= 1.5, \\ y_2' &= 3y_1 - y_1^2 y_2, & y_2(0) &= 3. \end{aligned}$$

Bubble. A model of cavitating bubble is

$$\begin{aligned} \tfrac{dy_1}{dx} &= y_2, & y_1(0) &= 1, \\ \tfrac{dy_2}{dx} &= \tfrac{5\exp\left(\frac{-x}{x^*}\right)-1-1.5y_2^2}{y_1} - \tfrac{ay_2+D}{y_1^2} + \tfrac{1+D}{y_1^{3\gamma+1}}. & y_2(0) &= 0. \end{aligned}$$

$$\begin{aligned} x^* &= \frac{0.029}{R_0}, \\ a &= \frac{4\times 10^{-5}}{R_0}, \\ D &= \frac{1.456\times 10^{-4}}{R_0}, \\ \gamma &= 1.4, \\ R_0 &= 10^{-3}. \end{aligned}$$

Square.

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} (\chi(y_1,y_2)-1)\,\mathrm{sgn}(y_2) \\ \chi(y_1,y_2)\mathrm{sgn}(y_1) \end{pmatrix} \qquad \begin{aligned} y_1(0) &= 1, \\ y_2(0) &= 0. \end{aligned}$$

where

$$\chi(y_1,y_2) = \begin{cases} 1 & \text{if } |y_1| > |y_2| \\ 0 & \text{otherwise} \end{cases}$$

with $x \in [0,8]$ so that the length of interval is exactly equal to the perimeter of a square of side two units.

PR. This problem is use to examine the stability behaviour of the numerical methods

$$y' = \lambda(y - f(x)) + f'(x), \quad \text{with true solution} \quad y(x) = f(x).$$

Here two cases are considered:

$$(i) f(x) = \exp(\mu x) \quad \text{where} \quad y(0) = 1, \quad \lambda = -0.1, \quad \mu = -0.1,$$
$$(ii) f(x) = \sin(x) \quad \text{where} \quad y(0) = 1, \quad \lambda = -4.$$

mild1.

$$y_1' = -16y_1 + 12y_2 + 16\cos(x) - 13\sin(x),$$
$$y_2' = 12y_1 - 9y_2 - 11\cos(x) + 9\sin(x),$$

with

$$y(0) = [1, 0]^{\mathsf{T}} \quad \text{and} \quad x \in [0, \pi].$$

Bibliography

[1] S. Ahmad, *Efficient implementation of implicit Runge–Kutta methods*, Masters thesis, The University of Auckland, 2011.

[2] R. Alexander, *Diagonally implicit Runge–Kutta methods for stiff ODEs*, SIAM J. Numer. Anal. 14, 1006-1021, 1977.

[3] G. Bader, P. Deuflhard, *A semi-implicit mid-point rule for stiff systems of ordinary differential equations*, Numer. Math. 41, 373-398, 1983.

[4] D. Barton, *On Taylor series and stiff equations*, ACM Trans. Math. Software 6, 280-294, 1980.

[5] D. Barton, I.M. Willers, R. V. M. Zahar, *The automatic solution of systems of ordinary differential equations by the method of Taylor series*, Comput. J. 14, 243-248, 1971.

[6] D. Barton, I.M. Willers, R. V. M. Zahar, *Taylor series methods for ordinary differential equations–an evaluation*, Mathematical Software, John Rice (Ed.), Academic Press, New York, 369-390, 1971.

[7] T. A. Bickart, Z. Picel, *High order stiffly stable composite multistep methods for numerical integration of stiff differential equations*, BIT 13, 272-286, 1973.

[8] P. N. Brown, G. D. Byrne, A. C. Hindmarsh, *VODE: a variable-coefficient ODE solver*, SIAM J. Sci. Statist. Comput. 10, 1038-1051, 1989.

[9] R. Bulirsch, J. Stoer, *Numerical treatment of ordinary differential equations by extrapolation methods*, Numer. Math. 8, 1-13, 1966.

[10] K. Burrage, J. C. Butcher, *Stability criteria for implicit Runge–Kutta methods*, SIAM J. Numer. Anal. 16, 46-57, 1979.

[11] K. Burrage, J. C. Butcher, *Non-linear stability of a general class of differential equation methods*, BIT 20, 185-203, 1980.

[12] K. Burrage, J. C. Butcher, F. H. Chipman, *An implementation of singly-implicit Runge–Kutta methods*, BIT 20, 326-340, 1980.

[13] J. C. Butcher, *Coefficients for the study of Runge–Kutta integration processes*, J. Austral. Math. Soc. 3, 185-201, 1963.

[14] J. C. Butcher, *Implicit Runge–Kutta Processes*, Math. Comp. 18, 50-64, 1964.

[15] J. C. Butcher, *On the convergence of numerical solutions to ordinary differential equations*, Math. Comp. 20, 1-10, 1966.

[16] J. C. Butcher, *An algebraic theory of integration methods*, Math. Comp. 26, 79-106, 1972.

[17] J. C. Butcher, *On the Implementation of Implicit Runge–Kuta Methods*, BIT 16, 237-240, 1976.

[18] J. C. Butcher, *Linear and non-linear stability for general linear methods*, Report 28, The University of Auckland, 1982.

[19] J. C. Butcher, *General linear methods: a survey*, Appl. Numer. Math. 1, 273-284, 1985.

[20] J. C. Butcher, *Optimal order and stepsize sequences*, IMA J. Numer. Anal. 6, 433-438, 1986.

[21] J. C. Butcher, *Linear and non–linear stability for general linear methods*, BIT 27, 182-189, 1987.

[22] J. C. Butcher, *The equivalence of algebraic stability and AN stability*, BIT 27, 510-533, 1987.

[23] J. C. Butcher, *Order, stepsize and stiffness switching*, Computing 44, 209-220, 1990.

[24] J. C. Butcher, *The adaptation of STRIDE to delay differential equations*, Appl. Numer. Math. 9, 415-425, 1992.

[25] J. C. Butcher, *Diagonally implicit multistage integration methods*, Appl. Numer. Math. 11, 347-363, 1993.

[26] J. C. Butcher, *General linear methods for the parallel solution of ordinary differential equations*, World Sci. Ser. Appl. Anal. 2, 99-111, 1993.

[27] J. C. Butcher, *A transformation for the analysis of DIMSIMs*, BIT 34, 25-32, 1994.

[28] J. C. Butcher, *An introduction to DIMSIMs*, Comput. Appl. Math. 14, No.1, 59-72, 1995.

[29] J. C. Butcher, *General Linear Methods*, Computers Math. Applic. 31, 105-112, 1996.

[30] J. C. Butcher, *An introduction to "Almost Runge–Kutta methods"*, Appl. Numer. Math. 24, 331-342, 1997.

[31] J. C. Butcher, *General linear methods for stiff differential equations*, BIT 41, No.2, 240-264, 2001.

[32] J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley and Sons, 2008.

[33] J. C. Butcher, *Software issues for ordinary differential equations*, Numerical Algorithms 31, 401-418, 2002.

[34] J. C. Butcher, *General linear methods*, Acta Numerica 15, 157-256, 2006.

[35] J. C. Butcher, *Thirty years of G–stability*, BIT 46, 479-489, 2006.

[36] J. C. Butcher, *General linear methods for ordinary differential equations*, Math. Comput. Simulation 79, 1834-1845, 2009.

[37] J. C. Butcher, P. Chartier, *Parallel general linear methods for stiff ordinary and differential algebraic equations*, Appl. Numer. Math. 17, 213-222, 1995.

[38] J. C. Butcher, P. Chartier, *A generalisation of singly-implicit Runge–Kutta methods*, Appl. Numer. Math. 24, 343-350, 1997.

[39] J. C. Butcher, P. Chartier, *The effective order of singly-implicit Runge–Kutta methods*, Numerical Algorithms 20, 269-284, 1999.

[40] J. C. Butcher, P. Chartier, Z. Jackiewicz, *Nordsieck representation of DIMSIMs*, Numerical Algorithms 16, 209-230, 1997.

[41] J. C. Butcher, P. Chartier, Z. Jackiewicz, *Experiments with variable–order type 1 DIMSIM code*, Numerical Algorithms 22, 237-261, 1999.

[42] J. C. Butcher, M. T. Diamantakis, *DESIRE: diagonally extended singly implicit Runge–Kutta effective order methods*, Numerical Algorithms 17, 121-145, 1998.

[43] J. C. Butcher, Y. Habib, A. T. Hill, T. J. T. Norton, *The control of parasitism in G–symplectic methods*, SIAM J. Numer. Anal. 52, 2440-2465, 2014.

[44] J. C. Butcher, A. D. Heard, *Stability of numerical methods for ordinary differential equations*, Numerical Algorithms 31, 75-85, 2002.

[45] J. C. Butcher, A. T. Hill, *Linear multistep methods as irreducible general linear methods*, BIT 46, 5-19, 2006.

[46] J. C. Butcher, Z. Jackiewicz, *Diagonally implicit general linear methods for ordinary differential equations*, BIT 33, 452-472, 1993.

[47] J. C. Butcher, 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.

[48] J. C. Butcher, Z. Jackiewicz, H. D. Mittelmann, *A nonlinear optimisation approach to the construction of general linear methods of high order*, J. Comput. Appl. Math. 81, 181-196, 1997.

[49] J. C. Butcher, Z. Jackiewicz, *Implementation of diagonally implicit multistage integration methods for ordinary differential equations*, SIAM J. Numer. Anal. 34, 2119-2141, 1997.

[50] J. C. Butcher, Z. Jackiewicz, *Construction of high order diagonally implicit multistage integration methods for ordinary differential equations*, Appl. Numer. Math. 27, 1-12, 1998.

[51] J. C. Butcher, Z. Jackiewicz, *A reliable error estimation for diagonally implicit multistage integration methods*, BIT 41, 656-665, 2001.

[52] J. C. Butcher, Z. Jackiewicz, *Error estimation for Nordsieck methods*, Numerical Algorithms 31, 75-85, 2002.

[53] J. C. Butcher, Z. Jackiewicz, *A new approach to error estimation for general linear methods*, Numer. Math. 95, 487-502, 2003.

[54] J. C. Butcher, Z. Jackiewicz, *Construction of general linear methods with Runge–Kutta stability properties*, Numerical Algorithms 36, 53-72, 2004.

[55] J. C. Butcher, Z. Jackiewicz, *Unconditionally stable general linear methods for ordinary differential equations*, BIT 44, 557-570, 2004.

[56] J. C. Butcher, Z. Jackiewicz, W. M. Wright, *Error propagation of general linear methods for ordinary differential equations*, J. Complexity 23, 560-580, 2007.

[57] J. C. Butcher, N. Rattenbury, *ARK methods for stiff problems*, Appl. Numer. Math. 53, 165-181, 2005.

[58] J. C. Butcher, H. Podhaisky, *On error estimation in general linear methods for stiff ODEs*, Appl. Numer. Math. 56, 345-357, 2006.

[59] J. C. Butcher, W. M. Wright, *The construction of practical general linear methods*, BIT 43, 695-721, 2003.

[60] J. C. Butcher, W. M. Wright, *A transformation relating explicit and diagonally implicit general linear methods*, Appl. Numer. Math. 44, 313-327, 2003.

[61] J. C. Butcher, W. M. Wright, *Application of doubly companion matrices*, Appl. Numer. Math. 56, 358-373, 2006.

[62] D. Chen, *The effective order of singly-implicit methods for stiff differential equations*, PhD thesis, The University of Auckland, 1998.

[63] G. Dahlquist, *Convergence and stability in the numerical integration of ordinary differential equations*, Math. Scand. 4, 33-53, 1956.

[64] G. Dahlquist, *A special stability problem for linear multistep methods*, BIT 18, 27-43, 1963.

[65] J. Donelson, E. Hansen, *Cyclic composite multistep predictor-corrector methods*, SIAM J. Numer. Anal. 8, 137-157, 1971.

[66] B. L. Ehle, *On Padé approximations to the exponential function and A–stable methods for the numerical solution of initial value problems*, Research Report CSRR 2010, Dept. AACS, Univ. of Waterloo, Ontario, Canada, 1969.

[67] C. W. Gear, *Algorithm 407, DIFSUB for solution of ordinary differential equations*, Communications of the ACM 14, 447-451, 1971.

[68] A. Gibbons, *A program for the automatic integration of differential equations using the method of Taylor series*, Comput. J. 3, 108-111, 1960.

[69] I. Gladwell, L.F. Shampine, R.W. Brankin, *Automatic selection for the initial stepsize for an ODE solver*, J. Comput. Appl. Math. 18, 175-192, 1987.

[70] A. Gorgey, *Extrapolation of symmetrized Runge–Kutta methods*, PhD thesis, The University of Auckland, 2012.

[71] K. Gustafsson, M. Lundh, G. Söderlind, *A PI stepsize control for the numerical solution of ordinary differential equations*, BIT 28, 270-287, 1988.

[72] K. Gustafsson, *Control theoretic techniques for the stepsize selection in implicit Runge–Kutta Methods*, ACM Trans. Math. Software 20, 496-517, 1994.

[73] E. Hairer, S.P. Nørsett, G. Wanner, *Solving Ordinary Differential Equations I*, Springer, Berlin, 1993.

[74] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II*, Springer, Berlin, 1996.

[75] E. Hairer, G. Wanner, *Stiff differential equations solved by Radau methods*, J. Comput. Appl. Math. 111, 93-111, 1999.

[76] A. C. Hindmarsh, *LSODE and LSODI, two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter 15, 10-11, 1980.

[77] T. E. Hull, W. H. Enright, B. M. Fellen, A. E. Sedgwick, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal. 9, No.4, 603-637, 1972.

[78] Z. Jackiewicz, *General linear methods for ordinary differential equations*, John Wiley and Sons, 2009.

[79] W. E. Milne, *Numerical solution of Differential Equations. Second revised and enlarged edition. With an appendix on partial differential equations by R.R. Reynolds*, Dover Publications Inc., New York, 1970.

[80] A. Nordsieck, *On numerical integration of ordinary differential equations*, Math. Comp. 16, 22-49, 1962.

[81] W. Romberg, *Vereinfachte numerische Integration*, Norske Vid. Selsk. Forh., Trondheim 28, 30-36, 1955.

[82] C. Runge, *Über die numerische Auflösung von Differentialgleichungen*, Math. Ann. 46, 167-178, 1895.

[83] G. Söderlind, L. Jay, M. Calvo, *Stiffness 1952–2012: Sixty years in search of a definition*, BIT 55, 531-558, 2015.

[84] J. Stoer, R. Bulirsch, *Introduction to numerical analysis*, 2nd edn. Berlin: Springer, 1992.

[85] W. Sweatman, *Vector rational interpolation algorithms of Bulirsch–Stoer–Neville form*, Proc. R. Soc. Lond. A 454, 1923-1932, 1998.

[86] W. M. Wright, *General linear methods with Inherent Runge–Kutta stability*, PhD thesis, The University of Auckland, 2002.